



Kivy Documentation

Release 1.9.2.dev0

www.kivy.org

Sumário

I ATENÇÃO	3
II COLABORADORES	5
III Bem-vindo ao Kivy	7
1 Guia de Usuário	9
1.1 Instalação	9
1.1.1 Versão Estável	10
Instala no Windows	10
Instalação	11
O que são Wheels, pip e Wheel	11
Instalação do Nightly Wheel	12
Dependências do Kivy	12
Linha de Comando	13
Use a Versão de Desenvolvimento	13
Compile Kivy	15
Instalando O Kivy num local alternativo	15
Tornando o Python disponível em qualquer lugar	16
Atualizando desde uma versão anterior do Kivy dist	16
Instalação no OS X	16
Usando o Homebrew com pip	17
Usando MacPorts com pip	17
Usando o <i>Kivy.app</i>	18
Instalação dos Módulos	19
Onde os módulos / arquivos estão instalados?	19
Para instalar os arquivos binários	19
Para incluir outros frameworks	19

Inicie qualquer aplicação Kivy	20
Inicie desde a Linha de Comando	20
Instalação no Linux	20
Usando pacotes de software	20
Ubuntu / Kubuntu / Xubuntu / Lubuntu (Saucy e acima)	20
Debian (Jessie ou versões recentes)	21
Linux Mint	22
Bodhi Linux	22
OpenSuSE	22
Fedora	22
Gentoo	23
Installation in a Virtual Environment	23
Dependências comuns	23
Cython	23
Dependencies with SDL2	24
Exemplo com Ubuntu	24
Instalação	24
Dependencies with legacy PyGame	25
Exemplo com Ubuntu	25
Fedora	25
OpenSuse	26
Instalação	26
Instale pacotes adicionais do Virtualenv	27
Inicie desde a Linha de Comando	27
Device permissions	29
Instalação no Android	29
Instalação no Raspberry Pi	29
Instalação Manual (No Raspbian Jessie)	30
Manual de instalação (No Raspbian Wheezy)	30
Distribuição KivyPie	31
Executando a Demonstração	31
Altere a tela padrão usando	32
Usando a tela de toque oficial RPi	32
Pra onde ir?	32
1.1.2 Versão de Desenvolvimento	32
Instalação de Dependências	33
Ubuntu	33
OS X	33
OSX HomeBrew	35
Windows	35
Instalando a versão de Desenvolvimento do Kivy	35
Roda o conjunto de testes	36
Desinstalando Kivy	36
1.2 Filosofia	37
1.2.1 Porque se importar?	37

Recente	37
Rápido	37
Flexível	38
Focado	38
Financiado	38
Gratuito	39
1.3 Contribuição	39
1.3.1 Avaliação	39
1.3.2 Reportar um requerimento	39
1.3.3 Contribuição Código	40
Estilo de Códificaçāo	40
Performance	40
Git e GitHub	40
Fluxo no Código	41
1.3.4 Contribuições para a Documentação	43
Docstrings	43
1.3.5 Contribuições para Teste Unitários	45
Testes Unitários	45
Como Isso Funciona	46
Testes Unitários GL	47
Escrevendo Testes Unitários GL	47
Relatórios de Cobertura	48
1.3.6 GSOC	48
Google Summer of Code - 2017	49
Prefácio	49
Requisitos	49
Como começar	50
Idéias do projeto	50
Projetos para iniciantes	50
Projetos Intermediários	51
Projetos avançados	52
Como entrar em contato com os desenvolvedores	54
Como ser um bom aluno	55
O que esperar se você for escolhido	56
1.4 FAQ	56
1.4.1 FAQ Técnico	56
Fatal Python error: (pygame parachute) Segmentation Fault	56
simbolo indefinido: glGenerateMipmap	57
ImportError: Nenhum módulo chamado evento	57
1.4.2 FAQ Android	57
Não possível extrair dados públicos	57
Dá pau na interação de toque na versão do Android 2.3.x	57
É possível ter um aplicativo Kosk no Android 3.0?	57
Qual é a diferença entre python-to-android do Kivy e SL4A?	58
1.4.3 FAQ Projeto	58
Por que você usa Python? O mesmo não é lento?	58

O Kivy suporta Python 3.x?	59
Como Kivy está relacionado ao PyMT?	59
Você aceita patches?	60
O projeto Kivy participa do Google Summer of Code?	60
1.5 Contate-nos	61
1.5.1 Rastreador de problemas	61
1.5.2 Correio	61
1.5.3 IRC	61
2 Guia de Programação	63
2.1 Básico do Kivy	63
2.1.1 Instalação do Ambiente Kivy	63
2.1.2 Criando um Aplicação	64
2.1.3 Ciclo de Vida de uma Aplicação Kivy	64
2.1.4 Executando o aplicativo	66
2.1.5 Personalizando a Aplicação	67
2.2 Controlando o Environment	69
2.2.1 Controle do caminho	69
2.2.2 Configuração	70
2.2.3 Restringir o núcleo à implementação específica	71
2.2.4 Métricas	71
2.2.5 Gráficos	72
2.3 Configurar p Kivy	72
2.3.1 Localizando o arquivo de configuração	73
2.3.2 Local configuration	73
2.3.3 Entendendo os Tokens de configuração	74
2.4 Visão geral da Arquitetura	74
2.4.1 Provedores Principais e Provedores de Entrada	75
2.4.2 Gráficos	76
2.4.3 Core	76
2.4.4 UIX (Widgets & Layouts)	77
2.4.5 Módulos	77
2.4.6 Eventos de Entrada (Toques)	77
2.4.7 Widgets e Despachadores de Eventos	78
2.5 Eventos e Propriedades	79
2.5.1 Introdução para o Despachador de Eventos	80
2.5.2 Main Loop	80
Agendando um Evento Repetitivo	81
Agendamento de um Evento Unico	82
Trigger Events	82
2.5.3 Eventos de Widgets	83
2.5.4 Criando um Evento Personalizado	83
2.5.5 Anexando callbacks	84
2.5.6 Introdução às Propriedades	84
2.5.7 Declaração de uma Propriedade	85
2.5.8 Despachando um Evento de Propriedade	85

2.5.9	Propriedades Compostas	89
2.6	Gerenciador de Entrada	90
2.6.1	Arquitetura de entrada	90
2.6.2	Perfis de Eventos de Movimento	91
2.6.3	Eventos de Toque	92
	Eventos Básicos de Toque	92
	Coordenadas	92
	Toque em Formas	93
	Duplo Toque	93
	Toque Triplo	94
	Pegando Eventos de Toques	94
	Gerenciamento de Eventos de Toque	95
2.6.4	Joystick events	95
	Joystick event basics	96
	Joystick input	96
	Xbox 360	97
	Joystick debugging	97
2.7	Widgets	97
2.7.1	Introdução ao Widgets	97
2.7.2	Manipulando a arvore de Widgets	97
2.7.3	Atravessando a árvore	98
2.7.4	Índice Z do Widgets	99
2.7.5	Organizar com Layouts	99
2.7.6	Adicionando um plano de fundo a um Layout	109
	Adicione uma cor ao plano de fundo de uma Classe/Regra de Layouts Personalizados	112
2.7.7	Layouts Aninhados	119
2.7.8	Medidas de tamanho e posição	119
2.7.9	Separação de Tela com Gerenciador de Tela	120
2.8	Gráficos	120
2.8.1	Introdução ao Canvas	120
2.8.2	Instrução de Contexto	121
2.8.3	Instruções de Desenho	121
2.8.4	Manipulando instruções	121
2.9	Kv language	122
2.9.1	Conceito por trás da linguagem KV	122
2.9.2	Como carregar KV	122
2.9.3	Regra do contexto	123
2.9.4	Sintaxe Especial	123
2.9.5	Instanciando Widget Filhos	124
2.9.6	Vinculação de Eventos	125
2.9.7	Ampliando o Canvas	126
2.9.8	Referenciando Widgets	126
2.9.9	Acessando Widgets definidos com a linguagem kv no seu código Python	127
2.9.10	Classes Dinâmicas	129

2.9.11	Reutilizando estilos em vários Widgets	130
2.9.12	Desenhando com a Linguagem Kivy	131
	O código vai em arquivos <i>*.py</i>	131
	O layout vai em <i>controller.kv</i>	132
2.10	Integrando com outros Frameworks	133
2.10.1	Usando o Twisted dentro do Kivy	133
	Server App	134
	Client App	135
2.11	Empacotando sua Aplicação	137
2.11.1	Criando um pacote para Windows	137
	Requisitos	137
2.11.2	Hook padrão do PyInstaller	137
	Empacotando uma simples aplicação	137
	Empacotando uma aplicação de vídeo com <i>Gstreamer</i>	139
2.11.3	Sobrescrevendo o hook (gancho) padrão	140
	Incluindo/excluindo vídeo e áudio e reduzindo o tamanho do aplicativo	140
	Instalação alternativa	141
2.11.4	Criando um pacote para o Android	141
	Buildozer	142
	Empacotando com Python-for-Android	143
	Empacotando seu aplicativo para executar no Kivy Launcher	143
	Instalação dos Exemplos	143
	Liberação no mercado	144
	Segmentação do Android	144
2.11.5	Máquina Virtual do Kivy com Android	144
	Prefácio	144
	Começando	145
	Construindo o APK	145
	Dicas e Truques	145
2.11.6	Kivy no Android	146
	Pacotes para Android'	147
	Depurando seu aplicativo na plataforma Android	147
	Usando APIs Android	147
	Plyer	148
	Pyjnius	148
	Módulo Android	149
	Status do Projeto e Dispositivos Testados	149
2.11.7	Criando um pacote para OS X	150
	Usando o Buildozer	150
	Usando o SDK do Kivy	151
	Instalação dos Módulos	151
	Onde os módulos / arquivos estão instalados?	152
	Para instalar os arquivos binários	152
	Para incluir outros frameworks	152
	Reduzindo o tamanho do aplicativo	152

Ajustando as Configurações	153
Criando um DMG	153
Usando o PyInstaller sem o Homebrew	153
Usando o PyInstaller e o Homebrew	154
Guia completo	155
Editando o arquivo <i>spec</i>	156
Construa o <i>spec</i> e crie um DMG	156
Bibliotecas adicionais	157
GStreamer	157
2.11.8 Criando um pacote para iOS	157
Pré-requisitos	157
Compile a distribuição	158
Criando um projeto no Xcode	158
Atualizando um projeto Xcode	159
Personalize	159
Problemas conhecidos	159
FAQ	159
A aplicação encerrou anormalmente!	159
Como a Apple pode aceitar um aplicativo do Python?	160
Já enviastes um aplicativo Kivy para a App Store?	160
2.11.9 Pré-requisitos para o iOS	160
Começando	160
Homebrew	160
2.12 Licença do Empacotamento	161
2.12.1 Dependências	161
2.12.2 Windows (PyInstaller)	162
Visual Studio Redistributables	162
Outras Bibliotecas	162
2.12.3 Linux	162
2.12.4 Android	163
2.12.5 Mac	163
2.12.6 iOS	163
2.12.7 Evitando binários	163
3 Tutoriais	165
3.1 Tutorial do Pong Game	165
3.1.1 Prefácio	165
3.1.2 Começando	166
3.1.3 Adicionando Simples Gráficos	167
Explicando a Sintaxe do Arquivo Kv	168
3.1.4 Adicionar Bola	170
Classe BolaPong	170
3.1.5 Adicionando Animação para a Bola	172
Agendando função para o Clock	172
Propriedade/Referência do Objetos	172
3.1.6 Conectando Eventos de Entrada	176

3.1.7	Para onde ir agora?	180
3.2	Um aplicativo de pintura simples	180
3.2.1	Considerações básicas	181
3.2.2	Widget Paint	181
	Estrutura Inicial	181
	Adicionando Comportamentos	182
	Bonus Points	188
3.3	Curso Rápido	191
3.3.1	Informação Básica	191
	Os tópicos cobertos pelo Crash Course incluem:	191
	Links:	192
4	Referência API	193
4.1	Framework Kivy	193
4.1.1	Animação	195
	Animação simples	195
	Múltiplas propriedades e transições	195
	Animação sequencial	195
	Animação paralela	196
	Repetindo animação	196
4.1.2	Aplicação	206
	Criando uma Aplicação	206
	Método usando sobreposição build()	206
	Método usando arquivo kv	207
	Configuração da aplicação	208
	Usar o arquivo de configuração	208
	Cria um painel de configurações	209
	Perfis com on_start e on_stop	211
	Customização de layout	212
	Modo pausado	213
4.1.3	Abertura de Dados Assíncrona	221
	Ajustando o Carregador Assíncrono	222
4.1.4	Atlas	224
	Definição de arquivos .atlas	225
	Como criar um Atlas	225
	Como utilizar um Atlas	226
	Manual de uso dos Atlas	227
4.1.5	Gerenciador de Cache	228
4.1.6	Objeto Clock	230
	Agenda Antes do Frame	232
	Eventos disparados	232
	Desagendamento	233
	Threading e ordem do Callback	234
	Detalhes Avançados de Clock	234
	Padrão Clock	235
	Clock Ininterruptível	235

Liberação do Clock	236
Liberação somente do Clock	236
Resumo	236
4.1.7 Módulo de Compatibilidade com Python 2.7 e > 3.3	242
4.1.8 Objeto de Configuração	243
Aplicando configurações	243
Utilização do objecto Config	243
Disponíveis símbolos de configuração	244
4.1.9 Contexto	251
4.1.10 Distribuidor de Eventos	251
4.1.11 Fábrica de Objetos	262
4.1.12 Utilitários geométricos	262
4.1.13 Reconhecimento de Gestos	263
4.1.14 Interactive launcher	265
Craindo um <i>InteractiveLauncher</i>	266
Desenvolvimento Interativo	266
Pausando a Aplicação Diretamente	267
Adicionando Atributos Dinamicamente	267
TODO	268
4.1.15 Kivy Base	269
Gerenciamento do Event Loop	269
4.1.16 Objeto Logger	272
Configuração do Logger	272
História do Logger	273
4.1.17 Métricas	273
Dimensões	273
Exemplos	274
Controle Manual de Métricas	274
4.1.18 Reconhecedor de gestos Multistroke	276
Visão Geral Conceitual	276
Exemplo de uso	277
Detalhes do Algoritmo	278
4.1.19 Utilitários Parser	289
4.1.20 Propriedades	290
Comparação entre Python vs Kivy	290
Simples exemplo	290
Profundidade sendo rastreada	291
Checagem de Valores	291
Tratamento de Erros	292
Conclusão	292
Modificação no Observe Property	292
Using Observe <i>bing()</i>	292
Observe usando ‘on_<propname>’	293
Vinculando a propriedade de propriedades.	293
4.1.21 Gerenciador de Recursos	305
Pesquisa de Recursos	306

	Personalizando o Kivy	306
4.1.22	Suporte	306
4.1.23	Util	307
4.1.24	Vetor	310
	Uso otimizado	310
	Operadores de Vetores	311
4.1.25	Método fraco	314
4.1.26	Proxy fraco	315
4.2	Adaptadores	315
4.2.1	O Conceito	315
4.2.2	Os componentes	316
4.2.3	Adapter	317
4.2.4	DictAdapter	318
4.2.5	Conversor de Lista de itens de Argumentos	320
	Uso simples	320
	Uso avançando	320
4.2.6	ListAdapter	321
4.2.7	SelectableDataItem	324
	Data Models	325
4.2.8	SimpleListAdapter	325
4.3	Núcleo de Abstração	326
4.3.1	Áudio	327
	Despacho de eventos e alterações de estado	327
4.3.2	Câmera	329
4.3.3	Área de Transferência	330
4.3.4	OpenGL	331
4.3.5	Imagem	331
	Carregamento de imagem pra memória	331
4.3.6	Spelling	336
4.3.7	Texto	337
	Text Layout	341
	Marcação de Texto	345
4.3.8	Vídeo	347
4.3.9	Janela	348
4.4	Módulo Kivy para dependências binárias	364
4.5	Efeitos	364
4.5.1	Efeito de rolagem amortecido	365
4.5.2	Efeito Kinetic	366
4.5.3	368
4.5.4	Scroll effect	368
4.6	Garden	369
4.6.1	Empacotamento	370
4.7	Gráficos	370
4.7.1	O básico	370
4.7.2	Mecanismo de Recarga GL	371
4.7.3	Canvas	397

4.7.4	CGL: standard C interface for OpenGL	403
4.7.5	Instrução de Contexto	404
4.7.6	Gerenciador de Contexto	409
4.7.7	FrameBuffer	409
	Recarregando o conteúdo da FBO	410
4.7.8	Instrução GL	413
	Limpa o FBO	413
4.7.9	Compilador Gráfico	414
	Reducindo as instruções de contexto	414
4.7.10	OpenGL	415
4.7.11	Utilitários do OpenGL	425
4.7.12	Instruções Scissor	427
4.7.13	Shader	429
	Inclusão do cabeçalho	429
	Programas de Shaders GLSL de arquivo único	430
4.7.14	Instruções do Stencil	431
	Limitações	432
	Exemplo de uso do <i>Stencil</i>	432
4.7.15	SVG	433
4.7.16	Tesselator	434
	Utilização	435
4.7.17	Textura	437
	Blitting dados personalizados	438
	Suporta BGR/BGRA	439
	Textura NPOT	439
	Atlas da textura	440
	Mipmapping	440
	Recarregando a Textura	440
4.7.18	Transformação	445
4.7.19	Instruções Vertex	449
	Atualizando Propriedades	450
4.8	Gerenciador de Entrada	462
4.8.1	Entrada de Pós Processamento	465
	Calibration	465
	Dejitter	466
	Double Tap	466
	Lista ignorada	467
	Manter o toque	467
	Triple Tap	467
4.8.2	Provedores	468
	Entrada do Provedor Joystick do Android	468
	Provedor de Entrada do Auto-Criador da entrada de configuração disponibilizada pela MT Hardware (Somente para Linux).	468
	Definições padrões para Provedores do Windows	469
	Leap Motion - somente dedo	469

Implementação do Provedor de Mouse	469
Usando interação com MultiTouch com o mouse	469
Suporte nativo para entrada HID do kernel do linux	470
Suporte nativo a dispositivos com Multitoques em Linux utilizando libmtdev	471
Suporte nativo do framework MultitouchSupport para Mac-Book (plataforma Mac OS X)	472
Suporte nativo do tablet Wacom do driver linuxwacom Drive de suporte nativo com linuxwacom da Wacom	472
Suporte para mensagens WM_PEN (plataforma Windows)	473
Suporte para mensagem WM_TOUCH (Plataforma Microsoft Windows)	473
Provedor de Entrada TUIO	473
Configura um provedor TUIO no arquivo <i>config.ini</i>	473
Configura um provedor TUIO na aplicação	473
4.8.3 Entrada de Gravação	475
Gravando Eventos	476
Reprodução manual	476
Gravando mais atributos	476
Limitações conhecidas	477
4.8.4 Eventos de Movimento	478
Evento de Movimento e Toque	478
Escuta para um Evento de Movimento	478
Perfis	479
4.8.5 Fábrica de Eventos de Movimentos	483
4.8.6 Provedor de Eventos de Movimento	484
4.8.7 Forma do Evento de Movimento	484
4.9 Linguagem Kivy	485
4.9.1 Visão Geral	485
4.9.2 Sintaxe de um Arquivo kv	485
4.9.3 Expressões de valor, expressões <i>on_property</i> , <i>ids</i> e palavras-chave reservadas	487
<i>ids</i>	488
Expressão Válida	489
4.9.4 Relação entre Valores e Propriedades	489
4.9.5 Instruções Gráficas	491
4.9.6 Classes dinâmicas	492
4.9.7 Modelos	493
Sintaxe de modelos	493
Exemplo de modelo	494
Limitações de modelo	496
4.9.8 Redefinindo um estilo de widget	496
4.9.9 Redefinindo um estilo de propriedade de uma ferramenta	497
4.9.10 Ordem dos <i>kwargs</i> e aplicação de regra <i>kv</i>	497
4.9.11 Diretivas da Linguagem <i>import <package></i>	498

<i>set <key> <expr></i>	499
<i>include <file></i>	499
4.9.12 Construtor	504
4.9.13 Parser	508
4.10 Bibliotecas Externas	509
4.10.1 GstPlayer	509
4.10.2 OSC	509
SEM DOCUMENTAÇÃO (módulo kivy.uix.recycleview)	510
simpleOSC 0.2	511
4.10.3 Arquivo de Biblioteca DDS	512
Formato DDS	512
4.10.4 Python mtdev	513
4.11 Módulos	513
4.11.1 Ativando um módulo	514
Ativar módulo no config	514
Ativar um módulo Python	514
Ativar o módulo via linha de comando	515
4.11.2 Criar seu próprio módulo	515
4.11.3 Console	515
Utilização	515
Navegação do Mouse	516
Navegação do Teclado	516
Informações adicionais	516
Addons	516
4.11.4 Inspetor	520
Utilização	520
4.11.5 Keybinding	521
Utilização	521
4.11.6 Módulo de Monitoria	522
Utilização	522
4.11.7 Módulo de Gravação	522
Configuração	522
Utilização	523
4.11.8 Tela	523
4.11.9 Toque	523
Configuração	524
Exemplo	524
4.11.10 Web Debugger	524
4.12 Suporte de Rede	524
4.12.1 URLRequest	524
4.13 Storage (Armazenamento)	528
4.13.1 Utilização	529
4.13.2 Exemplos	529
4.13.3 API síncrona/assíncrona	530
4.13.4 Tipo de container Síncrono	530
4.13.5 Dicionários de Armazenamento	533

4.13.6	Armazenamento JSON	533
4.13.7	Armazenamento Redis	534
4.14	Ferramentas	534
4.14.1	Scripts	534
4.14.2	Módulos	535
4.14.3	Outro	535
4.14.4	Empacotamento	535
	Pyinstaller hooks	535
	Hooks	535
	hiddenimports	536
	Gerador de Hook	536
4.15	Widgets	538
4.15.1	Comportamentos	539
	Mixin de classes de Comportamento	539
	Adicionando comportamentos	539
	Button Behavior	546
	Exemplo	547
	Comportamento de Navegação de Código	548
	Comportamento de Seleção Composto	549
	Conceitos de seleção compostos	549
	Mecanismo de Seleção	549
	Exemplo	549
	Cover Behavior	556
	Example	556
	Comportamento de Arrastar	558
	Exemplo	558
	Comportamento Emacs	560
	Teclas de Atalho Emacs	560
	Comportamento do Foco	561
	Gerenciamento do Foco	561
	Inicialização do Foco	562
	Kivy Namespaces	566
	Simples exemplo	567
	Definindo o namespace	567
	Namespace herdado	568
	Acessando o namespace	569
	Forking um namespace	570
	Comportamento ToggleButton	573
	Exemplo	573
4.15.2	RecycleView	575
	RecycleView Data Model	580
	RecycleView Layouts	581
	RecycleView Views	583
4.15.3	View abstrata	586
4.15.4	Acordeão	587
	Simples exemplo	588

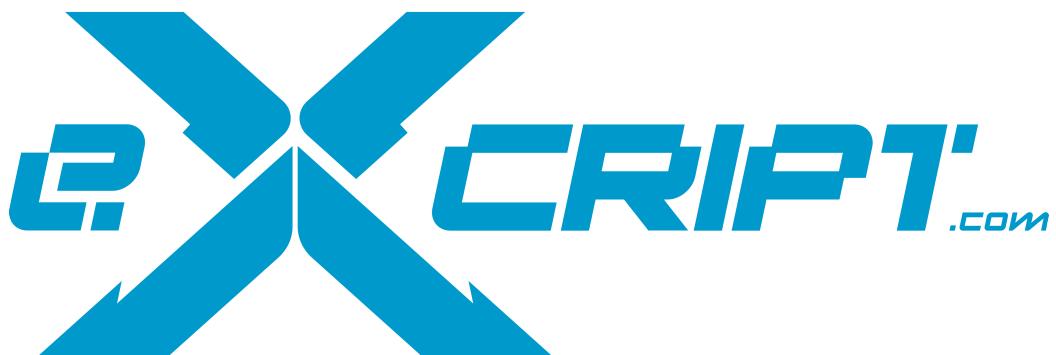
Personalizando o Accordion	588
4.15.5 Action Bar	592
4.15.6 Anchor Layout	598
4.15.7 Box Layout	599
4.15.8 Bubble	601
Simples exemplo	602
Customize o Bubble	603
4.15.9 Botão	605
4.15.10 Câmera	607
4.15.11 Carrossel	608
4.15.12 CheckBox	612
4.15.13 Entrada de Código	614
Exemplo de uso	615
4.15.14 Color Picker	615
4.15.15 Lista Drop-Down	618
Simples exemplo	618
Estendendo o dropdown em kv	619
4.15.16 EffectWidget	621
Diretrizes de Uso	622
Provedores de Efeitos	622
Criando Cores	623
4.15.17 FileChooser	626
Widgets Simples	626
Composição do Widget	628
Exemplo de uso	628
4.15.18 Float Layout	637
4.15.19 Superfície de Gestos	639
4.15.20 Grid Layout	643
Background	644
Largura da coluna e Altura da linha	644
Utilizando um GridLayout	644
4.15.21 Imagem	648
Abertura Assíncrona	648
Alinhamento	648
4.15.22 Rótulo	651
String e Conteúdo texto	652
Alinhamento do Texto e seu Envólucro	653
Marcação de Texto	653
Zona interativa no texto	654
Catering para linguagens Unicode	655
Exemplo de uso	655
4.15.23 Leiaute	665
Entendendo a propriedade <i>size_hint</i> do Widget	666
4.15.24 List View	667
Prefácio	668
Exemplo Básico	669

Usando um Adaptador	670
ListAdapter e DictAdapter	671
Usando um “Args Converter”	672
Um Exemplo com ListView	673
Usando um Custom Item View Class	674
Usando um Item View Template	675
Usando um CompositeListItem	676
Uso para Seleção	677
4.15.25 ModalView	680
Exemplos	681
Eventos do ModalView	681
4.15.26 PageLayout	683
4.15.27 Popup	684
Exemplos	685
Eventos Popup	685
4.15.28 Progress Bar	687
4.15.29 RecycleBoxLayout	688
4.15.30 RecycleGridLayout	688
4.15.31 RecycleLayout	689
4.15.32 Relative Layout	689
Sistema de Coordenadas	689
Coordenadas da Janela	689
Coordenadas Pai	690
Coordenadas Locais e de Widgets	691
Transformações de Coordenadas	691
Armadilhas Comuns	692
4.15.33 Renderizador reStructuredText	694
Uso com texto	694
Uso com fonte	695
4.15.34 Sandbox	698
4.15.35 Scatter	699
Utilização	700
Interações de Controle	700
Trazer automaticamente para frente	700
Limitação da Escala	700
Comportamento	701
4.15.36 Scatter Layout	704
4.15.37 Gerenciador de Vídeo	705
Uso básico	705
Alterando a Direção	707
Uso avançando	707
Alterando a transição	708
4.15.38 ScrollView	715
Comportamento de rolagem	715
Limitando ao eixo X ou Y	716
Gerenciando o tamanho e a posição do conteúdo	716

Efeitos de deslocamento	717
4.15.39 SelectableView	721
4.15.40 Configurações	722
Cria um painel a partir do JSON	723
Diferentes layouts de painel	725
4.15.41 Slider	734
4.15.42 Spinner	738
4.15.43 Splitter	740
4.15.44 Stack Layout	743
4.15.45 Stencil View	745
4.15.46 Switch	746
4.15.47 TabbedPane	747
Simples exemplo	747
Customize o Tabbed Panel	749
4.15.48 Entrada de Texto	754
Exemplo de uso	755
Seleção	756
Filtragem	756
Teclas de Atalho Padrões	757
4.15.49 Botão de alternar	768
4.15.50 Tree View	769
Prefácio	769
Criando seu próprio Widget de nó	771
4.15.51 Vídeo	776
4.15.52 Player de Vídeo	778
Anotações	779
Fullscreen (Tela Cheia)	780
Comportamento do final da execução	780
4.15.53 VKeyboard	784
Modos	785
Layouts	785
Solicitador de Teclado	786
4.15.54 Classe Widget	790
Propriedades utilizadas	791
Desenho básico	792
Bubbling do Evento de Toque do Widget	792
Utilização de Widget.center, Widget.right, e Widget.top	794
IV Apendice	809
5 Licença	811
Índice de Módulos Python	813

A tradução da documentação do projeto Kivy para o português é uma iniciativa da **eXcript Brasil**, idealizada e gerenciada por Cláudio Rogério Carvalho Filho junto com a comunidade

A **eXcript** também tem o orgulho de disponibilizar o primeiro treinamento de Kivy em Português. Para obter maiores informações, acesse o link a seguir: [Curso de Desenvolvimento Multiplataforma com Python e Kivy](#).



E-mail para contato: excriptbrasil@gmail.com

Parte I

ATENÇÃO

Esta tradução ainda não foi revisada, portanto, nesse momento deve haver dezenas, senão centenas de erros. Nos próximos meses serão lançadas novas versões com as devidas traduções e as atualizações conforme a documentação em Inglês evolui.

Obviamente, não há qualquer garantia e este material está licenciado com a mesma licença da versão em Inglês.

Parte II

COLABORADORES

Em ordem alfabética:

- Alexandre Ferreira
- Anderson Guerra
- Antonio Santos
- Caio J. Carvalho
- Cláudio Rogério Carvalho Filho
- Eduardo Melgaço
- Eudemir Vieira
- Everton Ventura
- Fabiano de Almeida
- Fausto Roger
- Felipe Nogueira de Souza
- Felype Bastos
- Flávio Andrei
- Gilberto dos Santos Alves
- Gregory Romano Casanova
- Henrique Nunes
- João Ponte

- Josafá
- Laender Oliveira
- Leandro Quadros Durães Braga
- Leonardo Castro
- Lucas Paim
- Lucas Rodrigues
- Lucas Teske
- Lucas Beneti
- Luciano Santos
- Luiz Guilherme Arruda
- Luke Feroz
- Marcelo Vieira Gonçalves
- Michel Ribeiro
- Paulo Henrique
- Paulo Santos
- Rafael Costa
- Robert Carlos
- Rodrigo Oliveira
- Sergio Junior
- Thadeu Santos
- Victor Sued
- Vinicius Ferreira de Souza
- Washington Guimaraes
- William da Rosa Garcia

Parte III

BEM-VINDO AO KIVY

Bem-vindo à documentação do Kivy. Kivy é uma biblioteca para o desenvolvimento de software com código-fonte aberto, voltado ao rápido desenvolvimento de aplicações que utilizam novas interfaces de usuário, e projetada para também ser utilizada na como aplicações que serão executadas em dispositivos multi-touch.

Recomendamos que comece com *Começando*. Em seguida, veja a seção *Guia de Programação*. Nós também temos um *Criando um Aplicação* caso estejas impaciente.

Provavelmente estas se perguntando por que deverias estar interessado em usar Kivy. Existe um documento que descreve a nossa *Filosofia* e lhe incentivamos a lê-la, há também um seção que explica em detalhes a *Visão geral da Arquitetura*.

Se quiseres contribuir com o projeto Kivy, veja como fazê-lo através de *Contribuição*. Se não encontrares o que estás buscando na documentação, sinta-se livre para *Contate-nos*.

Guia de Usuário

Esta parte da documentação explica as ideias básicas por trás do design do Kivy e por que você irá gostar de utilizá-lo. Em seguida continua com uma discussão da arquitetura e mostra como criar aplicativos impressionantes em um curto espaço de tempo usando o framework.

1.1 Instalação

Nós tentamos não reinventar a roda, mas sim, trazer algo inovador para o mercado. Como consequência, nós estamos focados no nosso próprio código e usando bibliotecas de terceiros de alta qualidade sempre que possível. Para suportar o conjunto completo e rico de recursos que o Kivy oferece, várias outras bibliotecas são necessárias. Se você não usar um recurso específico (por exemplo, reprodução de vídeo), você não precisa da dependência correspondente. Dito isto, há uma dependência que Kivy ** requer **: “Cython<<http://cython.org>>.

This version of **Kivy** requires at least **Cython** version **0.23**, and has been tested through 0.23. Later versions may work, but as they have not been tested there is no guarantee.

Além disso, você precisa do **Python** 2.x ($2.7 \leq x < 3.0$) ou do interpretador 3.x ($3.3 \leq x$). Se quiseres utilizar recursos como o windowing (isto é, abrir uma Janela), audio/video playback ou correção ortográfica, dependências adicionais devem estar disponíveis. Para estes, recomendamos **SDL2**, **Gstreamer 1.x** e **PyEnchant**, respectivamente.

Outras bibliotecas opcionais (mutualmente independente) são:

- **OpenCV 2.0** – Entra da Câmera .
- **Pillow** – Tela de Image e texto.
- **PyEnchant** – Correção Ortográfica.

Dito isso, *SEM PÂNICO**!

Nós não esperamos que você instale todas essas coisa por conta própria. Ao invés disso, nós criamos um ótimo pacote portável que você pode usar diretamente, e ele contem os pacotes necessários para sua plataforma. Nós apenas queremos que você saiba que existem alternativas para os padrões e lhe da uma visão geral das coisas que Kivy usa internamente.

1.1.1 Versão Estável

A última versão estável pode ser encontrada no site do Kivy em <http://kivy.org/#download>. Consulte as instruções de instalação específicas para a sua plataforma:

Instala no Windows

Começando com 1.9.1 nós fornecemos binário **wheels** para Kivy e todas as suas dependências para ser usado com uma instalação existente do Python. Veja: *Instalação*.

Também fornecemos Wheels que são todos os dias a noite gerados através do código oficial disponível em **master**. Veja: *ref:install-nightly-win-dist*. Veja também *Atualizando desde uma versão anterior do Kivy dist*. Se estiveres instalando o Kivy em um **local alternativo** e não no site-packages, consulte *Instalando O Kivy num local alternativo*.

Nota: For Python < 3.5 we use the MinGW compiler. However, for Python 3.5 on Windows we currently only support the microsoft MSVC compiler because of the following MinGW **issue**. Generally this should make no difference when using precompiled wheels.

Aviso: Support for Python 3.5 and higher isn't available with the current stable version (1.9.1). Compile the master branch or use the nightly wheels.

Para usar o Kivy, precisas do **Python**. Várias versões do Python podem ser instaladas lado a lado, mas o Kivy precisará ser instalado para cada versão do Python que desejas usar as bibliotecas do Kivy.

Instalação

Agora que o Python está instalado, abra o *Linha de Comando* e certifique-se de que o Python está disponível digitando `python --version`. Em seguida, faça o seguinte para instalar.

1. Assegure-se de que você tem o último pip e o Wheel:

```
python -m pip install --upgrade pip wheel setuptools
```

2. Install the dependencies (skip gstreamer (~120MB) if not needed, see *Dependências do Kivy*):

```
python -m pip install docutils pygments pypiwin32 kivy.deps.  
    →sdl2 kivy.deps.glew  
python -m pip install kivy.deps.gstreamer
```

For Python 3.5 only we additionally offer angle which can be used instead of glew and can be installed with:

```
python -m pip install kivy.deps.angle
```

3. Instale o Kivy:

```
python -m pip install kivy
```

That's it. You should now be able to `import kivy` in python or run a basic example:

```
python share\kivy-examples\demo\showcase\main.py
```

Nota: Se encontrares qualquer erro de **permissão negada**, tente abrir o prompt de comando ‘como administrador <<https://technet.microsoft.com/en-us/library/cc947813%28v=ws.10%29.aspx>>’ _ E tente novamente.

O que são Wheels, pip e Wheel

No Python, pacotes como o Kivy podem ser instalados com o gerenciador de pacotes Python, `pip`. Alguns pacotes como o Kivy requerem etapas adicionais, como a compilação, ao instalar usando o código-fonte do Kivy com o pip. Wheels (que possuem a extensão `*.whl`) são distribuições pré-construídas de um pacote que já foi compilado e por isso não requerem etapas adicionais no momento da instalação.

Quando hospedado em `pypi` é instalado uma Wheels usando o `pip`, por exemplo `python -m pip install kivy`. Ao fazer o download e instalar uma Wheel diretamente, o `python -m pip install wheel_file_name` é usado da seguinte forma:

```
python -m pip install C:\Kivy-1.9.1.dev-cp27-none-win_amd64.whl
```

Instalação do Nightly Wheel

Aviso: Usar a versão de desenvolvimento mais recente pode ser arriscado e poderá encontrar problemas durante o desenvolvimento. Se encontrares algum erro, informe-nos!

Snapshot wheels of current Kivy master are created on every commit to the *master* branch of kivy repository. They can be found [here](#). To use them, instead of doing `python -m pip install kivy` we'll install one of these wheels as follows.

- [Python 2.7, 32bit](#)
- [Python 3.4, 32bit](#)
- [Python 3.5, 32bit](#)
- [Python 3.6, 32bit](#)
- [Python 2.7, 64bit](#)
- [Python 3.4, 64bit](#)
- [Python 3.5, 64bit](#)
- [Python 3.6, 64bit](#)

1. Execute os passos 1 e 2 da seção de Instalação acima.
2. Faça o download do Wheel apropriado para o seu sistema.
3. Install it with `python -m pip install wheel-name` where `wheel-name` is the name of the renamed file and add deps to the *PATH*.

Kivy examples are separated from the core because of their size. The examples can be installed separately on both Python 2 and 3 with this single wheel:

- [Kivy examples](#)

Dependências do Kivy

We offer wheels for Kivy and its dependencies separately so only desired dependencies need be installed. The dependencies are offered as optional sub-packages of `kivy.deps`, e.g. `kivy.deps.sdl2`.

Currently on Windows, we provide the following dependency wheels:

- [gstreamer](#) for audio and video

- `glew` and/or `angle` (3.5 only) for OpenGL
- `sdl2` for control and/or OpenGL.

One can select which of these to use for OpenGL use using the `KIVY_GL_BACKEND` environment variable by setting it to `glew` (the default), `angle`, or `sdl2`. `angle` is currently in an experimental phase as a substitute for `glew` on Python 3.5 only.

`gstreamer` is an optional dependency which only needs to be installed if video display or audio is desired. `ffpyplayer` is an alternate dependency for audio or video.

Linha de Comando

Conheça como funciona a sua linha de comando. Para executar qualquer um dos comandos `pip` ou “wheel”, é necessário uma ferramenta de linha de comando e o Python deve estar no Path do sistema. A linha de comando padrão do Windows é o **Prompt de Command**, e a maneira mais rápida de abri-lo é pressionar `Win + R` no teclado. Em seguida digite `cmd` na janela que se abriu, e então pressione Enter.

Os shells de comandos alternativos no estilo linux que recomendamos são [Git para Windows](#) que oferece uma linha de comando bash como `well` como '`git <https://try.github.io>'`. Note que o CMD ainda pode ser usado mesmo que o bash tenha sido instalado.

Andando pelo caminho! Para adicionar a sua instalação do Python ao PATH do sistema, abra a linha de comando e, em seguida, digite o comando `cd` para mudar o diretório atual e então vá até o diretório onde o Python está instalado, por exemplo, `cd C:\python27`. Alternativamente, se tiveres apenas uma versão do Python instalada, adicione permanentemente o diretório Python ao PATH no `cmd` ou no `bash`.

Use a Versão de Desenvolvimento

Aviso: Usar a versão de desenvolvimento mais recente pode ser arriscado e poderá encontrar problemas durante o desenvolvimento. Se encontrares algum erro, informe-nos!

Para compilar e instalar o Kivy utilizando o [código-fonte](#) ou para usar o Kivy com o git ao invés do Wheel existente, será necessário executar alguns passos adicionais:

1. Both the `python` and the `Python\Scripts` directories **must** be on the path. They must be on the path every time you recompile kivy.
2. Certifique-se que você possui a última versão do pip e Wheel com:

```
python -m pip install --upgrade pip wheel setuptools
```

3. Get the compiler. For Python < 3.5 we use mingwpy as follows.

- (a) Crie o arquivo `python\Lib\distutils\distutils.cfg` e adicione 2 linhas:

```
[build]
compiler = mingw32
```

- (b) Instale o MinGW com:

```
python -m pip install -i https://pypi.anaconda.org/carlkl/
→simple mingwpy
```

For Python 3.5 we use the MSVC compiler. For 3.5, **Visual Studio 2015** is required, which is available for free. Just download and install it and you'll be good to go.

Visual Studio is very big so you can also use the smaller, **Visual C Build Tools instead**.

4. Configure as variáveis de ambiente. No Windows faça:

```
set USE SDL2=1
set USE GSTREAMER=1
```

No Bash faça:

```
export USE SDL2=1
export USE GSTREAMER=1
```

Essas variáveis devem ser definidas sempre que você recompilar o kivy.

5. Instale as outras dependências, bem como as suas versões dev (podes ignorar o Gstreamer e o gstreamer_dev se não fores utilizar recursos de vídeo/áudio):

```
python -m pip install cython docutils pygments pypiwin32 kivy.
→deps.sdl2 \
kivy.deps.glew kivy.deps.gstreamer kivy.deps.glew_dev kivy.deps.
→sdl2_dev \
kivy.deps.gstreamer_dev
```

6. Se baixastes ou clonastes o Kivy num local alternativo e não deseja instalá-lo em pacotes de sites, leia a próxima seção.
7. Finally compile and install kivy with `pip install filename`, where `filename` can be a url such as `https://github.com/kivy/kivy/archive/master.zip` for kivy master, or the full path to a local copy of a kivy.

Compile Kivy

1. Start installation of Kivy cloned from GitHub:

```
python -m pip install kivy\.
```

If the compilation succeeds without any error, Kivy should be good to go. You can test it with running a basic example:

```
python share\kivy-examples\demo\showcase\main.py
```

Instalando O Kivy num local alternativo

In development Kivy is often installed to an alternate location and then installed with:

```
python -m pip install -e location
```

That allows Kivy to remain in its original location while being available to python, which is useful for tracking changes you make in Kivy for example directly with Git.

To achieve using Kivy in an alternate location extra tweaking is required. Due to this [issue](#) wheel and pip install the dependency wheels to `python\Lib\site-packages\kivy`. So they need to be moved to your actual kivy installation from site-packages.

Depois de instalar as dependências do Kivy e fazer o download ou a clonagem do Kivy no diretório local desejado, faça o seguinte:

1. Mova o conteúdo de `python\Lib\site-packages\kivy\deps` para “`your-pathkivydeps`” onde `your-path` é o caminho onde o kivy está localizado.
2. Remova completamente o diretório `python\Lib\site-packages\kivy`.
3. From `python\Lib\site-packages` move `all kivy.deps.*.dist-info` directories to `your-path` right next to `kivy`.

Now you can safely compile kivy in its current location with one of these commands:

```
> make  
> mingw32-make  
> python -m pip install -e .  
> python setup.py build_ext --inplace
```

If **kivy fails to be imported**, you probably didn't delete all the `*.dist-info` folders and the `kivy` or `kivy.deps*` folders from site-packages.

Tornando o Python disponível em qualquer lugar

Há 2 métodos para executar arquivos Python no seu sistema *.py.

Se tiveres apenas uma versão do Python instalada, poderás associar todos os arquivos *.py ao executável do Python, caso ainda não tenhas feito, e então executá-lo clicando duas vezes. Ou você só pode fazê-lo uma vez caso queiras ser capaz de escolher qual a versão do Python utilizar a cada execução:

1. Clique com o botão direito do mouse no arquivo Python (cuja extensão do arquivo é *.py) do aplicativo que você deseja iniciar
2. No menu de contexto que aparece, selecione *Abrir com*
3. Procure no seu disco rígido e encontre o arquivo `python.exe` que deseja usar. Selecione-o.
4. Selecione “Sempre abrir o arquivo com ...” se não quiseres repetir esse procedimento toda vez que clicares duas vezes num arquivo .py.
5. Você terminou. Abra o arquivo.

Pode iniciar um arquivo *.py com o nosso Python usando o menu Send-to:

1. Procure o arquivo `python.exe` que desejas utilizar. Clique com o botão direito sobre ele e copie-o.
2. Abra o Windows Explorer (Explorador de arquivos no Windows 8) e vá para o endereço ‘shell: sendto’. Deverás obter o diretório especial do Windows `SendTo`
3. Cole o arquivo `python.exe` previamente copiado como sendo **um atalho**.
4. Renomeie-o para Python <python-version>. Por exemplo: `python27-x64`

You can now execute your application by right clicking on the `.py` file -> “Send To” -> “python <python-version>”.

Atualizando desde uma versão anterior do Kivy dist

Para instalar as novas Wheels numa distribuição antiga do Kivy, todos os arquivos e pastas, exceto a pasta Python, deverão ser excluídos da distribuição. Esta pasta Python será então tratada como um sistema normal instalado o Python e todas as etapas descritas em [Instalação](#) podem ser continuado.

Instalação no OS X

Nota: This guide describes multiple ways for setting up Kivy. Installing with Homebrew and pip is recommended for general use.

Usando o Homebrew com pip

Você pode instalar o Kivy com o Homebrew e pip usando as seguintes etapas:

1. Instalar os requisitos usando [homebrew](#):

```
$ brew install sdl2 sdl2_image sdl2_ttf sdl2_mixer  
→gstreamer
```

2. Instale o Cython 0.23 e o Kivy usando o pip (certifique-se de definir a variável env e USE OSX_FRAMEWORKS = 0, o comando pode variar dependendo do shell que estás utilizando):

```
$ pip install -I Cython==0.23  
$ USE OSX_FRAMEWORKS=0 pip install kivy
```

- Para instalar a versão de desenvolvimento, use isso na segunda etapa:

```
$ USE OSX_FRAMEWORKS=0 pip install https://github.com/  
→kivy/kivy/archive/master.zip
```

Usando MacPorts com pip

Nota: Terás que instalar manualmente o suporte ao GStreamer se desejas oferecer suporte à reprodução de vídeo em seu aplicativo Kivy. Os documentos mais recentes de como portar mostram o seguinte [py-gst-python port](#).

Podes instalar o Kivy com Macports e o pip usando as seguintes etapas:

1. Instale [Macports](#)
2. Instale e defina o Python 3.4 como sendo o padrão:

```
$ port install python34  
$ port select --set python python34
```

3. Instale e defina o pip como o parfão:

```
$ port install pip-34  
$ port select --set pip pip-34
```

4. Instale os requisitos usando [Macports](#):

```
$ port install libsdl2 libsdl2_image libsdl2_ttf  
→libsdl2_mixer
```

5. Instale o Cython 0.23 e o Kivy usando o pip (certifique-se de definir a variável env e USE OSX_FRAMEWORKS = 0, o comando pode variar dependendo do shell que estás utilizando):

```
$ pip install -I Cython==0.23  
$ USE OSX FRAMEWORKS=0 pip install kivy
```

- Para instalar a versão de desenvolvimento, use isso na segunda etapa:

```
$ USE OSX FRAMEWORKS=0 pip install https://github.com/  
→kivy/kivy/archive/master.zip
```

Usando o *Kivy.app*

Nota: Este método só foi testado no OS X 10.7 e superior (64 bits). Para versões anteriores a 10.7 ou 10.7 32 bits, deverás instalar os componentes por conta própria. Sugerimos usar [homebrew](#) para instalação dos pacotes.

For OS X 10.7 and later, we provide packages with all dependencies bundled in a virtual environment, including a Python 3 interpreter for Kivy3.app. These bundles are primarily used for rapid prototyping, and currently serve as containers for packaging Kivy apps with Buildozer.

Para instalar o Kivy, você deve:

1. Navigate to the latest Kivy release at <https://kivy.org/downloads/> and download *Kivy-*-osx-python*.7z*.
2. Extraía isso usando um programa como por exemplo o [Keka](#).
3. Copie o Kivy2.app ou Kivy3.app como Kivy.app para /Applications. Cole a seguinte linha no terminal:

```
$ sudo mv Kivy2.app /Applications/Kivy.app
```

4. Crie um link simbólico chamado *kivy* para facilmente conseguir abrir aplicação com kivy venv:

```
$ ln -s /Applications/Kivy.app/Contents/Resources/script /usr/  
→local/bin/kivy
```

5. Exemplos e todas as ferramentas normais do Kivy estão presentes no diretório 'Kivy.app/Contents/Resources/kivy'.

Agora deves ter um script *kivy* que poderás usar para iniciar o seu aplicativo Kivy a partir do terminal.

Pode simplesmente arrastar e soltar o seu *main.py* para executar o seu aplicativo.

Instalação dos Módulos

O Kivy SDK no OS X usa seu próprio env virtual que é ativado quando executas o aplicativo usando o comando *kivy*. Para instalar qualquer módulo, precisarás instalar o módulo da seguinte forma:

```
$ kivy -m pip install <modulename>
```

Onde os módulos / arquivos estão instalados?

Dentro do venv do aplicativo, digite:

```
Kivy.app/Contents/Resources/venv/
```

Se instalares um módulo que instala um binário, por exemplo, como o *kivy-jardim*. Esse binário só estará disponível a partir do venv acima, como em depois de você

```
kivy -m pip install kivy-garden
```

O lib do Garden só estará disponível quando você ativar este env:

```
source /Applications/Kivy.app/Contents/Resources/venv/bin/activate  
garden install mapview  
deactivate
```

Para instalar os arquivos binários

Apenas copie o binário para o diretório /Applications/Kivy.app/Contents/Resources/venv/bin/

Para incluir outros frameworks

O Kivy.app é provido com o framework SDL2 e Gstreamer . Para incluir frameworks diferentes daqueles que são fornecidos, faça o seguinte:

```
git clone http://github.com/tito/osxrelocator  
export PYTHONPATH=~/path/to/osxrelocator  
cd /Applications/Kivy.app  
python -m osxrelocator -r . /Library/Frameworks/<Framework_name>.  
↳framework/ \  
@executable_path/./Frameworks/<Framework_name>.framework/
```

Não se esqueça de substituir <Framework_name> pela estrutura do seu framework. Essa ferramenta ‘osxrelocator’ basicamente altera o caminho das libs no framework

de modo que as mesmas sejam relativas ao executável dentro do .app, tornando o framework portável com o .app.

Inicie qualquer aplicação Kivy

Podes executar qualquer aplicativo Kivy simplesmente arrastando o arquivo principal do aplicativo para o ícone Kivy.app. Podes experimentar trabalhar dessa forma utilizando qualquer arquivo Python contido na pasta de exemplos.

Inicie desde a Linha de Comando

Se quiseres usar o Kivy desde a linha de comando, clique duas vezes no script **Make Symlinks** e depois de ter arrastado o Kivy.app para a pasta Applications. Faça isso para testar se funcionou:

1. Abra o *Terminal.app* e digite:

```
$ kivy
```

Você deve obter um prompt do Python.

2. Lá, digite:

```
>>> import kivy
```

Se ele só vai para a próxima linha sem erros, ele funcionou.

3. Executar qualquer aplicação Kivy a partir da linha de comando passou a ser simplesmente uma questão de executar um comando como no exemplo a seguir:

```
$ kivy yourapplication.py
```

Instalação no Linux

Usando pacotes de software

Para instalar os pacotes relativos para a distribuição .deb/.rpm / ...

Ubuntu / Kubuntu / Xubuntu / Lubuntu (Saucy e acima)

1. Adicione um dos PPAs como você preferir

```
stable builds $ sudo add-apt-repository ppa:kivy-team/kivy
```

```
nightly builds $ sudo add-apt-repository ppa:kivy-team/kivy-daily
```

2. Atualize o seu package lista usando o seu gerenciador de pacotes \$ sudo apt-get update

3. Instalação do Kivy

Python2 - python-kivy \$ sudo apt-get install python-kivy

Python3 - python3-kivy \$ sudo apt-get install python3-kivy

optionally the examples - kivy-examples \$ sudo apt-get install python-kivy-examples

Debian (Jessie ou versões recentes)

1. Adicione um dos PPAs ao seu *sources.list* no apt manualmente ou via Synaptic

- Jessie/Testando:

stable builds deb <http://ppa.launchpad.net/kivy-team/kivy/ubuntu> trusty main

daily builds deb <http://ppa.launchpad.net/kivy-team/kivy-daily/ubuntu> trusty main

- Sid/Instável:

stable builds deb <http://ppa.launchpad.net/kivy-team/kivy/ubuntu> utopic main

daily builds deb <http://ppa.launchpad.net/kivy-team/kivy-daily/ubuntu> utopic main

Aviso: O Wheezy não é suportado - Você precisará atualizar, pelo menos para a versão Jessie!

2. Adiciona a chave GPG ao seu chaveiro apt executando.

como usuário:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com  
--recv-keys A863D2D6
```

como root:

```
apt-key adv --keyserver keyserver.ubuntu.com  
--recv-keys A863D2D6
```

3. Atualize a sua lista de pacotes e instale **python-kivy** e/ou **python3-kivy** e, opcionalmente, os exemplos encontrados em **kivy-examples**

Linux Mint

1. Find out on which Ubuntu release your installation is based on, using this [over-view](#).
2. Continuando conforme descrito acima para o Ubuntu, dependendo de qual versão a sua instalação for baseada.

Bodhi Linux

1. Descubra qual versão da distribuição estás executando e use a tabela abaixo para descobrir em qual Ubuntu LTS se baseiar.

Bodhi 1 Ubuntu 10.04 LTS aka Lucid (Sem pacotes, apenas instalação manual)

Bodhi 2 Ubuntu 12.04 LTS aka Precise

Bodhi 3 Ubuntu 14.04 LTS aka Trusty

Bodhi 4 Ubuntu 16.04 LTS aka Xenial

2. Continuando conforme descrito acima para o Ubuntu, dependendo de qual versão a sua instalação for baseada.

OpenSuSE

1. Para instalar o Kivy vá para <http://software.opensuse.org/package/python-Kivy> e use o “1 Click Install” para a sua versão do openSuse. Talvez precise construir a versão do Kivy mais recente aparecer na lista clicando em “Show unstable packages”. Nós preferimos usar pacotes “devel:languages:python”.
2. Se quiseres acessar os exemplos, selecione **python-Kivy-examples** no assistente de instalação.

Fedora

1. Adicionando o repositório através do terminal:

Fedora 18

```
$ sudo yum-config-manager --add-repo=http://download.
  ↪opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_18/
  ↪home:thopiekar:kivy.repo
```

Fedora 17

```
$ sudo yum-config-manager --add-repo=http://download.
→opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_17/
→home:thopiekar:kivy.repo
```

Fedora 16

```
$ sudo yum-config-manager --add-repo=http://download.
→opensuse.org\
/repositories/home:/thopiekar:/kivy/Fedora_16/
→home:thopiekar:kivy.repo
```

2. Use o seu gerenciador de pacotes preferidos para atualizar o seu packagelists
3. Instale o ****python-Kivy **** e opcionalmente os exemplos, como podem ser encontrados em ****python-Kivy-examples ****

Gentoo

1. Existe um kivy ebuild (versão estável kivy)

emerge Kivy

2. disponível USE-flags são:

cairo: Standard flag, let kivy use cairo graphical libraries. camera: Install libraries needed to support camera. doc: Standard flag, will make you build the documentation locally. examples: Standard flag, will give you kivy examples programs. garden: Install garden tool to manage user maintained widgets. gstreamer: Standard flag, kivy will be able to use audio/video streaming libraries. spell: Standard flag, provide enchant to use spelling in kivy apps.

Installation in a Virtual Environment

Dependências comuns

Cython

As versões diferentes de Kivy foram testadas somente até uma determinada versão do Cython. Não há garantia se irá funcionar com versões posteriores.

Kivy	Cython
1.8	0.20.2
1.9	0.21.2
1.9.1	0.23

Dependencies with SDL2

Exemplo com Ubuntu

No comando a seguir utilize “python” e “python-dev” para o Python 2, ou “python3” e “python3-dev” para o Python 3.

```
# Install necessary system packages
sudo apt-get install -y \
    python-pip \
    build-essential \
    git \
    python \
    python-dev \
    ffmpeg \
    libsdl2-dev \
    libsdl2-image-dev \
    libsdl2-mixer-dev \
    libsdl2-ttf-dev \
    libportmidi-dev \
    libswscale-dev \
    libavformat-dev \
    libavcodec-dev \
    zlib1g-dev
```

Nota: Dependendo da versão do Linux, poderás receber mensagens de erro relacionadas ao pacote “ffmpeg”. Nesse caso, use “libav-tools” no lugar de “ffmpeg” (acima) ou use um PPA (conforme mostrado abaixo):

```
- sudo add-apt-repository ppa:mc3man/trusty-media
- sudo apt-get update
- sudo apt-get install ffmpeg
```

Instalação

```
# Make sure Pip, Virtualenv and Setuptools are updated
sudo pip install --upgrade pip virtualenv setuptools

# Then create a virtualenv named "kivyinstall" by either:

# 1. using the default interpreter
virtualenv --no-site-packages kivyinstall

# or 2. using a specific interpreter
# (this will use the interpreter in /usr/bin/python2.7)
virtualenv --no-site-packages -p /usr/bin/python2.7 kivyinstall
```

```

# Enter the virtualenv
. kivyinstall/bin/activate

# Use correct Cython version here
pip install Cython==0.23

# Install stable version of Kivy into the virtualenv
pip install kivy
# For the development version of Kivy, use the following command instead
# pip install git+https://github.com/kivy/kivy.git@master

```

Dependencies with legacy PyGame

Exemplo com Ubuntu

```

# Install necessary system packages
sudo apt-get install -y \
    python-pip \
    build-essential \
    mercurial \
    git \
    python \
    python-dev \
    ffmpeg \
    libsdl-image1.2-dev \
    libsdl-mixer1.2-dev \
    libsdl-ttf2.0-dev \
    libsmpeg-dev \
    libSDL1.2-dev \
    libportmidi-dev \
    libswscale-dev \
    libavformat-dev \
    libavcodec-dev \
    zlib1g-dev

```

Fedora

```

$ sudo yum install \
    make \
    mercurial \
    automake \
    gcc \

```

```
gcc-c++ \
SDL_ttf-devel \
SDL_mixer-devel \
KHRplatform-devel \
mesa-libGLES \
mesa-libGLES-devel \
gstreamer-plugins-good \
gstreamer \
gstreamer-python \
mtdev-devel \
python-devel \
python-pip
```

OpenSuse

```
$ sudo zypper install \
    python-distutils-extra \
    python-gstreamer-0_10 \
    python-enchant \
    gstreamer-0_10-plugins-good \
    python-devel \
    Mesa-devel \
    python-pip
$ sudo zypper install -t pattern devel_C_C++
```

Instalação

```
# Make sure Pip, Virtualenv and Setuptools are updated
sudo pip install --upgrade pip virtualenv setuptools

# Then create a virtualenv named "kivyinstall" by either:

# 1. using the default interpreter
virtualenv --no-site-packages kivyinstall

# or 2. using a specific interpreter
# (this will use the interpreter in /usr/bin/python2.7)
virtualenv --no-site-packages -p /usr/bin/python2.7 kivyinstall

# Enter the virtualenv
. kivyinstall/bin/activate

pip install numpy
```

```

pip install Cython==0.23

# If you want to install pygame backend instead of sdl2
# you can install pygame using command below and enforce using
# export USE SDL2=0. If kivy's setup can't find sdl2 libs it will
# automatically set this value to 0 then try to build using pygame.
pip install hg+http://bitbucket.org/pygame/pygame

# Install stable version of Kivy into the virtualenv
pip install kivy
# For the development version of Kivy, use the following command
→ instead
pip install git+https://github.com/kivy/kivy.git@master

```

Instale pacotes adicionais do Virtualenv

```

# Install development version of buildozer into the virtualenv
pip install git+https://github.com/kivy/buildozer.git@master

# Install development version of plyer into the virtualenv
pip install git+https://github.com/kivy/plyer.git@master

# Install a couple of dependencies for KivyCatalog
pip install -U pygments docutils

```

Inicie desde a Linha de Comando

Enviamos alguns exemplos que estão prontos para serem executados. No entanto, estes exemplos são empacotado dentro do pacote. Isso significa que deverás primeiro saber onde o easy_install instalou seu atual pacote Kivy e, em seguida, vá para o diretório *examples*:

```
$ python -c "import pkg_resources; print(pkg_resources.resource_
→filename('kivy', '../share/kivy-examples'))"
```

E deverás ter um caminho semelhante a:

```
/usr/local/lib/python2.6/dist-packages/Kivy-1.0.4_beta-py2.6-linux-
→x86_64.egg/share/kivy-examples/
```

Então poderás ir para o diretório de exemplo, e executar:

```
# launch touchtracer
$ cd <path to kivy-examples>
$ cd demo/touchtracer
$ python main.py

# launch pictures
$ cd <path to kivy-examples>
$ cd demo/pictures
$ python main.py
```

Se estiveres familiarizado com Unix e links simbólicos, poderás criar um link diretamente em seu diretório pessoal para facilitar o acesso. Por exemplo:

1. Obtenha o caminho de exemplo a partir da linha de comando acima
2. Cole-a em seu console:

```
$ ln -s <path to kivy-examples> ~/
```

3. Então, poderás acessar a pasta kivy-examples diretamente no seu diretório home:

```
$ cd ~/kivy-examples
```

Se desejas iniciar seus programas Kivy como scripts (digitando `./Main.py`) ou clicando duas vezes neles, irás querer definir a versão correta do Python lincando a mesma para ela. Algo como:

```
$ sudo ln -s /usr/bin/python2.7 /usr/bin/kivy
```

Ou, se estiveres executando o Kivy dentro de um virtualenv, o link para o interpretador Python dele, será algo como:

```
$ sudo ln -s /home/your_username/Envs/kivy/bin/python2.7 /usr/bin/
˓→kivy
```

Em seguida, dentro de cada arquivo `main.py`, adicione uma nova linha sendo esta a primeira do arquivo:

```
#!/usr/bin/kivy
```

NOTA: Cuidado com arquivos Python armazenados com terminação de linha em estilo Windows (CR-LF). O Linux não ignorará o <CR> e tentará usá-lo como parte do nome do arquivo. Isso torna as mensagens de erro confusas. Converta para terminação de linha de comando Unix.

Device permissions

When your app starts, Kivy uses `Mtdev` to scan for available multi-touch devices to use for input. Access to these devices is typically restricted to users or group with the appropriate permissions.

If you do not have access to these devices, Kivy will log an error or warning specifying these devices, normally something like:

```
Permission denied: '/dev/input/eventX'
```

In order to use these devices, you need to grant the user or group permission. This can be done via:

```
$ sudo chmod u+r /dev/input/eventX
```

for the user or:

```
$ sudo chmod g+r /dev/input/eventX
```

for the group. These permissions will only be effective for the duration of your current session. A more permanent solution is to add the user to a group that has these permissions. For example, in Ubuntu, you can add the user to the 'input' group:

```
$ sudo adduser $USER input
```

Note that you need to log out then back in again for these permissions to be applied.

Instalação no Android

O Kivy é um framework normal do Python, e simplesmente instalá-lo em um dispositivo Android da mesma forma que você faz num Desktop nada irá acontecer. No entanto, podes compilar um aplicativo Kivy para um APK Android padrão que será executado como um aplicativo Java normal em (mais ou menos) executá-lo em qualquer dispositivo.

Fornecemos várias ferramentas diferentes para ajudá-lo a executar o código em um dispositivo Android, a documentação pode ser acessada em [Documentação de Empacotamento de Apps Android](#). Estes incluem a criação de um APK totalmente autônomo que pode ser enviado para lojas Android, bem como a capacidade de executar seus aplicativos Kivy sem uma etapa de compilação usando o aplicativo Kivy Launcher pré-preparado.

Instalação no Raspberry Pi

Podes instalar o Kivy manualmente, ou podes baixar e inicializar o KivyPie no Raspberry Pi. Ambas as opções estão descritas abaixo.

Instalação Manual (No Raspbian Jessie)

1. Instalação das dependências:

```
sudo apt-get update
sudo apt-get install libsdl2-dev libsdl2-image-dev libsdl2-
    ↪mixer-dev libsdl2-ttf-dev \
        pkg-config libgl1-mesa-dev libgles2-mesa-dev \
        python-setuptools libgstreamer1.0-dev git-core \
        gstreamer1.0-plugins-{bad,base,good,ugly} \
        gstreamer1.0-{omx,alsa} python-dev libmtdev-dev \
        xclip
```

2. Install a new enough version of Cython:

```
sudo pip install -I Cython==0.23
```

3. Instale o Kivy globalmente em seu sistema:

```
sudo pip install git+https://github.com/kivy/kivy.git@master
```

4. Ou construa e use o kivy inplace (melhor para o desenvolvimento):

```
git clone https://github.com/kivy/kivy
cd kivy

make
echo "export PYTHONPATH=$(pwd):$PYTHONPATH" >> ~/.profile
source ~/.profile
```

Manual de instalação (No Raspbian Wheezy)

1. Adicione as fontes do APT para o Gstreamer 1.0 in */etc/apt/sources.list*:

```
deb http://vontaene.de/raspbian-updates/ . main
```

2. Adicionar as chave APT para vontaene.de

```
gpg --recv-keys 0C667A3E
gpg -a --export 0C667A3E | sudo apt-key add -
```

3. Instalação das dependências:

```
sudo apt-get update
sudo apt-get install libsdl2-dev libsdl2-image-dev libsdl2-
    ↪mixer-dev libsdl2-ttf-dev \
        pkg-config libgl1-mesa-dev libgles2-mesa-dev \
```

```
python-setuptools libgstreamer1.0-dev git-core \
gstreamer1.0-plugins-{bad,base,good,ugly} \
gstreamer1.0-{omx,alsa} python-dev
```

4. Instale o pip desde origem

```
wget https://raw.github.com/pypa/pip/master/contrib/get-pip.py
sudo python get-pip.py
```

5. Instale o Cython desde os fontes (o pacote Debian está desatualizado):

```
sudo pip install cython
```

6. Instale o Kivy globalmente em seu sistema:

```
sudo pip install git+https://github.com/kivy/kivy.git@master
```

7. Ou construa e use o kivy inplace (melhor para o desenvolvimento):

```
git clone https://github.com/kivy/kivy
cd kivy

make
echo "export PYTHONPATH=$(pwd):$PYTHONPATH" >> ~/.profile
source ~/.profile
```

Distribuição KivyPie

KivyPie é uma distribuição compacta e leve baseada Raspbian que vem com o Kivy instalado e pronto para ser executado. É o resultado da aplicação das etapas de instalação manual descritas acima, com mais algumas ferramentas extras. Podes baixar a imagem do link <http://kivypie.mitako.eu/kivy-download.html> e iniciá-lo no Raspberry PI.

Executando a Demonstração

Vá para a sua pasta *kivy/examples*, lá terás dezenas de exemplos que poderás estudar e realizar tentes.

Você poderia começar o Showcase:

```
cd kivy/examples/demo/showcase
python main.py
```

O demo 3D Monkey também é divertido de vê-lo:

```
cd kivy/examples/3Drendering  
python main.py
```

Altere a tela padrão usando

Você pode definir uma variável de ambiente chamada `KIVY_BCM_DISPMANX_ID` para alterar a exibição usada para executar o Kivy. Por exemplo, para forçar a exibição a ser HDMI, use:

```
KIVY_BCM_DISPMANX_ID=2 python main.py
```

Veja [Controlando o Environment](#) para ver todos os valores possíveis.

Usando a tela de toque oficial RPi

Se estiveres usando a tela de toque oficial do Raspberry Pi, precisarás configurar o Kivy para usá-lo como fonte de entrada. Para fazer isso, edite o arquivo `~/.kivy/config.ini` e vá para a seção `[input]`. Adicione isso:

```
mouse = mouse  
mtdev_(name)s = probesysfs,provider=mtdev  
hid_(name)s = probesysfs,provider=hidinput
```

Para obter mais informações sobre como configurar o Kivy, consulte [Configurar p Kivy](#)

Pra onde ir?

Fizemos alguns jogos usando GPIO/entrada física que tivemos durante o Pycon 2013: um botão e uma inclinação. Faça checkout do <https://github.com/kivy/piki>. Você precisará adaptar o pino GPIO no código.

Existe um vídeo para que vejas o que estávamos fazendo com ele: <http://www.youtube.com/watch?v=NVM09gaX6pQ>

1.1.2 Versão de Desenvolvimento

A versão de desenvolvimento é para desenvolvedores e testadores. Observe que ao executar uma versão de desenvolvimento, você está executando código potencialmente quebrado por sua conta e risco. Para usar a versão de desenvolvimento, primeiro você precisará instalar as dependências. Depois disso, você precisará configurar o Kivy no seu computador de uma maneira que permita um fácil desenvolvimento. Para isso, consulte nosso documento: ref: *Contributing* .

Instalação de Dependências

Para instalar as dependências do Kivy, siga o guia abaixo conforme a plataforma que estejas utilizando. Também poderás precisar desses pacotes para os componentes RST e lexing:

```
$ sudo pip install pygments docutils
```

Ubuntu

Para versões do Ubuntu 12.04 ou superior (testado para 14.04), basta digitar o seguinte comando será instalado todos os pacotes necessários:

```
$ sudo apt-get install python-setuptools python-pygame python-
˓→opengl \
  python-gst0.10 python-enchant gstreamer0.10-plugins-good python-
˓→dev \
  build-essential libgl1-mesa-dev-lts-quantal libgles2-mesa-dev-lts-
˓→quantal\
  python-pip
```

Para o Ubuntu 15.04 e versões anteriores a 12.04, este deve funcionar:

```
$ sudo apt-get install python-setuptools python-pygame python-
˓→opengl \
  python-gst0.10 python-enchant gstreamer0.10-plugins-good python-
˓→dev \
  build-essential libgl1-mesa-dev libgles2-mesa-dev zlib1g-dev_
˓→python-pip
```

Kivy requer uma versão recente do Cython, então é melhor usar a última versão de suporte do pypi:

```
$ sudo pip install --upgrade Cython==0.23
```

OS X

Sem usar *brew* podes instalar as dependências do Kivy colando manualmente os seguintes comandos num terminal:

```
curl -O -L https://www.libsdl.org/release/SDL2-2.0.4.dmg
curl -O -L https://www.libsdl.org/projects/SDL_image/release/SDL2_
˓→image-2.0.1.dmg
curl -O -L https://www.libsdl.org/projects/SDL_mixer/release/SDL2_
˓→mixer-2.0.1.dmg
curl -O -L https://www.libsdl.org/projects/SDL_ttf/release/SDL2_ttf-
˓→2.0.13.dmg
```

```
curl -O -L http://gstreamer.freedesktop.org/data/pkg/osx/1.7.1/
↳ gstreamer-1.0-1.7.1-x86_64.pkg
curl -O -L http://gstreamer.freedesktop.org/data/pkg/osx/1.7.1/
↳ gstreamer-1.0-devel-1.7.1-x86_64.pkg
hdiutil attach SDL2-2.0.4.dmg
sudo cp -a /Volumes/SDL2/SDL2.framework /Library/Frameworks/
```

Isso deverá pedir a sua senha de root, forneça-a e cole as seguintes linhas no seu terminal:

```
hdiutil attach SDL2_image-2.0.1.dmg
sudo cp -a /Volumes/SDL2_image/SDL2_image.framework /Library/
↳ Frameworks/
hdiutil attach SDL2_ttf-2.0.13.dmg
sudo cp -a /Volumes/SDL2_ttf/SDL2_ttf.framework /Library/Frameworks/
hdiutil attach SDL2_mixer-2.0.1.dmg
sudo cp -a /Volumes/SDL2_mixer/SDL2_mixer.framework /Library/
↳ Frameworks/
sudo installer -package gstreamer-1.0-1.7.1-x86_64.pkg -target /
sudo installer -package gstreamer-1.0-devel-1.7.1-x86_64.pkg -
↳ target /
pip install --upgrade --user cython pillow
```

Agora que tens todas as dependências do Kivy, precisarás ter a certeza de que tem as ferramentas de linha de comando instaladas:

```
xcode-select --install
```

Vá para um diretório apropriado como por exemplo:

```
mkdir ~/code
cd ~/code
```

Agora você pode instalar o próprio kivy:

```
git clone http://github.com/kivy/kivy
cd kivy
make
```

Isso deve compilar o Kivy, para torná-lo acessível em seu ambiente virtual Python, apenas aponte o seu PYTHONPATH para este diretório:

```
export PYTHONPATH=~/code/kivy:$PYTHONPATH
```

Para verificar se o Kivy está instalado, digite o seguinte comando em seu terminal:

```
python -c "import kivy"
```

Isso deverá lhe dar uma saída semelhante ao que temos a seguir:

```
$ python -c "import kivy"
[INFO    ] [Logger      ] Record log in /Users/quanon/.kivy/logs/
          ↴kivy_15-12-31_21.txt
[INFO    ] [Screen      ] Apply screen settings for Motorola Droid 2
[INFO    ] [Screen      ] size=480x854 dpi=240 density=1.5
          ↴orientation=portrait
[INFO    ] [Kivy       ] v1.9.1-stable
[INFO    ] [Python      ] v2.7.10 (default, Oct 23 2015, 18:05:06)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)]
```

OSX HomeBrew

Se preferires utilizar o homebrew: instale os requerimentos usando [homebrew](#):

```
$ brew install sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer
```

Windows

Veja [Use a Versão de Desenvolvimento](#).

Instalando a versão de Desenvolvimento do Kivy

Agora que já instalastes todas as dependências necessárias, é hora de baixar e compilar uma versão de desenvolvimento do Kivy:

Download do Kivy desde o GitHub:

```
$ git clone git://github.com/kivy/kivy.git
$ cd kivy
```

Compilar:

```
$ python setup.py build_ext --inplace -f
```

Se tiveres o comando `make` disponível, também poderás usar o seguinte atalho para fazer a compilação (faz o mesmo que o último comando):

```
$ make
```

Aviso: Por padrão, as versões 2.7 até 2.7.2 do Python usam o compilador GCC fornecido com versões anteriores do XCode. A partir da versão 4.2, somente o compilador Clang é fornecido com XCode por padrão. Isso significa que se construirás

usando o XCode 4.2 ou superior, precisarás ter instalado pelo menos a versão do Python 2.7.3, de preferência utilize a versão mais recente (2.7.5 no momento da escrita deste tutorial).

Se desejas modificar o código do Kivy você mesmo, defina a [variável de ambiente PYTHONPATH para o clone que fizestes do repositório](#). Desta forma, não precisarás instalar (`setup.py install`) após cada simples modificação. O Python irá importar o Kivy do seu clone.

Alternativamente, se não quiseres fazer nenhuma alteração no código do Kivy, também pode executar (como admin, por exemplo, com sudo)

```
$ python setup.py install
```

Se desejas contribuir com algum código (patches ou novos recursos) para a base de código do Kivy, por favor, leia esse documento [Contribuição](#).

Roda o conjunto de testes

Para ajudar a detectar problemas e mudanças de comportamento em Kivy, um conjunto de testes unitários são fornecidos. Uma boa coisa a fazer é executá-los logo após a instalação Kivy, e cada vez que você pretende executar uma mudança. Se você acha que algo foi interrompido em Kivy, talvez um teste irá mostrar isso. (Se não, pode ser um bom momento para escrever um.)

Os testes do Kivy são baseados no nosetest, que podes instalar a partir do seu gerenciador de pacotes ou usando pip:

```
$ pip install nose
```

Para rodar o conjunto de testes faça:

```
$ make test
```

Desinstalando Kivy

Se estiveres misturando várias instalações do Kivy, podes ficar confuso de onde cada versão do Kivy está localizada. Observe que talvez seja necessário seguir essas etapas várias vezes se tiveres várias versões do Kivy instaladas no PATH da biblioteca do Python. Para encontrar a versão atual instalada, podes usar a seguinte linha de comando:

```
$ python -c 'import kivy; print(kivy.__path__)'
```

Em seguida, remove esse diretório recursivamente.

Se você instalastes o Kivy com o `easy_install` no Linux, no diretório da instalação talvez exista um diretório “egg”. Remova-o também:

```
$ python -c 'import kivy; print(kivy.__path__)'  
['/usr/local/lib/python2.7/dist-packages/Kivy-1.0.7-py2.7-linux-x86_  
-64.egg/kivy']  
$ sudo rm -rf /usr/local/lib/python2.7/dist-packages/Kivy-1.0.7-py2.  
-7-linux-x86_64.egg
```

Se tiveres instalado usando o `apt-get`, faça o seguinte:

```
$ sudo apt-get remove --purge python-kivy
```

1.2 Filosofia

No caso de você estar se perguntando o que é Kivy e o que o diferencia de outras soluções, este documento é para você.

1.2.1 Porque se importar?

Por que você quer usar Kivy? Afinal, existem muitos outros bons toolkits (ou frameworks, ou plataformas) disponíveis por aí - gratuitamente. Você tem Qt e Flash, para citar apenas duas boas escolhas para o desenvolvimento de aplicativos. Muitas destas numerosas soluções já suportam Multi-Touch, então o que é que faz Kivy especial a ponto de valer a pena usá-lo?

Recente

Kivy é feito para o hoje e o amanhã. Novos métodos de entrada como o Multi-Touch tornaram-se cada vez mais importantes. Criamos Kivy a partir do zero, especificamente para esse tipo de interação. Isso significa que fomos capazes de repensar muitas coisas em termos de interação entre homem-computador, enquanto que os kits de ferramentas mais antigos (que não significam “ultrapassados”, e sim “bem estabelecidos”) carregam seu legado, o que por vezes é um fardo. Nós não estamos tentando forçar esta nova abordagem para que usem um computador no espartilho de modelos existentes (digamos, interação de mouse de um único ponteiro). Queremos deixá-lo florescer e deixá-lo explorar as possibilidades. Isso é o que realmente diferencia Kivy.

Rápido

Kivy é rápido. Isso se aplica tanto ao desenvolvimento de aplicativos quanto à velocidade de execução de aplicativos. Nós otimizamos Kivy de muitas maneiras. Implementamos a funcionalidade de tempo crítico no nível de C para alavancar o poder

dos compiladores existentes. Mais importante, também usamos algoritmos inteligentes para minimizar operações dispendiosas. Também usamos a GPU sempre que ela faz sentido no nosso contexto. O poder computacional das placas gráficas atuais supera, de longe, o das CPUs atuais para algumas tarefas e algoritmos, especialmente o drawing. É por isso que tentamos deixar a GPU fazer o máximo de trabalho possível, aumentando assim o desempenho consideravelmente.

Flexível

Kivy é flexível. Isso significa que ele pode ser executado em uma variedade de dispositivos diferentes, incluindo smartphones e tablets com Android. Suportamos todos os principais sistemas operacionais (Windows, Linux, OS X). Ser flexível também significa que o desenvolvimento rápido de Kivy permite que ele se adapte às novas tecnologias rapidamente. Mais de uma vez adicionamos suporte para novos dispositivos externos e protocolos de software, às vezes até antes de serem lançados. Por último, Kivy também é flexível na medida em que é possível usá-lo em combinação com um grande número de diferentes soluções de terceiros. Por exemplo, no Windows nós suportamos WM_TOUCH, o que significa que qualquer dispositivo que tenha drivers de Pen & Touch do Windows 7 funcionará no Kivy. No OS X, você pode usar os dispositivos compatíveis com a Multi-Touch da Apple, como trackpads e mouse. No Linux, você pode usar eventos de entrada de kernel HID. Além disso, apoiamos o TUO (Tangible User Interface Objects) e uma variedade de outras fontes de entrada.

Focado

Kivy é focado. Você pode escrever um aplicativo simples com algumas linhas de código. Programas Kivy são criados usando a linguagem de programação Python, que é incrivelmente versátil e poderosa, contudo, fácil de usar. Além disso, criamos nossa própria linguagem de descrição, a linguagem Kivy, para criar interfaces de usuário sofisticadas. Esta linguagem permite-lhe configurar, conectar e organizar os seus elementos de aplicação rapidamente. Sentimos que permitir que você se concentre na essência de sua aplicação é mais importante do que forçá-lo a mexer com as configurações de compilador. Nós tiramos esse fardo de seus ombros.

Financiado

Kivy é desenvolvido ativamente por profissionais que atuam em seus campos. Kivy é uma community-influenced, com desenvolvido profissional e comercialmente apoiado por solução. Alguns dos nossos principais desenvolvedores ganha a vida desenvolvendo o projeto Kivy. Kivy está aqui para ficar. Não é um pequeno projeto de estudante.

Gratuito

Kivy é livre para ser usado. Você não tem que pagar por isso. Você não tem sequer que pagar por isso mesmo que você esteja ganhando dinheiro com a venda de um aplicativo que usa Kivy.

1.3 Contribuição

Existem muitas maneiras que você pode contribuir para o Kivy. Pacotes de código são apenas uma coisa entre outras que você pode enviar para ajudar o projeto. Também recebemos comentários, relatórios de bugs, solicitações de recursos, melhorias na documentação, propaganda e defesa, testes, contribuições gráficas e diversas outras ideias. Apenas fale com a gente se você quer ajudar, e iremos ajudá-lo a nos ajudar.

1.3.1 Avaliação

Esta é de longe a maneira mais fácil de contribuir com algo. Se você estiver usando Kivy para seu próprio projeto, não hesite em compartilhar. Ele não tem que ser um aplicativo corporativo de alta classe, obviamente. É simplesmente incrivelmente motivador saber que as pessoas usam as coisas que você desenvolve e o quê lhes permite fazer. Se você tem algo que gostaria de nos dizer, por favor, não hesite. Screenshots e vídeos também são muito bem-vindos! Também estamos interessados nos problemas que você teve ao começar. Sinta-se encorajado a relatar quaisquer obstáculos que você encontrou, como falta de documentação, instruções enganosas ou similares. Somos perfeccionistas, por isso, mesmo que seja apenas um erro de digitação, avise-nos.

1.3.2 Reportar um requerimento

Se encontrou algo incorreto, uma interrupção, falha memória, documentação faltante, erro ortográfico ou exemplos desconexos, por favor use 2 minutos e preencha o relatório de solicitação.

1. Mova seu nível de registro para depurar editando `<user_directory>/.kivy/config.ini`:

```
[kivy]
log_level = debug
```

2. Execute seu código novamente e copie/cole a saída completa para <http://gist.github.com/>, incluindo o log de Kivy e o python backtrace.
3. Abra <https://github.com/kivy/kivy/issues/>
4. Defina o título da solicitação

5. Explique exatamente o que fazer para reproduzir o problema e cole o link da saída publicado em <http://gist.github.com/>
6. Valide a solicitação e pronto terminou!

Se você está se sentindo à vontade, você também pode tentar resolver o bug, e contribuir enviando-nos o patch :) Leia a próxima seção para descobrir como fazer isso.

1.3.3 Contribuição Código

As contribuições de código (patches, novos recursos) são a maneira mais óbvia de ajudar no desenvolvimento do projeto. Uma vez que isso é tão comum, pedimos que você siga nosso fluxo de trabalho para trabalhar com mais eficiência conosco. Aderir ao nosso fluxo de trabalho garante que sua contribuição não será esquecida ou perdida. Além disso, seu nome sempre será associado com a mudança que você fez, o que basicamente significa fama eterna no nosso histórico de código (você pode anular se você não quiser isso).

Estilo de Códificação

- Caso ainda não tenha lido, por favor leia [PEP8](#) sobre estilo de codificação em python.
- Ative a verificação PEP8 no git commit assim:

```
make hook
```

Isso passará o código adicionado à zona de teste do `git` (prestes a ser confirmado) por meio de um programa de verificação PEP8 quando fizeres um `commit` e certifique-se de que não introduziu erros PEP8. Caso haja erros, o `commit` será rejeitado: corrija os erros e tente novamente.

Performance

- Cuide de problemas de desempenho: leia [Dicas de desempenho do Python](#)
- Partes intensivas da cpu Kivy são escritas em cython: se você estiver fazendo um monte de computação, considere usá-lo também.

Git e GitHub

Usamos o `git` como nosso sistema de controle de versão para nossa base de código. Se nunca usastes o `git` ou um DVCS semelhante (ou mesmo qualquer VCS) antes, sugerimos que dê uma olhada na grande documentação disponível para o `git` online. O *Git Community Book* <<http://book.git-scm.com/>> _ ou os [Git Videos](#) são ótimas maneiras de aprender como o `git` funciona. Confie em nós quando dizemos que o `git` é uma

ótima ferramenta. Pode parecer assustadora no início, mas depois de um tempo (espero) irás amá-lo tanto quanto nós. Ensinar a usar o `git`, no entanto, está muito além do escopo deste documento.

Além disso, usamos *GitHub* <<http://github.com>> para hospedar nosso código. Na sequência, vamos supor que você tem uma conta (gratuita) GitHub. Embora esta parte seja opcional, permite uma integração firme entre os seus trechos e a nossa base de código upstream. Se você não quiser usar o GitHub, nós assumimos que você sabe o que você está fazendo de qualquer maneira.

Fluxo no Código

Então aqui está a configuração inicial para começar com o nosso fluxo de trabalho (você só precisa fazer isso uma vez para instalar o Kivy). Basicamente, você segue as instruções de instalação de: ref: *dev-install*, mas você não clona nosso repositório, você o exclui. Aqui estão os passos:

1. Log no GitHub
2. Crie um fork do [Kivy repository](#) clicando no botão *fork*.
3. Clone o fork do nosso repositório para o seu computador. Seu fork terá o nome remoto `git 'origin'` e estarás no ramo '`master`':

```
git clone https://github.com/username/kivy.git
```

4. Compile e configure `PYTHONPATH` ou instale (ver [Instalando a versão de Desenvolvimento do Kivy](#))
5. Instale nosso hook de pré-commit que garante que seu código não viole nosso guia de estilo executando `make hook` no diretório raiz do seu clone. Isso executará a nossa checagem de guia de estilo sempre que fizeres um commit, e se houver violações nas partes que alterastes, o seu `commit` será abortado. Corrija e tente novamente.
6. Adicionar repositório `kivy` como repositório remoto:

```
git remote add kivy https://github.com/kivy/kivy.git
```

Agora, quando desejar criar um patch, siga os seguintes passos:

1. Veja se existe um tiquete no tracker com a sua solicitação para correção ou funcionalidade adicional e marque que você está trabalhando nele, caso não tenha ninguém vinculado.
2. Criar um novo, apropriadamente nomeado branch eu seu repositório local para a funcionalidade específica ou correção de erro. (Mantenha o novo branch por funcionalidade, o que irá permitir facilmente fazer pull de suas atualizações sem interferir em nenhum outro aspecto fora do pretendido.):

```
git checkout -b new_feature
```

3. Modificar o código para fazer o que deseja (ex. correção).
4. Testar o código. Tente fazer isso com pequenas correções. Nunca irá saber se introduziu algum efeito colateral se não testar.
5. Fazer um ou mais, commits mínimos por correção ou por funcionalidade. Mínimo/Atômico significa *manter o commit claro*. Não faça atividades que não pertençam logicamente à correção ou à funcionalidade. **Não** quer dizer que deve fazer um commit por linha modificada. Usar `git add -p` se necessário.
6. Associe a cada commit uma mensagem de commit apropriada, assim outros que não estejam familiar com o assunto podem ter uma boa ideia do que foi modificado.
7. Quando estiver satisfeito com suas modificações, faça pull para o nosso repositório e faça o merge com seu repositório local. Nós iremos carregar suas coisas, mas desde que saiba exatamente o que modificou, faça o merge:

```
git pull kivy master
```

8. Faça um push do seu branch local em um repositório remoto no GitHub:

```
git push origin new_feature
```

9. Envie um *Pull Request* com a descrição do que modificou via o botão no interface do GitHub do seu repositório. (Isso é porque nós fazemos o fork inicialmente. Seu repositório está ligado ao nosso).

Aviso: Se você alterar partes da base de código que exigem compilação, você terá que recompilar para que as alterações entrem em vigor. O comando `make` fará isso por você (veja o `Makefile` se você quiser saber o que ele faz). Se você precisa limpar seu diretório atual de arquivos compilados, execute `make clean`. Se você quiser se livrar de **todos** arquivos que não estão sob controle de versão, execute `make distclean` (**Atenção: **Se suas alterações não estiverem sob controle de versão, este comando as eliminará!)

Agora receberemos seu pedido de `pull`. Vamos verificar se as suas alterações estão limpas e fazem sentido (se falastes conosco antes de fazer tudo isso, teremos dito se faz sentido ou não). Se assim for, vamos puxá-los e terás um karma instantâneo. Parabéns, você é um herói!

1.3.4 Contribuições para a Documentação

As contribuições de documentação geralmente seguem o mesmo fluxo de trabalho que as contribuições de código, mas são apenas um pouco mais relaxadas.

1. Siga as instruções acima,
 - (a) Fork o repositório.
 - (b) Clone seu fork em seu computador.
 - (c) Configurar o repositório Kivy como uma fonte remota.
2. Instale o Python-Sphinx. (Veja `docs/README` para assistência.)
3. Utilize [ReStructuredText Markup](#) para fazer alterações na documentação HTML em `docs/sources`.

Para enviar uma atualização da documentação, use as seguintes etapas:

1. Crie um novo branch nomeado apropriadamente em seu repositório local:

```
git checkout -b my_docs_update
```

2. Modifique a documentação com a sua correção ou melhoramento.
3. Re-crie as páginas HTML, e reveja sua atualização:

```
make html
```

4. Associe a cada commit uma mensagem de commit apropriada, assim outros que não estejam familiar com o assunto podem ter uma boa ideia do que foi modificado.
5. Mantenha cada commit focado num único tema relacionado. Não commit outros trabalhos que não pertencem logicamente a este trabalho.
6. Envie para o seu repositório remoto no GitHub:

```
git push
```

7. Envie um *Pull Request* com uma descrição do que você modificou através do botão na interface do GitHub do seu repositório.

Não pedimos que você passe por todos os problemas apenas para corrigir um único erro de digitação, mas para contribuições mais complexas, por favor, siga o fluxo de trabalho sugerido.

Docstrings

Cada módulo/classe/método/função precisa de um docstring, então utilize as seguintes palavras chaves quando relevante:

- `.. versionadded::` para marcar a versão na qual o recurso foi adicionado.
- `.. versionchanged::` para marcar a versão na qual o comportamento do recurso foi alterado.
- `.. note::` para adicionar informação sobre como usar o recurso ou o recurso descrito.
- `.. warning::` para indicar um problema potencial que o usuário pode executar ao usar o recurso.

Exemplos:

```
def my_new_feature(self, arg):
    """
    New feature is awesome

    .. versionadded:: 1.1.4

    .. note:: This new feature will likely blow your mind

    .. warning:: Please take a seat before trying this feature
    """
```

Resultará em:

`def my_new_feature(self, arg):` Recursos novos são incríveis
 Novo na versão 1.1.4.

Nota: Este novo recurso provavelmente vai “explodir” a sua mente

Aviso: Por favor, sente-se antes de experimentar este recurso

Ao se referir a outras partes da API use:

- `:mod:`~kivy.module`` para se referir a um módulo
- `:class:`~kivy.module.Class`` para se referir a uma classe
- `:meth:`~kivy.module.Class.method`` para se referir a um método
- `:doc:`api-kivy.module`` para se referir a um módulo da documentação (mesmo para uma classe e um método)

Obviamente substituindo *module Class* e *method* com seu nome real, e usando `‘.’` para separar módulos referentes a módulos imbricados, por exemplo:

```
:mod:`~kivy.uix.floatlayout`
:class:`~kivy.uix.floatlayout.FloatLayout`
```

```
:meth:`~kivy.core.window.WindowBase.toggle_fullscreen`  
:doc:``/api-kivy.core.window`
```

Resultará em:

floatlayout FloatLayout toggleFullscreen() Janela

:doc: e *:mod:* são essencialmente os mesmos, com exceção de uma âncora na URL que torna *:doc:* preferido para obter uma URL mais limpa.

Para construir sua documentação, execute:

```
make html
```

Se você atualizou sua instalação do kivy e tiver problemas para compilar documentos, execute:

```
make clean force html
```

Os documentos serão gerados em `docs build/html`. Para obter mais informações sobre a formatação de docstring, consulte a documentação oficial do Sphinx '<http://sphinx-doc.org/>'.

1.3.5 Contribuições para Teste Unitários

Para a equipe de testes, temos o documento `contribuer-unitest` que explica como os testes de unidade do Kivy funcionam e como podes criar o seu próprio. Use a mesma abordagem do *Code Workflow* para enviar novos testes.

Testes Unitários

Os testes estão localizados na pasta `kivy/tests`. Caso encontres um bug no Kivy, uma boa coisa a se fazer é escrever um caso mínimo demonstrando o problema e perguntar aos desenvolvedores principais se o comportamento ocorrido é o esperado ou realmente trata-se de um bug. Se escreveres seu código como sendo um *teste unitário* <http://docs.python.org/2/library/unittest.html>, isso evitará que o mesmo reapareça no futuro, e estarás contribuindo com o projeto Kivy, fazendo como que o mesmo fique mais confiável e estável. Escrever um `UnitTest` pode ser uma excelente forma para familiarizar-se com o Kivy ao mesmo tempo em que estás fazendo algo útil.

Os testes unitários são separados em dois casos:

- Testes unitários não gráficos: estes são testes unitários padrão que podem ser executados no console
- Testes Unitários Gráficos: estes necessitam um contexto GL, e funcionam através da comparação de imagens

Para ser capaz de executar testes unitários, precisas instalar o *Nose* (<http://code.google.com/p/python-nose/>), e o *Coverage* (<http://nedbatchelder.com/code/coverage/>). Podes utilizar o *easy_install* para realizar a instalação ou então o pip:

```
sudo easy_install nose coverage
```

Em seguida, no diretório Kivy:

```
make test
```

Como Isso Funciona

Todos os testes estão armazenados no diretório *kivy/tests*, e o nome do arquivo inicia co *test_<name>.py*. O *Nose* irá automaticamente reunir todos os arquivos e classes dentro desta pasta, e usá-los para gerar os testes unitários.

Para escrever um teste, crie um arquivo que respeite a nomenclatura anterior e, em seguida, inicie com este template:

```
import unittest

class XXXTestCase(unittest.TestCase):

    def setUp(self):
        # import class and prepare everything here.
        pass

    def test_YYY(self):
        # place your test case here
        a = 1
        self.assertEqual(a, 1)
```

Substitua XXX por um nome apropriado que cubra seus casos de teste e, em seguida, substitua 'YYY' pelo nome do seu teste. Caso tenhas dúvidas, observe a nomenclatura utilizada nos outros testes.

Em seguida, para executá-los, basta rodar:

```
make test
```

Se desejas executar esse arquivo apenas, podes executar da seguinte forma:

```
nosetests kivy/tests/test_your testcase.py
```

Testes Unitários GL

Teste de unidade GL são mais difíceis. Deves saber que, mesmo que o OpenGL seja um padrão, a saída/renderização não é. Tudo depende da sua GPU e do driver instalado. Para estes testes, o objetivo é salvar a saída da renderização no Frame X e compará-la com uma imagem de referência.

Atualmente, as imagens são geradas com 320x240 pixels e salvas no formato *png*.

Nota: Atualmente, a comparação de imagens é feita por pixel. Isso significa que a imagem de referência que você gerará só será correta para a sua GPU/driver. Se alguém pudem implementar a comparação de imagens que tenha suporte “delta”, este melhoramento será muito bem-vindo :)

Para executar testes unitários de GL, precisas criar um diretório:

```
mkdir kivy/tests/results  
make test
```

O diretório de resultados conterá todas as imagens de referência e as imagens geradas. Após a primeira execução, se o diretório de resultados estiver vazio, nenhuma comparação será feita. Ele usará as imagens geradas como referência. Após a segunda execução, todas as imagens serão comparadas às imagens de referência.

Um arquivo HTML está disponível para mostrar a comparação antes/depois do teste e um Snipper do teste de unidade é associado. Ele será gerado em:

kivy/tests/build/index.html

Nota: O diretório de compilação é limpo após cada chamada a *make test*. Caso não desejes, use o comando *nosetests*.

Escrevendo Testes Unitários GL

A idéia é criar um Widget raiz, como faria em *build()*, ou em *kivy.base.runTouchApp()*. Você vai dar esse Widget raiz para uma função de renderização que irá capturar a saída em X Frames.

Aqui tem um exemplo:

```
from common import GraphicUnitTest  
  
class VertexInstructionTestCase(GraphicUnitTest):  
  
    def test_ellipse(self):
```

```

from kivy.uix.widget import Widget
from kivy.graphics import Ellipse, Color
r = self.render

# create a root widget
wid = Widget()

# put some graphics instruction on it
with wid.canvas:
    Color(1, 1, 1)
    self.e = Ellipse(pos=(100, 100), size=(200, 100))

# render, and capture it directly
r(wid)

# as alternative, you can capture in 2 frames:
r(wid, 2)

# or in 10 frames
r(wid, 10)

```

Cada chamada a `self.render` (ou `r` no nosso exemplo) irá gerar uma imagem chamada da seguinte forma:

`<classname>_<funcname>-<r-call-count>.png`

`r-call-count` representa o número de vezes que `self.render` é chamado dentro da função de teste.

As imagens de referência são nomeadas:

`ref_<classname>_<funcname>-<r-call-count>.png`

Podes facilmente substituir a imagem de referência por uma nova, se desejas.

Relatórios de Cobertura

A cobertura é baseada na execução de testes anteriores. As estatísticas sobre cobertura de código são calculadas automaticamente durante a execução. Pode gerar um relatório HTML da cobertura com o comando:

`make cover`

Então, abra `kivy/htmlcov/index.html` com o seu navegador favorito.

1.3.6 GSOC

Prefácio

O Kivy é uma biblioteca multiplataforma de código aberto do Python, amigável, acelerada por GPU para rápido desenvolvimento de aplicações que faz uso das inovadoras interfaces de usuário como as de aplicativos multitoque.

A organização Kivy supervisiona diversos grandes projetos:

- The [Kivy](#) GUI Library
- Ferramenta de Compilação [Python-For-Android](#) .
- Ferramenta de Compilação para [Kivy-iOS](#).
- A biblioteca [PyJNIus](#) é utilizada para interagir com o Java a partir do Python.
- A biblioteca [PyOBJus](#) para interagir com o Objective-C a partir do Python.
- O wrapper Python [Plyer](#) independente de plataforma para APIs dependentes de plataforma.
- [Buildozer](#) - Um criador de pacotes genérico de Python para Android, iOS e desktop.
- [KivEnt](#) - Um 2d Game Engine que fornece métodos otimizados de manipulação de grandes quantidades de dados visuais dinâmicos.
- [Kivy Designer](#) - Um designer gráfico GUI para Kivy construído em Kivy.

Ao todo, esses projetos permitem que o usuário crie aplicativos para cada sistema operacional principal que faça uso de qualquer API nativa presente. Nossa objetivo é permitir o desenvolvimento de aplicativos Python que rodem em todos os lugares com o mesmo código base e façam uso de APIs dependentes de plataforma e recursos que os usuários de sistemas operacionais específicos têm aguardado.

Dependendo do projeto que você escolher, talvez você precise conhecer Cython, OpenGL ES2, Java, Objective-C ou C, além do Python. Fazemos uso pesado de Cython e OpenGL para computação e desempenho gráfico onde importa, e as outras linguagens são normalmente envolvidas no acesso ao SO ou provedora de nível API.

We are hoping to participate in Google Summer of Code 2017. This page showcases some ideas for GSoC projects and corresponding guidelines for students contributing to the Kivy Framework.

Requisitos

Presume-se que o aluno entrante ‘novo aluno’ atenda alguns requisitos básicos como destacado aqui:

- Nível intermediário familiaridade com Python.

- Confortável com git e github (Kivy e seus projetos irmãos são todos gerenciados no github) se você nunca usou github antes, você pode estar interessado neste [tutorial](#).
- Confortável com a programação orientada a eventos.
- Tem ferramentas / ambiente adequado para Kivy ou o projeto irmão que você está indo trabalhar à respeito. Por exemplo, para poder trabalhar no PyOJus, você iria precisar de acesso a um dispositivo iOS, OS X com Xcode e uma licença de desenvolvedor, para trabalhar em PyJNIus você precisaria de um dispositivo Android e para trabalhar em plyer você precisaria de acesso a hardware para ambas as plataformas.

Outras habilidades desejadas podem ser listadas com projetos específicos.

Familiarize-se com o '[guia de contribuição 'http://kivy.org/docs/contribute.html'](http://kivy.org/docs/contribute.html)' Podemos ajudá-lo a chegar até a velocidade, no entanto os alunos que demonstram habilidade com antecedência será dada preferência.

Como começar

Para Kivy, a maneira mais fácil é seguir as instruções de instalação da versão de desenvolvimento para sua plataforma específica:

<http://kivy.org/docs/installation/installation.html#development-version>

Para o resto é geralmente suficiente para instalar o projeto relevante do git e adicioná-lo ao seu PYTHONPATH.

Por exemplo, para o PyJNIus:

```
git clone http://github.com/kivy/pyjnius  
export PYTHONPATH=/path/to/pyjnius:$PYTHONPATH
```

Idéias do projeto

Aqui estão algumas ideias prospectivas provenientes da equipe de desenvolvimento Kivy, se nenhum desses projetos lhe interessar, venha falar conosco no #kivy-dev a respeito de uma ideia de projeto de sua preferência.

Projetos para iniciantes

Estes projetos devem ser adequados para qualquer pessoa com uma familiaridade de nível universitário com Python e requerem pouco conhecimento específicos da plataforma.

Projetos Intermediários

Esses projetos podem envolver conhecimento de nível superficial de vários detalhes a nível de Sistemas Operacionais, alguma interação com o OpenGL ou outros tópicos que podem ser um pouco fora de contexto da maioria dos programadores Python.

Plyer:

Descrição: Plyer é uma plataforma independente da API Python para usar os recursos comumente encontrados no desktop e plataformas móveis suportadas por Kivy. A ideia é fornecer uma API estável ao usuário para acessar recursos de seu desktop ou dispositivo móvel.

O aluno substituiria algum código *.java* presente no projeto p4a para um lugar mais apropriado em Plyer. Além disso, o aluno trabalharia na melhoria do acesso a recursos específicos da plataforma através do Plyer, incluindo acessibilidade, como Bluetooth Low Energy, acesso e edição de contatos, compartilhamento, NFC, navegador da aplicação, Wi-Fi (habilitar, desativar, acesso ao Wi-Fi, (Wi-Fi direct, network accessibility, current IP info on network etc.) captura de câmera (vídeo), exibição de câmera, integração do Google Play, interface de chamada de telefone, interface de SMS, geolocalização, interação com as notificações, internacionalização (I18N) e todas as implementações da faltantes da plataforma a partir dos recursos existentes.

Sob o capô, você usará PyJNIus no Android, PyOBJus no OS X e iOS, ctypes no Windows e APIs nativas no Linux. Isso provavelmente também inclui a melhoria do PyOBJus e PyJNIus para lidar melhor com interfaces que eles nesse momento não tem implementado.

Referências:

- <https://github.com/kivy/plyer>
- <https://github.com/kivy/pyjnius>
- <https://github.com/kivy/pyobjus>
- <https://github.com/kivy/python-for-android>
- <https://github.com/kivy/kivy-ios>

Resultado esperado: Um resultado bem sucedido incluiria mover o código Java/PyOBJus do p4a/kivy-ios para o Plyer e implementar algumas ou todas as novas facades a serem decididas com o aluno.

- **Mentors:** Akshay Arora
- **Requirements:** Acesso a Linux, Windows, OS X, dispositivo iOS, dispositivo Android.
- **Task level:** intermediário

- **Desired Skills:** Familiaridade com o PyJNIus, PyOBJus.

Font Reshaping and Font Fallback Support

Descrição: Currently Kivy does not support reshaping for alphabets such as Arabic, Persian, Thai, or Devanagari. The solution is to integrate a text shaping and layout engine (Pango and Harfbuzz). You would need to ensure that Pango and Harfbuzz can be compiled on every platform, and integrate it as a core text provider.

The second part of the same project would involve font fallback support. If a particular character/glyph is missing, currently we show a [] box. The solution for this would involve either using an OS API if available or maintaining a hashtable for the default fonts on each OS which can be used for glyph fallback.

Referências:

- <http://www.pango.org>
- <https://www.freedesktop.org/wiki/Software/HarfBuzz/>
- <https://github.com/kivy/kivy/tree/master/kivy/core/text>

Resultado esperado: Font fallback and text reshaping support in Kivy, compilation recipes for Python-For-Android and packaging on desktop platforms.

- **Mentors:** Akshay Arora, Jacob Kovac, Matthew Einhorn
- **Requirements:** Access to a desktop OS and ideally at least one mobile platform
- **Task level:** intermediário
- **Desired Skills:** Familiarity with text rendering, Pango, HarfBuzz and Kivy's provider abstraction.

Projetos avançados

Esses projetos podem envolver um conhecimento muito profundo dos componentes internos existentes do Kivy, os detalhes cabeludos da compilação entre plataformas ou outros tópicos bastante avançados. Se você está confortável com a parte interna do Python, se trabalha com código C e utiliza o Cython para construir suas próprias extensões C, estes projetos podem ser os ideias para você!!

Kivent: Chipmunk 7 Integration

Descrição: KivEnt é um motor de jogo baseado em entidade componente modular construído em cima da biblioteca Kivy. KivEnt fornece uma abordagem altamente performática para a construção de jogos com

Python que evita algumas das piores sobrecargas do Python usando construções Cython especializado.

At the moment, KivEnt internally makes use of the cymunk library (<https://github.com/tito/cymunk>) for physics simulation and collision detection. Cymunk is based on Chipmunk2d 6.x, recently Chipmunk 7 has released and brought many previously premium features into the core library. In addition to the API changes present in the newest Chipmunk, the KivEnt - Cymunk bridging does not make most efficient use of the KivEnt API for handling C level objects and data. The student will be responsible for creating a new wrapper over Chipmunk2d 7 that better matches KivEnt's approach to handling game data.

Referências:

- <http://chipmunk-physics.net/>
- <https://github.com/kivy/kivent>

Resultado esperado: Uma saída bem sucedida envolve um novo módulo kivent_tiled sendo lançada para o motor de jogo KivEnt.

- **Mentors:** Jacob Kovac
- **Requisitos:** Acesso a pelo menos uma plataforma Kivy.
- **Task level:** Avançado
- **Habilidades Desejadas:** Familiaridade com Cython, Python e conceitos de matemática relacionados ao desenvolvimento de jogos.

KV Compiler: A compiler for the KV language

Descrição: The KV language is a fundamental component of Kivy. The KV language allows one to describe a GUI; from the creation of a Widget tree to the actions that should be taken in response value changes and events. In effect it is a concise way to create rule bindings using the Kivy properties and events. Internally, python code that reflects these rules are created and bound to the properties and events. Currently, these bindings are not at all optimized because upon each widget creation all of these rules are re-evaluated and bound. This process can be significantly optimized by pre-compiling the kv code, especially the bindings. A compiler would also allow us to update and fix some of the long-standing kv language issues.

Work on a kv-compiler has already progressed quite far, in fact a PR in the pre-alpha stage, is currently open. However, it is out of sync with the current codebase due to some unrelated kv changes in the meantime. Also, that PR would require a significant re-write to make things more modular, self-contained, and extensible. So there is much work still to be done on it.

Theming has also been a prepatual issue in Kivy, a KV compiler may help implement bindings that facilitate theming.

Referências:

- <https://kivy.org/docs/guide/lang.html>
- <https://github.com/kivy/kivy/pull/3456>
- <https://github.com/kivy/kivy/wiki/KEP001:-Instantiate-things-other-than-widgets-from-kv>
- <https://github.com/kivy/kivy/issues/691>
- <https://github.com/kivy/kivy/issues/2727>

Resultado esperado: A successful outcome would be a compiler which compiles kv code into python code. The compiler should be modular and extensible so that we can continue to improve the kv language. The compiler should have the common debug/optimization options. The compiled code should also be human readable so issues could be traced back to the original kv code. The compiler should also be a drop in replacement for the current KV runtime compiler, and would require extensive testing.

- **Mentors:** Matthew Einhorn
- **Requisitos:** Acesso a pelo menos uma plataforma Kivy.
- **Task level:** Avançado
- **Desired Skills:** Familiarity with Cython, Python, and Kivy. Familiarity with typical computer science concepts and data structures is also desired.

Como entrar em contato com os desenvolvedores

Toda a comunicação deve acontecer através de canais públicos, e-mails privados e mensagens IRC são desencorajados.

Faça suas perguntas no fórum de usuários do Kivy <https://groups.google.com/group/kivy-users> ou envie um e-mail para kivy-users@googlegroups.com

Certifique-se também, de participar do grupo de usuários do kivy-dev: <https://groups.google.com/forum/#!forum/kivy-dev>.

Você também pode tentar entrar em contato conosco no IRC (bate-papo online), para obter os identificadores IRC dos desenvolvedores mencionados acima, visite <https://kivy.org/#aboutus>.

Certifique-se de ler o [IRC rules](#) antes de conectar. [Connect to webchat](#).

A maioria dos nossos desenvolvedores estão localizados na Europa, Índia e América do Norte, portanto, tenha em mente as horas de vigília típicas dessas áreas.

Como ser um bom aluno

Se você quiser participar como aluno e quiser maximizar suas chances de ser aceito, comece a falar conosco hoje e tente corrigir alguns problemas menores para se acostumar com o nosso fluxo de trabalho. Se sabermos que você pode trabalhar bem conosco, você terá chances muito melhores de ser selecionado.

Aqui há uma lista de verificação:

- Certifique-se de ler o site e, pelo menos, skim a documentação.
- Olhe para o código-fonte.
- Leia nossas diretrizes de contribuição.
- Faça uma contribuição! Kivy gostaria de ver como você se envolve com o processo de desenvolvimento. Dê uma olhada no rastreador de problemas para um projeto do Kivy que lhe interessa e envie um pedido de solicitação. Pode ser um bug simples ou uma mudança de documentação. Estamos procurando ter uma idéia de como você trabalha, não avaliando suas capacidades. Não se preocupe em tentar escolher algo para nos impressionar.
- Escolha uma ideia que você acha que é interessante a partir da lista de ideias ou vir acima com sua própria ideia.
- Faça alguma pesquisa **você mesmo**. GSoC diz respeito a receber, não apenas uma interação unilateral. Trata-se de você tentar alcançar metas acordadas com o nosso apoio. A força motriz principal neste trabalho deve ser, obviamente, você mesmo. Muitos alunos aparecem e perguntam o que devem fazer. Você deve basear essa decisão em seus interesses e suas habilidades. Mostre-nos que você é sério e que tens iniciativa.
- Escreva um rascunho **proondo** sobre o que você quer fazer. Inclua o que você entende do estado atual do projeto, o que gostaria de melhorar, como irás implementar e etc.
- Discuta essa proposta conosco em tempo hábil. Obtenha feedback.
- Seja paciente! Especialmente no IRC. Vamos tentar chegar até você se estivermos disponíveis. Se não, envie um e-mail e espere. A maioria das perguntas já estão respondidas nos documentos ou em outro lugar e podem ser encontradas com alguma pesquisa. Suas perguntas devem refletir que você realmente pensou através do que estás perguntando e fez alguma pesquisa ainda que básica.
- Acima de tudo, não se esqueça de se divertir e interagir com a comunidade. A comunidade é uma parte tão grande do Open Source quanto o próprio código.

O que esperar se você for escolhido

- Todos os alunos devem se juntar aos canais #kivy e #kivy-dev irc diariamente, assim é como a equipe de desenvolvimento se comunica internamente e com os usuários.
- Você e seus mentores concordarão em dois marcos miliários para a duração do verão.
- O desenvolvimento ocorrerá em seu fork do ramo master do Kivy, esperamos que você envie pelo menos um PR por semana do seu branch em um branch reservado para você no repo principal. Este será o seu fórum para relatar o progresso, bem como documentar quaisquer luta que você possa vir a encontrar.
- Faltar 2 PR semanais ou 2 milestones resultará em sua perda, a menos que haja circunstâncias atenuantes. Se alguma coisa surgir, por favor informe seus mentores o mais rápido possível. Se um alguma meta parece fora de alcance, trabalharemos com você para reavaliar os objetivos.
- Suas alterações serão mescladas no master quando o projeto for concluído e tivermos testado minuciosamente em todas as plataformas que forem relevantes.

1.4 FAQ

Existe uma quantidade de questões que repetidamente precisam ser respondidas. O documento a seguir tenta responder algumas delas.

1.4.1 FAQ Técnico

Fatal Python error: (pygame parachute) Segmentation Fault

Na maioria das vezes, este problema é devido ao uso de drivers de gráficos antigos. Instale a última versão do driver gráfico disponível da sua placa gráfica, e o mesmo deverá funcionar.

Caso não funcione, isso significa que você provavelmente disparou algum código OpenGL sem um contexto OpenGL disponível. Se estiveres carregando imagens, atlas ou usando instruções gráficas, precisarás criar primeira uma Janela:

```
# method 1 (preferred)
from kivy.base import EventLoop
EventLoop.ensure_window()

# method 2
from kivy.core.window import Window
```

Caso contrário, por favor informe um problema detalhado no github seguindo as instruções na seção *Reportar um requerimento* da documentação *Contribuição*. Isso é muito importante para nós porque esse tipo de erro pode ser muito difícil de depurar. Dê-nos todas as informações que você pode dar sobre seu ambiente e execução.

simbolo indefinido: glGenerateMipmap

Sua placa gráfica ou seus drivers podem ser antigos. Atualize seus drivers gráficos para a última versão disponibilizada pelo fabricante e tente novamente.

ImportError: Nenhum módulo chamado evento

Se utilizas o Kivy na versão que está em desenvolvimento, precisarás compila-la antes de poder utilizar. Vá no diretório Kivy e faça o seguinte:

```
make force
```

1.4.2 FAQ Android

Não possível extrair dados públicos

Esta mensagem de erro pode ocorrer sob várias circunstâncias. Assegure que:

- você tem um telefone com um SDCard
- você não está atualmente no modo “USB Mass Storage”
- Você tem permissões para escrever no *sdcards*

No caso do erro de modo “USB Mass Storage”, e se não quiseres manter desconectando o dispositivo, defina a opção USB como sendo Power (ligado).

Dá pau na interação de toque na versão do Android 2.3.x

Houve relatos de falhas em dispositivos baseados em Adreno 200/205. Apps de outra forma funcionam bem, mas falham quando interagiram com/através da tela.

Esses relatórios também mencionaram o problema sendo resolvido quando movidos para um ICS ou ROM superior.

É possível ter um aplicativo Kosk no Android 3.0?

Thomas Hansen escreveu uma resposta detalhada na lista de discussão do kivy-users:

https://groups.google.com/d/msg/kivy-users/QKoCekAR1c0/yV-85Y_iAwoJ

Basicamente, você precisa da raiz ‘root’ do dispositivo, remover o pacote SystemUI, adicionar algumas linhas para a configuração xml, e você estará pronto.

Qual é a diferença entre python-to-android do Kivy e SL4A?

Apesar de ter o mesmo nome, python-for-android de Kivy não está relacionado com o projeto python-android de SL4A, Py4A ou android-python27. São projetos distintamente diferentes com objetivos diferentes. Você pode usar Py4A com Kivy, mas nenhum código ou esforço foi feito para fazê-lo. A equipe de Kivy sente que nosso python-for-android é a melhor solução para nós indo para frente, e as tentativas de integrar e suportar Py4A não é um bom uso do nosso tempo.

1.4.3 FAQ Projeto

Por que você usa Python? O mesmo não é lento?

Vamos tentar fornecer uma resposta completa; Por favor, tenha um pouco de paciência conosco.

O Python é uma linguagem super ágil e que permite implementarmos grandes trabalhos num curto período de tempo (em comparação a outras linguagens). Em muitos cenários de desenvolvimento, preferimos escrever nosso aplicativo da forma mais rápida possível utilizando uma linguagem de alto nível, como Python, testa-lo, em seguida, caso seja necessário, otimizar a aplicação.

Certo, e a performance? Se você comparar a velocidade de execução de implementações para um determinado conjunto de algoritmos (esp. Número crunching) irás achar que o Python é muito mais lento do que o C++. Pode estar ainda mais pensando não ser uma boa ideia em nosso caso usar o Python. Desenhar gráficos sofisticados (e não estamos falando sobre o OpenGL da sua avó aqui) é computacionalmente muito caro e dado que muitas vezes queremos fazer isso para experiências de usuário complexas, que seria um argumento justo. **Mas**, em praticamente todos os casos, sua aplicação acaba gastando a maior parte do tempo (de longe) executando a mesma parte do código. Em Kivy, por exemplo, essas partes são a invocação de eventos e o desenho gráfico. Agora, o Python permite que você faça algo para tornar essas partes muito mais rápidas.

Usando Cython, poderás compilar seu código até o nível C, e de lá seu compilador C usual otimizará as coisas. Este é um processo que não deveria dar problemas se adicionares algumas dicas ao seu código, o resultado se torna ainda mais rápido. Estamos falando de uma aceleração no desempenho por um fator de qualquer coisa entre 1x e até mais de 1000x (depende muito do seu código). Em Kivy, fizemos isso para você e implementamos as partes de nosso código, onde a eficiência realmente é crítica, no nível C.

Para o desenho gráfico, também alavancamos as GPUs de hoje, que são, para algumas

tarefas, como a rasterização gráfica, muito mais eficiente do que uma CPU. O Kivy faz tanto quanto for razoável na GPU para maximizar o desempenho. Se usares nossa API de tela para fazer o desenho, existe até um compilador que inventamos otimizando automaticamente o código de desenho. Se mantiveres seu desenho principalmente na GPU, grande parte da velocidade de execução do programa não será determinada pela linguagem de programação utilizadas, mas pelo hardware gráfico onde o mesmo está sendo executado.

Acreditamos que essas (e outras) otimizações que o Kivy faz para você já tornam a maioria das aplicações suficientemente rápida o bastante. Muitas vezes você ainda vai querer limitar a velocidade da aplicação, a fim de não desperdiçar recursos. Mas mesmo que isso não seja suficiente, você ainda tem a opção de usar o Cython para o seu próprio código para acelerar *muito*.

Confie em nós quando dizemos que nós damos a isto um pensamento muito cuidadoso. Realizamos vários benchmarks diferentes e elaboramos algumas otimizações inteligentes para fazer o seu aplicativo funcionar sem problemas.

O Kivy suporta Python 3.x?

Sim! A partir da versão 1.8.0, o Kivy suporta Python ≥ 2.7 e Python ≥ 3.3 com o mesmo código base. O Python 3 agora também é suportado pelo python-para-android.

No entanto, esteja certo de que, embora o Kivy seja executado em Python 3.3+, nossas ferramentas de compilação do iOS ainda requerem a versão do Python 2.7.

Como Kivy está relacionado ao PyMT?

Nossos desenvolvedores são profissionais e são muito experientes em sua área de especialização. No entanto, antes de Kivy surgir havia (e ainda há) um projeto chamado PyMT que foi liderado por nossos desenvolvedores principais. Aprendemos muito com esse projeto durante o tempo em que o desenvolvemos. Nos mais de dois anos de pesquisa e desenvolvimento, encontramos muitas maneiras interessantes de melhorar o design do nosso framework. Realizamos inúmeros benchmarks e, como resultado, para conseguir a grande velocidade e flexibilidade que Kivy tem, tivemos que reescrever uma grande parte do código-base, tornando esta uma decisão sem voltas, mas visando o futuro. Os mais notáveis são os aumentos de desempenho, que são simplesmente incríveis. Kivy inicia e opera muito mais rápido, devido a estas otimizações pesadas. Também tivemos a oportunidade de trabalhar com empresas e associações usando PyMT. Fomos capazes de testar o nosso produto em uma grande diversidade de configurações e o PyMT trabalhou em todos elas. Escrever um sistema como Kivy ou PyMT é uma coisa. Fazê-lo funcionar sob todas essas condições diferentes é outra. Nós temos uma boa experiência aqui, e a levamos para o Kivy.

Além disso, uma vez que alguns dos nossos principais desenvolvedores decidiram abandonar seus empregos em tempo integral e se dedicar a este projeto integralmente,

foi decidido que uma fundação mais profissional tinha que ser estabelecida. O Kivy é uma fundação. Isso supõem ser um produto estável e profissional. Tecnicamente, Kivy não é realmente um sucessor para PyMT porque não há caminho de migração fácil entre ambos os códigos. No entanto, o objetivo é o mesmo: Produzir aplicativos de alta qualidade para novas interfaces de usuário. É por isso que incentivamos todos a basear novos projetos em Kivy ao invés de utilizar o PyMT. O desenvolvimento ativo do PyMT foi interrompido. A manutenção e patches ainda são aceitos.

Você aceita patches?

Sim, nós amamos trechos. A fim de garantir uma integração harmoniosa das suas preciosas mudanças, no entanto, certifique-se de ler as nossas orientações de contribuição. Obviamente, não aceitamos todos trechos. Seu trecho tem que ser consistente com o nosso guia de estilo e, mais importante, fazer sentido. Faz sentido falar com a gente antes que você venha com maiores mudanças, especialmente novos recursos.

O projeto Kivy participa do Google Summer of Code?

Potenciais estudantes perguntam se participamos do GSoC. A resposta clara é: óbvio!!

Se você quiser participar como aluno e quiser maximizar suas chances de ser aceito, comece a conversar conosco hoje e tente corrigir alguns problemas menores (ou maiores, se você conseguir ;-) para se acostumar ao nosso fluxo de trabalho. Se entendermos que você pode trabalhar bem conosco, isso seria uma grande vantagem.

Aqui há uma lista de verificação:

- Certifique-se de ler o site e, pelo menos, skim a documentação.
- Olhe para o código-fonte.
- Leia nossas diretrizes de contribuição.
- Escolha uma ideia que acreditas ser interessante a partir da nossas lista de ideias (veja o link acima) ou então, traga a sua própria ideia!
- Faça alguma pesquisa **você mesmo**. O GSoC não diz respeito a nós lhe ensinando algo e você ganhará pago por isso. Trata-se de você tentar alcançar metas acordadas por si mesmo com o nosso apoio. A força motriz principal deverá ser, obviamente, a sua! Muitos alunos vêm e perguntam o que devem fazer. Bem, não sabemos porque não conhecemos nem seus interesses nem suas habilidades. Mostre-nos que és sério e que desejas realmente fazer algo interessante e que tens iniciativa.
- Escreva uma proposta preliminar sobre o que você pretende implementar. Inclua o que você entende de como funciona neste momento o que desejas melhorar (pode ser bem grosseiramente), e o que gostarias de melhorar e como pretendes fazer isso e etc....

- Discuta essa proposta conosco em tempo hábil. Obtenha feedback.
- Seja paciente! Especialmente no IRC. Vamos tentar chegar até você se estivermos disponíveis. Se não, envie um e-mail e espere. A maioria das perguntas já são respondidas nos documentos ou em outro lugar e podem ser encontradas com alguma pesquisa. Se suas perguntas não refletem o que você realmente pensou através do que você está solicitando, isso pode não ser bem recebido.

Boa sorte!:-)

1.5 Contate-nos

Você pode contactar-nos de diferente formas:

1.5.1 Rastreador de problemas

Se você encontrou um problema com o código ou tem uma solicitação de recurso, consulte nosso 'Rastreador de problemas <<https://github.com/kivy/kivy/issues>>'. Se não houver nenhum problema ainda que corresponda ao seu inquérito, sinta-se livre para criar um novo. Por favor, certifique-se de receber os e-mails que o github envia se comentarmos sobre o problema caso necessitemos de mais informações. Para erros, forneça todas as informações necessárias, como o sistema operacional que está usando, a **mensagem de erro completa**, ou algum outro registro, uma descrição do que você fez para acionar o erro e qual foi o real erro, bem como qualquer outra coisa que possa ser de interesse. Obviamente, somente poderemos ajudá-lo se nos contar precisamente qual o real problema.

1.5.2 Correio

Para utilizadores de nossa estrutura, existe uma lista de discussão para consultas de suporte no grupo de utilizadores do 'kivy-users Google Group <<https://groups.google.com/group/kivy-users>>'. Use esta lista se você tiver problemas com seu aplicativo baseado em Kivy. Nós também temos uma lista de discussão para assuntos que tratam do desenvolvimento do código de estrutura do Kivy no 'kivy-dev Google Group <<https://groups.google.com/group/kivy-dev>>_.

1.5.3 IRC

#Kivy on irc.freenode.net

IRC é ótimo para comunicação em tempo real, mas por favor certifique-se de **esperar** depois de fazer sua pergunta. Se você apenas se juntar, perguntar e sair, não temos **nenhuma maneira** de saber quem você era e onde devemos enviar nossa resposta. Além

disso, tenha em mente que estamos na sua maior parte baseados na Europa, portanto, leve em consideração quaisquer problemas de fuso horário. Se você foi infeliz mais de uma vez, tente a lista de discussão.

Se você não tem uma conta no IRC, você pode também usar o bate papo da web da Freenode '<http://webchat.freenode.net/>', mas por favor, não feche a janela do navegador muito cedo. Basta digitar "#kivy" no campo de canais.

Por favor, leia nossas 'Diretrizes da Comunidade <<https://github.com/kivy/kivy/wiki/Community-Guidelines>>', antes de pedir ajuda na lista de discussão ou no canal do IRC.

Guia de Programação

2.1 Básico do Kivy

2.1.1 Instalação do Ambiente Kivy

O Kivy depende de muitas bibliotecas do Python, como o PyGame, GStreamer, PIL, Cairo e várias outras. Essas bibliotecas não são todas necessários, mas dependendo da plataforma que estiveres trabalhando, elas poderão se tornar uma dor de cabeça para serem instaladas. No Windows e no MacOS X, fornecemos um pacote portátil que bastará apenas descompactar e usar.

- *Instala no Windows*
- *Instalação no OS X*
- *Instalação no Linux*

Se desejas instalar tudo sozinho, certifique-se de ter pelo menos o [Cython](#) e o [Pygame](#). Uma típica instalação usando o pip seria algo do tipo:

```
pip install cython  
pip install hg+http://bitbucket.org/pygame/pygame  
pip install kivy
```

A [versão de desenvolvimento](#) pode ser instalado com a ferramenta git:

```
git clone https://github.com/kivy/kivy  
make
```

2.1.2 Criando um Aplicação

Criar uma aplicação kivy é tão simples como:

- Subclasse a classe `App`
- Implementando o seu métodos `build()` para que ele devolva uma instância da `Widget` (a raiz de sua árvore de widgets)
- instanciando esta classe e invocando este método `run()`.

Aqui temos um exemplo de aplicação mínima:

```
import kivy
kivy.require('1.0.6') # replace with your current kivy version !

from kivy.app import App
from kivy.uix.label import Label


class MyApp(App):

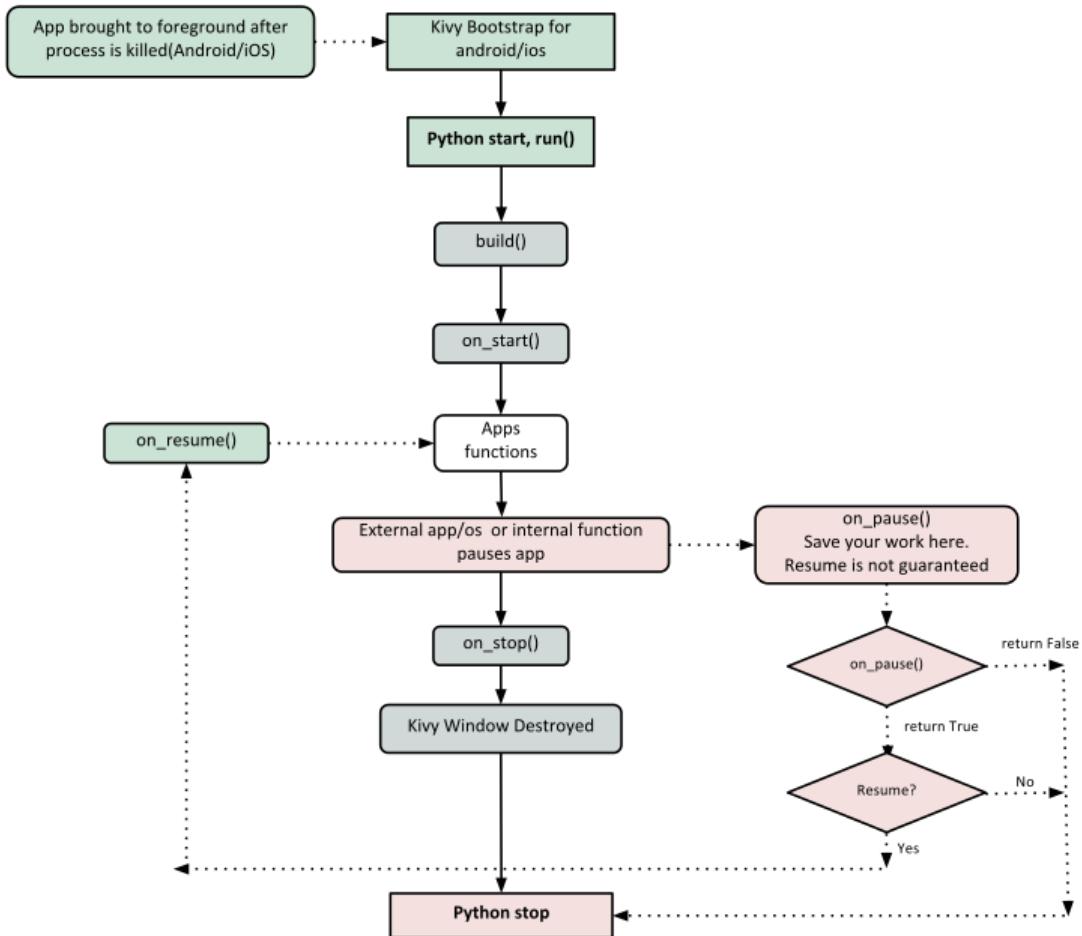
    def build(self):
        return Label(text='Hello world')

if __name__ == '__main__':
    MyApp().run()
```

Podes salvar isso em um arquivo de texto, `main.py`, por exemplo, e executá-lo.

2.1.3 Ciclo de Vida de uma Aplicação Kivy

Primeiro, vamos nos familiarizar com o ciclo de vida de um aplicativo Kivy.



Como pode ser visto acima, para todos os efeitos, o ponto de entrada numa App é o método `run()`, e em nosso caso que temos uma “`MyApp().Run ()`”. Voltaremos a falar disso, mas começaremos a partir da terceira linha:

```
from kivy.app import App
```

É necessário que a classe base do seu aplicativo herde da classe `App`. Ela está disponível em `kivy_installation_dir/kivy/app.py`.

Nota: Vá em frente, abra esse arquivo caso queiras se aprofundar no que a classe App do Kivy faz e como tudo está implementado. Incentivamos você a abrir o código e lê-lo. O Kivy é baseado no Python e usa o Sphinx para documentação, portanto a documentação para cada classe está junto a cada classe e a cada função.

Similarmente na linha 2:

```
from kivy.uix.label import Label
```

Uma coisa importante a observar aqui é a forma como os pacotes/classes são definidos. O módulo `uix` é a seção que contém os elementos da interface do usuário como os Layouts e os Widgets.

Passando para a linha 5:

```
class MyApp(App):
```

É aqui que **definimos** a Classe Base da nossa aplicação Kivy. Você só precisará alterar o nome do aplicativo *MyApp* nesta linha.

Mais adiante na linha 7:

```
def build(self):
```

Conforme destacado na imagem acima, veja o *Kivy App Life Cycle*, esta é a função onde deves inicializar e retornar o seu *Root Widget*. Isto é o que fazemos na linha 8:

```
return Label(text='Hello world')
```

Aqui inicializamos um Label com o texto 'Hello World' e retornamos sua instância. Este Label será o Widget Raiz desta Aplicação.

Nota: O Python usa indentação para denotar blocos de código, portanto, note que no código fornecido acima, na linha 9 a definição da classe e da função terminam.

Agora a parte que fará nosso aplicativo rodar na linha 11 e 12:

```
if __name__ == '__main__':
    MyApp().run()
```

Aqui a classe *MyApp* é inicializada e seu método *run()* é invocado. Isso inicializa e começa uma aplicação Kivy.

2.1.4 Executando o aplicativo

Para executar o aplicativo, siga as instruções para o seu sistema operacional:

Linux Siga as instrução para *executar uma aplicação Kivy no Linux*:

```
$ python main.py
```

Windows Siga as instrução para *executar uma aplicação Kivy no Windows*:

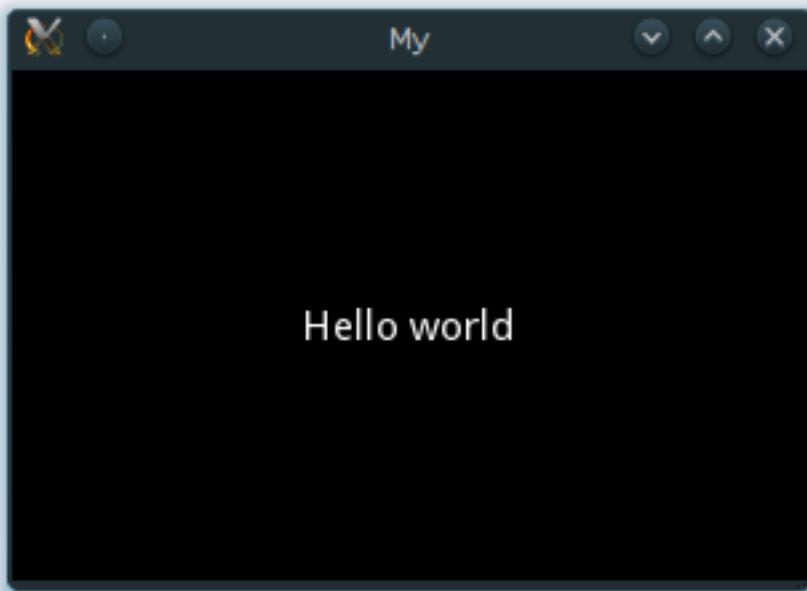
```
$ python main.py
# or
C:\appdir>kivy.bat main.py
```

Mac OS X Siga as instruções para *executar uma aplicação Kivy no OS X*:

```
$ kivy main.py
```

Android Seu aplicativo precisa de alguns arquivos complementares para poder rodar no Android. Consulte a documentação oficial [Criando um pacote para o Android](#) para obter mais informações.

Uma janela deverá ser aberta, mostrando um único Label (com o texto 'Hello World') que cobre toda a área da janela. Isso é tudo que há para uma aplicação.



2.1.5 Personalizando a Aplicação

Vamos estender esta aplicação um pouquinho, vamos dizer que temos uma simples página de UserName/Password.

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput

class LoginScreen(GridLayout):
    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.cols = 2
```

```

        self.add_widget(Label(text='User Name'))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
        self.add_widget(Label(text='password'))
        self.password = TextInput(password=True, multiline=False)
        self.add_widget(self.password)

class MyApp(App):

    def build(self):
        return LoginScreen()

if __name__ == '__main__':
    MyApp().run()

```

Na linha 2 nós importamos uma GridLayout:

```
from kivy.uix.gridlayout import GridLayout
```

Esta classe é usada como Base para o nosso Widget Raiz (LoginScreen) definido na linha 9:

```
class LoginScreen(GridLayout):
```

Na linha 12 da tela de Login, sobrecrevemos o método `__init__()` para adicionar Widgets e definir o seu comportamento:

```
def __init__(self, **kwargs):
    super(LoginScreen, self).__init__(**kwargs)
```

Não se pode esquecer de invocar o super para implementar a funcionalidade da classe original que está sendo sobrecrevada. Observe também que é uma boa prática não omitir o `kwargs` ao invocar super, já que eles são às vezes usados internamente.

Passando para a Linha 15 e continuando:

```

self.cols = 2
self.add_widget(Label(text='User Name'))
self.username = TextInput(multiline=False)
self.add_widget(self.username)
self.add_widget(Label(text='password'))
self.password = TextInput(password=True, multiline=False)
self.add_widget(self.password)

```

Pedimos ao GridLayout para gerenciar seus filhos em duas colunas e adicionamos um `Label` e um `TextInput` para que seja informado o nome de usuário e senha.

Executando o código acima terás uma janela que deverá se parecer com algo do tipo:



Tente re-dimensionar a janela e verás que os Widgets na tela se ajustam de acordo com o tamanho da janela sem que precises fazer absolutamente nada. Isso ocorre porque os Widgets usam as propriedades de tamanho *hint_size* por padrão.

O código acima não manipula a entrada do usuário, e não faz nenhuma validação ou qualquer outra coisa. Nas próximas seções, vamos nos aprofundar **Widget** no tamanho e no posicionamento dos Widgets.

2.2 Controlando o Environment

Muitas das variáveis de ambiente estão disponíveis para controlar a inicialização e o comportamento do Kivy.

Por exemplo, para restringir a renderização de texto da implementação do PIL:

```
$ KIVY_TEXT=pil python main.py
```

Variáveis de ambiente devem ser definidas antes de importar o Kivy:

```
import os
os.environ['KIVY_TEXT'] = 'pil'
import kivy
```

2.2.1 Controle do caminho

Novo na versão 1.0.7.

You can control the default directories where config files, modules and kivy data are located.

KIVY_DATA_DIR Localização dos dados do Kivy, o padrão é <kivy path>/data

KIVY_MODULES_DIR Local dos módulos do Kivy, o padrão é <kivy path>/modules

KIVY_HOME Local da pasta Home do Kivy. Este diretório será utilizado para salvar as configurações locais e deverá estar num local gravável.

O padrão é:

- Desktop: <user home>/.kivy
- Android: <android app path>/.kivy
- iOS: <user home>/Documents/.kivy

Novo na versão 1.9.0.

KIVY SDL2_PATH Se definido, as bibliotecas e cabeçalhos SDL2 desse caminho são usados para compilar o Kivy em vez daqueles instalados em todo o sistema. Para usar as mesmas bibliotecas ao executar uma aplicação Kivy, esse caminho deve ser adicionado no início da lista da variável de ambiente PATH.

Novo na versão 1.9.0.

Aviso: Este caminho é necessário para compilar o Kivy. Porém, não é necessário para a execução do programa.

2.2.2 Configuração

KIVY_USE_DEFAULTCONFIG Se esse nome for encontrado em *environ*, o Kivy não lerá o arquivo de configuração do usuário.

KIVY_NO_CONFIG Se definido, nenhum arquivo de configuração será lido ou gravado. Isso também se aplica ao diretório de configuração do usuário.

KIVY_NO_FILELOG Se definido, os logs não serão impressos em um arquivo

KIVY_NO_CONSOLELOG Se definido, os logs não serão impressos no console

KIVY_NO_ARGS Se definido, o argumento passado pela linha de comando não será analisado e nem utilizado pelo Kivy. Ou seja, você pode com segurança fazer um script ou um aplicativo com seus próprios argumentos sem exigir o – como sendo o delimitador:

```
import os
os.environ["KIVY_NO_ARGS"] = "1"
import kivy
```

Novo na versão 1.9.0.

2.2.3 Restringir o núcleo à implementação específica

kivy.core tenta selecionar a melhor implementação disponível para sua plataforma. Para testes ou instalação personalizada, convém restringir o seletor a uma implementação específica.

KIVY_WINDOW Implementação a ser usada para criar a Janela

Valores: *SDL2, PyGame, X11, EGL_RPI*

KIVY_TEXT Implementação a ser usada para renderizar o texto

Valores: *SDL2, PIL, PyGame, sdlttf*

KIVY_VIDEO Implementação a ser usada para renderizar o vídeo.

Values: *gstplayer, ffpyplayer, ffmpeg, null*

KIVY_AUDIO Implementação a ser usada para reproduzir áudio

Values: *sdl2, gstplayer, ffpyplayer, pygame, avplayer*

KIVY_IMAGE Implementação a ser usada para ler a imagem

Valores: *sdl2, pil, pygame, imageio, tex, dds, gif*

KIVY_CAMERA Implementação a ser usada para a leitura de câmeras

Values: *avfoundation, android, opencv*

KIVY SPELLING Implementação a ser usada para usar na ortografia

Valores: *enchant, osxappkit*

KIVY_CLIPBOARD Implementação a ser utilizada para o gerenciamento da área de transferência

Valores: *sdl2, pygame, dummy, android*

2.2.4 Métricas

KIVY_DPI Se definido, o valor será usado para `Metrics.dpi`.

Novo na versão 1.4.0.

KIVY_METRICS_DENSITY Se definido, o valor será usado para `Metrics.density`.

Novo na versão 1.5.0.

KIVY_METRICS_FONTSCALE

Se definido, o valor será usado para `Metrics.fontscale`.

Novo na versão 1.5.0.

2.2.5 Gráficos

KIVY_GL_BACKEND The OpenGL backend to use. See [cgl](#).

KIVY_GL_DEBUG Whether to log OpenGL calls. See [cgl](#).

KIVY_GRAPHICS Whether to use OpenGL ES2. See [cgl](#).

KIVY_GLES_LIMITS Se as restrições a GLES2 são aplicadas (o padrão, ou se definido como 1). Se definido como *False*, o Kivy não será totalmente compatível com GLES2.

Segue uma lista das potenciais incompatibilidades que geralmente ocorrem quando definido igual a *True*.

Índice da Malha	Se True, o número de índices em um Mesh (malha) será limitado a 65535.
Textura Blit	Quando blitting para uma textura, o formato de dados (cor e buffer) devem ser o mesmo formato que o usado na criação da textura. No Desktop, a conversão das cores diferentes é tratada corretamente pelo driver, enquanto no Android, a maioria dos dispositivos não conseguem fazê-lo. Ref: https://github.com/kivy/kivy/issues/1600

Novo na versão 1.8.1.

KIVY_BCM_DISPMANX_ID Altere o padrão exibido para utilizar o Raspberry Pi .

A lista de valor disponível está acessível em *vc_dispmanx_types.h*. O valor padrão é 0:

- 0: DISPMANX_ID_MAIN_LCD
- 1: DISPMANX_ID_AUX_LCD
- 2: DISPMANX_ID_HDMI
- 3: DISPMANX_ID_SDTV
- 4: DISPMANX_ID_FORCE_LCD
- 5: DISPMANX_ID_FORCE_TV
- 6: DISPMANX_ID_FORCE_OTHER

2.3 Configurar p Kivy

O arquivo de configuração do Kivy é possui o nome igual a *config.ini* e utiliza o formato [padrão dos arquivos *.ini](#).

2.3.1 Localizando o arquivo de configuração

A localização do arquivo de configuração é controlado pela variável de ambiente `KIVY_HOME`:

```
<KIVY_HOME>/config.ini
```

No Desktop, o padrão é:

```
<HOME_DIRECTORY>/.kivy/config.ini
```

Portanto, se o nome do usuário for “tito”, o arquivo estará em:

- Windows: `C:\Users\tito\.kivy\config.ini`
- OS X: `/Users/tito/.kivy/config.ini`
- Linux: `/home/tito/.kivy/config.ini`

No Android, o padrão é:

```
<ANDROID_APP_PATH>/.kivy/config.ini
```

Se o nome do seu aplicativo for “org.kivy.launcher”, o arquivo estará em:

```
/data/data/org.kivy.launcher/files/.kivy/config.ini
```

No iOS, o padrão é:

```
<HOME_DIRECTORY>/Documents/.kivy/config.ini
```

2.3.2 Local configuration

Sometimes it's desired to change configuration only for certain applications or during testing of a separate part of Kivy for example input providers. To create a separate configuration file you can simply use these commands:

```
from kivy.config import Config  
  
Config.read(<file>)  
# set config  
Config.write()
```

When a local configuration of single `.ini` file isn't enough, e.g. when you want to have separate environment for `garden`, kivy logs and other things, you'll need to change the the `KIVY_HOME` environment variable in your application to get desired result:

```
import os  
os.environ['KIVY_HOME'] = <folder>
```

or before each run of the application change it manually in the console:

1. Windows:

```
set KIVY_HOME=<folder>
```

2. Linux & OSX:

```
export KIVY_HOME=<folder>
```

After the change of KIVY_HOME, the folder will behave exactly the same as the default .kivy/ folder mentioned above.

2.3.3 Entendendo os Tokens de configuração

Todos os tokens da configuração são explicados no módulo *kivy.config*.

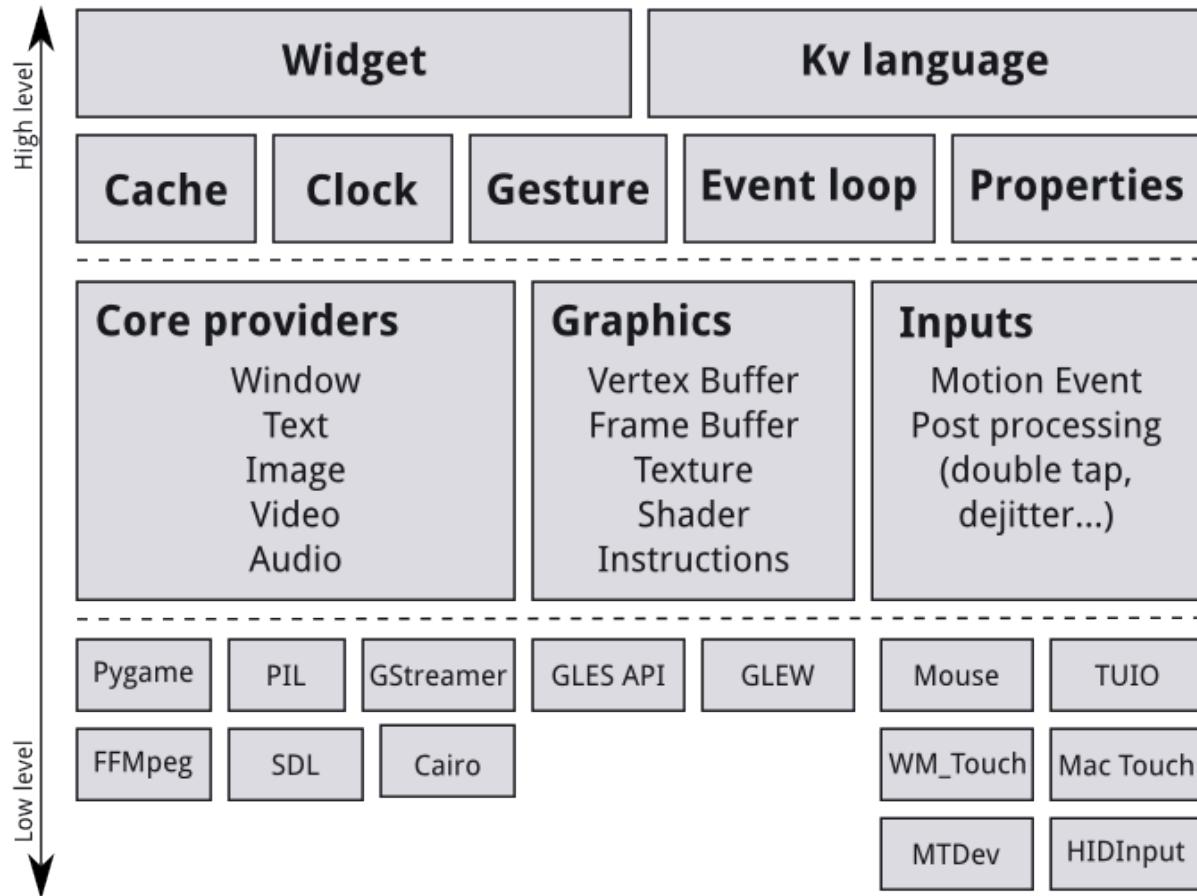
2.4 Visão geral da Arquitetura

Gostaríamos de aproveitar um momento para explicar como criamos o Kivy do ponto de vista da Engenharia de Software. Isso é fundamental para entender como tudo funciona em conjunto. Se apenas olhares para o código, as chances são de que terás uma ideia aproximada, porém, em vista de que essa abordagem é um pouco assustadora para a maioria dos usuários, nesta seção, explicaremos as idéias básicas da implementação em mais detalhes. Pode pular esta seção e voltar a ela mais tarde, porém, sugerimos que pelo menos tenhas uma visão geral do funcionamento!

O Kivy é composto de vários blocos de construção que vamos explicar em breve. Aqui está um resumo gráfico da arquitetura:



Kivy Architecture



2.4.1 Provedores Principais e Provedores de Entrada

Uma das ideias fundamentais para entender os componentes internos do Kivy é a modularidade e a abstração. Tentamos abstrair tarefas básicas como abrir uma janela, exibir imagens e texto, reproduzir áudio, obter imagens de uma câmera, correção ortográfica e assim por diante. Chamamos essas tarefas de *core*. Isso torna a API fácil de usar e fácil de estender. Mais importante ainda, isso nos permite usar - o que chamamos - provedores específicos para os respectivos cenários nos quais a sua aplicação está sendo executada. Por exemplo, em OSX, Linux e Windows, existem diferentes APIs nativas para as diferentes tarefas principais. Um pedaço de código usa uma dessas APIs específicas para conversar com o sistema operacional de um lado e do outro, temos um pedaço de código que conversa com o Kivy (atuando como uma camada de comunicação intermediária) é o que chamamos de um *fornecedor central*. A vantagem de usar provedores de núcleo especializados para cada plataforma é que podemos aproveitar totalmente a funcionalidade disponibilizada pelo sistema operacional e atuar da forma mais eficiente possível. Isso também oferece aos usuários uma escolha. Além disso, usando bibliotecas que são enviadas com qualquer plataforma,

reduzimos efetivamente o tamanho da distribuição Kivy e facilitamos o empacotamento. Isso também facilita portar o Kivy para outras plataformas. A port feito para o Android se beneficiou muito com isso.

Seguimos o mesmo conceito com manipulação dos dados de entrada. *Um provedor de entrada* é um pedaço de código que adiciona suporte para um dispositivo de entrada específico, como um Trackpads da Apple, TUIO ou um emulador de mouse. Se precisares adicionar suporte para algum dispositivo de entrada novo, basta fornecer uma nova classe que lê os dados de entrada fornecido pelo seu dispositivo e transforma-los em eventos básicos do Kivy.

2.4.2 Gráficos

A API gráfica do Kivy é nossa abstração do OpenGL. No nível mais baixo, o Kivy emite comandos de desenho com aceleração de hardware usando o OpenGL. Escrever código OpenGL, no entanto, pode ser um pouco confuso, especialmente para os recém-chegados. É por isso que fornecemos a API de gráficos que permite desenhar coisas usando simples metáforas que não existem de fato em OpenGL (por exemplo, Canvas, Rectangle, etc.).

Todos os nossos Widgets utilizam essa API gráfica, que está implementada a nível de código C por motivos de desempenho.

Outra vantagem da API gráfica é a capacidade de otimizar automaticamente os comandos de desenho que seu código emite. Isto é especialmente útil se não és um especialista em tuning OpenGL. Isso torna seu código de desenho muito mais eficiente para a maiores dos casos.

Você pode, naturalmente, ainda utilizar comandos do OpenGL diretamente (crus), caso desejes. A versão que utilizamos é OpenGL 2.0 ES (GLES2) para todos os dispositivos, por isso, se pretender manter compatibilidade entre plataformas, recomendamos que utilize apenas as funções GLES2.

2.4.3 Core

O código do pacote principal fornece recursos comumente usados, por exemplo:

Clock Você pode usar a classe Clock para agendar eventos de tempo. São suportados tanto temporizadores que disparam um única vez como também, temporizadores periódicos.

Cache Se você precisa armazenar em cache algo que será usado com freqüência, podes usar nossa classe ao invés de escrever a sua própria.

Detecção de Gestos Implementamos um simples reconhecedor de gesto que poderás usar para detectar vários tipos de traços, como também círculos ou retângulos. Poderás também treiná-lo para detectar suas próprias formas de desenho.

Linguagem Kivy A linguagem Kivy é usada para escrever interfaces de usuário de forma fácil e eficiente.

Propriedades Estas não são as propriedades normais que já utilizas em seus códigos Python. Essas são nossas próprias classes de propriedade que vinculam o código do seu Widget com a descrição da interface de usuário.

2.4.4 UIX (Widgets & Layouts)

O módulo UIX contém Widgets e Layouts comumente usados que poderás reutilizar para criar rapidamente uma interface de usuário.

Widgets Widgets são elementos da interface de usuário que podes adicionar ao seu programa para fornecer algum tipo de funcionalidade. Um Widget pode ou não estar visíveis. Exemplo de Widget seria um navegador de arquivos, botões, controles deslizantes, listas e assim por diante. Widgets recebem MotionEvents.

Layouts Você utiliza Layouts para organizar os Widgets. É claro que é possível calcular as posições dos seus Widgets em pixel, por exemplo, mas na maior parte das vezes, será mais conveniente usar um dos nossos Layouts que já estão prontos. Exemplos seriam o GridLayouts ou então o BoxLayouts. Você também pode adicionar Layouts dentro de outros Layouts e assim, construir estruturas mais complexas.

2.4.5 Módulos

Se já utilizastes um navegador Web moderno e personalizastes-o com alguns Add-ons, então já conheces a ideia básica por trás das nossas classes de módulo. Os módulos podem ser usados para injetar funcionalidade em programas Kivy, mesmo se o autor original não os tenha incluído.

Um exemplo seria o módulo que sempre mostra o FPS do aplicativo atual e algum gráfico representando o FPS ao longo do tempo.

Você também pode escrever seus próprios módulos.

2.4.6 Eventos de Entrada (Toques)

O Kivy abstrai diferentes tipos de entrada e fontes como exemplos, a entrada de toques, mouse, TUIO e outras entradas similares. O que todos esses tipos de entrada têm em comum é que podemos associa-las a uma posição 2D na tela com qualquer evento de entrada individual. (Há outros dispositivos de entrada, como acelerômetros, onde não podemos facilmente encontrar uma posição 2D, por exemplo, uma inclinação do

seu dispositivo. Este tipo de entrada é tratada separadamente. Em seguida, descrevemos os tipos anteriores)

Todos esses tipos de entrada são representados por instâncias da classe `Touch()`. (Note que isso não se refere apenas a toques de dedo, mas também, a todos os outros tipos de entrada. Nós apenas chamamos de *Touch* por uma questão de simplicidade. Pense nisso de algo que *toques* na interface do usuário ou na tela.) Uma instância de toque, ou objeto, pode estar em um dos três estados. Quando um toque entra em um desses estados, o programa é informado de que o evento ocorreu. Os três estados em que um toque pode estar são:

Baixo Um toque pra baixo apenas uma vez, no momento em que aparece pela primeira vez.

Movimento Um toque pode estar nesse estado por um tempo potencialmente ilimitado. Um toque não precisa estar nesse estado durante sua vida útil. Um ‘Move’ acontece sempre que a posição 2D de um toque sofrer alteração (mudar).

Cima Um toque sobe no máximo uma vez, ou então nenhuma. Na prática, você quase sempre receberá um evento porque ninguém vai segurar o dedo na tela por toda a eternidade, porém, não existe garantia de que o usuário irá soltar o dedo da tela. Se souberes as fontes de entrada que os seus usuários estarão usando, saberás se podes ou não confiar nesse estado que está recebendo dados de entrada.

2.4.7 Widgets e Despachadores de Eventos

O termo *Widget* é frequentemente usado em contextos de programação GUI para descrever parte do programa com o qual o usuário interage. Em Kivy, um Widget é um objeto que recebe eventos de entrada. Não necessariamente precisa ter uma representação visual na tela. Todos os Widgets são organizados em uma *árvore de Widgets* (que é uma estrutura de dados em forma de árvore, nomenclatura estudada nas aulas de Ciência da Computação): Um Widget pode ter qualquer número de Widgets filhos ou nenhum. Existe exatamente um único *Widget raiz*, ou *Widget principal* que estará no topo da árvore e que não terá qualquer Widget pai, e todos os outros Widgets serão direta ou indiretamente filhos deste Widget principal (razão pela qual ele é chamado de *raiz*).

Quando novos dados de entrada estão disponíveis, o Kivy envia um evento por toque. O Widget raiz da árvore de Widgets será o primeiro a receber o evento. Dependendo do estado do toque, o evento `on_touch_down`, `on_touch_move` ou `on_touch_up` será despachado (com um toque como o argumento) para o Widget raiz, o que resulta no correspondente `on_touch_down`, `on_touch_move` ou `on_touch_up` do gerenciador de eventos do Widget raiz que está sendo chamado.

Cada Widget (incluindo o Widget raiz) na árvore pode escolher entre “consumir” ou re-transmitir o evento. Se um manipulador de eventos retornar `True`, isso significa

que o evento foi “consumido” e tratado adequadamente. Neste caso, nenhum processamento posterior acontecerá com esse evento. Caso contrário, o manipulador de eventos passa o Widget para seus próprios filhos, chamando a implementação da superclasse do respectivo manipulador de eventos. Isto faz todo o caminho até a classe Widget base, que - em seus manipuladores de evento de toque - não faz nada, mas passa os eventos de toques para seus filhos:

```
# This is analogous for move/up:  
def on_touch_down(self, touch):  
    for child in self.children[:]:  
        if child.dispatch('on_touch_down', touch):  
            return True
```

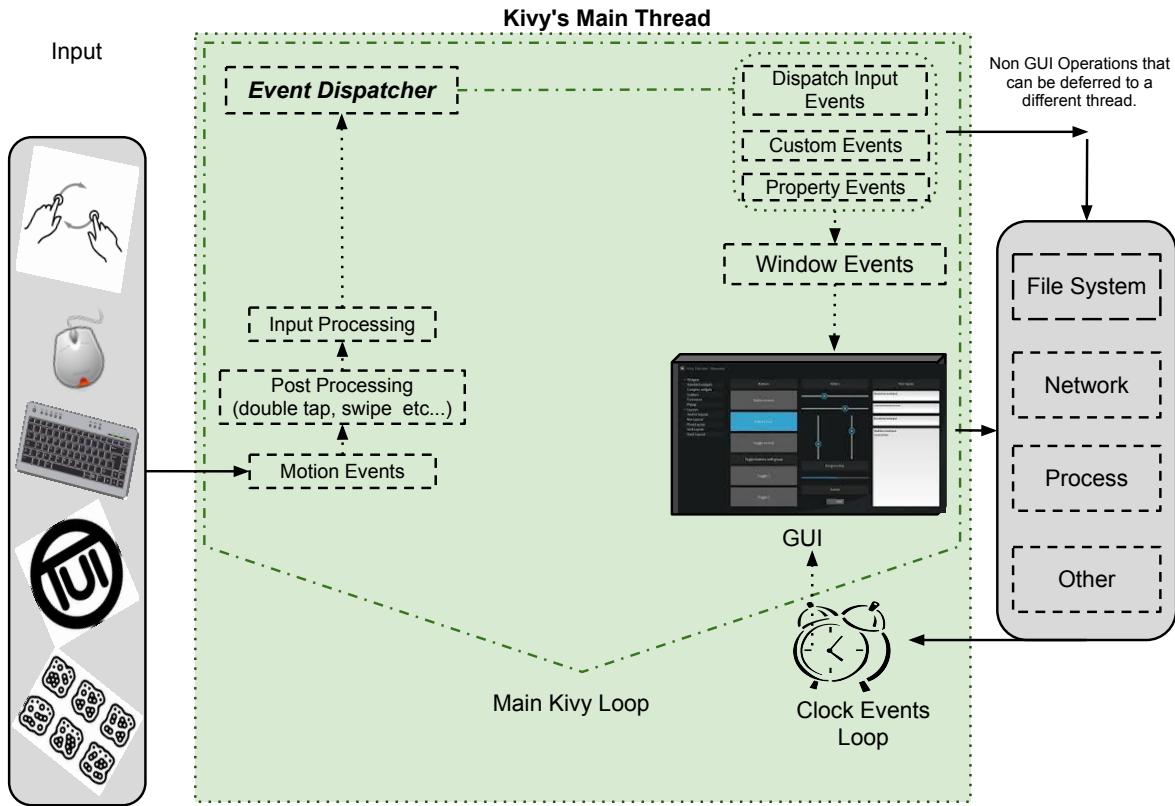
Isso é realmente muito mais simples do que parece. Um bom exemplo de como isso pode ser usado para criar bons aplicativos rapidamente será apresentado na próxima seção.

Muitas vezes irás querer restringir a *área* na tela que um Widget de relógios para toques. Podes usar o método *collide_point()* de um Widget para conseguir isso. Você simplesmente passa a posição do toque e será retornado *True* se o toque estiver dentro da ‘área de observação’ ou *False* caso contrário. Por padrão, isso verifica a região retangular na tela que é descrita pela posição (posição e *x* & *y*) do tamanho do Widget (largura e altura), mas poderás substituir esse comportamento em sua própria classe.

2.5 Eventos e Propriedades

Os eventos são uma parte importante da programação Kivy. Isso não deve causar surpresa àqueles com experiência em desenvolvimento de aplicações que utilizam GUI, porém, é um conceito importante para os recém chegados. Depois de entender como os eventos funcionam e como vincular funções ao mesmos, passarás a vê-los em todos os lugares da biblioteca Kivy. Estes facilitam a construção de qualquer comportamento que quiseres implementar com a biblioteca Kivy.

A ilustração a seguir mostra como os eventos são tratados pelo framework Kivy.



2.5.1 Introdução para o Despachador de Eventos

Uma das classes de base mais importantes do framework é a classe ***EventDispatcher***. Essa classe permite que você registre tipos de evento e despache-os para partes interessadas (geralmente outros distribuidores de eventos). As classes ***Widget***, ***Animation*** e ***Clock*** são exemplos de despachadores de eventos.

Os objetos ***EventDispatcher*** dependem do “main loop” (loop principal) para gerar e manipular eventos.

2.5.2 Main Loop

Conforme descrito na ilustração acima, o Kivy tem um *main loop* (loop principal). Este loop está em execução durante toda a vida útil do aplicativo e só é finalizado quando saímos do aplicativo.

Dentro do loop, e em cada iteração, os eventos são gerados a partir das entradas do usuário, através dos sensores de hardware ou através de qualquer outra fonte de entrada, e é onde os Frames são renderizados para que o display possa exibi-los no vídeo.

A sua aplicação especificará callbacks (mais sobre esse tema na sequência), que serão invocados pelo *main loop*. Se um callback demorar muito ou não finalizar, o *main loop* irá parar de funcionar e a sua aplicação não funcionará corretamente.

Em aplicações Kivy, você tem que evitar loops (laços de repetição) longos/infinitos ou pior ainda, fazer a sua aplicação dormir por um período de tempo. Por exemplo, veja o código a seguir:

```
while True:  
    animate_something()  
    time.sleep(.10)
```

Quando executares isso, o programa nunca sairá do seu loop, impedindo o Kivy de fazer todas as outras coisas que precisam ser feitas. Como resultado, tudo o que irás ver é uma janela preta e não serás capaz de interagir com a mesma. Em vez disso, precisas “agendar” a sua função `animate_something()` para que seja chamada repetidamente.

Agendando um Evento Repetitivo

Podes chamar uma função ou um método a cada X vezes por segundo usando `schedule_interval()`. Aqui temos um exemplo da invocação de uma função cujo nome é `my_callback()` e que está sendo invocada 30 vezes por segundo:

```
def my_callback(dt):  
    print 'My callback is called', dt  
event = Clock.schedule_interval(my_callback, 1 / 30.)
```

Tens várias maneiras de cancelar um evento agendado anteriormente. Uma dessas formas é usando o método `cancel()` ou `unschedule()`:

```
event.cancel()
```

ou:

```
Clock.unschedule(event)
```

Como alternativa, podes retornar `False` para o seu callback, e então, o evento deixará de ser propagado automaticamente:

```
count = 0  
def my_callback(dt):  
    global count  
    count += 1  
    if count == 10:  
        print 'Last call of my callback, bye bye !'  
        return False  
    print 'My callback is called'  
Clock.schedule_interval(my_callback, 1 / 30.)
```

Agendamento de um Evento Unico

Usando `schedule_once()`, podes invocar uma função “mais tarde”, como no próximo Frame, ou daqui a X segundos:

```
def my_callback(dt):
    print 'My callback is called !'
Clock.schedule_once(my_callback, 1)
```

Isso chamará o `my_callback` em 1 segundo. O segundo argumento é a quantidade de tempo de espera antes de chamar a função, em segundos. No entanto, podes obter alguns outros resultados com valores especiais para o segundo argumento:

- Se X for maior que 0, o callback será chamado em X segundos
- Se X for 0, o callback será chamado após o próximo Frame
- Se X for -1, o callback será chamado antes do próximo Frame

O -1 é usado principalmente quando já estás em um evento agendado e caso queiras agendar uma chamada ANTES do próximo Frame que está acontecendo.

Um segundo método para repetir a invocação de uma função é inicialmente agendar a invocação de um callback uma vez com `schedule_once()` e uma segunda invocação a esta função dentro do próprio callback:

```
def my_callback(dt):
    print 'My callback is called !'
    Clock.schedule_once(my_callback, 1)
Clock.schedule_once(my_callback, 1)
```

Enquanto o *main loop* tentar manter o agendamento conforme solicitado, há alguma incerteza quanto a quando exatamente um callback programado será invocado. Às vezes, outro callback ou alguma outra tarefa no aplicativo demorará mais do que o previsto e, portanto, o tempo pode ficar um pouco fora.

Na última solução para o problema callbacks repetitivos, a próxima iteração será chamada pelo menos um segundo após a última iteração ter terminado. Com `schedule_interval()` no entanto, o callback é chamado a cada segundo.

Trigger Events

Às vezes, podes querer agendar uma função para ser invocada apenas uma vez no próximo Frame, impedindo assim chamadas duplicadas. Pode ser tentado a conseguir fazer isso dessas forma:

```
# First, schedule once.
event = Clock.schedule_once(my_callback, 0)

# Then, in another place you will have to unschedule first
```

```
# to avoid duplicate call. Then you can schedule again.
Clock.unschedule(event)
event = Clock.schedule_once(my_callback, 0)
```

Esta maneira de programar um gatilho (trigger) é bastante custosa, desde que sempre chamarás o desagendamento, mesmo que o evento já tenha terminado. Além disso, um novo evento será criado cada vez. Use um trigger em vez disso:

```
trigger = Clock.create_trigger(my_callback)
# later
trigger()
```

Cada vez que invocares `trigger()`, o mesmo agendará um único callback. Caso já tenha sido agendado, o mesmo não será agendado novamente

2.5.3 Eventos de Widgets

Por padrão, um Widget tem 2 tipos de eventos:

- Evento de propriedade: se um Widget tem alguma de suas propriedades alteradas, tais como, a sua posição ou tamanho, um evento é disparado.
- Evento definido pelo Widget: Um evento será disparado por um botão quando o mesmo for pressionado ou quando o botão for liberado.

Para uma discussão sobre como os eventos de toque dos Widget são gerenciados e propagados, veja a documentação na seção [Widget touch event bubbling](#) section.

2.5.4 Criando um Evento Personalizado

Para criar um dispatcher de eventos com eventos personalizados, será necessário registrar o nome do evento na classe e, em seguida, criar um método com o mesmo nome.

Veja o seguinte exemplo:

```
class MyEventDispatcher(EventDispatcher):
    def __init__(self, **kwargs):
        self.register_event_type('on_test')
        super(MyEventDispatcher, self).__init__(**kwargs)

    def do_something(self, value):
        # when do_something is called, the 'on_test' event will be
        # dispatched with the value
        self.dispatch('on_test', value)
```

```
def on_test(self, *args):
    print "I am dispatched", args
```

2.5.5 Anexando callbacks

Para usar eventos, terás que vincular um callbacks a eles. Quando o evento for despatchado, seus callbacks serão invocados com os parâmetros relevantes para esse evento específico.

Um callback pode ser qualquer callable Python, mas precisarás garantir que o mesmo aceite os argumentos que o evento emite. Para isso, geralmente é mais seguro trabalhar com os argumento `*args`, que capturará todos os argumentos na lista 'args'.

Exemplo:

```
def my_callback(value, *args):
    print "Hello, I got an event!", args

ev = MyEventDispatcher()
ev.bind(on_test=my_callback)
ev.do_something('test')
```

Consulte a documentação do método `kivy.event.EventDispatcher.bind()` para obter mais exemplos de como vincular callbacks.

2.5.6 Introdução às Propriedades

As propriedades são uma ótima maneira de definir eventos e vincular funções ao mesmos. Essencialmente, eles produzem eventos tais que quando um atributo de seu objeto sofrer alguma alteração, todas as propriedades que fazem referência a esse atributo são automaticamente atualizadas.

Existem diferentes tipos de propriedades que descrevem o tipo de dados que você deseja manipular.

- `StringProperty`
- `NumericProperty`
- `BoundedNumericProperty`
- `ObjectProperty`
- `DictProperty`
- `ListProperty`
- `OptionProperty`

- *AliasProperty*
- *BooleanProperty*
- *ReferenceListProperty*

2.5.7 Declaração de uma Propriedade

Para declarar propriedades, precisarás declará-las no nível de classe. A classe fará o trabalho para instanciar os atributos reais quando o objeto for criado. Estas propriedades não são somente atributos: são mecanismos para criar eventos com base nos seus atributos:

```
class MyWidget(Widget):
    text = StringProperty('')
```

Ao substituir `__init__`, sempre aceite `**kwargs` e utilize `super()` para invocar o método `__init__` da superclasse, passando a instância da classe atual:

```
def __init__(self, **kwargs):
    super(MyWidget, self).__init__(**kwargs)
```

2.5.8 Despachando um Evento de Propriedade

As propriedades Kivy, por padrão, fornecem um evento `on_<property_name>`. Esse evento é chamado quando o valor da propriedade é alterado.

Nota: Caso o novo valor da propriedade seja igual ao valor atual, o evento `on_<property_name>` não será chamado.

Por exemplo, considere o seguinte código:

```
1 class CustomBtn(Widget):
2
3     pressed = ListProperty([0, 0])
4
5     def on_touch_down(self, touch):
6         if self.collide_point(*touch.pos):
7             self.pressed = touch.pos
8             return True
9         return super(CustomBtn, self).on_touch_down(touch)
10
11    def on_pressed(self, instance, pos):
12        print ('pressed at {pos}'.format(pos=pos))
```

No código acima na linha 3:

```
pressed = ListProperty([0, 0])
```

Definimos a propriedade *pressed* do tipo *ListProperty*, dando-lhe um valor padrão igual a [0, 0]. A partir deste ponto, o evento *on_pressed* será invocado sempre que o valor desta propriedade for alterado.

Na linha 5:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):
        self.pressed = touch.pos
        return True
    return super(CustomBtn, self).on_touch_down(touch)
```

Substituímos o método *on_touch_down()* da classe Widget. Aqui, verificamos a colisão do *touch* com o nosso Widget.

Se o toque cair dentro da área do nosso Widget, alteramos o valor de *pressed* para *touch.pos* e retornamos True, indicando que consumimos o toque e não queremos que o mesmo continue se propagando.

Finalmente, se o toque cair fora da área do nosso Widget, chamamos o evento original usando *super (...)* e retornamos o resultado. Isso permite que a propagação de eventos de toque continue como normalmente ocorreria.

Finalmente na linha 11:

```
def on_pressed(self, instance, pos):
    print ('pressed at {}'.format(pos))
```

Definimos uma função *on_pressed* que será invocada pela propriedade sempre que o valor dessa propriedade for alterado.

Nota: Este evento *on_<prop_name>* é chamado dentro da classe onde a propriedade está definida. Para monitorar/observar qualquer alteração numa propriedade fora da classe onde a mesma está definida, precisarás vincular a propriedade como mostrado abaixo.

Vinculando a uma Propriedade

Como monitorar mudanças numa propriedade quando tudo o que tens acesso é uma instância de widget? Você *vincula* um callback a essa propriedade:

```
your_widget_instance.bind(property_name=function_name)
```

Por exemplo, considere o seguinte código:

```

1  class RootWidget(BoxLayout):
2
3      def __init__(self, **kwargs):
4          super(RootWidget, self).__init__(**kwargs)
5          self.add_widget(Button(text='btn 1'))
6          cb = CustomBtn()
7          cb.bind(pressed=self.btn_pressed)
8          self.add_widget(cb)
9          self.add_widget(Button(text='btn 2'))
10
11     def btn_pressed(self, instance, pos):
12         print ('pos: printed from root widget: {pos}'.format(pos=.
13             pos))

```

Se executares o código como está, notarás duas instruções de impressão no console. Uma do evento *on_pressed* que é chamado dentro da classe *CustomBtn* e outro da função *btn_pressed* que vinculamos junto a alteração de propriedade.

A razão pela qual ambas as funções são invocadas é simples. Vincular não significa substituir. Ter ambas as funções é redundante e geralmente precisarás usar apenas um dos métodos de ouvir/reagir às alterações de propriedade.

Você também deve observar os parâmetros que são passados para o evento *on <property name>* ou a função vinculada a essa propriedade.

```
def btn_pressed(self, instance, pos):
```

O primeiro parâmetro será *self*, que é a instância da classe onde esta função está definida. Pode usar uma função in-line da seguinte maneira:

```

1  cb = CustomBtn()
2
3  def _local_func(instance, pos):
4      print ('pos: printed from root widget: {pos}'.format(pos=pos))
5
6  cb.bind(pressed=_local_func)
7  self.add_widget(cb)

```

O primeiro parâmetro seria a *instance* da classe onde a propriedade é definida.

O segundo parâmetro seria o *valor*, que será o novo valor da propriedade.

Aqui temos um exemplo completo, derivado dos Snippets acima, que poderás usar para copiar e colar em um editor para então experimentar.

```

1  from kivy.app import App
2  from kivy.uix.widget import Widget
3  from kivy.uix.button import Button
4  from kivy.uix.boxlayout import BoxLayout
5  from kivy.properties import ListProperty

```

```

6
7     class RootWidget(BoxLayout):
8
9         def __init__(self, **kwargs):
10            super(RootWidget, self).__init__(**kwargs)
11            self.add_widget(Button(text='btn 1'))
12            cb = CustomBtn()
13            cb.bind(pressed=self.btn_pressed)
14            self.add_widget(cb)
15            self.add_widget(Button(text='btn 2'))
16
17        def btn_pressed(self, instance, pos):
18            print ('pos: printed from root widget: {pos}'.
19                  format(pos=pos))
20
21    class CustomBtn(Widget):
22
23        pressed = ListProperty([0, 0])
24
25        def on_touch_down(self, touch):
26            if self.collide_point(*touch.pos):
27                self.pressed = touch.pos
28                # we consumed the touch. return False here to propagate
29                # the touch further to the children.
30                return True
31            return super(CustomBtn, self).on_touch_down(touch)
32
33        def on_pressed(self, instance, pos):
34            print ('pressed at {}'.format(pos))
35
36    class TestApp(App):
37
38        def build(self):
39            return RootWidget()
40
41    if __name__ == '__main__':
42        TestApp().run()

```

Executando o código acima lhe dará a seguinte saída:



Nosso CustomBtn não possui qualquer representação visual e, portanto, aparece preto. Podes tocar/clicar na área preta para ver a saída no seu console.

2.5.9 Propriedades Compostas

Ao definir uma **AliasProperty**, normalmente definirás um getter e uma função setter você mesmo. Aqui, competirá a você definir quando as funções getter e setter são invocadas usando o argumento *bind*.

Considere o seguinte código.

```
1 cursor_pos = AliasProperty(_get_cursor_pos, None, bind=
2     'cursor', 'padding', 'pos', 'size', 'focus',
3     'scroll_x', 'scroll_y'))
4 '''Current position of the cursor, in (x, y).
5
6 :attr:`cursor_pos` is a :class:`~kivy.properties.AliasProperty`, ↴
7     read-only.
'''
```

Aqui *cursor_pos* é uma **AliasProperty** e que utiliza um getter *get_cursor_pos* com a parte *setter* definida como sendo igual a *None*, implicando que esta será uma propriedade apenas de leitura.

O argumento *bind* no final define que o evento *on_cursor_pos* será despachado quando qualquer uma das propriedades usadas no *bind* = alterar o argumento.

2.6 Gerenciador de Entrada

2.6.1 Arquitetura de entrada

O Kivy é capaz de lidar com a maioria dos tipos de entrada: mouse, touchscreen, acelerômetro, giroscópio, etc. Ele lida com os protocolos nativos multitouch das seguintes plataformas: TUIO, WM_Touch, MacMultitouchSupport, MT Protocolo A/B e Android.

A arquitetura global pode ser vista como:

```
Input providers -> Motion event -> Post processing -> Dispatch to  
    ↴Window
```

A classe de todos os eventos de entrada é *MotionEvent*. Gera 2 tipos de eventos:

- Eventos de toque: um evento de movimento que contém pelo menos uma posição X e Y. Todos os eventos de toque são enviados através da árvore de Widget.
- Eventos sem toque: todo o restante. Por exemplo, o acelerômetro é um evento contínuo, sem posição. Nunca começa ou pára. Esses eventos não são enviados através da árvore de Widget.

Um evento *Motion* é gerado por *Input Provider*. Um provedor de entrada é responsável pela leitura do evento de entrada do sistema operacional, da rede ou mesmo de outro aplicativo. Existem vários fornecedores de entrada, tais como:

- *TuioMotionEventProvider*: criar um servidor UDP e escuta as mensagens TUIO/OSC.
- *WM_MotionEventProvider*: usa a API do Windows para ler informações MultiTouch e envia-as para o Kivy.
- *ProbeSysfsHardwareProbe*: No Linux, é iterado sobre todo o hardware conectado ao computador e é anexado um provedor de entrada multitouch para cada dispositivo multitouch encontrado.
- e muito mais!

Quando escreves um aplicativo, não é necessário criar um provedor de entrada. O Kivy tenta detectar automaticamente o hardware disponível. No entanto, se pretenderes suportar algum hardware personalizado, terás que configurar o kivy para o faze-lo se comunicar corretamente com o hardware especial.

Antes do recém-criado Evento de Movimento ser passado para o usuário, o Kivy aplica um pós-processamento à entrada. Cada evento de movimento é analisado para detectar e corrigir entradas defeituosas, bem como fazer interpretações significativas como:

- Detecção de toque duplo/triplo, de acordo com um limite de distância e tempo.
- Tornando os eventos mais precisos quando o hardware não é preciso

- Reduz a quantidade de eventos gerados do hardware de toque nativo que está enviando eventos com quase a mesma posição

Após o processamento, o evento de movimento é enviado para a janela. Conforme explicado anteriormente, nem todos os eventos são enviados para toda a árvore de Widgets: a janela filtra-os. Para um determinado evento:

- Se for apenas um evento de movimento, ele será despachado para: meth: ~
kivy.core.window.WindowBase.on_motion
- Se for um evento de toque, a posição (x, y) do toque a (escala 0-1) será dimensionada para o tamanho da janela (largura/altura) e enviada para:
 - *on_touch_down()*
 - *on_touch_move()*
 - *on_touch_up()*

2.6.2 Perfis de Eventos de Movimento

Dependendo do seu hardware e dos provedores de entrada usados, mais informações podem ser disponibilizadas a você. Por exemplo, uma entrada de toque tem uma posição (x, y), mas também pode ter informações de pressão, tamanho do blob, um vetor de aceleração, etc.

Um profile (perfil) é uma sequência de caracteres que indica quais recursos estão disponíveis dentro do evento de movimento. Vamos imaginar que você está num método *on_touch_move*:

```
def on_touch_move(self, touch):
    print(touch.profile)
    return super(..., self).on_touch_move(touch)
```

Poderia ser impresso:

```
[ 'pos' , 'angle' ]
```

Aviso: Muitas pessoas misturam o nome do perfil e o nome da propriedade correspondente. Só porque '*angle*' está disponível no perfil não significa que o objeto de evento *touch* terá uma propriedade *angle*.

Para o perfil '*pos*', as propriedades *pos*, *x* e *y* estarão disponíveis. Com o perfil '*angle*', a propriedade *a* estará disponível. Como dissemos, para os eventos de toque '*pos*' é um perfil obrigatório, mas não '*angle*'. Pode estender sua interação verificando se o perfil '*angle*' existe:

```
def on_touch_move(self, touch):
    print('The touch is at position', touch.pos)
    if 'angle' in touch.profile:
        print('The touch angle is', touch.a)
```

Você pode encontrar uma lista de perfis disponíveis na documentação [motionevent](#).

2.6.3 Eventos de Toque

Um evento de toque é um especializado em [MotionEvent](#) onde a propriedade `is_touch` é avaliada como `True`. Para todos os eventos de toque, as posições X e Y estarão automaticamente disponíveis, dimensionadas para a largura e altura da Janela. Em outras palavras, todos os eventos de toque têm no profile '`pos`'.

Eventos Básicos de Toque

Por padrão, os eventos de toque são despachados para todos os Widgets atualmente exibidos. Isso significa que os Widgets recebem o evento de toque se estes ocorrem dentro ou fora da sua área física.

Isso pode ser contra intuitivo se tiveres experiência com outros kits de ferramentas GUI. Estes tipicamente dividem a tela em áreas geométricas e somente enviam eventos de toque ou mouse para o Widget se a coordenada estiver dentro da área de determinado Widget.

Este requisito torna-se muito restritivo quando se trabalha com entrada de toque. Swipes, Pinches e longos toques de tela podem muito bem originarem-se fora do Widget que quer saber sobre eles e ainda assim reagir a esses eventos.

A fim de fornecer a máxima flexibilidade, o Kivy despacha os eventos para todos os Widgets e permite que eles decidam como reagir a cada evento. Se só desejas responder a eventos de toque dentro do Widget, veja um exemplo de como fazê-lo:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):
        # The touch has occurred inside the widgets area. Do stuff!
        pass
```

Coordenadas

Deves cuidar da transformação matricial em seu toque ao utilizar um Widget que tenha transformação matricial. Alguns Widgets como os [Scatter](#) possuem a sua própria transformação de matriz, o que significa que o toque deve ser multiplicado pela matriz de dispersão para poder enviar corretamente as posições de toque para os filhos do Scatter.

- Obtém a coordenada do espaço do seu pai para o espaço local: `to_local()`
- Obtém a coordenada do espaço local para o espaço do seu parent (pai): `to_parent()`
- Obtém a coordenada de espaço local para o espaço da janela: `to_window()`
- Obtém a coordenada do espaço da janela para o espaço local: `to_widget()`

Deves usar um deles para dimensionar as coordenadas corretamente para o contexto. Vejamos a implementação do Widget Scatter:

```
def on_touch_down(self, touch):
    # push the current coordinate, to be able to restore it later
    touch.push()

    # transform the touch coordinate to local space
    touch.apply_transform_2d(self.to_local)

    # dispatch the touch as usual to children
    # the coordinate in the touch is now in local space
    ret = super(..., self).on_touch_down(touch)

    # whatever the result, don't forget to pop your transformation
    # after the call, so the coordinate will be back in parent space
    touch.pop()

    # return the result (depending what you want.)
    return ret
```

Toque em Formas

Se o toque tiver uma forma, o mesmo será refletido na propriedade `shape`. Agora, apenas uma :class:`~kivy.input.shape.ShapeRect` poderá ser exposta:

```
from kivy.input.shape import ShapeRect

def on_touch_move(self, touch):
    if isinstance(touch.shape, ShapeRect):
        print('My touch have a rectangle shape of size',
              (touch.shape.width, touch.shape.height))
    # ...
```

Duplo Toque

Um toque duplo é a ação de tocar duas vezes num intervalo de tempo numa distância bastante próxima. Isso é calculado pelo módulo de pós-processamento DoubleTap. Pode testar se o toque atual é um toque duplo ou não da seguinte maneira:

```
def on_touch_down(self, touch):
    if touch.is_double_tap:
        print('Touch is a double tap !')
        print(' - interval is', touch.double_tap_time)
        print(' - distance between previous is', touch.double_tap_
distance)
    # ...
```

Toque Triplo

Um toque triplo é a ação de tocar três vezes dentro num curto intervalo de tempo num distância bastante próxima. Isso é calculado pelo módulo de pós-processamento TripleTap. Pode testar se o toque atual é um toque triplo ou não, veja o exemplo a seguir:

```
def on_touch_down(self, touch):
    if touch.is_triple_tap:
        print('Touch is a triple tap !')
        print(' - interval is', touch.triple_tap_time)
        print(' - distance between previous is', touch.triple_tap_
distance)
    # ...
```

Pegando Eventos de Toques

É possível que o Widget pai envie um evento de toque para um Widget filho de dentro de `on_touch_down`, mas não de `on_touch_move` ou `on_touch_up`. Isso poderá acontecer em certos cenários, como quando um movimento de toque está fora da caixa delimitadora do pai, de modo que o pai decide por não notificar os seus filhos sobre esse movimento.

Mas podes querer fazer algo em `on_touch_up`. Digamos que iniciastes algo no evento `on_touch_down`, como reproduzir um som e gostaria de terminar tudo no evento `on_touch_up`. Grabbing é o que você precisa.

Quando “pegas” um toque, sempre receberás o movimento e até o evento. Mas existem algumas limitações para “agarrar”:

- Receberás o evento pelo menos duas vezes: uma vez do seu pai (o evento normal) e uma vez da janela (grab).
- Pode receber um evento com um toque grabbed “agarrado”, mas não de você: pode ser porque o pai enviou o toque para seus filhos enquanto ele estava no estado “grabbed”.
- A coordenada de toque não é traduzida para o espaço do Widget porque o toque está vindo diretamente da janela. É o seu trabalho converter a coordenada para

o seu espaço local do seu Widget.

Aqui está um exemplo de como usar o grabbing:

```
def on_touch_down(self, touch):
    if self.collide_point(*touch.pos):

        # if the touch collides with our widget, let's grab it
        touch.grab(self)

        # and accept the touch.
        return True

def on_touch_up(self, touch):
    # here, you don't check if the touch collides or things like_
    # that.
    # you just need to check if it's a grabbed touch event
    if touch.grab_current is self:

        # ok, the current touch is dispatched for us.
        # do something interesting here
        print('Hello world!')

        # don't forget to ungrab ourself, or you might have side_
        # effects
        touch.ungrab(self)

        # and accept the last up
        return True
```

Gerenciamento de Eventos de Toque

Para ver como os eventos de toque são controlados e propagados entre os Widgets, consulte a seção [Widget touch event bubbling](#).

2.6.4 Joystick events

A joystick input represents raw values received directly from physical or virtual controllers through the SDL2 provider via these events:

- `SDL_JOYAXISMOTION`
- `SDL_JOYHATMOTION`
- `SDL_JOYBALLMOTION`
- `SDL_JOYBUTTONDOWN`
- `SDL_JOYBUTTONUP`

Every motion event has a minimum, maximum and default value which can reach:

Event	Minimum	Maximum	Default
on_joy_axis	-32767	32767	0
on_joy_hat	(-1, -1)	(1, 1)	(0, 0)
on_joy_ball	Unknown	Unknown	Unknown

Button events, on the other hand represent basically only a state of each button i.e. *up* and *down*, therefore no such values are present.

- on_joy_button_up
- on_joy_button_down

Joystick event basics

Unlike touch events, joystick events are dispatched directly to the Window, which means there's only a single value passed for e.g. a specified axis, not multiple ones. This makes things harder if you want to separate input to different widgets, yet not impossible. You can use [Multiple dropfile example](#) as an inspiration.

To get a joystick event, you first need to bind some function to the Window joystick event like this:

```
Window.bind(on_joy_axis=self.on_joy_axis)
```

Then you need to fetch the parameters specified in `Window` for each event you use, for example:

```
def on_joy_axis(self, win, stickid, axisid, value):
    print(win, stickid, axisid, value)
```

A variable *stickid* is an id of a controller that sent the value, *axisid* is an id of an axis to which the value belongs.

Joystick input

Kivy should be able to fetch input from any device specified as *gamepad*, *joystick* or basically any other type of game controller recognized by the SDL2 provider. To make things easier, here are layouts of some common controllers together with ids for each part.

Xbox 360



#	ID	#	ID
1	axis 1	2	axis 0
3	hat Y	4	hat X
5	axis 4	6	axis 3
7	axis 2	9	axis 5
9	button 4	10	button 5
X	button 2	Y	button 3
A	button 0	B	button 1
back	button 6	start	button 7
center	button 10		

Joystick debugging

Mostly you'd want to debug your application with multiple controllers, or test it against _other_ types of controllers (e.g. different brands). As an alternative you might want to use some of the available controller emulators, such as [vJoy](#).

2.7 Widgets

2.7.1 Introdução ao Widgets

O [Widget](#) é o bloco básico de construção de interfaces GUI com o Kivy. Fornece um [Canvas](#) que pode ser usado para desenhar na tela. Recebe eventos e reage a eles. Para uma explicação detalhada sobre a classe [Widget](#), consulte a documentação do módulo.

2.7.2 Manipulando a arvore de Widgets

Widgets em Kivy são organizados em árvores. Seu aplicativo terá um 'Widget root', que normalmente terá [children](#) que poderão ter seus próprios [children](#). Os filhos de um Widget são representadas como os atributos de um [children](#), um Kivy [ListProperty](#).

A árvore de Widget pode ser manipulada com os seguintes métodos:

- [add_widget\(\)](#): adiciona um Widget como um Widget filho
- [remove_widget\(\)](#): remove um widget da lista de filhos
- [clear_widgets\(\)](#): remove todas os Widgets filhos de um widget

Por exemplo, se você quer adicionar um botão dentro do BoxLayout, você pode fazer:

```
layout = BoxLayout(padding=10)
button = Button(text='My first button')
layout.add_widget(button)
```

O botão é adicionado ao Layout: a propriedade pai do botão será definida como layout; O Layout terá um botão adicionado à sua lista de filhos. Para remover o botão do layout:

```
layout.remove_widget(button)
```

Com a remoção, a propriedade pai do botão será definida como None e o Layout terá o botão removido de sua lista de filhos.

Se quiseres limpar todas os filhos de dentro de um Widget, use o método `clear_widgets()` method:

```
layout.clear_widgets()
```

Aviso: Nunca manipule a lista de filhos você mesmo, a menos que realmente saibas o que estás fazendo. A árvore de Widgets está associada a uma árvore gráfica. Por exemplo, se adicionares um Widget à lista de filhos sem adicionar seu Canvas à árvore de gráficos, o Widget será um filho, sim, mas nada será desenhado na tela. Além disso, poderás ter problemas em outras chamadas de `add_widget()`, `remove_widget()` e `clear_widgets()`.

2.7.3 Atravessando a árvore

A instância da classe Widget `children` propriedade de lista contém todas os filhos. Podes facilmente atravessar a árvore fazendo:

```
root = BoxLayout()
# ... add widgets to root ...
for child in root.children:
    print(child)
```

No entanto, isso deve ser usado com cuidado. Se pretendes modificar a lista de filhos com um dos métodos mostrados na seção anterior, deverás usar uma cópia da lista como temos no código a seguir:

```
for child in root.children[:]:
    # manipulate the tree. For example here, remove all widgets_
    # that have a
    # width < 100
    if child.width < 100:
        root.remove_widget(child)
```

Widgets não influenciam o tamanho/pos de seus filhos por padrão. O atributo *pos* é a posição absoluta em coordenadas de tela (a menos que, uses o *relativelayout*). Falaremos mais sobre isso na sequência) e *size*, é um tamanho absoluto.

2.7.4 Índice Z do Widgets

A ordem do desenho do Widget é baseada na posição do Widget na árvore de Widgets. O método *add_widget* tem um parâmetro *index* que pode ser usado para especificar sua posição na árvore de Widgets:

```
root.add_widget(widget, index)
```

Os Widgets indexados abaixo serão desenhados acima daqueles com um índice mais alto. Tenha em mente que o padrão para *index* é 0, portanto Widgets adicionados posteriormente são desenhados em cima dos outros, a menos que seja especificado o contrário.

2.7.5 Organizar com Layouts

layout é um tipo especial de Widget que controla o tamanho e a posição de seus filhos. Existem diferentes tipos de Layouts, permitindo a organização automática diferente de seus filhos. Layouts utilizam as propriedades *size_hint* e *pos_hint* para determinar o *size* e a *pos* de seus filhos.

O **BoxLayout**: Arranja Widgets de maneira adjacente (vertical ou horizontalmente), para preencher todo o espaço. A propriedade *size_hint* dos filhos pode ser usada para alterar as proporções definidas para cada filho, ou defina um tamanho fixo para alguns dos Widgets.

Box Layout

vertical

vertical

Grid Layout

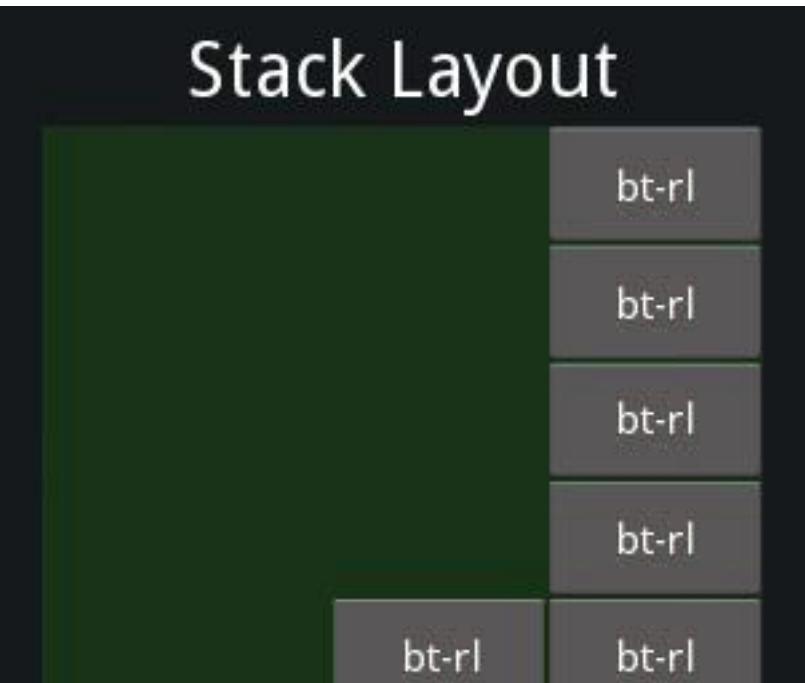
cols = 3 cols = 3 cols = 3

cols = 3 cols = 3 cols = 3

cols = 3 cols = 3 cols = 3

cols = 3 cols = 3

Stack Layout



Anchor Layout





GridLayout: Arranja os Widgets em uma grade. Deves especificar pelo menos uma dimensão da grade para que o Kivy possa calcular o tamanho dos elementos e como organizá-los.

StackLayout: Arranja os Widgets adjacentes um ao outro, mas com um tamanho de conjunto em uma das dimensões, sem tentar fazê-los caber dentro de todo o espaço. Isso é útil para exibir filhos do mesmo tamanho predefinido.

AnchorLayout: Um layout simples apenas somente gerenciando as posições dos Widgets filhos. Permite colocar os Widgets em uma posição relativa a borda de outros layout. A propriedade *size_hint* não é contemplada.

FloatLayout: Permite colocar os filhos em locais e tamanhos arbitrários, absolutos ou relativos ao tamanho do Layout. O padrão *size_hint* (1, 1) fará com que todas os filhos tenham o mesmo tamanho do Layout inteiro, então, provavelmente desejarás alterar esse valor se tiveres mais de um filho. Poderás definir o *size_hint* pra (None, None) e utilizar tamanho absoluto com *size*. Este Widget também utiliza a propriedade *pos_hint*, que é uma posição relativa ao Layout pai em que os Widgets filhos estão contidos.

RelativeLayout: comporta-se como o FloatLayout, exceto a posição dos filhos que são relativas à posição de layout, não a tela.

Examine a documentação individual dos Layouts para uma compreensão mais aprofundada.

size_hint and *pos_hint*:

- *floatlayout*

- *boxlayout*
- *gridlayout*
- *stacklayout*
- *relativelayout*
- *anchorlayout*

size_hint é uma *ReferenceListProperty* de *size_hint_x* e *size_hint_y*. Ele aceita valores no intervalo entre 0 a 1 ou *None* e o padrão é (1, 1). Isso significa que se o Widget está no layout, o layout será alocado no local o quanto possível em ambas as direções (em relação ao tamanho dos layouts).

Definir o *size_hint* como sendo (0,5, 0,8), por exemplo, fará com que o Widget ocupe 50% da largura e 80% da altura do tamanho disponível para o *Widget* dentro de um layout.

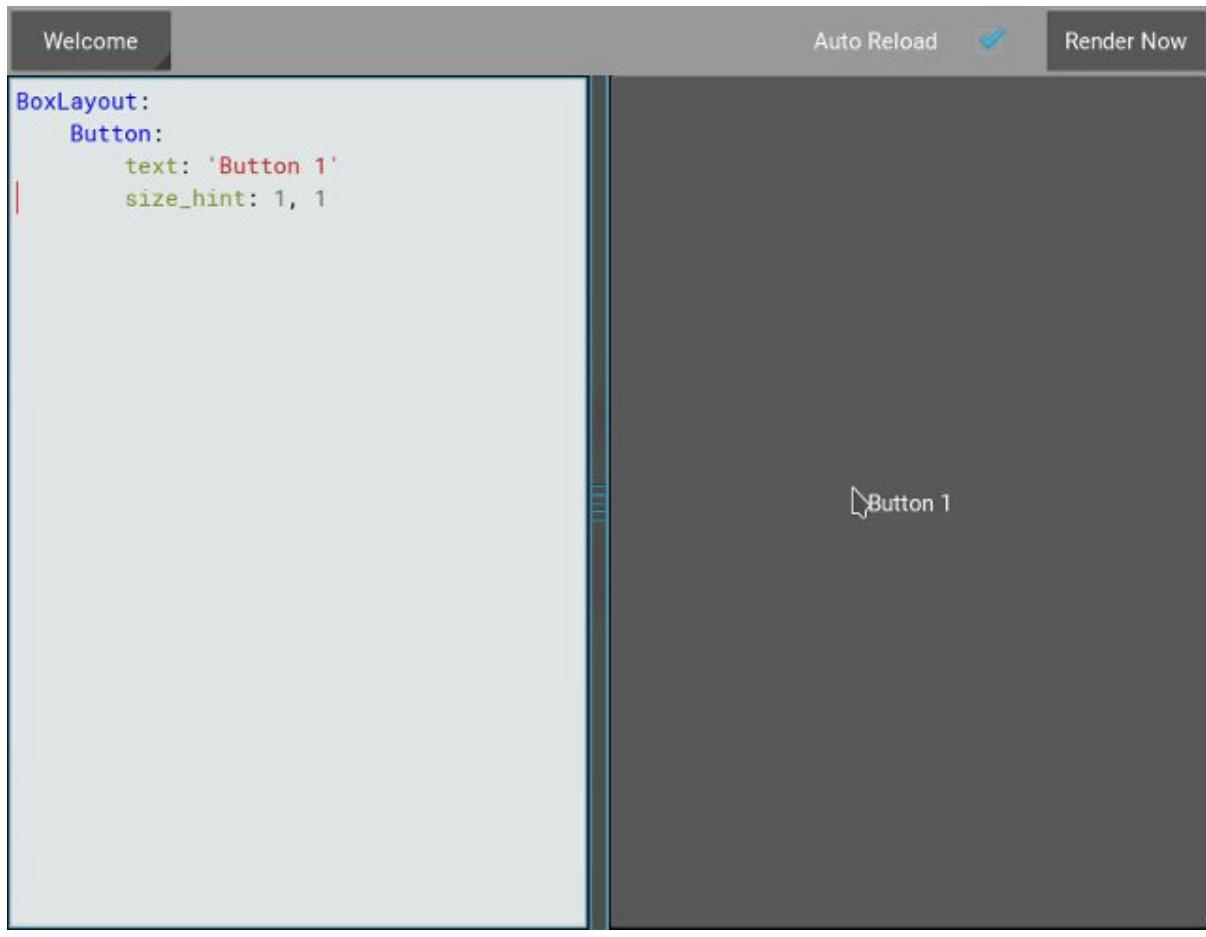
Considere o exemplo a seguir:

```
BoxLayout:
    Button:
        text: 'Button 1'
        # default size_hint is 1, 1, we don't need to specify it_
        ↪explicitly
        # however it's provided here to make things clear
        size_hint: 1, 1
```

Carregando o KivyCatalog

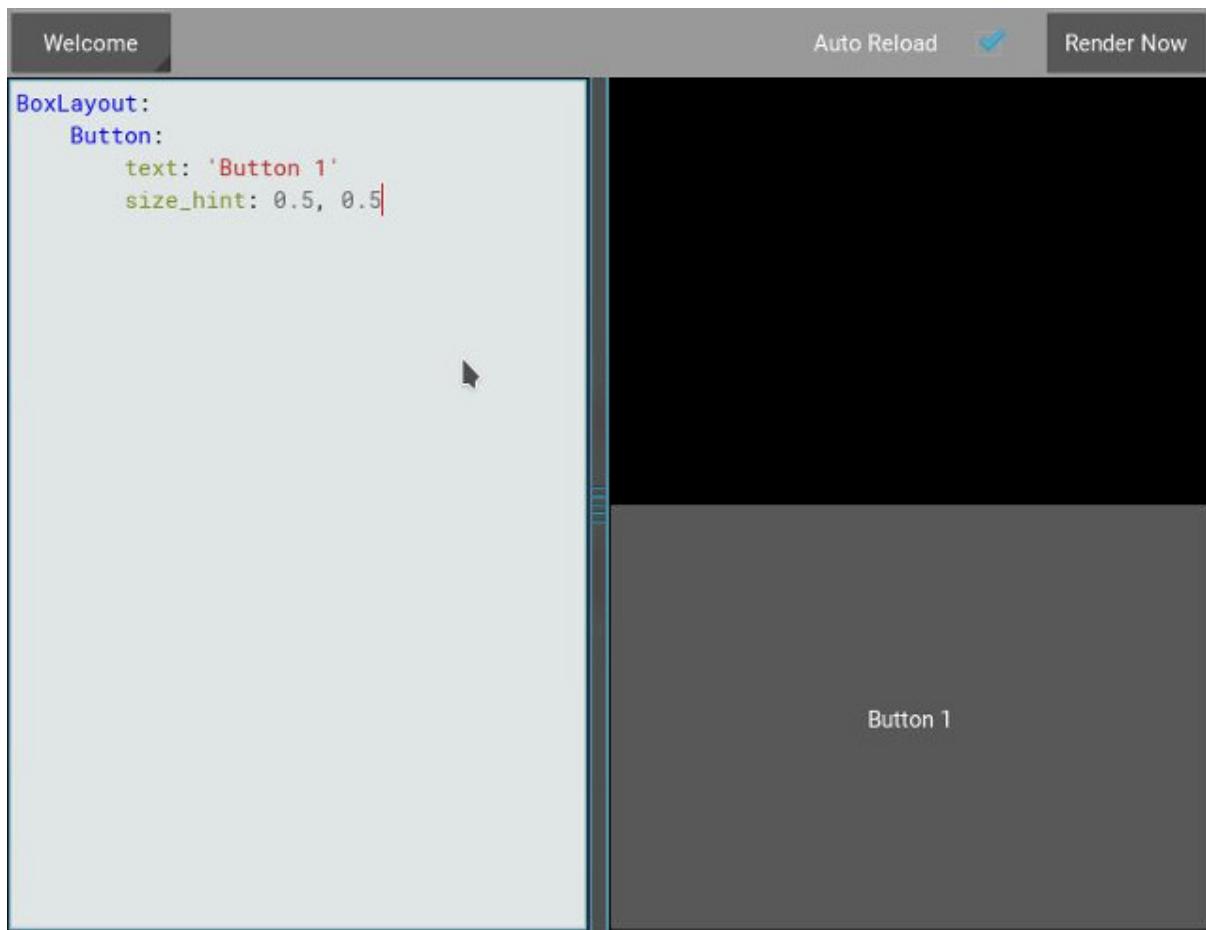
```
cd $KIVYDIR/examples/demo/kivycatalog
python main.py
```

Substitua \$KIVYDIR pelo diretório de sua instalação do Kivy. Clique no botão “Layout de caixa” a esquerda. Agora cole o código de cima no painel do editor à direita.



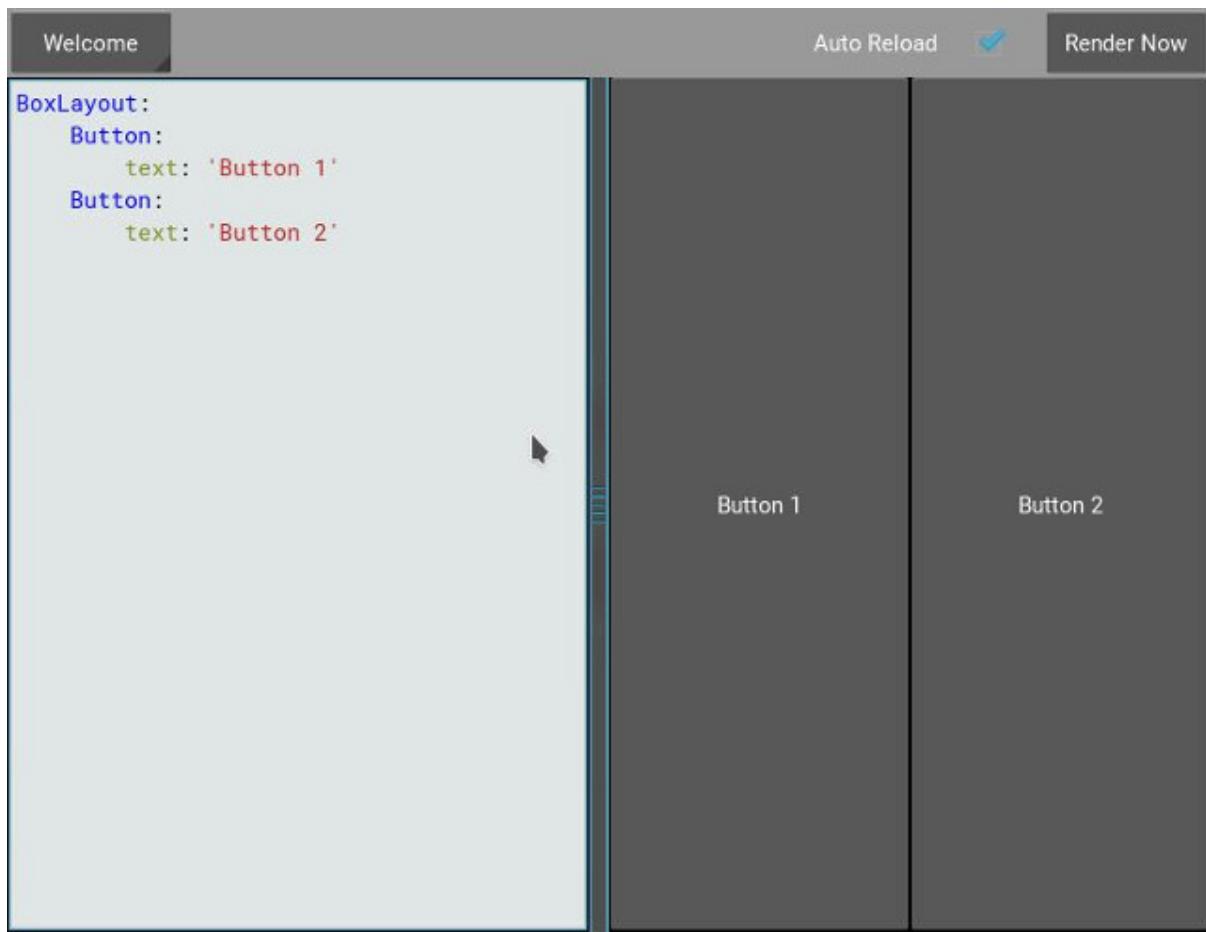
Como podes ver na imagem acima, o *Button* ocupa 100% do *size* do Layout.

Alterando o *size_hint_x*/*size_hint_y* pra .5 fará o *Widget* ocupar 50% do *layout width/height*.

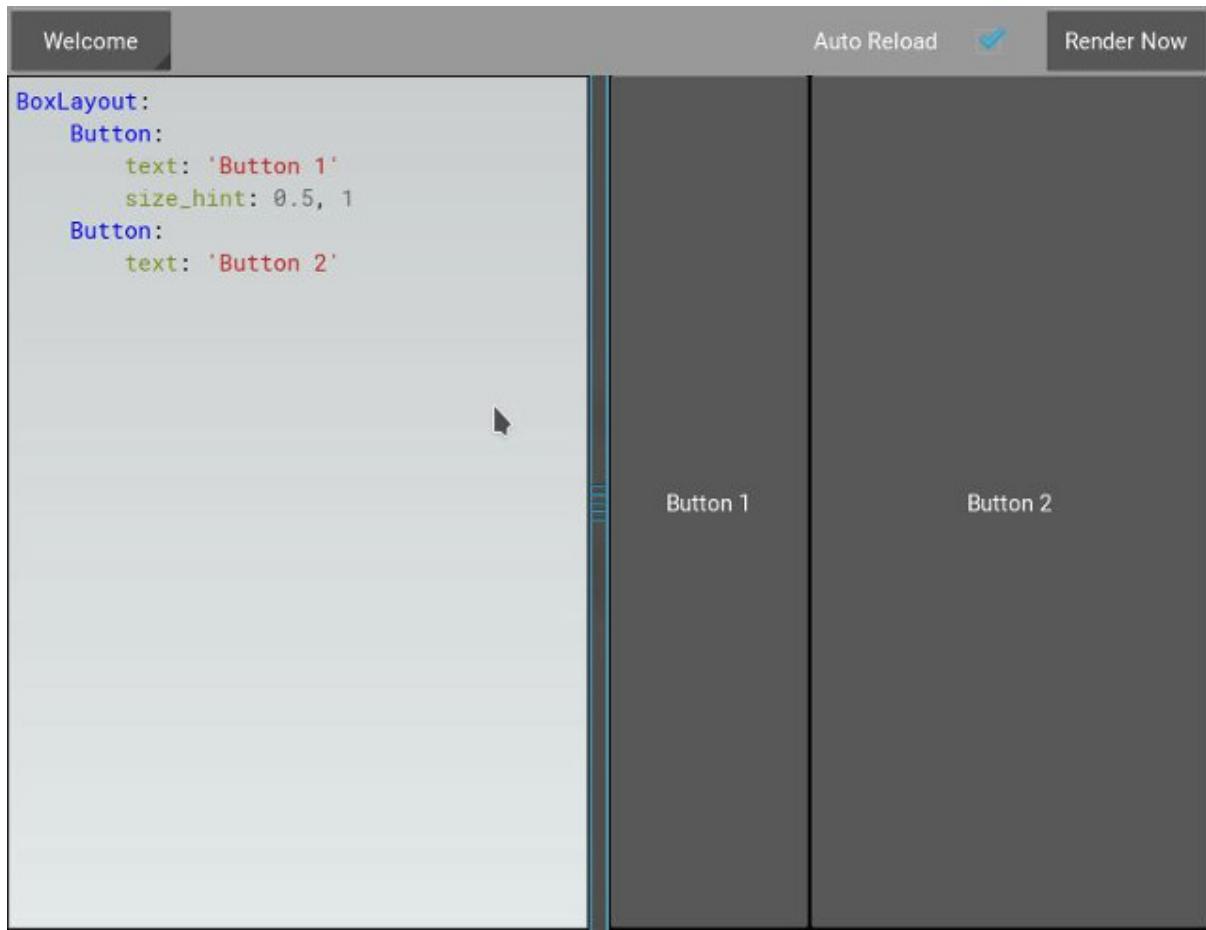


You can see here that, although we specify `size_hint_x` and `size_hint_y` both to be .5, only `size_hint_y` seems to be honored. That is because `boxlayout` controls the `size_hint_y` when `orientation` is *vertical* and `size_hint_x` when `orientation` is 'horizontal'. The controlled dimension's size is calculated depending upon the total no. of `children` in the `boxlayout`. In this example, one child has `size_hint_y` controlled (.5/.5 = 1). Thus, the widget takes 100% of the parent layout's height.

Vamos adicionar outro `Button` ao `layout` e ver o que acontece.



O **boxlayout** pela sua própria natureza, divide o espaço disponível entre os seus **children** igualmente. No nosso exemplo, a proporção é 50-50, porque temos dois **children**. Vamos usar *size_hint* em um dos filhos e ver os resultados.



Se uma filhos determina o `size_hint`, isso especifica quanto espaço o `Widget` irá ocupar do `size` fornecido a ele pelo `boxlayout`. Em nosso exemplo, o primeiro `Button` especifica .5 para o `size_hint_x`. O espaço do Widget é calculado da seguinte forma:

```
first child's size_hint divided by
first child's size_hint + second child's size_hint + ...n(no of_
→children)
.5/(.5+1) = .333...
```

O restante do `width` do `BoxLayout` é dividido entre o resto dos `children`. Em nosso exemplo, isso significa que o segundo `Button` ocupa 66,66% do `width` do `layout`.

Experimente com `size_hint` para sentir-se confortável usando-o.

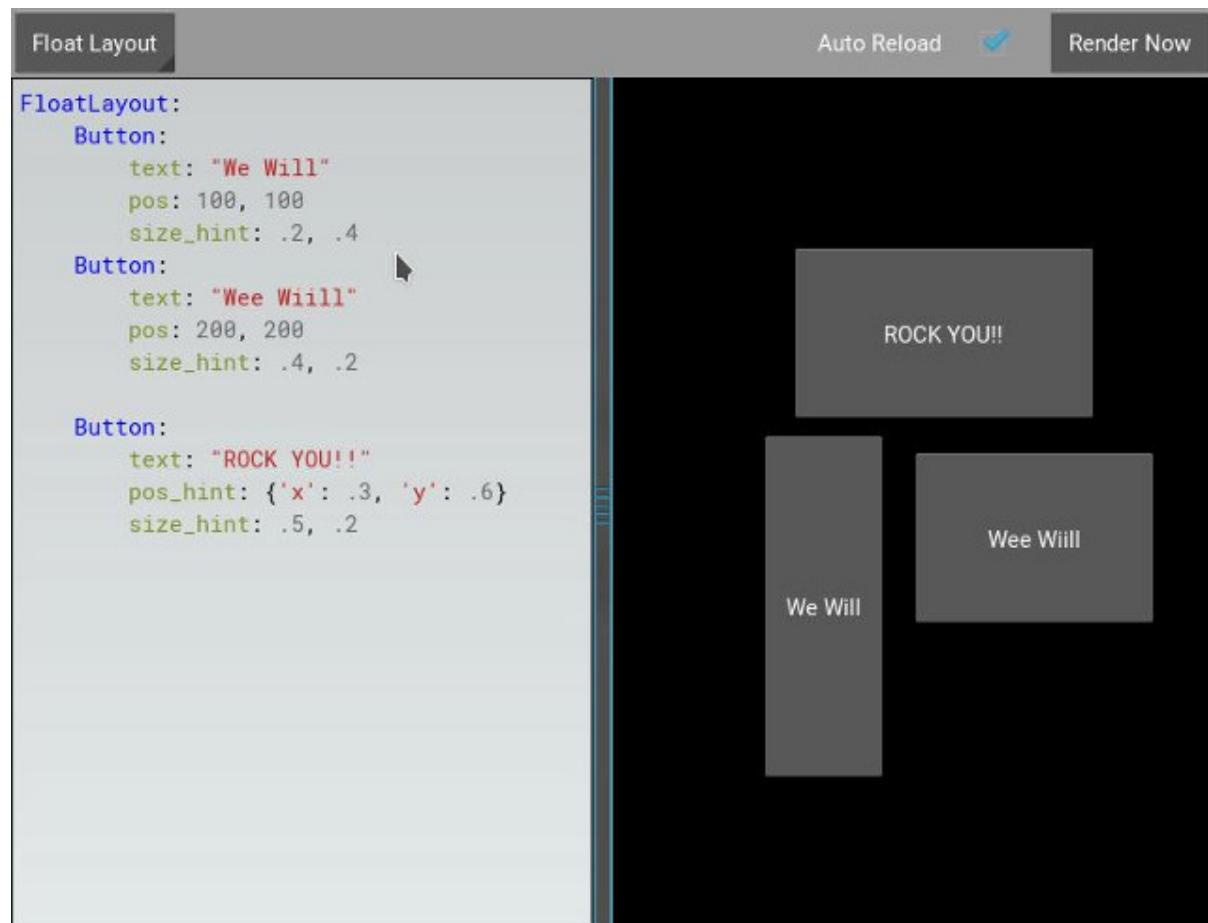
Se quiseres controlar o `size` absoluto de um `Widget`, poderás definir o `size_hint_x`/`size_hint_y` ou ambos como sendo igual a `None` para que a `width` e a `height` dos atributos dos Widgets sejam utilizados.

`pos_hint` é um dict, cujo o padrão é vazio. Igual o que temos para o `size_hint`, Layouts utilizam o `pos_hint` de forma diferente, mas geralmente poderás adicionar valores a qualquer `pos` do atributos (`x, y, right, top, center_x, center_y`) para ter o `Widget` posicionados em relação ao `parent`.

Vamos experimentar o seguinte código do KivyCatalog para entender o *pos_hint* visualmente:

```
FloatLayout:  
    Button:  
        text: "We Will"  
        pos: 100, 100  
        size_hint: .2, .4  
    Button:  
        text: "Wee Wiill"  
        pos: 200, 200  
        size_hint: .4, .2  
  
    Button:  
        text: "ROCK YOU!!"  
        pos_hint: {'x': .3, 'y': .6}  
        size_hint: .5, .2
```

Isso nos dá:



Como com *size_hint*, deverás experimentar o *pos_hint* para entender o efeito que tem sobre as posições dos Widget.

2.7.6 Adicionando um plano de fundo a um Layout

Uma das perguntas mais frequentes sobre os Layouts é:

"How to add a background image/color/video/... to a Layout"

Layouts por sua natureza não têm representação visual: os mesmos não possuem instruções de Canvas por padrão. No entanto, poderás adicionar instruções de Canvas a uma instância de Layout facilmente, como por exemplo, adicionar um fundo colorido:

Em Python:

```
from kivy.graphics import Color, Rectangle

with layout_instance.canvas.before:
    Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-
    ↵255
    self.rect = Rectangle(size=layout_instance.size,
                          pos=layout_instance.pos)
```

Infelizmente, isso só desenhará um retângulo na posição inicial do layout e tamanho. Para certificar-se de que o rect é desenhado dentro do layout, quando o tamanho do layout/pos forem alterados, precisamos ouvir as alterações e atualizar o *size* e a *pos*. Podemos fazer da seguinte forma:

```
with layout_instance.canvas.before:
    Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-
    ↵255
    self.rect = Rectangle(size=layout_instance.size,
                          pos=layout_instance.pos)

def update_rect(instance, value):
    instance.rect.pos = instance.pos
    instance.rect.size = instance.size

# listen to size and position changes
layout_instance.bind(pos=update_rect, size=update_rect)
```

Em kv:

```
FloatLayout:
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            # self here refers to the widget i.e BoxLayout
            pos: self.pos
            size: self.size
```

A declaração kv define uma vínculo implícito: as últimas duas linhas kv asseguram que os valores de *pos* e *size* do retângulo serão atualizados quando o *pos* do *floatlayout* sofrer alguma alteração.

Agora, colocamos os Snippets acima no shell do aplicativo Kivy.

Modo com puro Python:

```
from kivy.app import App
from kivy.graphics import Color, Rectangle
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.button import Button

class RootWidget(FloatLayout):

    def __init__(self, **kwargs):
        # make sure we aren't overriding any important functionality
        super(RootWidget, self).__init__(**kwargs)

        # let's add a Widget to this layout
        self.add_widget(
            Button(
                text="Hello World",
                size_hint=(.5, .5),
                pos_hint={'center_x': .5, 'center_y': .5}))


class MainApp(App):

    def build(self):
        self.root = root = RootWidget()
        root.bind(size=self._update_rect, pos=self._update_rect)

        with root.canvas.before:
            Color(0, 1, 0, 1) # green; colors range from 0-1 not 0-255
            self.rect = Rectangle(size=root.size, pos=root.pos)
        return root

    def _update_rect(self, instance, value):
        self.rect.pos = instance.pos
        self.rect.size = instance.size

if __name__ == '__main__':
    MainApp().run()
```

Usando a linguagem kv:

```
from kivy.app import App
from kivy.lang import Builder

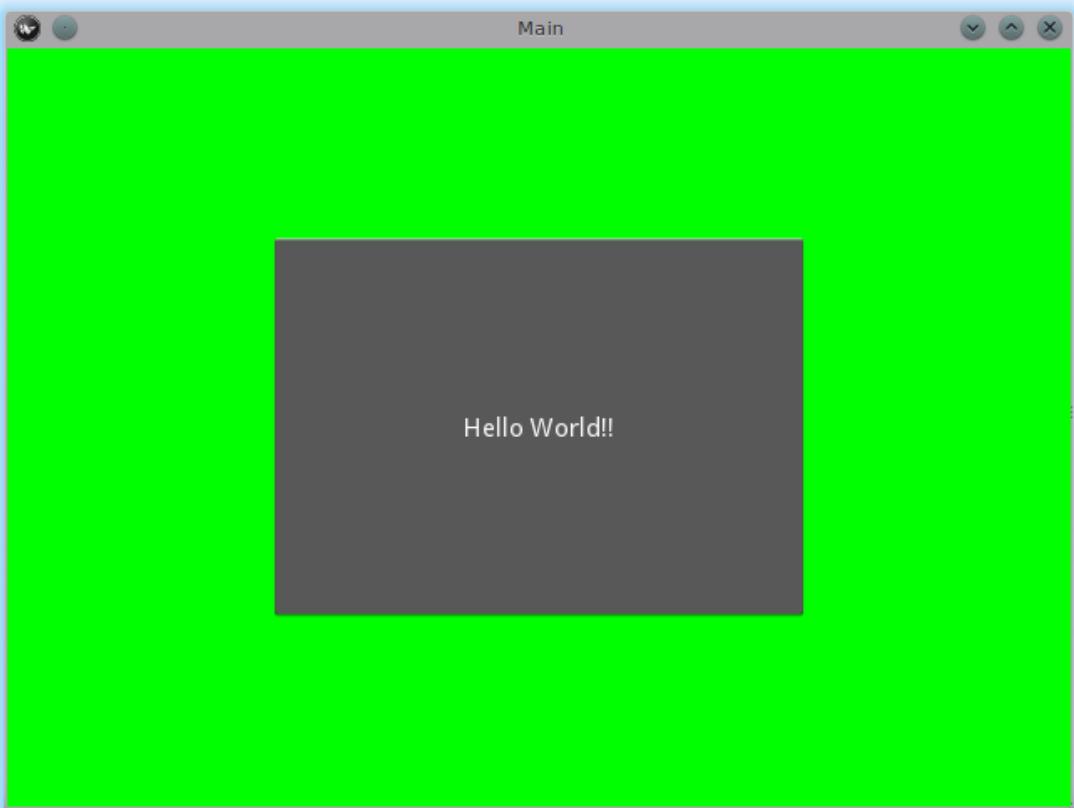

root = Builder.load_string('''
FloatLayout:
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            # self here refers to the widget i.e FloatLayout
            pos: self.pos
            size: self.size
    Button:
        text: 'Hello World!!!'
        size_hint: .5, .5
        pos_hint: {'center_x':.5, 'center_y': .5}
''')

class MainApp(App):

    def build(self):
        return root

if __name__ == '__main__':
    MainApp().run()
```

Ambos os aplicativos devem ser parecidos com isto:



Adicione uma cor ao plano de fundo de uma **Classe/Regra de Layouts Personalizados**

A forma como adicionamos o plano de fundo à instância do layout pode rapidamente se tornar complicada se precisarmos usar vários Layouts. Para ajudar com isso, podes usar uma subclasse de Layout e criar seu próprio Layout que adiciona um plano de fundo.

Usando Python:

```
from kivy.app import App
from kivy.graphics import Color, Rectangle
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.image import AsyncImage

class RootWidget(BoxLayout):
    pass

class CustomLayout(FloatLayout):
```

```

def __init__(self, **kwargs):
    # make sure we aren't overriding any important functionality
    super(CustomLayout, self).__init__(**kwargs)

    with self.canvas.before:
        Color(0, 1, 0, 1) # green; colors range from 0-1
→ instead of 0-255
        self.rect = Rectangle(size=self.size, pos=self.pos)

    self.bind(size=self._update_rect, pos=self._update_rect)

def _update_rect(self, instance, value):
    self.rect.pos = instance.pos
    self.rect.size = instance.size


class MainApp(App):

    def build(self):
        root = RootWidget()
        c = CustomLayout()
        root.add_widget(c)
        c.add_widget(
            AsyncImage(
                source="http://www.everythingzoomer.com/wp-content/
→uploads/2013/01/Monday-joke-289x277.jpg",
                size_hint=(1, .5),
                pos_hint={'center_x':.5, 'center_y':.5}))
        root.add_widget(AsyncImage(source='http://www.
→stuffistumbledupon.com/wp-content/uploads/2012/05/Have-you-seen-
→this-dog-because-its-awesome-meme-puppy-doggy.jpg'))
        c = CustomLayout()
        c.add_widget(
            AsyncImage(
                source="http://www.stuffistumbledupon.com/wp-
→content/uploads/2012/04/Get-a-Girlfriend-Meme-empty-wallet.jpg",
                size_hint=(1, .5),
                pos_hint={'center_x':.5, 'center_y':.5}))
        root.add_widget(c)
        return root

if __name__ == '__main__':
    MainApp().run()

```

Usando a linguagem kv:

```

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout

```

```

from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder


Builder.load_string('''
<CustomLayout>
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        Rectangle:
            pos: self.pos
            size: self.size

<RootWidget>
    CustomLayout:
        AsyncImage:
            source: 'http://www.everythingzoomer.com/wp-content/
            uploads/2013/01/Monday-joke-289x277.jpg'
            size_hint: 1, .5
            pos_hint: {'center_x':.5, 'center_y': .5}
        AsyncImage:
            source: 'http://www.stuffistumbledupon.com/wp-content/
            uploads/2012/05/Have-you-seen-this-dog-because-its-awesome-meme-
            puppy-doggy.jpg'
        CustomLayout
            AsyncImage:
                source: 'http://www.stuffistumbledupon.com/wp-content/
                uploads/2012/04/Get-a-Girlfriend-Meme-empty-wallet.jpg'
                size_hint: 1, .5
                pos_hint: {'center_x':.5, 'center_y': .5}
    ...)

class RootWidget(BoxLayout):
    pass

class CustomLayout(FloatLayout):
    pass

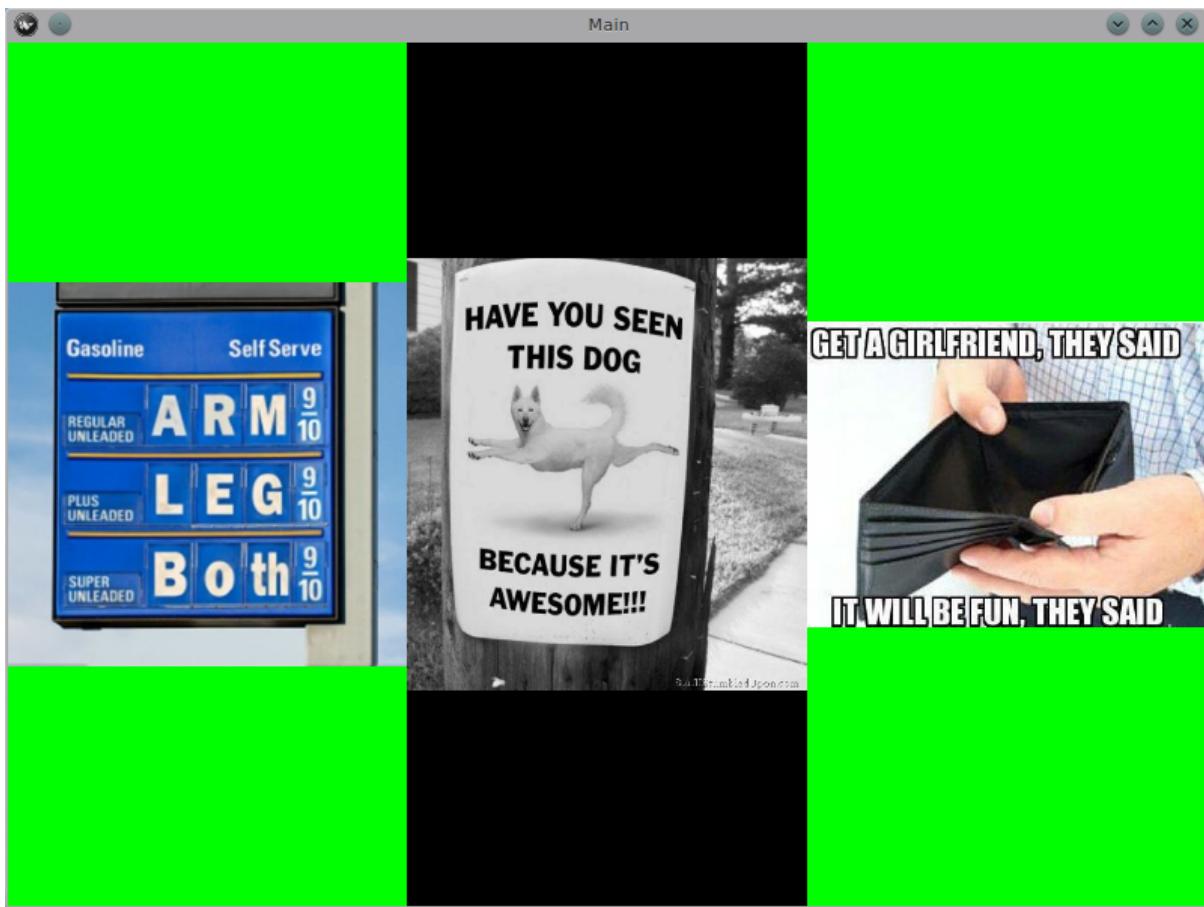
class MainApp(App):

    def build(self):
        return RootWidget()

if __name__ == '__main__':
    MainApp().run()

```

Ambos os aplicativos devem ser parecidos com isto:



Definir o plano de fundo na classe personalizada de Layout, garante que a mesma será usada em todas as instâncias de *CustomLayout*.

Agora, para adicionar uma imagem ou uma cor ao fundo de um Layout incorporado do Kivy, **globalmente**, precisamos substituir a regra kv do layout em questão. Considere o GridLayout:

```
<GridLayout>
    canvas.before:
        Color:
            rgba: 0, 1, 0, 1
        BorderImage:
            source: '../examples/widgets/sequenced_images/data/
                    images/button_white.png'
            pos: self.pos
            size: self.size
```

Então, quando colocamos este Snippet em um App Kivy:

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder

Builder.load_string('''
```

```

<GridLayout>
    canvas.before:
        BorderImage:
            # BorderImage behaves like the CSS BorderImage
            border: 10, 10, 10, 10
            source: '../examples/widgets/sequenced_images/data/
→images/button_white.png'
            pos: self.pos
            size: self.size

<RootWidget>
    GridLayout:
        size_hint: .9, .9
        pos_hint: {'center_x': .5, 'center_y': .5}
        rows:1
        Label:
            text: "I don't suffer from insanity, I enjoy every_
→minute of it"
            text_size: self.width-20, self.height-20
            valign: 'top'
        Label:
            text: "When I was born I was so surprised; I didn't_
→speak for a year and a half."
            text_size: self.width-20, self.height-20
            valign: 'middle'
            halign: 'center'
        Label:
            text: "A consultant is someone who takes a subject you_
→understand and makes it sound confusing"
            text_size: self.width-20, self.height-20
            valign: 'bottom'
            halign: 'justify'
    ...)

class RootWidget(FloatLayout):
    pass

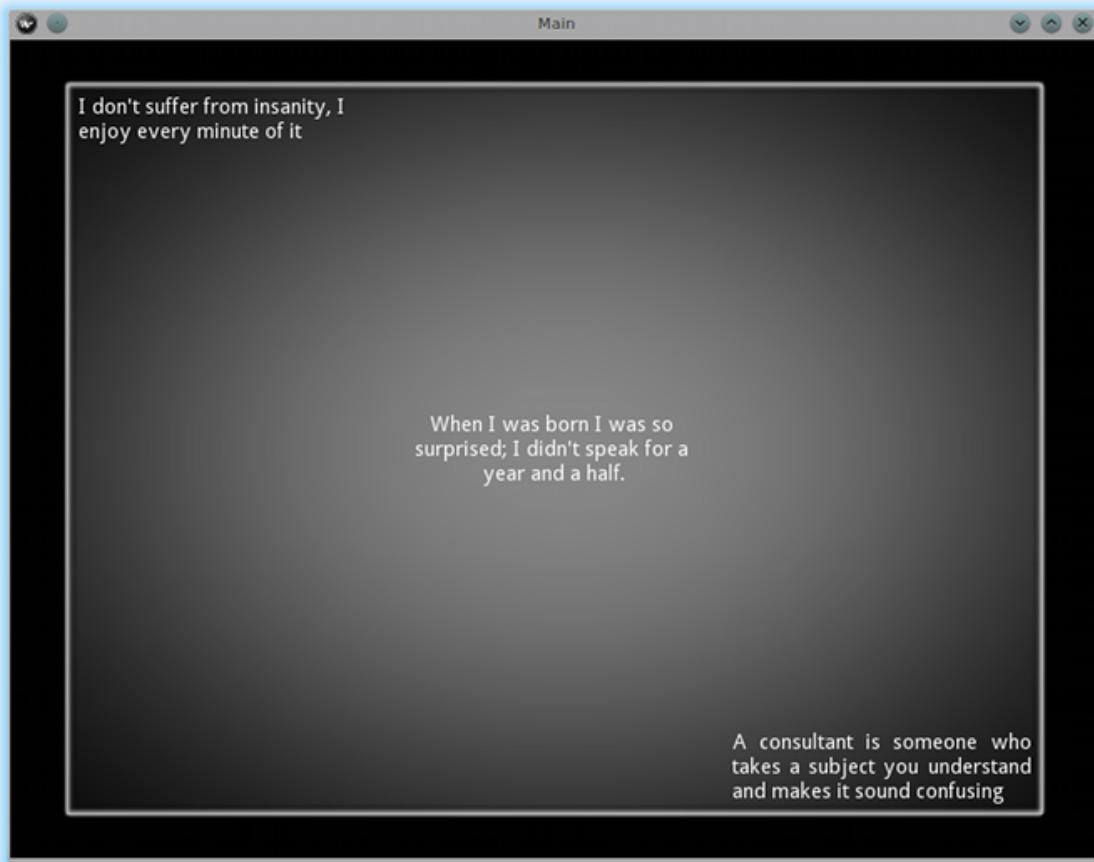
class MainApp(App):

    def build(self):
        return RootWidget()

if __name__ == '__main__':
    MainApp().run()

```

O resultado deverá ser algo como isto:



Como estamos substituindo a regra da classe GridLayout, qualquer uso desta classe em nosso aplicativo exibirá essa imagem.

Como cerca de um **Animated background?** (How about an **Animated background?**)

Pode definir as instruções de desenho como Rectangle/BorderImage/Ellipse / ... para usar uma textura particular:

```
Rectangle:  
    texture: reference to a texture
```

Usamos isso para exibir um fundo animado

```
from kivy.app import App  
from kivy.uix.floatlayout import FloatLayout  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.image import Image  
from kivy.properties import ObjectProperty  
from kivy.lang import Builder  
  
Builder.load_string('''  
<CustomLayout>
```

```

    canvas.before:
        BorderImage:
            # BorderImage behaves like the CSS BorderImage
            border: 10, 10, 10, 10
            texture: self.background_image.texture
            pos: self.pos
            size: self.size

<RootWidget>
    CustomLayout:
        size_hint: .9, .9
        pos_hint: {'center_x': .5, 'center_y': .5}
        rows:1
        Label:
            text: "I don't suffer from insanity, I enjoy every_
        ↪minute of it"
            text_size: self.width-20, self.height-20
            valign: 'top'
        Label:
            text: "When I was born I was so surprised; I didn't_
        ↪speak for a year and a half."
            text_size: self.width-20, self.height-20
            valign: 'middle'
            halign: 'center'
        Label:
            text: "A consultant is someone who takes a subject you_
        ↪understand and makes it sound confusing"
            text_size: self.width-20, self.height-20
            valign: 'bottom'
            halign: 'justify'
    ...)

class CustomLayout(GridLayout):
    background_image = ObjectProperty(
        Image(
            source='..../examples/widgets/sequenced_images/data/
        ↪images/button_white_animated.zip',
            anim_delay=.1))

class RootWidget(FloatLayout):
    pass

class MainApp(App):

```

```
def build(self):
    return RootWidget()

if __name__ == '__main__':
    MainApp().run()
```

Para tentar entender o que está acontecendo aqui, comece a ler o código desde a linha 13:

```
texture: self.background_image.texture
```

Isso especifica que a propriedade *texture* de' BorderImage' será atualizada sempre que a propriedade *texture* de' background_image' for atualizada. Definimos a propriedade *background_image* na linha 40:

```
background_image = ObjectProperty(...)
```

Isso configura *background_image* como um *ObjectProperty* em que adicionamos um *Image* ferramenta. Um Widget de imagem possui uma propriedade *texture*; Onde você vê *self.background_image.texture*, isso define uma referência,'texture', para esta propriedade. A *Image* o Widget suporta a animação: a textura da imagem é atualizada sempre que a animação é alterada e a textura da instrução BorderImage for atualizada no processo.

Também poderás apenas blit dados personalizados para a textura. Para obter detalhes, consulte a documentação de *Texture*.

2.7.7 Layouts Aninhados

SiM! É bastante divertido ver como o processo pode ser extensível.

2.7.8 Medidas de tamanho e posição

A unidade padrão de comprimento do Kivy é o pixel, todos os tamanhos e posições são expressos em pixel por padrão. Pode expressá-los em outras unidades, o que é útil para obter uma melhor consistência entre os dispositivos (eles serão convertidos para o tamanho em pixels automaticamente).

As unidades disponíveis são *pt*, *mm*, *cm*, *inch*, *dp* e *sp*. Pode aprender sobre seu uso de *metrics* na documentação.

Também podes experimentar com o *screen* para simular várias telas de dispositivos pra sua aplicação.

2.7.9 Separação de Tela com Gerenciador de Tela

Se o seu aplicativo é composto de várias telas, provavelmente queres uma maneira fácil de navegar entre um *Screen* para outro. Felizmente, existe a classe *ScreenManager*, que permite que definas telas separadamente, e também definir a *TransitionBase* entre as várias telas.

2.8 Gráficos

2.8.1 Introdução ao Canvas

A representação gráfica dos Widgets é feita usando um Canvas, que poderás ver tanto como uma prancheta de desenho ilimitada, ou como um conjunto de instruções de desenho. Existem várias instruções diferentes que poderás aplicar (adicionar) ao seu Canvas, mas existem dois tipos principais, são eles:

- *context instructions*
- *vertex instructions*

As instruções de contexto não desenham nada, mas alteram os resultados das instruções do vértice.

Canvasses pode conter dois subconjuntos de instruções. Eles são os grupos de instruções *canvas.before* e o *canvas.after*. As instruções nesses grupos serão executadas antes e depois do grupo *canvas*, respectivamente. Isso significa que eles aparecerão sob (em baixo) (se executado antes) e acima (se executado depois) deles. Esses grupos não são criados até que o usuário os acesse.

Para adicionar uma instrução canvas ao a um widget, use o contexto canvas:

```
class MyWidget(Widget):
    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        with self.canvas:
            # add your instruction for main canvas here

        with self.canvas.before:
            # you can use this to add instructions rendered before

        with self.canvas.after:
            # you can use this to add instructions rendered after
```

2.8.2 Instrução de Contexto

As instruções de contexto manipulam o contexto OpenGL. Pode girar, transladar e dimensionar o seu Canvas. Podes também anexar uma textura ou alterar a cor do desenho. Este é o comumente usado, mas outros também são muito úteis:

```
with self.canvas.before:  
    Color(1, 0, .4, mode='rgb')
```

2.8.3 Instruções de Desenho

As instruções de desenho vão desde as mais simples, como desenhar uma linha ou um polígono, até as mais complexas, como malhas ou curvas de Bezier:

```
with self.canvas:  
    # draw a line using the default color  
    Line(points=(x1, y1, x2, y2, x3, y3))  
  
    # lets draw a semi-transparent red square  
    Color(1, 0, 0, .5, mode='rgba')  
    Rectangle(pos=self.pos, size=self.size)
```

2.8.4 Manipulando instruções

Às vezes, irás querer atualizar ou remover as instruções que adicionaste a um Canvas, isso pode ser feito de várias maneiras dependendo de suas necessidades:

Pode manter uma referência às suas instruções e atualizá-las:

```
class MyWidget(Widget):  
    def __init__(self, **kwargs):  
        super(MyWidget, self).__init__(**kwargs)  
        with self.canvas:  
            self.rect = Rectangle(pos=self.pos, size=self.size)  
  
            self.bind(pos=self.update_rect)  
            self.bind(size=self.update_rect)  
  
    def update_rect(self, *args):  
        self.rect.pos = self.pos  
        self.rect.size = self.size
```

Ou podes limpar a tela e começar de novo:

```
class MyWidget(Widget):  
    def __init__(self, **kwargs):
```

```

super(MyWidget, self).__init__(**kwargs)
self.draw_my_stuff()

self.bind(pos=self.draw_my_stuff)
self.bind(size=self.draw_my_stuff)

def draw_my_stuff(self):
    self.canvas.clear()

    with self.canvas:
        self.rect = Rectangle(pos=self.pos, size=self.size)

```

Observe que a atualização das instruções é considerada a melhor prática, pois envolve menos despesas gerais e evita a criação de novas instruções.

2.9 Kv language

2.9.1 Conceito por trás da linguagem

À medida que seu aplicativo cresce e se torna mais complexo, é comum que a construção de árvores de Widget e a declaração explícita de bindings, se torne verboso e difícil de manter. A Linguagem 'KV' é uma tentativa de superar esses tipos de problemas.

A linguagem KV (às vezes chamada kvlang ou Kivy), permite criar a árvore de Widgets de forma declarativa e vincular as propriedades dos Widgets entre si ou vinculando a callbacks de maneira natural. A linguagem permite uma rápida prototipagem e mudanças ágeis em sua interface de usuário. A mesma também facilita a boa separação entre a lógica da sua aplicação e da sua interface de usuário.

2.9.2 Como carregar KV

Existem duas maneiras de carregar o código Kv em seu aplicativo:

- Por convenção de nome:

O Kivy procura um arquivo Kv com o mesmo nome da sua classe App em minúsculas, menos a palavra "App" no caso de o nome terminar com 'App', por exemplo:

MyApp -> my.kv

Se este arquivo define um *Root Widget* o mesmo será anexado ao atributo *root* da App e usado como sendo a base da árvore de Widgets do aplicativo.

- **Builder**: podes dizer ao Kivy para carregar diretamente uma String ou um arquivo. Se esta sequência de caracteres ou arquivo definir um Widget raiz, o mesmo será retornado pelo método:

```
Builder.load_file('path/to/file.kv')
```

ou:

```
Builder.load_string(kv_string)
```

2.9.3 Regra do contexto

Um código-fonte KV é constituído por *rules*, que são usadas para descrever o conteúdo dos Widget, podes ter uma regra *root* e um número infinito de regras como *class*.

A regra *root* é declarada pela declaração de classe do seu Widget principal, sem qualquer indentação, seguida por um sinal de dois pontos :'*e será definido como o atributo 'root* da instância do App:

```
Widget:
```

Uma regra *class*, declarada pelo nome de uma classe de Widget entre <> 'e seguida por ':, definirá como todas as instâncias dessa classe serão representada graficamente:

```
<MyWidget>:
```

Regras usam indentação para a delimitação, como o Python, a indentação deve ser feita com quatro espaços por nível, como as recomendado nas boas práticas de desenvolvimento com Python (PEP8).

Existem três palavras-chave específicas da linguagem kv:

- *app*: sempre se refere a instância da sua aplicação.
- *root*: refere-se ao Widget base na regra atual
- *self*: sempre se refere o widget atual

2.9.4 Sintaxe Especial

Existem duas sintaxes especiais para definir valores pra todo contexto kv:

Para acessar módulos e classes do Python de dentro do código kv:

```
#:import name x.y.z
#:import isdir os.path.isdir
#:import np numpy
```

é equivalente a:

```
from x.y import z as name
from os.path import isdir
import numpy as np
```

Em Python.

Para definir um valor global,

```
#:set name value
```

é equivalente a:

```
name = value
```

Em Python.

2.9.5 Instanciando Widget Filhos

Para declarar que um Widget tem um Widget filho, uma instância de alguma classe, basta declarar este Widget dentro da regra do outro Widget:

```
MyRootWidget:
    BoxLayout:
        Button:
        Button:
```

O exemplo acima define que o nosso Widget principal, uma instância de *MyRootWidget*, tem um filho que é uma instância de *BoxLayout*. E que um *BoxLayout* ainda possui dois filhos e que são instâncias da classe :class:`~kivy.uix.button.Button`.

Um código Python equivalente deste código poderia ser:

```
root = MyRootWidget()
box = BoxLayout()
box.add_widget(Button())
box.add_widget(Button())
root.add_widget(box)
```

Que podes achar menos agradável, tanto para ler e escrever.

Naturalmente, em Python, podes passar argumentos por palavras-chave para seus Widgets na criação especificando assim o seu comportamento. Por exemplo, para definir o número de colunas de um *gridlayout*, faríamos:

```
grid = GridLayout(cols=3)
```

Para fazer a mesma coisa com a linguagem kv, podemos definir a propriedades do Widget filho diretamente na regra:

```
GridLayout:  
    cols: 3
```

O valor é avaliado como uma expressão Python, e todas as propriedades usadas na expressão serão observadas, o que significa que se você tivesse algo parecido com isto em Python (isso assume que `self` é um Widget com uma `ListProperty`):

```
grid = GridLayout(cols=len(self.data))  
self.bind(data=grid.setter('cols'))
```

Para que sua exibição seja atualizada quando seus dados mudem, agora precisarás fazer apenas:

```
GridLayout:  
    cols: len(root.data)
```

Nota: A nomenclatura dos Widgets devem começar com letras maiúsculas, enquanto os nomes de propriedade devem começar com letras minúsculas. Encorajamos vocês a seguir as regras definidas na convenção [PEP8 Naming Conventions](#).

2.9.6 Vinculação de Eventos

Você pode vincular O eventos no Kv usando a sintaxe ":", ou seja, associando um callback a um evento:

```
Widget:  
    on_size: my_callback()
```

Podes passar os valores despachados pelo sinal usando a palavra-chave `args`:

```
TextInput:  
    on_text: app.search(args[1])
```

Expressões mais complexas pode ser utilizadas, como esta:

```
pos: self.center_x - self.texture_size[0] / 2., self.center_y -  
    ↴self.texture_size[1] / 2.
```

Esta expressão escuta uma alteração em `center_x`, `center_y` e `texture_size`. Caso alguma delas sofra alterações, a expressão será reavaliada para atualizar o campo `pos`.

Também podes lidar com eventos `on_` dentro da sua linguagem kv. Por exemplo, a classe `TextInput` possui uma propriedade `focus` cujo evento auto-gerado `on_focus` pode

ser acessado de dentro da linguagem kv da seguinte forma:

```
TextInput:  
    on_focus: print(args)
```

2.9.7 Ampliando o Canvas

A linguagem kv pode ser usado para definir as instruções de tela do seu Widget como este:

```
MyWidget:  
    canvas:  
        Color:  
            rgba: 1, .3, .8, .5  
        Line:  
            points: zip(self.data.x, self.data.y)
```

Eles são atualizados quando os valores das propriedades mudam.

Claro que você pode usar *canvas.before* e' *canvas.after*'.

2.9.8 Referenciando Widgets

Em uma árvore de Widgets, muitas vezes existe uma necessidade de acessar/referenciar outros Widgets. A linguagem kv fornece uma maneira de fazer isso usando id's. Pense nelas como variáveis de nível de classe que só podem ser usadas na linguagem kv. Considere o seguinte código:

```
<MyFirstWidget>:  
    Button:  
        id: f_but  
    TextInput:  
        text: f_but.state  
  
<MySecondWidget>:  
    Button:  
        id: s_but  
    TextInput:  
        text: s_but.state
```

Um *id* tem um escopo limitado à regra em que foi declarado, portanto, no código acima, *s_but* não pode ser acessado fora da regra *<MySecondWidget>*.

Aviso: Ao atribuir um valor *id*, lembre-se de que o valor não é uma String. Não há citações: good -> *id: value*, bad -> *id: 'value'*

Um *id* é uma *weakref* de um Widget e não para o próprio Widget. Como consequência, armazenar o *id* não é o suficiente para evitar que um Widget seja coletado pelo Garbage Collector. Veja a seguinte demonstração:

```
<MyWidget>:  
    label_widget: label_widget  
    Button:  
        text: 'Add Button'  
        on_press: root.add_widget(label_widget)  
    Button:  
        text: 'Remove Button'  
        on_press: root.remove_widget(label_widget)  
    Label:  
        id: label_widget  
        text: 'widget'
```

Embora uma referência a *label_widget* esteja armazenada em *MyWidget*, isso não é suficiente para manter o objeto vivo depois que outras referências foram removidas porque a referência é uma *weakref*. Portanto, depois que o botão Remover for clicado (o que remove qualquer referência direta ao Widget) e a janela for redimensionada (que chama o coleto de lixo resultando na exclusão dos *label_widget*), quando o botão adicionar é clicado para adicionar o Widget novamente, uma exceção *ReferenceError: weakly-referenced object no longer exists* será levantada.

Para manter o Widget vivo, uma referência direta ao mesmo deverá ser mantida em *label_widget*. Isto é obtido quando usamos *id .__self__* ou *label_widget.__self__* neste caso. A maneira correta de fazer isso seria:

```
<MyWidget>:  
    label_widget: label_widget.__self__
```

2.9.9 Acessando Widgets definidos com a linguagem kv no seu código Python

Considere o código abaixo em my.kv:

```
<MyFirstWidget>:  
    # both these variables can be the same name and this doesn't  
    # lead to  
    # an issue with uniqueness as the id is only accessible in kv.  
    txt_inpt: txt_inpt  
    Button:  
        id: f_but  
    TextInput:  
        id: txt_inpt  
        text: f_but.state  
        on_text: root.check_status(f_but)
```

Em myapp.py:

```
...
class MyFirstWidget(BoxLayout):
    txt_inpt = ObjectProperty(None)

    def check_status(self, btn):
        print('button state is: {}'.format(state=btn.state))
        print('text input text is: {}'.format(txt=self.txt_inpt))
...
```

txt_inpt está definido como uma *ObjectProperty* inicializada sendo igual a *None* dentro da classe:

```
txt_inpt = ObjectProperty(None)
```

Nesse ponto, *self.txt_inpt* é *None*. Em kvlang esta propriedade é atualizada para manter a instância da **TextInput** referenciado pelo id *txt_inpt*:

```
txt_inpt: txt_inpt
```

A partir deste ponto, *self.txt_inpt* contém uma referência ao Widget identificado pelo id *txt_input* e pode ser usado em qualquer parte da classe, bem como na função *check_status*. Em contraste com este método, também podes passar simplesmente o *id* para a função que precisa manipula-lo, como no caso de *f_but* utilizado no código acima.

Existe uma maneira mais simples para acessar objetos com tags *id* no kv usando o objeto de pesquisa de *ids*. Pode fazer isso da seguinte maneira:

```
<Marvel>
    Label:
        id: loki
        text: 'loki: I AM YOUR GOD!'
    Button:
        id: hulk
        text: "press to smash loki"
        on_release: root.hulk_smash()
```

Em seu código Python:

```
class Marvel(BoxLayout):
    def hulk_smash(self):
        self.ids.hulk.text = "hulk: puny god!"
        self.ids["loki"].text = "loki: >_<!!! # alternative syntax"
```

Quando o seu arquivo kv for analisado, o Kivy coletará todos os Widgets marcados

com id's e os colocará no dicionário de propriedade `self.ids`. Isso significa que também poderás iterar sobre esses Widgets e acessá-los da mesma forma em que acessamos qualquer dicionário:

```
for key, val in self.ids.items():
    print("key={0}, val={1}".format(key, val))
```

Nota: Embora o método `self.ids` seja bastante conciso, geralmente é considerado como senod uma ‘melhor prática’ usar `ObjectProperty`. Isso cria uma referência direta, fornece acesso mais rápido e é mais explícito.

2.9.10 Classes Dinâmicas

Considere o código abaixo:

```
<MyWidget>:
    Button:
        text: "Hello world, watch this text wrap inside the button"
        text_size: self.size
        font_size: '25sp'
        markup: True
    Button:
        text: "Even absolute is relative to itself"
        text_size: self.size
        font_size: '25sp'
        markup: True
    Button:
        text: "Repeating the same thing over and over in a comp =_"
    ↵fail"
        text_size: self.size
        font_size: '25sp'
        markup: True
    Button:
```

Em vez de ter que repetir os mesmos valores para cada botão, podemos apenas usar um Template em vez disso, veja o código a seguir:

```
<MyBigButt@Button>:
    text_size: self.size
    font_size: '25sp'
    markup: True

<MyWidget>:
    MyBigButt:
        text: "Hello world, watch this text wrap inside the button"
```

```

MyBigButt:
    text: "Even absolute is relative to itself"
MyBigButt:
    text: "repeating the same thing over and over in a comp =_
←fail"
MyBigButt:

```

Esta classe, criada apenas pela declaração desta regra, herda da classe Button e permite-nos alterar valores padrão e criar vínculos para todas as suas instâncias sem adicionar nenhum novo código no lado do Python.

2.9.11 Reutilizando estilos em vários Widgets

Considere o código abaixo em my.kv:

```

<MyFirstWidget>:
    Button:
        on_press: root.text(txt_inpt.text)
    TextInput:
        id: txt_inpt

<MySecondWidget>:
    Button:
        on_press: root.text(txt_inpt.text)
    TextInput:
        id: txt_inpt

```

Em myapp.py:

```

class MyFirstWidget(BoxLayout):

    def text(self, val):
        print('text input text is: {}'.format(txt=val))

class MySecondWidget(BoxLayout):

    writing = StringProperty('')

    def text(self, val):
        self.writing = val

```

Como ambas as classes compartilham o mesmo style .kv, esse design pode ser simplificado se reutilizarmos o style para ambos os Widgets. Pode fazer isso em .kv da seguinte maneira. Em my.kv:

```

<MyFirstWidget,MySecondWidget>:
    Button:

```

```
    on_press: root.text(txt_inpt.text)
TextInput:
    id: txt_inpt
```

Ao separar os nomes de classe com uma vírgula, todas as classes listadas na declaração terão as mesmas propriedades kv.

2.9.12 Desenhando com a Linguagem Kivy

Um dos objetivos da linguagem Kivy é separar as implementações da parte de apresentação e da implementação da lógica. **Separate the concerns**. O lado da apresentação o (layout) é endereçado pelo seu arquivo kv e a lógica pelo seu arquivo Python.

O código vai em arquivos `*.py`

Vamos começar com um pequeno exemplo. Primeiro, o arquivo Python chamado `main.py`:

```
import kivy
kivy.require('1.0.5')

from kivy.uix.floatlayout import FloatLayout
from kivy.app import App
from kivy.properties import ObjectProperty, StringProperty


class Controller(FloatLayout):
    '''Create a controller that receives a custom widget from the
    kv lang file.

    Add an action to be called from the kv lang file.
    '''

    label_wid = ObjectProperty()
    info = StringProperty()

    def do_action(self):
        self.label_wid.text = 'My label after button press'
        self.info = 'New info text'


class ControllerApp(App):

    def build(self):
        return Controller(info='Hello world')
```

```
if __name__ == '__main__':
    ControllerApp().run()
```

Neste exemplo, estamos criando uma classe Controller com 2 propriedades:

- `info` para receber algum texto
- `label_wid` para receber o Widget Label

Além disso, estamos criando um método `do_action()` que usará ambas as propriedades. Ele irá alterar o texto `info` e alterará o texto no Widget `label_wid`.

O layout vai em `controller.kv`

Executar este aplicativo sem um arquivo `.kv` correspondente funcionará, mas nada será exibido na tela. Isso é esperado, porque a classe `Controller` não tem Widgets definido, o mesmo é apenas um `BoxLayout`. Podemos criar a interface do usuário em torno da classe `Controlador` em um arquivo chamado `controller.kv`, que será carregado quando executarmos o `ControllerApp`. Como isso é funciona e é feito e quais arquivos são carregados é descrito no método `kivy.app.App.load_kv()`.

```
#:kivy 1.0

<Controller>:
    label_wid: my_custom_label

    BoxLayout:
        orientation: 'vertical'
        padding: 20

        Button:
            text: 'My controller info is: ' + root.info
            on_press: root.do_action()

        Label:
            id: my_custom_label
            text: 'My label before button press'
```

Um `Label` e um `Button` em um `BoxLayout` vertical. Parece muito simples. Há 3 coisas acontecendo aqui:

1. Usando dados do `Controlador`. Assim que a propriedade `info` for alterada no controlador, a expressão `text: 'Minha informação do controlador é:' + root.info` será automaticamente reavaliada, alterando o texto no `Button`.
2. Dando dados ao `Controlador`. A expressão `id: my_custom_label` está atribuindo um `Label` criado ao id de `my_custom_label`. Então, usando `my_custom_label` na expressão `label_wid: my_custom_label` dá a instância desse Widget `Label` para o seu `Controller`.

3. Criando um callback personalizado no `Button` usando o método `on_press` do `Controller`.

- `root` e `self` são palavras-chave reservadas, que podem ser utilizadas em qualquer lugar. `root` representa o Widget superior na regra e `self` representa o Widget atual.
- Pode usar qualquer id declarado nas regras da mesma forma que utilizar `root` e `self`. Por exemplo, poderias fazer isso no evento `on_press ()`:

```
Button:  
    on_press: root.do_action(); my_custom_label.font_size = 18
```

E é isso. Agora, ao executarmos `main.py`, `controller.kv` será carregado para que o `Button` e o `Label` apareçam e respondam aos nossos eventos de toque.

2.10 Integrando com outros Frameworks

Novo na versão 1.0.8.

2.10.1 Usando o Twisted dentro do Kivy

Nota: Pode usar a função `kivy.support.install_twisted_reactor` para instalar um reator do Twisted que será executado dentro do `main loop` de eventos do Kivy.

Todos os argumentos ou argumentos de palavra-chave passados pra esta função serão passados na função de intercalação de reatores `threadedselect`. Esses são os argumentos que normalmente se passariam para o `reactor.startRunning` do Twisted.

Aviso: Ao contrário do reactor padrão do Twisted, o reactor instalado não manipulará quaisquer sinais a menos que você defina o argumento de palavra-chave `installSignalHandlers` como sendo igual a 1 explicitamente. Isto é feito pra permitir que o Kivy manipule os sinais como habitualmente, a menos que pretendas especificamente que o reactor do Twisted manipule os sinais (por exemplo, SIGINT).

Os exemplos Kivy incluem um pequeno exemplo de um servidor Twisted e de um Client. O aplicativo de Servidor tem um servidor Twisted simples em execução e que registra todas as mensagens. O aplicativo Client pode enviar mensagens ao servidor e imprimirá sua mensagem e a resposta obtida. Os exemplos baseiam-se principalmente

no exemplo simples de Echo da documentação do Twisted, que poderás encontrar aqui:

- http://twistedmatrix.com/documents/current/_downloads/simpleserv.py
- http://twistedmatrix.com/documents/current/_downloads/simpleclient.py

Para testar o exemplo, execute primeiro *echo_server_app.py* e, em seguida, inicie *echo_client_app.py*. O servidor responderá com mensagens de Echo simples a qualquer coisa que o aplicativo Client envia quando alguma coisa for digitada na caixa de texto e a tecla Enter for pressionada.

Server App

```
# install_twisted_reactor must be called before importing and using
# the reactor
from kivy.support import install_twisted_reactor
install_twisted_reactor()

from twisted.internet import reactor
from twisted.internet import protocol

class EchoProtocol(protocol.Protocol):
    def dataReceived(self, data):
        response = self.factory.app.handle_message(data)
        if response:
            self.transport.write(response)

class EchoFactory(protocol.Factory):
    protocol = EchoProtocol

    def __init__(self, app):
        self.app = app

from kivy.app import App
from kivy.uix.label import Label

class TwistedServerApp(App):
    def build(self):
        self.label = Label(text="server started\n")
        reactor.listenTCP(8000, EchoFactory(self))
        return self.label

    def handle_message(self, msg):
```

```

    self.label.text = "received: %s\n" % msg

    if msg == "ping":
        msg = "pong"
    if msg == "plop":
        msg = "kivy rocks"
    self.label.text += "responded: %s\n" % msg
    return msg

if __name__ == '__main__':
    TwistedServerApp().run()

```

Client App

```

# install_twisted_reactor must be called before importing the reactor
from kivy.support import install_twisted_reactor
install_twisted_reactor()

# A simple Client that send messages to the echo server
from twisted.internet import reactor, protocol

class EchoClient(protocol.Protocol):
    def connectionMade(self):
        self.factory.app.on_connection(self.transport)

    def dataReceived(self, data):
        self.factory.app.print_message(data)

class EchoFactory(protocol.ClientFactory):
    protocol = EchoClient

    def __init__(self, app):
        self.app = app

    def clientConnectionLost(self, conn, reason):
        self.app.print_message("connection lost")

    def clientConnectionFailed(self, conn, reason):
        self.app.print_message("connection failed")

from kivy.app import App

```

```

from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.boxlayout import BoxLayout

# A simple kivy App, with a textbox to enter messages, and
# a large label to display all the messages received from
# the server
class TwistedClientApp(App):
    connection = None

    def build(self):
        root = self.setup_gui()
        self.connect_to_server()
        return root

    def setup_gui(self):
        self.textbox = TextInput(size_hint_y=.1, multiline=False)
        self.textbox.bind(on_text_validate=self.send_message)
        self.label = Label(text='connecting...\n')
        self.layout = BoxLayout(orientation='vertical')
        self.layout.add_widget(self.label)
        self.layout.add_widget(self.textbox)
        return self.layout

    def connect_to_server(self):
        reactor.connectTCP('localhost', 8000, EchoFactory(self))

    def on_connection(self, connection):
        self.print_message("connected successfully!")
        self.connection = connection

    def send_message(self, *args):
        msg = self.textbox.text
        if msg and self.connection:
            self.connection.write(str(self.textbox.text))
            self.textbox.text = ""

    def print_message(self, msg):
        self.label.text += msg + "\n"

if __name__ == '__main__':
    TwistedClientApp().run()

```

2.11 Empacotando sua Aplicação

2.11.1 Criando um pacote para Windows

Nota: Este documento só é válido para versões 1.9.1 ou superior do Kivy.

Embalar seu aplicativo para a plataforma Windows só pode ser feito dentro do sistema operacional Microsoft Windows. O seguinte processo foi testado no Windows com a instalação de Kivy **Wheels**, ver no final para instalações algumas alternativas.

O pacote será para 32 ou 64 bits, dependendo de qual versão do Python você executar com ele.

Requisitos

- Última versão do Kivy (instalado como descrito em *Instala no Windows*).
- PyInstaller 3.1+ (`pip install --upgrade pyinstaller`).

2.11.2 Hook padrão do PyInstaller

Esta seção se aplica ao PyInstaller (>= 3.1) que inclui os Hooks (ganchos) do Kivy. Para substituir o hook (ganchos) padrão, os seguintes exemplos precisam ser ligeiramente modificados. Veja *Sobrescrevendo o hook (gancho) padrão*.

Empacotando uma simples aplicação

Para este exemplo, iremos empacotar o projeto de exemplo **touchtracer** e incorporar um ícone personalizado ao mesmo. A localização dos exemplos do Kivy é, ao usar as Wheels, instalada em `python\\share\\kivy-examples` e ao usar o código-fonte direto do Github instalado em `kivy\\examples`. Nós apenas iremos nos referir ao caminho completo que leva aos exemplos como `examples-path`. O exemplo do touchtracer está em `examples-path\\demo\\touchtracer` e o arquivo principal é chamado de `main.py`.

1. Abra o seu shell de comando e certifique-se de que o Python está no PATH (ou seja, de que o comando `python` funcione).
2. Crie uma pasta dentro da pasta da qual a aplicação empacotada que será criada. Por exemplo, crie uma pasta `TouchApp` e altere para o diretório <<http://www.computerhope.com/cdhlp.htm>>_ com e.g. `cd TouchApp`. Então digite:

```
python -m PyInstaller --name touchtracer examples-
  ↪path\demo\touchtracer\main.py
```

Também podes adicionar um ícone *icon.ico* à pasta do aplicativo para criar um ícone do executável. Se não tiveres um arquivo .ico disponível, poderás converter seu arquivo *icon.png* para ICO usando o aplicativo da web *ConvertICO* <<http://www.convertico.com>> _. Salve o ‘*icon.ico* no diretório do touchtracer e digite:

```
python -m PyInstaller --name touchtracer --icon examples-
    ↪path\demo\touchtracer\icon.ico examples-
    ↪path\demo\touchtracer\main.py
```

Para mais opções, por favor, consulte o [Manual do PyInstaller](#).

3. O arquivo spec será *touchtracer.spec* localizado em *TouchApp*. Agora precisamos editar o arquivo spec para adicionar os (Hooks) ganchos de dependências para construir corretamente o *.exe. Abra o arquivo spec com seu editor favorito e adicione essas linhas no início da especificação (supondo que SDL2 está sendo usado, o padrão agora):

```
from kivy.deps import sdl2, glew
```

Em seguida, encontre *COLLECT()* e adicione os dados para o touchtracer (*touchtracer.kv*, *particle.png*, ...): Altere a linha para adicionar um objeto *Tree()*, por exemplo. *Tree('examples-path\demo\touchtracer\')*. Esta Árvore irá pesquisar e adicionar todos os arquivos encontrados no diretório touchtracer ao seu pacote final.

Para adicionar as dependências, antes do primeiro argumento de palavra-chave em *COLLECT*, adicione um objeto *Tree* para cada Path das dependências. Por exemplo. *[*Tree(p)* for *p* in (*sdl2.dep_bins* + *glew.dep_bins*)] por isso isso se parecerá com algo do tipo:

```
coll = COLLECT(exe, Tree('examples-path\\demo\\touchtracer\\'),
                 a.binaries,
                 a.zipfiles,
                 a.datas,
                 *[Tree(p) for p in (sdl2.dep_bins + glew.dep_
bins)],
                 strip=False,
                 upx=True,
                 name='touchtracer')
```

4. Agora nós construímos o arquivo de especificações em *TouchApp* com

```
python -m PyInstaller touchtracer.spec
```

5. O pacote compilado estará no diretório *TouchApp\dist\touchtracer*.

Empacotando uma aplicação de vídeo com *Gstreamer*

A seguir, modificamos um pouco o exemplo acima para empacotar um aplicativo que usa GStreamer para vídeo. Usaremos o exemplo `videoplayer` encontrado em `examples-path\widgets\videoplayer.py`. Crie uma pasta em algum lugar chamada `VideoPlayer` e na linha de comando mude seu diretório atual para aquela pasta e faça:

```
python -m PyInstaller --name gstvideo examples-  
-path\widgets\videoplayer.py
```

para criar o arquivo `gstvideo.spec`. Edite como acima e desta vez também inclua a dependência GStreamer:

```
from kivy.deps import sdl2, glew, gstreamer
```

E adicione o `Tree()` para incluir os arquivos de vídeo, por exemplo, `Tree('examples-path \widgets')` assim como as dependências GStreamer para que se pareça algo como do tipo:

```
coll = COLLECT(exe, Tree('examples-path\\widgets'),  
                a.binaries,  
                a.zipfiles,  
                a.datas,  
                *[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins +  
-gstreamer.dep_bins)],  
                strip=False,  
                upx=True,  
                name='gstvideo')
```

Em seguida, crie o arquivo `spec` no `VideoPlayer` com:

```
python -m PyInstaller gstvideo.spec
```

E deverás encontrar `GSTvideo.exe` em `VideoPlayer\dist\gstvideo`, que quando executado irá reproduzir um vídeo.

Nota: Se estiveres usando o PyGame e precisares do PyGame no empacotamento do seu App, terás que adicionar o seguinte código ao seu arquivo de spec devido ao Kivy issue #1638. Após as importações adicione o seguinte:

```
def getResource(identifier, *args, **kwargs):  
    if identifier == 'pygame_icon.tiff':  
        raise IOError()  
    return _original_getResource(identifier, *args, **kwargs)  
  
import pygame.pkgdata  
_original_getResource = pygame.pkgdata.getResource  
pygame.pkgdata.getResource = getResource
```

2.11.3 Sobrescrevendo o hook (gancho) padrão

Incluindo/excluindo vídeo e áudio e reduzindo o tamanho do aplicativo

O PyInstaller inclui um (Hook) gancho para o Kivy que, por padrão, adiciona **todos** os módulos principais usados pelo Kivy, por exemplo, o Áudio, Vídeo, Ortografia e etc. (você ainda precisa compactar as dlls do GStreamer e suas dependências manualmente com *Tree()* - veja o exemplo acima). Se o (Hook) gancho não estiver instalado ou para reduzir o tamanho do aplicativo, alguns desses módulos podem ser excluídos, por exemplo, se nenhum áudio/vídeo for utilizado, com um (Hook) ganho alternativo.

O Kivy fornece o (Hook) ganho alternativo em *hookspath()*. Além disso, se e somente se o PyInstaller não tiver os (Hooks) ganchos padrão *runtime_hooks()* este também deverá ser fornecido. Ao substituir o (Hook) ganho, este último normalmente não será necessário ser substituído.

O Hook alternativo *hookspath()* não inclui nenhum dos provedores Kivy. Para adicioná-los, os mesmos devem ser adicionados com *get_deps_minimal()* ou *get_deps_all()*. Consulte a respectiva documentação e *pyinstaller_hooks* para obter mais detalhes. Mas basicamente *get_deps_all()* adiciona todos os provedores como no Hook padrão enquanto *get_deps_minimal()* adiciona somente aqueles que são carregados quando o aplicativo é executado. Cada método fornece uma lista de importações Kivy ocultas e importações excluídas que podem ser passadas para *Analysis*.

Pode-se também gerar um (Hook) ganho alternativo que literalmente listará cada módulo que é provado pelo Kivy e aqueles não necessários poderão ser comentados. Veja *pyinstaller_hooks*.

Para usar os Hooks alternativos com os exemplos acima, modifique o seguinte para adicionar os hooks com *hookspath()* e *runtime_hooks* (se necessário) e *get_deps_minimal()* ou *get_deps_all()* para especificar os provedores.

For example, add the import statement:

```
from kivy.tools.packaging.pyinstaller_hooks import get_deps_minimal,  
    get_deps_all, hookspath, runtime_hooks
```

and then modify *Analysis* as follows:

```
a = Analysis(['examples-path\\demo\\touchtracer\\main.py'],  
            ...  
            hookspath=hookspath(),  
            runtime_hooks=runtime_hooks(),  
            ...  
            **get_deps_all())
```

Para incluir tudo como o Hook padrão. Ou:

```
a = Analysis(['examples-path\\demo\\touchtracer\\main.py'],
    ...
    hookspath=hookspath(),
    runtime_hooks=runtime_hooks(),
    ...
    **get_deps_minimal(video=None, audio=None))
```

Por exemplo, para excluir os provedores de áudio e vídeo e para os outros módulos principais só usar aqueles carregados.

Os pontos-chave é fornecer o alternativo `hookspath()` que não lista por padrão todos os provedores Kivy e manualmente em `hiddenimports` adiciona os provedores necessários ao remover os indesejados (áudio e Vídeo neste exemplo) com `get_deps_minimal()`.

Instalação alternativa

Os exemplos anteriores utilizados, por exemplo, `*[Tree(p) for p in (sdl2.dep_bins + glew.dep_bins + gstreamer.dep_bins)]`, para fazer o PyInstaller adicionar todas as DLLs usadas por essas dependências. Se o Kivy não foi instalado usando o método de Wheels, estes comandos não funcionarão ou seja, `Kivy.deps.sdl2` deixará de importar. Em vez disso, é preciso localizar a localização dessas dlls e passá-las manualmente para a classe `Tree` de uma forma semelhante à do exemplo.

2.11.4 Criando um pacote para o Android

Pode criar um pacote para o Android usando o projeto [python-for-android](#). Esta página explica como fazer o download e usá-lo diretamente em seu próprio computador (veja Empacotando seu aplicativo no APK), use a imagem pré-construída [Kivy Android VM](#) ou use a ferramenta [Buildozer](#) para automatizar todo o processo. Também podes ver Empacotando o seu aplicativo com o Kivy Launcher para executar programas Kivy sem que haja a necessidade de compilá-los.

Para novos usuários, recomendamos usar [Buildozer](#) como a maneira mais fácil de fazer um APK completo. Também podes executar seu aplicativo Kivy sem a etapa de compilação [Kivy Launcher](#).

As aplicações do Kivy podem ser [lançadas em mercados de Apps Android](#), como a PlayStore, bastando somente alguns passos extras para criar um APK e que esteja totalmente assinado.

O projeto Kivy inclui ferramentas para acessar as APIs do Android para realizar vibração, acesso ao sensor, envio de mensagens de texto etc. Essas informações, bem

como as informações sobre a depuração no dispositivo, estão documentadas em [main Android page](#).

Nota: O suporte ao Python 3 no Android já está disponível experimentalmente.

Buildozer

O Buildozer é uma ferramenta que automatiza todo o processo de construção. Ele baixa e configura todos os pré-requisitos para o python-for-android, incluindo o Android SDK e o NDK, feito isso cria um APK que pode ser automaticamente instalado em qualquer dispositivo.

Atualmente o Buildozer funciona apenas no Linux, e é uma versão alfa, porém, já funciona muito bem e pode simplificar significativamente a compilação do APK.

Podes obter o Buildozer em <https://github.com/kivy/buildozer>:

```
git clone https://github.com/kivy/buildozer.git  
cd buildozer  
sudo python2.7 setup.py install
```

Isso irá instalar buildozer em seu sistema Linux. Em seguida, navegue até o diretório do projeto e execute:

```
buildozer init
```

Isso cria um arquivo *buildozer.spec* controlando sua configuração de compilação. Deves editar adequadamente o arquivo adicionando o nome do seu aplicativo, etc. Pode definir variáveis para controlar a maioria ou todos os parâmetros passados para o python-for-android.

Instalação das ‘ dependências do buildozer’s <<https://buildozer.readthedocs.io/en/latest/installation.html#targeting-android>>’.

Finalmente, conecte seu dispositivo Android e execute:

```
buildozer android debug deploy run
```

para construir, dar push e executar automaticamente o APK no seu dispositivo.

O Buildozer tem muitas opções disponíveis e várias ferramentas para ajudá-lo, os passos acima são apenas a maneira mais simples de construir e executar o seu APK. A documentação completa está disponível [aqui](#). Você também pode verificar o README do Buildozer em <https://github.com/kivy/buildozer>.

Empacotando com Python-for-Android

Também podes empacotar diretamente com python-for-android, que pode lhe dar mais controle, porém, exige que você faça o download manual de partes da ferramenta Android.

Veja o [python-for-android documentation](#) para maiores informações.

Empacotando seu aplicativo para executar no Kivy Launcher

The [Kivy launcher](#) é um aplicativo Android que executa todos os exemplos do Kivy armazenados no cartão SD. Para instalar o Launcher do Kivy, precisarás:

1. Vá para [a página do Kivy Launcher](#) na página do Google PlayStore
2. Clique em Instalar
3. Selecione seu telefone... e está pronto!

Se não tiveres acesso à Google PlayStore em seu telefone/tablet, poderás fazer o download e instalar o APK manualmente em <http://kivy.org/#download>.

Uma vez instalado o Kivy, poderás colocar seus aplicativos Kivy no diretório Kivy em seu diretório de armazenamento externo (geralmente disponível em `/sdcard`, funciona até mesmo em dispositivos onde a memória interna):

```
/sdcard/kivy/<yourapplication>
```

`<yourapplication>` deve ser um diretório contendo:

```
# Your main application file:  
main.py  
# Some info Kivy requires about your app on android:  
android.txt
```

O arquivo `android.txt` deve conter:

```
title=<Application Title>  
author=<Your Name>  
orientation=<portrait|landscape>
```

Estas opções são apenas uma configuração bem básica. Se criares o seu próprio APK usando as ferramentas acima, poderás escolher muitas outras configurações.

Instalação dos Exemplos

O Kivy vem com vários exemplos, e estes podem ser um ótimo lugar para começar a experimentar o Kivy Launcher. Podes executá-los como é feito no código abaixo:

```
#. Download the `Kivy demos for Android <https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/kivy/kivydemo-for-android.zip>`_
#. Unzip the contents and go to the folder `kivydemo-for-android`
#. Copy all the subfolders here to
```

/sdcard/kivy

1. Execute o lançador e selecione uma das imagens, Showcase, Touchtracer, Cy-munk ou outras demonstrações...

Liberação no mercado

Se criastes o seu próprio APK com Buildozer ou com python-for-android, poderás criar uma versão de lançamento que pode ser disponibilizada na PlayStore ou em outros mercados Android.

To do this, you must run Buildozer with the `release` parameter (e.g. `buildozer android release`), or if using python-for-android use the `--release` option to build.py. This creates a release APK in the `bin` directory, which you must properly sign and zipalign. The procedure for doing this is described in the Android documentation at <https://developer.android.com/studio/publish/app-signing.html#signing-manually> - all the necessary tools come with the Android SDK.

Segmentação do Android

O Kivy é projetado para operar de forma idêntica entre plataformas e, como resultado, deverá tomar algumas decisões previsíveis com o design. Ele inclui seu próprio conjunto de Widgets e, por padrão, cria um APK com todas as dependências e bibliotecas necessárias.

É possível direcionar funcionalidades específicas uma versão específica do Android, tanto diretamente, como também de maneira multi-plataforma (de alguma forma) . Veja a seção *Usando as APIs do Android* na documentação [Kivy on Android documentation](#) para obter mais detalhes.

2.11.5 Máquina Virtual do Kivy com Android

Prefácio

Atualmente, as aplicações Android Kivy só podem ser construídas num ambiente Linux configurado com o python-for-android, o Android SDK e o Android NDK. Como este ambiente não só é complicado de configurar, como também, é impossível de construí-lo em alguns sistemas operacionais como o Windows ou OS X, oferecemos

uma imagem de disco **VirtualBox** completamente configurada para facilitar a produção de empacotamentos.

Se não estiveres familiarizado com a virtualização, recomendamos que leias a página [Wikipedia Virtualization page](#).

Começando

1. Baixe a imagem do disco de [aqui](#), na seção *Virtual Machine*. O download possui mias do 2GB (aproximadamente 6GB após extraído). Extraia o arquivo e lembre-se do local onde o arquivo *.VDI foi extraído.
2. Baixe a versão do VirtualBox para o seu computador e instale-o [VirtualBox](#).
3. Inicie o VirtualBox, clique em “Novo” no canto superior esquerdo. Em seguida, selecione “linux” e “Ubuntu 64-bit”.
4. Em “Hard drive”, escolha “Use an existing virtual hard drive file”. Procure o seu arquivo *.vdi e selecione-o.
5. Vá para as “Configurações” da sua máquina virtual. Na seção “Exibir -> Vídeo”, aumente a quantidade de RAM de vídeo para 32 MB ou mais. Ative a aceleração 3D para melhorar a experiência do usuário.
6. Inicie a máquina virtual e siga as instruções do arquivo Readme que está na área de trabalho.
7. Go to <https://github.com/kivy/buildozer#buildozer-virtual-machine> to see the current issues with the VM.

Construindo o APK

Uma vez que a VM é carregada, podes seguir as instruções de Empacotando o seu aplicativo em APK. Não precisas fazer o download com *git clone*, até porque o python-for-android já está instalado e configurado no diretório Home da máquina virtual.

Dicas e Truques

1. Pasta Compartilhada

Geralmente, seu ambiente de desenvolvimento e o conjunto de ferramentas são configurados em sua máquina host, mas o APK é construído em sua máquina Guest. O VirtualBox tem um recurso chamado ‘Pastas compartilhadas’, que permite ao seu convidado acesso direto a uma pasta em seu host.

Caso seja conveniente usar esse recurso (geralmente com opções ‘Permanentes’ e ‘Montagem automática’) para copiar o APK construído para a máquina host, ele poderá fazer parte do seu ambiente normal

de desenvolvimento. Um script simples pode automatizar facilmente o processo de compilação e cópia/mover.

Currently, VirtualBox doesn't allow symlink anymore in a shared folder. Adjust your buildozer.spec to build outside the shared folder. Also, ensure the *kivy* user is in the *vboxsf* group.

2. Copie e Cole

Por padrão, não serás capaz de compartilhar itens da área de transferência entre o host e a máquina convidada. Poderás conseguir isso ativando a opção de clipboard e compartilhamento “bi-direcional” em “Configurações -> Geral -> Avançado”.

3. Snapshots

Se estiveres trabalhando na branch de desenvolvimento do Kivy, puxe a versão mais recente poderá, por vezes, quebrar as coisas (nos esforçamos para que isso não aconteça). Poderás se proteger contra isso tirando snapshot antes de puxar os arquivos do servidor. Isso permitirá que restaures facilmente a sua máquina para seu estado anterior caso seja necessário.

4. Memória Insuficiente

Definir uma quantidade insuficiente de memória a Máquina Virtual poderá resultar numa falha de compilação com erros enigmáticos, como por exemplo:

```
arm-linux-androideabi-gcc: Internal error: Killed (program  
cc1)
```

Caso isso acontece, verifique a quantidade de memória livre na VM do Kivy e aumente a quantidade de RAM alocada para a mesma, caso seja necessário.

5. No space left

Read the section about resizing the VM at <https://github.com/kivy/buildozer#buildozer-virtual-machine>

2.11.6 Kivy no Android

Você pode rodar as aplicações do Kivy no Android, em (mais ou menos) qualquer dispositivo com OpenGL ES 2.0 (Android 2.2 mínimo requerido). Isso é padrão em dispositivos modernos; O Google informa que a exigência é atendida por 99,9% dos dispositivos<<https://developer.android.com/about/dashboards/index.html>>.

Os APKs com Kivy são aplicativos Android normais que você pode distribuir como qualquer outro, inclusive em lojas como a Play Store. Eles se comportam corretamente

quando o aparelho é pausados ou reiniciados, podem utilizar os serviços do Android e ter acesso à maioria da API do nativa do Java, conforme descrito a seguir.

Siga as instruções abaixo para aprender a:
:ref:empacotar seu aplicativo para Android <package_for_android>, depure seu código no dispositivo, e utilizas as APIs do Android tal como sensores de vibração e de leitura.

Pacotes para Android'

O projeto Kivy fornece todas as ferramentas necessárias para empacotar seu aplicativo no Android, incluindo a criação de seu próprio APK autônomo que pode ser distribuído em mercados como o PlayStore. Isso está totalmente descrito na documentação *:ref:packaging_android*.

Depurando seu aplicativo na plataforma Android

Pode ver a saída normal do seu código (stdout, stderr), bem como os logs normais do Kivy, através do fluxo LogCat do Android. Isso é acessado através de ADB, fornecido pelo *Android SDK <http://developer.android.com/sdk/index.html>* . Pode ser necessário ativar o ADB nas opções de desenvolvedor do dispositivo e, em seguida, conectar o dispositivo ao computador e executar:

```
adb logcat
```

Verás todos os logs, incluindo seu stdout/stderr e Kivy logger.

Se você empacotar o seu aplicativo com o Buildozer, a ferramenta *adb* pode não estar no seu \$ PATH e o comando acima poderá não funcionar. Em vez disso, pode executar:

```
buildozer android logcat
```

para executar a versão instalada pelo Buildozer ou localize as ferramentas do SDK em \$HOME/.buildozer/android/platform.

Você também pode executar e depurar seu aplicativo usando o **Kivy Launcher**. Se executares seu aplicativo desta forma, encontrarás arquivos de log dentro da subpasta “/.kivy/logs” dentro da pasta do aplicativo.

Usando APIs Android

Embora o Kivy seja um framework Python, o projeto Kivy mantém ferramentas para usar facilmente as APIs normais do Java, é possível acessar tudo, desde o vibrador a sensores, até o envio de mensagens por SMS ou e-mail.

Para novos usuários, recomendamos que utilizem o [Plyer](#). Para um acesso mais avançado ou para APIs que não estão atualmente Wrapped, podes usar o [Pyjnius](#) diretamente. O Kivy também fornece um [módulo Android](#) para acessar as funcionalidade básicas do Android.

O código e exemplos do Android fornecidos pelo usuário estão disponíveis em [Kivy wiki](#).

Plyer

Plyer <<https://github.com/kivy/plyer>> é uma API pythonica, independente de plataforma para usar recursos comumente encontrados em várias plataformas, principalmente as plataformas móveis. A ideia é fazer com que o seu aplicativo possa invocar funções simplesmente Plyer, seja para apresentar uma notificação ao usuário, enquanto que o Plyer cuidará de como conversar com a plataforma corretamente, e isso tudo, independentemente da plataforma ou sistema operacional. Internamente, o Plyer utiliza o Pyjnius (no Android), Pyobjus (no iOS) e algumas APIs específicas para plataformas Desktop, como Microsoft Windows e Mac OS X.

Por exemplo, o código a seguir faria vibrar seu dispositivo Android ou elevaria uma mensagem de erro `NotImplementedError` que você pode manipular adequadamente em outras plataformas, como Desktops que não possuem hardware apropriado:

```
from plyer import vibrator
vibrator.vibrate(10) # vibrate for 10 seconds
```

Plyer's contém a lista de APIs suportadas que está crescendo muito rapidamente, podes ver a lista completa em [README](#).

Pyjnius

Pyjnius é um módulo Python que permite acessar classes Java diretamente do seu código Python, convertendo automaticamente os argumentos para o tipo certo e permitindo que você converta facilmente os resultados recebidos em Java para Python.

Pyjnius pode ser obtido em [github](#), e ele tem a sua [própria documentação](#).

Aqui está um exemplo simples mostrando a capacidade do Pyjnius de acessar a API do Android de vibração, o mesmo resultado do código Plyer acima:

```
# 'autoclass' takes a java class and gives it a Python wrapper
from jnius import autoclass

# Context is a normal java class in the Android API
Context = autoclass('android.content.Context')

# PythonActivity is provided by the Kivy bootstrap app in python-
→for-android
```

```

PythonActivity = autoclass('org.renpy.android.PythonActivity')

# The PythonActivity stores a reference to the currently running
activity
# We need this to access the vibrator service
activity = PythonActivity.mActivity

# This is almost identical to the java code for the vibrator
vibrator = activity.getSystemService(Context.VIBRATOR_SERVICE)

vibrator.vibrate(10000) # The value is in milliseconds - this is
10s

```

Este código invoca funções diretamente da API Java para por exemplo, chamar o vibrador, com Pyjnius, automaticamente traduzindo a API para código Python e as nossas chamadas de callbacks equivalente em Java. É muito mais verboso e java-like do que a versão de Plyer, para nenhum benefício neste caso, embora Plyer não envolve cada API disponível para Pyjnius.

Pyjnius também tem poderosas habilidades para implementar interfaces Java, o que é importante para envolver algumas APIs, mas estas não estão documentadas aqui - podes encontrar a documentação do Pyjnius em <http://pyjnius.readthedocs.org/en/latest/> .

Módulo Android

O Python-for-android inclui um módulo Python (na verdade um Cython, wrapping Java) para acessar um conjunto limitado de APIs do Android. Isso foi amplamente substituído pelo Pyjnius e Plyer como visto acima, que são mais flexíveis, mas que podem, ainda, ocasionalmente ser útil. As funções disponíveis são dadas na documentação [python-for-android](#).

Isso inclui código para billing/IAP e criação/acesso a serviços Android, que ainda não está disponível nas outras ferramentas acima.

Status do Projeto e Dispositivos Testados

Estas seções descreviam anteriormente a existência das ferramentas de compilação do Kivy Android, com suas limitações e alguns dispositivos que eram conhecidos por funcionarem.

As ferramentas do Android estão agora bastante estáveis, e devem funcionar com praticamente qualquer dispositivo; Nossos requisitos mínimos são OpenGL ES 2.0 e Android 2.2. Neste momento, essas versões já são bem comuns - o Kivy está funcionando mesmo em SmartWatch com Android!

Uma limitação técnica atual é que as ferramentas de compilação do Android compilam apenas APKs ARM, que não serão executados em dispositivos Android com processadores x86 (estes são bem raros). Isso deverá ser adicionado em breve.

Como a Kivy funciona bem na maioria dos dispositivos, a lista de telefones/tablets suportados foi retirada - todos os dispositivos Android provavelmente funcionarão se atenderem às condições acima.

2.11.7 Criando um pacote para OS X

Nota: Empacotar os aplicativos Kivy com os seguintes métodos devem ser feitos dentro do OS X, e também, temos que as plataformas de 32 bits não são mais suportadas.

Usando o Buildozer

```
pip install git+http://github.com/kivy/buildozer cd  
/to/where/I/Want/to/package buildozer init
```

Edito o *buildozer.spec* e adicione os detalhes para o seu aplicativo. Dependências podem ser adicionadas à seção *requirements* =.

Por padrão, a versão especificada do Kivy nos requisitos será ignorada.

Se tens um *Kivy.app* em /Applications/Kivy.app, em seguida, o que é usado, para a embalagem. Caso contrário, a versão mais recente do kivy.org usando o Kivy master será baixada e usada.

Se quiseres empacotar para a versão do Python 3.x.x, basta baixar o pacote chamado Kivy3.7z na seção de download em kivy.org e extraí-lo para Kivy.app em /Applications, em seguida, executar:

```
buildozer osx debug
```

Uma vez que o aplicativo está empacotado, poderás querer remover pacotes desnecessários como o GStreamer, se não precisares de suporte a vídeo. A mesma lógica aplica-se a outras coisas que não fores utilizar, reduza o pacote ao seu estado mínimo que seja somente o necessário para que o aplicativo possa ser executado.

Como exemplo, incluímos o exemplo do ShowCase empacotado usando este método para o Python 2 (9.xMB) e 3 (15.xMB), poderás encontrar os pacotes aqui: https://drive.google.com/drive/folders/0B1WO07-OL50_a1FzSXJUajBFdnc.

É isso aí. Aprecie!

O Buildozer agora utiliza o SDK do Kivy para empacotar o aplicativo. Se desejas controlar mais detalhes em relação ao seu aplicativo que o buildozer atualmente oferece, então poderás usar o SDK diretamente, conforme detalhado na seção abaixo.

Usando o SDK do Kivy

Desde a versão 1.9.0, o Kivy é lançado para a plataforma OS X em uma distribuição autônoma e portátil.

Os aplicativos podem ser empacotados e distribuídos com o Kivy SDK usando o método descrito abaixo, facilitando a inclusão de frameworks como SDL2 e GStreamer.

1. Verifique se tens o Kivy SDK (Kivy.app) não modificado da página de download.
2. Execute os seguintes comandos:

```
> mkdir packaging
> cd packaging
packaging> git clone https://github.com/kivy/kivy-sdk-packager
packaging> cd kivy-sdk-packager/osx
osx> cp -a /Applications/Kivy.app ./Kivy.App
```

Nota: Esta etapa acima é importante, deves se certificar de preservar os caminhos e permissões. Um comando como `cp -rf` copiará, mas tornará o aplicativo inutilizável e levará a erro mais tarde.

3. Agora tudo o que precisas fazer é incluir seu aplicativo compilado no Kivy.app executando o seguinte comando:

```
osx> ./package-app.sh /path/to/your/<app_folder_name>/
```

Onde `<app_folder_name>` é o nome da sua aplicação.

Isso copia Kivy.app para `<app_folder_name>.app` e inclui uma cópia compilada do seu aplicativo neste pacote.

4. É isso, seu pacote independente está pronto para ser instalado em qualquer dispositivo! Agora, podes personalizar ainda mais seu aplicativo, conforme descrito abaixo.

Instalação dos Módulos

O pacote Kivy no OS X usa seu próprio *env virtual* que é ativado quando você executa o aplicativo usando o comando *kivy*. Para instalar qualquer módulo, faça da seguinte forma:

```
$ kivy -m pip install <modulename>
```

Onde os módulos / arquivos estão instalados?

Dentro do venv do aplicativo, digite:

```
Kivy.app/Contents/Resources/venv/
```

Se instalares um módulo que instala um binário, por exemplo, como kivy-garden, esse binário estará disponível a partir do venv acima, como depois de fazer:

```
kivy -m pip install kivy-garden
```

O lib garden só estará disponível quando você ativar este env.

```
source /Applications/Kivy.app/Contents/Resources/venv/bin/activate  
garden install mapview deactivate
```

Para instalar os arquivos binários

Basta copiar o binário para o diretório Kivy.app/Contents/Resources/venv/bin/.

Para incluir outros frameworks

O Kivy.app é provido com o framework SDL2 e Gstreamer . Para incluir frameworks diferentes daqueles que são fornecidos, faça o seguinte:

```
git clone http://github.com/tito/osxrelocator  
export PYTHONPATH=~/path/to/osxrelocator  
cd Kivy.app  
python -m osxrelocator -r ./Library/Frameworks/<Framework_name>.  
→framework/ \  
@executable_path/./Frameworks/<Framework_name>.framework/
```

Não se esqueça de substituir <Framework_name> pela estrutura do seu framework. Essa ferramenta ‘osxrelocator’ basicamente altera o caminho das libs no framework de modo que as mesmas sejam relativas ao executável dentro do .app, tornando o framework portável com o .app.

Reduzindo o tamanho do aplicativo

Nesse momento, o aplicativo possui um tamanho considerável, porém, tudo que for desnecessárias poderá ser removidas do pacote.

Por exemplo, se não fores utilizar o GStreamer, basta removê-lo de YourApp.app/Contents/Frameworks. Da mesma forma, podes remover a pasta de exem-

plos de dentro de /Applications/Kivy.app/Contents/Resources/kivy/examples/ ou kivy/tools, kivy/docs etc.

Desta forma, o pacote pode ser feito para incluir apenas as peças que são necessárias para o seu aplicativo.

Ajustando as Configurações

Os ícones e outras configurações de seu aplicativo podem ser alterados editando YourApp/Contents/info.plist para atender às suas necessidades.

Criando um DMG

Para fazer um DMG do seu aplicativo, use o seguinte comando:

```
osx> ./create-osx-dmg.sh YourApp.app
```

Observe a falta de / no final. Isto deve gerar um *.DMG comprimido que reduzirá ainda mais o tamanho do seu aplicativo para ser distribuído.

Usando o PyInstaller sem o Homebrew

Primeiro instale o Kivy e suas dependências sem usar o Homebrew como mencionado aqui: <http://kivy.org/docs/installation/installation.html#development-version>.

Depois de ter o Kivy e as suas dependências instaladas, precisarás instalar o PyInstaller.

Vamos supor que estamos usamos a pasta *testpackaging*:

```
cd testpackaging
git clone http://github.com/pyinstaller/pyinstaller
```

Crie um arquivo chamado *touchtracer.spec* neste diretório e adicione o código a ele:

```
# -*- mode: python -*-
block_cipher = None
from kivy.tools.packaging.pyinstaller_hooks import get_deps_all,
hookspath, runtime_hooks

a = Analysis(['/path/to/yout/folder/containing/examples/demo/
touchtracer/main.py'],
pathex=['/path/to/yout/folder/containing/testpackaging
'],
binaries=None,
win_no_prefer_redirects=False,
```

```

        win_private_assemblies=False,
        cipher=block_cipher,
        hookspath=hookspath(),
        runtime_hooks=runtime_hooks(),
        **get_deps_all())
pyz = PYZ(a.pure, a.zipped_data,
           cipher=block_cipher)
exe = EXE(pyz,
          a.scripts,
          exclude_binaries=True,
          name='touchtracer',
          debug=False,
          strip=False,
          upx=True,
          console=False )
coll = COLLECT(exe, Tree('../kivy/examples/demo/touchtracer/'),
               Tree('/Library/Frameworks/SDL2_ttf.framework/
→Versions/A/Frameworks/FreeType.framework'),
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=False,
               upx=True,
               name='touchtracer')
app = BUNDLE(coll,
              name='touchtracer.app',
              icon=None,
              bundle_identifier=None)

```

Altere os caminhos com seus caminhos relevantes:

```

a = Analysis(['/path/to/yout/folder/containing/examples/demo/
→touchtracer/main.py'],
            pathex=['/path/to/yout/folder/containing/testpackaging
→'],
...
...
coll = COLLECT(exe, Tree('../kivy/examples/demo/touchtracer/'),

```

Em seguida, execute o seguinte comando:

```
pyinstaller/pyinstaller.py touchtracer.spec
```

Substitua o *touchtracer* pelo nome do seu aplicativo, quando apropriado. Isso lhe dará um *<yourapp>.app* na pasta dist /.

Usando o PyInstaller e o Homebrew

Nota: Empacotando a sua aplicação para a versão mais antiga do OS X que desejas oferecer suporte.

Guia completo

1. Instalando o [Homebrew](#)
2. Instalando o Python:

```
$ brew install python
```

Nota: Para usar o Python 3, `brew install python3` e substitua `pip` por `pip3`' na guia abaixo.

3. (Re)instale suas dependências com `--build-bottle` para ter certeza de que podem ser usadas em outras máquinas:

```
$ brew reinstall --build-bottle sdl2 sdl2_image sdl2_ttf sdl2_
→mixer
```

Nota: Se o seu projeto depender do GStreamer ou de outras bibliotecas adicionais (re) instale-as com `--build-bottle` as described [below](#).

4. Instalando o Cython e o Kivy:

```
$ pip install -I Cython==0.23
$ USE OSX_FRAMEWORKS=0 pip install -U kivy
```

5. Instalando o PyInstaller:

```
$ pip install -U pyinstaller
```

6. Empacote seu aplicativo usando o Path para o seu `main.py`:

```
$ pyinstaller -y --clean --windowed --name touchtracer \
--exclude-module _tkinter \
--exclude-module Tkinter \
--exclude-module enchant \
--exclude-module twisted \
/usr/local/share/kivy-examples/demo/touchtracer/main.py
```

Nota: Isso ainda não copiará os arquivos adicionais de imagem ou som. Precisarás adaptar o arquivo `.spec` criado pra fazer isso.

Editando o arquivo *spec*

O arquivo specs é chamado `touchtracer.spec` e está localizado no diretório onde você executou o comando `pyinstaller`.

Precisas alterar a chamada `COLLECT()` para adicionar os dados de `touchtracer` (`touchtracer.kv`, `particle.png`, ...). Altere a linha para adicionar um objeto `Tree()`. Esta Árvore irá pesquisar e adicionar todos os arquivos encontrados no diretório `touchtracer` ao seu pacote final. Sua seção `COLLECT` deve ser algo como o que temos a seguir:

```
coll = COLLECT(exe, Tree('/usr/local/share/kivy-examples/demo/  
→touchtracer/'),  
                a.binaries,  
                a.zipfiles,  
                a.datas,  
                strip=None,  
                upx=True,  
                name='touchtracer')
```

Isso adicionará os (hooks) ganchos necessários para que o PyInstaller obtenha os arquivos Kivy necessários. Nós acabamos aqui! Sua especificação está pronta para ser executada.

Construa o *spec* e crie um DMG

1. Abra um console.
2. Vá para o diretório `PyInstaller` e crie o `spec`

```
$ pyinstaller -y --clean --windowed touchtracer.spec
```

3. Execute:

```
$ pushd dist  
$ hdiutil create ./Touchtracer.dmg -srcfolder touchtracer.app -  
→ov  
$ popd
```

4. Agora você terá um arquivo `Touchtracer.dmg` disponível no diretório `dist`.

Bibliotecas adicionais

GStreamer

Se seu projeto depender do GStreamer:

```
$ brew reinstall --build-bottle gstreamer gst-plugins-{base,good,  
bad,ugly}
```

Nota: Se o seu projeto precisar de suporte a Ogg Vorbis, adicione a opção `--with-libvorbis` ao comando acima.

Se estiveres usando o Python do Homebrew, também precisará do seguinte passo até que *este ‘pull request <<https://github.com/Homebrew/homebrew/pull/46097>>’* seja mesclado:

```
$ brew reinstall --with-python --build-bottle https://github.com/  
cbenhagen/homebrew/raw/patch-3/Library/Formula/gst-python.rb
```

2.11.8 Criando um pacote para iOS

Nota: Atualmente, os pacotes para iOS só podem ser gerados com o Python 2.7. O suporte a Python 3.3+ está a caminho.

O processo geral para criar um pacote para iOS pode ser explicado em 4 etapas:

1. Compile Python + módulos para iOS
2. Crie um projeto Xcode e vincule seu código-fonte
3. Personalize

Pré-requisitos

Precisas instalar algumas dependências, como o Cython, autotools, etc. Nós encorajamos você a usar o **Homebrew** para instalar essas dependências:

```
brew install autoconf automake libtool pkg-config  
brew link libtool  
sudo easy_install pip  
sudo pip install cython==0.23
```

Para obter mais detalhes, consulte [IOS Prerequisites](#). Antes de iniciar o segundo passo, certifique-se de que tudo está funcionando!

Compile a distribuição

Abra um terminal e digite:

```
$ git clone git://github.com/kivy/kivy-ios  
$ cd kivy-ios  
$ ./toolchain.py build kivy
```

A maioria das distribuições do Python é compactada como *python27.zip*. Se você tiver algum problema, consulte nosso [grupo de usuários](#) ou a [página do projeto kivy-ios](#).

Criando um projeto no Xcode

Antes de prosseguir para a próxima etapa, verifique se o ponto de entrada do aplicativo é um arquivo chamado *main.py*.

Fornecemos um Script que cria um novo projeto do Xcode. Na linha de comando abaixo, substitua *test* pelo nome do seu projeto. Precisa ser um nome sem espaços ou e sem caracteres ilegais:

```
$ ./toolchain.py create <title> <app_directory>  
$ ./toolchain.py create Touchtracer ~/code/kivy/examples/demo/  
↳ touchtracer
```

Nota: Deves usar um caminho totalmente qualificado para o diretório do aplicativo.

Um diretório chamado *<title>-ios* será criado, com um projeto Xcode contido dentro do mesmo. Poderás abrir o projeto no Xcode:

```
$ open touchtracer-ios/touchtracer.xcodeproj
```

Em seguida, clique em “Play” e divirta-se.

Nota: Todas as vezes que pressionares *Play*, o diretório do aplicativo será sincronizado com o diretório *<title>-ios/YourApp*. Não faça alterações diretamente no diretório *-ios*.

Atualizando um projeto Xcode

Digamos que desejas adicionar a biblioteca NumPy ao seu projeto, mas você não irá compila-lo antes de criar o seu projeto no XCode. Inicialmente, certifique-se de que o mesmo esteja construído:

```
$ ./toolchain.py build numpy
```

Em seguida, atualize seu projeto Xcode:

```
$ ./toolchain.py update touchtracer-ios
```

Todas as bibliotecas/frameworks necessários para executar toda e qualquer receitas compilada serão adicionadas ao seu projeto Xcode.

Personalize

There are various ways to customize and configure your app. Please refer to the [kivy-ios](#) documentation for more information.

Problemas conhecidos

All known issues with packaging for iOS are currently tracked on our [issues](#) page. If you encounter an issue specific to packaging for iOS that isn't listed there, please feel free to file a new issue, and we will get back to you on it.

While most are too technical to be written here, one important known issue is that removing some libraries (e.g. SDL_Mixer for audio) is currently not possible because the kivy project requires it. We will fix this and others in future versions.

FAQ

A aplicação encerrou anormalmente!

Por padrão, todas as instruções de impressão para o Console e os arquivos são ignorados. Se tiveres um problema ao executar seu aplicativo, poderás ativar o log comentando esta linha no arquivo *main.m*:

```
putenv("KIVY_NO_CONSOLELOG=1");
```

Então deves ver todo o registro do Kivy no console do Xcode.

Como a Apple pode aceitar um aplicativo do Python?

Conseguimos mesclar o binário do aplicativo com todas as bibliotecas em um único binário, chamado libpython. Isso significa que todos os módulos binários são carregados previamente, portanto nada é carregado dinamicamente.

Já enviastes um aplicativo Kivy para a App Store?

Sim, verifique:

- [Defletouch on iTunes](#),
- [ProcessCraft on iTunes](#)

Para uma lista mais completa, visite o Wiki do [Kivy](#) em.

2.11.9 Pré-requisitos para o iOS

O guia seguinte assume que:

- XCode 5.1 ou superior
- OS X 10.9 ou superior

Sua experiência pode variar com versões diferentes.

Começando

Para enviar qualquer aplicativo para a loja do iTunes, precisarás de uma ‘licença de desenvolvedor iOS <<https://developer.apple.com/programs/ios/>>’. Para testar, poderás usar um dispositivo físico ou o emulador XCode iOS.

Por favor, observe que, para testar no dispositivo, será necessário registrar antes esse dispositivo e instalar o seu “profile provisioning” no mesmo. Consulte o Guia de Introdução da Apple <https://developer.apple.com/programs/ios/gettingstarted/> para obter mais informações.

Homebrew

Usamos o gerenciador de pacotes [Homebrew](#) no OSX para instalar algumas das dependências e ferramentas usadas pelo Kivy. É uma ferramenta realmente útil e é um projeto Open Source hospedado no [Github](#).

Devido à natureza do gerenciamento de pacotes (complicações com versões e sistemas operacionais), esse processo pode ser propenso a erros e causar falhas no processo de compilação. A mensagem **Missing requirement: <pkg> is not installed!** é um erro clássico.

A primeira coisa é garantir que tenhas executado os seguintes comandos

```
brew install autoconf automake libtool pkg-config mercurial  
brew link libtool  
brew link mercurial  
sudo easy_install pip  
sudo pip install cython
```

Se ainda receberes erros de compilação, verifique se o seu Homebrew está em um estado saudável:

```
brew doctor
```

Para obter mais ajuda, consulte o [Homebrew wiki](#).

O último, final e desesperado passo para que as coisas funcionem pode ser remover o Homebrew completamente, obter a versão mais recente, instala-la novamente e, em seguida, reinstalar as dependências.

[Como desinstalar e remover Homebrew para Mac OSX](#)

2.12 Licença do Empacotamento

Aviso: This is not a legally authoritative guide! The Kivy organisation, authors and contributors take no responsibility for any lack of knowledge, information or advice presented here. The guide is merely informative and is meant to protect inexperienced users.

Your code alone may not require including licensing information or copyright notices of other included software, but binaries are something else. When a binary (.exe, .app, .apk, ...) is created, it includes Kivy, its dependencies and other packages that your application uses.

Some of them are licensed in a way that requires including a copyright notice somewhere in your app (or more). Before distributing any of the binaries, please **check all the created files** that don't belong to your source (.dll, .pyd, .so, ...) and include the appropriate copyright notices if required by the license the files belong to. This way you may satisfy licensing requirements of the Kivy deps.

2.12.1 Dependências

All of the dependencies will be used at least partially on each platform Kivy supports. You therefore need to comply to their licenses, which mostly requires only pasting a copyright notice in your app and not pretending you wrote the code.

- [docutils](#)
- [pygments](#)
- [sdl2](#)
- [glew](#)
- [gstreamer](#) (if used)
- Bibliotecas de Image & Áudio (por exemplo [SDL_mixer has them](#))

You'll probably need to check image and audio libraries manually (most begin with `lib`). The `LICENSE*` files that belong to them should be included by PyInstaller, but are not included by python-for-android and you need to find them.

2.12.2 Windows (PyInstaller)

To access some Windows API features, Kivy uses the [pypiwin32](#) package. This package is released under the [PSF license](#).

Visual Studio Redistributables

Python compiled with Visual Studio (official) includes files from Microsoft and you are only allowed to redistribute them under specific conditions listed in the [CRTlicense](#). You need to include the names of the files and a reworded version of [Py2 CRT license](#) or [Py3 CRT license](#) (depending which interpreter you use) and present these to the end-user of your application in order to satisfy their requirements.

- [List of redistributables](#)

Outras Bibliotecas

- [zlib](#)

Nota: Please add the attributions for other libraries that you *don't use directly* but are present after packaging with e.g. PyInstaller on Windows.

2.12.3 Linux

Linux has many distributions which means there's no correct guide for all of the distributions. This applies to the RPi too. However, it can be simplified in two ways depending on how you create a package (also with PyInstaller): with or without including binaries.

If the binaries are included, you should check every file (e.g. `.so`) that's not your source and find the license it belongs to. According to that license, you'll probably need to put an attribution into your application or possibly more, depending on the requirements of that license.

If the binaries are not included (which allows packaging your app as e.g. a `.deb` package), there's a *situation bad for your user*. It's up to you to decide whether you satisfy the conditions of other licenses and, for example, include copyright attributions into your app or not.

2.12.4 Android

As APK is just an archive of files: you can extract files from it and (as in Windows redistributables) check all the files.

`APK/assets/private.mp3/private.mp3/` contém todos os arquivos incluídos. A maioria deles está relacionada com o Kivy, Python ou a sua fonte, mas aqueles que não são necessidade de verificação.

Pacotes conhecidos:

- `pygame` (Se o `old_toolchain` for usado)
- `sqlite3`
- `six`

There are other included libraries, included either by Kivy directly or through Pygame/SDL2, that are located in `APK/lib/armv7a/`. Most of them are related to dependencies or are produced by `python-for-android` and are part of its source (and licensing).

- `libapplication.so`

2.12.5 Mac

Perdido.

2.12.6 iOS

Perdido.

2.12.7 Evitando binários

There might be a way how to avoid this licensing process by avoiding creating a distribution with third-party stuff completely. With Python you can create a module, which is only your code with `__main__.py + setup.py` that only lists required dependencies.

This way, you can still distribute your app - your *code* - and you might not need to care about other licenses. The combination of your code and the dependencies could be specified as a “usage” rather than a “distribution”. The responsibility of satisfying licenses, however, most likely transfers to your user, who needs to assemble the environment to even run the module. If you care about your users, you might want to slow down a little and read more about the [consequences](#).

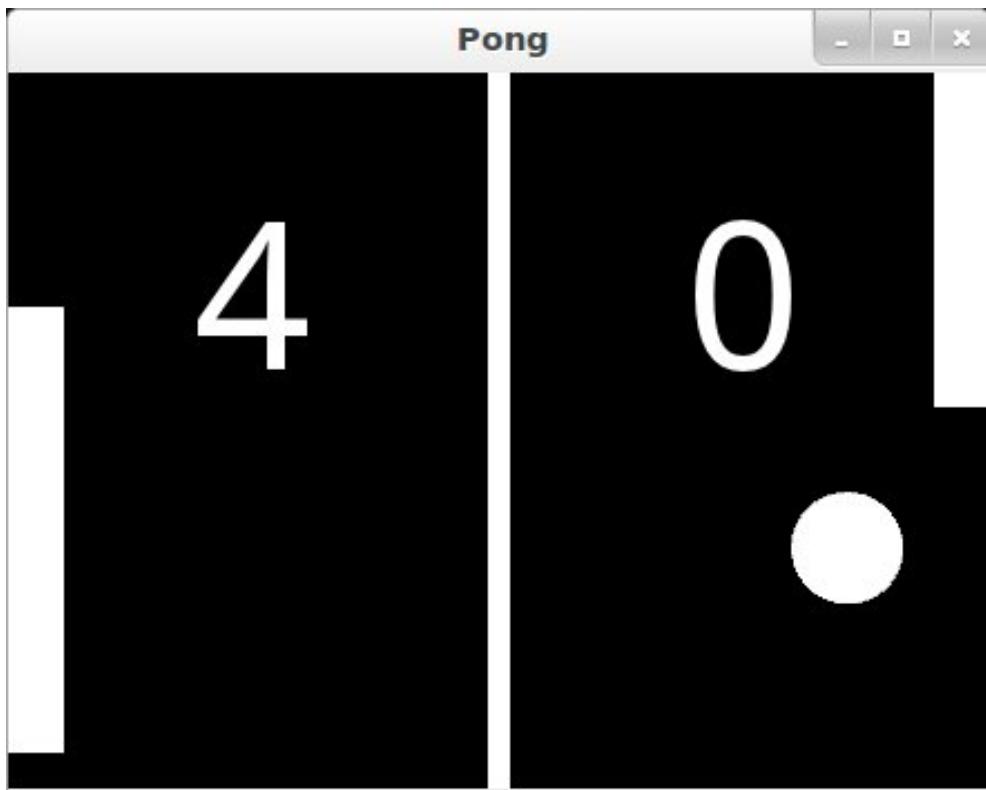
Tutoriais

3.1 Tutorial do Pong Game

3.1.1 Prefácio

Seja bem-vindo ao tutorial Pong

Este tutorial irá ensiná-lo a escrever um jogo de pong usando Kivy. Vamos começar com um aplicativo básico como o descrito em: ref: *quickstart* e transformá-lo em um jogo de pong jogável, descrevendo cada passo ao longo do caminho.



Aqui está uma lista de verificação antes de iniciar este tutorial:

- Você tem uma instalação Kivy em funcionamento. Veja a seção: [doc:/ installation/installation](#) para descrições detalhadas.
- Você sabe como executar uma aplicação básica Kivy. Veja: [ref: quickstart](#) se você não souber.

Se você leu o guia de programação e comprehende os conceitos básicos do Widget ([Um aplicativo de pintura simples](#)) e os conceitos básicos da linguagem kv ([Kv language](#)), você provavelmente pode ignorar os 2 primeiros passos e seguir direto para o passo 3.

Nota: Você pode encontrar todo o código-fonte e arquivos de código-fonte para cada etapa no diretório de exemplos do Kivy em [tutorials/pong/](#)

Pronto? Então vamos começar!

3.1.2 Começando

Começando

Vamos começar tendo um aplicativo Kivy realmente simples instalado e funcionando. Crie um diretório para o jogo e um arquivo chamado `*main.py *`

```
1 from kivy.app import App  
2 from kivy.uix.widget import Widget
```

```

3
4
5 class PongGame(Widget):
6     pass
7
8
9 class PongApp(App):
10    def build(self):
11        return PongGame()
12
13
14 if __name__ == '__main__':
15     PongApp().run()

```

Vá em frente e execute o aplicativo. Deverá apenas mostrar uma janela preta neste ponto. O que fizemos é criar uma classe Kivy *App* muito simples, que cria uma instância da nossa classe de Widget *PongGame* e a devolve como o elemento raiz da UI das aplicações, e que você deve imaginar neste momento, como que é uma árvore hierárquica de Widgets. O Kivy coloca esta árvore de Widgets na Janela padrão. Na próxima etapa, vamos desenhar o fundo do Pong e a pontuações, definindo assim o Widget *PongGame* *widget*.

3.1.3 Adicionando Simples Gráficos

Criação do *pong.kv*

Usaremos um arquivo .kv para definir a aparência da classe *PongGame*. Uma vez que a classe *App* é chamada *PongApp*, podemos simplesmente criar um arquivo chamado *pong.kv* no mesmo diretório que será carregado automaticamente quando o aplicativo for executado. Então crie um novo arquivo chamado “*pong.kv*” e adicione o seguinte conteúdo.

```

1 #:kivy 1.0.9
2
3 <PongGame>:
4     canvas:
5         Rectangle:
6             pos: self.center_x - 5, 0
7             size: 10, self.height
8
9     Label:
10        font_size: 70
11        center_x: root.width / 4
12        top: root.top - 50
13        text: "0"
14
15     Label:

```

```
16     font_size: 70
17     center_x: root.width * 3 / 4
18     top: root.top - 50
19     text: "0"
```

Nota: ERRO COMUM: O nome do arquivo kv, por exemplo, pong.kv, deve combinar com o nome do aplicativo, por exemplo, PongApp (a escrita que vem antes de App - no arquivo python).

Se você executar o aplicativo agora, você deverá ver uma barra vertical no meio e dois zeros onde as pontuações do jogador serão exibidas.

Explicando a Sintaxe do Arquivo Kv

Antes de passar para a próxima etapa, você pode querer dar uma olhada no conteúdo do arquivo kv que acabamos de criar e descobrir o que está acontecendo. Se você entender o que está acontecendo, você provavelmente pode pular para o próximo passo.

Na primeira linha temos:

```
#:kivy 1.0.9
```

Esta primeira linha é necessária em cada arquivo kv. Ele deve começar com `#:kivy` seguido de um espaço e da versão Kivy mínima exigida (para que Kivy possa ter certeza de ter pelo menos a versão necessária ou lidar com a compatibilidade com versões anteriores mais tarde).

Depois disso, começamos a definir regras que são aplicadas a todas as instâncias PongGame:

```
<PongGame>:
```

```
    ...
```

Como o Python, os arquivos kv usam indentação para definir blocos aninhados. Um bloco definido com um nome de classe dentro dos caracteres `< e >` é uma regra **Widget**. Ele será aplicado a qualquer instância da classe nomeada. Se você substituiu `PongGame` por `Widget` no nosso exemplo, todas as instâncias do `Widget` teriam a linha vertical e os dois Widgets `Label` dentro deles, isso porque o mesmo definiria essas regras para todas as instâncias do `Widget`.

Dentro de uma seção de regra, você pode adicionar vários blocos para definir o estilo e o conteúdo dos widgets aos quais serão aplicados. Você pode:

- Definir valores de propriedade,
- adicionar widget filho

- define uma seção *canvas* na qual poderás adicionar instruções Graphics que definem como o Widget será renderizado.

O primeiro bloco que temos dentro da regra <PongGame> é um bloco *canvas*:

```
<PongGame>:
    canvas:
        Rectangle:
            pos: self.center_x - 5, 0
            size: 10, self.height
```

Portanto, este bloco de Canvas diz que o Widget **PongGame** deverá desenhar alguns gráficos primitivos. Nesse caso, adicionamos um retângulo à tela. Definimos a *pos* do retângulo como sendo igual 5 pixels à esquerda do centro horizontal do Widget e 0 para y. O tamanho do retângulo é definido como sendo 10 pixels de largura e a altura do Widget em altura. A coisa agradável sobre como definir gráficos como este, é que o retângulo processado será atualizado automaticamente quando as propriedades de quaisquer Widgets que for usado na expressão de valor sofrer alguma alteração.

Nota: Tente redimensionar a janela do aplicativo e observe o que acontece. É isso mesmo, a interface de usuário inteira é redimensionada automaticamente. O comportamento padrão da janela é redimensionar um elemento com base em sua propriedade *size_hint*. O Widget de tamanho padrão *size_hint* é (1,1), significando que ele será esticado 100% tanto na direção *x* como na direção *y* e, portanto, preencherá o espaço disponível. Uma vez que a *pos* e o tamanho do retângulo e *center_x* e o topo dos marcadores de pontuação foram definidos no contexto da classe **PongGame**, estas propriedades atualizarão automaticamente quando as propriedades dos Widget correspondentes forem alteradas. Usando a linguagem kv terás a vinculação de propriedade automática.

As duas últimas seções que adicionamos parecem bastante semelhantes. Cada uma delas adiciona um Widget **Label** como um Widget filho ao Widget **PongGame**. Por enquanto, o texto em ambos é apenas definido como “0”. Nós vamos ligar isso até a pontuação real, uma vez que temos a lógica implementada, mas os Labels já ficaram melhores ao definirmos um *font_size* maior, e posicioná-los relativamente ao Widget raiz. A palavra-chave **root** pode ser usada dentro do bloco filho para se referir novamente ao Widget pai/raiz que a regra se aplica a (**PongGame** neste caso):

```
<PongGame>:
    # ...

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: "0"
```

```

Label:
    font_size: 70
    center_x: root.width * 3 / 4
    top: root.top - 50
    text: "0"

```

3.1.4 Adicionar Bola

Adicionar Bola

Ok, então temos uma arena básica de Pongue para jogar, mas ainda precisamos dos jogadores e uma bola para bater de um lado ao outro. Vamos começar com a bola. Vamos adicionar uma nova classe *PongBall* para criar um Widget que será a nossa bola e fazê-lo saltar ao redor do tabuleiro.

Classe BolaPong

Aqui está o código Python para a classe *PongBall*:

```

1 class PongBall(Widget):
2
3     # velocity of the ball on x and y axis
4     velocity_x = NumericProperty(0)
5     velocity_y = NumericProperty(0)
6
7     # referencelist property so we can use ball.velocity as
8     # a shorthand, just like e.g. w.pos for w.x and w.y
9     velocity = ReferenceListProperty(velocity_x, velocity_y)
10
11    # ``move`` function will move the ball one step. This
12    # will be called in equal intervals to animate the ball
13    def move(self):
14        self.pos = Vector(*self.velocity) + self.pos

```

E aqui está a regra kv usada para desenhar a bola como um círculo branco:

```

<PongBall>:
    size: 50, 50
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size

```

Para que tudo funcione, também teremos que adicionar as importações para as classes de propriedade *Propriedades* e a *Vector*.

Aqui está todo o código Python atualizado e o arquivo kv para desenvolvido nesta etapa:

main.py:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.properties import NumericProperty,
4     ReferenceListProperty
5 from kivy.vector import Vector
6
7 class PongBall(Widget):
8     velocity_x = NumericProperty(0)
9     velocity_y = NumericProperty(0)
10    velocity = ReferenceListProperty(velocity_x, velocity_y)
11
12    def move(self):
13        self.pos = Vector(*self.velocity) + self.pos
14
15
16 class PongGame(Widget):
17     pass
18
19
20 class PongApp(App):
21     def build(self):
22         return PongGame()
23
24
25 if __name__ == '__main__':
26     PongApp().run()
```

pong.kv:

```
1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10    <PongGame>:
11        canvas:
12            Rectangle:
13                pos: self.center_x-5, 0
14                size: 10, self.height
```

```

15
16     Label:
17         font_size: 70
18         center_x: root.width / 4
19         top: root.top - 50
20         text: "0"
21
22     Label:
23         font_size: 70
24         center_x: root.width * 3 / 4
25         top: root.top - 50
26         text: "0"
27
28     PongBall:
29         center: self.parent.center
30

```

Note que não só uma regra de Widget `<PongBall>` foi adicionada, mas também um Widget filho `PongBall` na regra de Widget `<PongGame>`.

3.1.5 Adicionando Animação para a Bola

Mover a Bola

Legal, então agora temos uma bola, e ela ainda tem uma função `move...` mas ainda não está se movendo. Vamos corrigir isso.

Agendando função para o Clock

Precisamos do método `move` da nossa bola para que esse seja chamado regularmente. Felizmente, o Kivy torna isso muito fácil, deixando-nos programar qualquer função que queremos usando o `Clock` e especificando o intervalo:

```
Clock.schedule_interval(game.update, 1.0/60.0)
```

Essa linha, por exemplo, faria com que a função “`update`” do objeto de jogo fosse chamada uma vez a cada 60 de segundo (60 vezes por segundo).

Propriedade/Referência do Objetos

Temos outro problema. Gostaríamos de certificar-nos de que o `PongBall` tem a sua função `move` invocada regularmente, mas no nosso código não temos nenhuma referência ao objeto bola, já que acabamos de adicioná-la através do arquivo `kv` dentro da regra `kv` para o classe `PongGame`. A única referência ao nosso jogo é a que retornamos no método de compilação de aplicativos.

Uma vez que vamos ter que fazer algo mais do que apenas mover a bola (por exemplo, saltar para fora das paredes e mais tarde a raquete de jogadores), provavelmente vamos precisar de um método `update` para a nossa classe `PongGame`. Além disso, uma vez que já temos uma referência ao objeto do jogo, podemos facilmente agendar seu novo método `update` quando o aplicativo for construído:

```
1 class PongGame(Widget):
2
3     def update(self, dt):
4         # call ball.move and other stuff
5         pass
6
7 class PongApp(App):
8
9     def build(self):
10        game = PongGame()
11        Clock.schedule_interval(game.update, 1.0/60.0)
12        return game
```

No entanto, isso ainda não altera o fato de que não temos uma referência ao Widget filho `PongBall` criado pela regra kv. Para corrigir isso, podemos adicionar uma `ObjectProperty` à classe `PongGame` e conectá-la ao Widget criado na regra kv. Uma vez que isso é feito, podemos facilmente referenciar a propriedade `bola` dentro do método `update` e até mesmo fazê-la saltar fora das bordas:

```
1 class PongGame(Widget):
2     ball = ObjectProperty(None)
3
4     def update(self, dt):
5         self.ball.move()
6
7         # bounce off top and bottom
8         if (self.ball.y < 0) or (self.ball.top > self.height):
9             self.ball.velocity_y *= -1
10
11         # bounce off left and right
12         if (self.ball.x < 0) or (self.ball.right > self.width):
13             self.ball.velocity_x *= -1
```

Não se esqueça de conectar-lo no arquivo kv, dando ao Widget filho um id e definindo `ObjectProperty bola` do `PongGame` para esse id:

```
<PongGame>:
    ball: pong_ball

    # ... (canvas and Labels)

PongBall:
```

```
    id: pong_ball  
    center: self.parent.center
```

Nota: Neste ponto tudo está ligado para a bola para saltar ao redor. Se estás codificando junto como conosco, poderás querer saber porque a esfera não está se movendo em qualquer lugar. A velocidade da bola é ajustada para 0 em ambos x e y. Na listagem de código abaixo, um método `serve_ball` é adicionado à classe `PongGame` e chamado no método `build` do aplicativo. Ele define uma velocidade aleatória x e y para a bola, e também redefine a posição, para que possamos usá-la mais tarde para redefinir a bola quando um jogador marcou um ponto.

Aqui está o código inteiro desta etapa:

`main.py`:

```
1  from kivy.app import App  
2  from kivy.uix.widget import Widget  
3  from kivy.properties import NumericProperty, _  
   ReferenceListProperty,\n   ObjectProperty  
4  from kivy.vector import Vector  
5  from kivy.clock import Clock  
6  from random import randint  
7  
8  
9  
10 class PongBall(Widget):  
11     velocity_x = NumericProperty(0)  
12     velocity_y = NumericProperty(0)  
13     velocity = ReferenceListProperty(velocity_x, velocity_y)  
14  
15     def move(self):  
16         self.pos = Vector(*self.velocity) + self.pos  
17  
18  
19 class PongGame(Widget):  
20     ball = ObjectProperty(None)  
21  
22     def serve_ball(self):  
23         self.ball.center = self.center  
24         self.ball.velocity = Vector(4, 0).rotate(randint(0, _  
   360))  
25  
26     def update(self, dt):  
27         self.ball.move()  
28  
29         # bounce off top and bottom
```

```

30     if (self.ball.y < 0) or (self.ball.top > self.height):
31         self.ball.velocity_y *= -1
32
33     # bounce off left and right
34     if (self.ball.x < 0) or (self.ball.right > self.width):
35         self.ball.velocity_x *= -1
36
37
38 class PongApp(App):
39     def build(self):
40         game = PongGame()
41         game.serve_ball()
42         Clock.schedule_interval(game.update, 1.0 / 60.0)
43         return game
44
45
46 if __name__ == '__main__':
47     PongApp().run()

```

pong.kv:

```

1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10 <PongGame>:
11     ball: pong_ball
12
13     canvas:
14         Rectangle:
15             pos: self.center_x-5, 0
16             size: 10, self.height
17
18     Label:
19         font_size: 70
20         center_x: root.width / 4
21         top: root.top - 50
22         text: "0"
23
24     Label:
25         font_size: 70
26         center_x: root.width * 3 / 4
27         top: root.top - 50

```

```

28     text: "0"
29
30 PongBall:
31     id: pong_ball
32     center: self.parent.center
33

```

3.1.6 Conectando Eventos de Entrada

Adicionando jogadores e reagindo a entrada de toques

Doce, nossa bola está saltando ao redor. As únicas coisas que faltam agora são as raquetes de jogadores móveis e manter o controle da pontuação. Não iremos examinar todos os detalhes da criação das regras de classe e kv novamente, uma vez que esses conceitos já foram abordados nas etapas anteriores. Em vez disso, vamos nos concentrar em como mover os Widgets do Player em resposta à entrada do usuário. Poderás obter o código inteiro e as regras kv para a classe `PongPaddle` no final desta seção.

Em Kivy, um Widget pode reagir à entrada implementando os métodos `on_touch_down`, o `on_touch_move` e o `on_touch_up`. Por padrão, a classe Widget implementa esses métodos chamando apenas o método correspondente em todos os seus Widgets filho para transmitir o evento até que um dos filhos retorne True.

Pong é muito simples. As raquetes só precisam se mover para cima e para baixo. Na verdade, é tão simples, que nem sequer precisa realmente ter os Widgets jogador para lidar com os próprios eventos. Vamos apenas implementar a função `on_touch_move` para a classe `PongGame` e definir a posição do jogador esquerdo ou direito com base no fato de o toque ter ocorrido no lado esquerdo ou direito da tela.

Verifique o manipulador `on_touch_move`:

```

1 def on_touch_move(self, touch):
2     if touch.x < self.width/3:
3         self.player1.center_y = touch.y
4     if touch.x > self.width - self.width/3:
5         self.player2.center_y = touch.y

```

Vamos manter a pontuação para cada jogador em uma `NumericProperty`. Os marcadores de pontuação do `PongGame` são mantidos atualizados alterando o `score` de `NumericProperty`, que por sua vez atualiza a propriedade de texto de Labels filho de `PongGame`. Essa vinculação ocorre porque o Kivy `properties` se liga automaticamente a qualquer referência em seus arquivos kv correspondentes. Quando a bola escapa para fora dos lados, vamos atualizar a pontuação e servir a bola novamente, alterando o método `update` na classe `PongGame`. A classe `PongPaddle` também implementa um método `bounce_ball`, para que a bola rejeite de forma diferente com base no local onde atinge a raquete. Aqui está o código da classe `PongPaddle`:

```

1 class PongPaddle(Widget):
2
3     score = NumericProperty(0)
4
5     def bounce_ball(self, ball):
6         if self.collide_widget(ball):
7             speedup = 1.1
8             offset = 0.02 * Vector(0, ball.center_y - self.center_y)
9             ball.velocity = speedup * (offset - ball.velocity)

```

Nota: Este algoritmo para a bola que salta é muito simples, mas terá o comportamento estranho se a bola bater na pá/raquete do lado ou na parte inferior...isto é algo que você poderia tentar reparar se você quiser.

E aqui está no contexto. Muito bem feito:

main.py:

```

1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.properties import NumericProperty,_
4     ReferenceListProperty,\n    ObjectProperty
5 from kivy.vector import Vector
6 from kivy.clock import Clock
7
8
9 class PongPaddle(Widget):
10     score = NumericProperty(0)
11
12     def bounce_ball(self, ball):
13         if self.collide_widget(ball):
14             vx, vy = ball.velocity
15             offset = (ball.center_y - self.center_y) / (self.
16                 height / 2)
17             bounced = Vector(-1 * vx, vy)
18             vel = bounced * 1.1
19             ball.velocity = vel.x, vel.y + offset
20
21
22 class PongBall(Widget):
23     velocity_x = NumericProperty(0)
24     velocity_y = NumericProperty(0)
25     velocity = ReferenceListProperty(velocity_x, velocity_y)
26
27     def move(self):
28         self.pos = Vector(*self.velocity) + self.pos

```

```

28
29
30 class PongGame(Widget):
31     ball = ObjectProperty(None)
32     player1 = ObjectProperty(None)
33     player2 = ObjectProperty(None)
34
35     def serve_ball(self, vel=(4, 0)):
36         self.ball.center = self.center
37         self.ball.velocity = vel
38
39     def update(self, dt):
40         self.ball.move()
41
42         # bounce of paddles
43         self.player1.bounce_ball(self.ball)
44         self.player2.bounce_ball(self.ball)
45
46         # bounce ball off bottom or top
47         if (self.ball.y < self.y) or (self.ball.top > self.top):
48             self.ball.velocity_y *= -1
49
50         # went of to a side to score point?
51         if self.ball.x < self.x:
52             self.player2.score += 1
53             self.serve_ball(vel=(4, 0))
54         if self.ball.x > self.width:
55             self.player1.score += 1
56             self.serve_ball(vel=(-4, 0))
57
58     def on_touch_move(self, touch):
59         if touch.x < self.width / 3:
60             self.player1.center_y = touch.y
61         if touch.x > self.width - self.width / 3:
62             self.player2.center_y = touch.y
63
64
65 class PongApp(App):
66     def build(self):
67         game = PongGame()
68         game.serve_ball()
69         Clock.schedule_interval(game.update, 1.0 / 60.0)
70         return game
71
72
73 if __name__ == '__main__':
74     PongApp().run()

```

pong.kv:

```
1 #:kivy 1.0.9
2
3 <PongBall>:
4     size: 50, 50
5     canvas:
6         Ellipse:
7             pos: self.pos
8             size: self.size
9
10 <PongPaddle>:
11     size: 25, 200
12     canvas:
13         Rectangle:
14             pos: self.pos
15             size: self.size
16
17 <PongGame>:
18     ball: pong_ball
19     player1: player_left
20     player2: player_right
21
22     canvas:
23         Rectangle:
24             pos: self.center_x-5, 0
25             size: 10, self.height
26
27     Label:
28         font_size: 70
29         center_x: root.width / 4
30         top: root.top - 50
31         text: str(root.player1.score)
32
33     Label:
34         font_size: 70
35         center_x: root.width * 3 / 4
36         top: root.top - 50
37         text: str(root.player2.score)
38
39     PongBall:
40         id: pong_ball
41         center: self.parent.center
42
43     PongPaddle:
44         id: player_left
45         x: root.x
46         center_y: root.center_y
```

```
48 PongPaddle:  
49     id: player_right  
50     x: root.width-self.width  
51     center_y: root.center_y  
52
```

3.1.7 Para onde ir agora?

Divirta-se

Bem, o jogo pong está considerado completo. Se você entendeu todas as coisas que são abordadas neste tutorial, dê um tapinha nas costas e pense em como você poderia melhorar o jogo. Aqui estão algumas ideias de coisas que você poderia fazer:

- Adicione alguns gráficos/imagens mais agradáveis. (Dica: verifique a propriedade **source** nas instruções gráficas como **circle** ou **Rectangle**, para definir uma imagem como a textura.)
- Faça o jogo terminar depois de uma certa pontuação. Talvez uma vez que um jogador tenha 10 pontos, poderás exibir um Label grande “PLAYER 1 WINS” e/ou adicionar um menu principal para iniciar, pausar e redefinir o jogo. (Sugestão: confira as class:~*kivy.uix.button.Button* e **Label** e descubra como usar as suas funções *add_widget* e *remove_widget* para adicionar ou remover Widgets dinamicamente.
- Torne o jogo Pong para 4 jogadores. A maioria dos tablets tem suporte Multi-Touch, então não seria legal ter um jogador de cada lado e ter quatro pessoas jogando ao mesmo tempo?
- Corrigir a verificação de colisão simplista para bater a bola com uma extremitade da pá/raquete resulta em um salto mais realista.

Nota: Você pode encontrar todo o código fonte e arquivos de código-fonte para cada etapa no diretório de exemplos Kivy em tutorials/pong/

3.2 Um aplicativo de pintura simples

No tutorial a seguir, você será guiado através da criação de seu primeiro widget. Isso fornece um poderoso e importante conhecimento ao programar aplicativos Kivy, pois permite criar interfaces de usuário completamente novas com elementos personalizados para sua finalidade específica.

3.2.1 Considerações básicas

Quando estiveres criando um aplicativo, você deve se fazer três perguntas importantes:

- Quais são os dados que o meu aplicativo processa?
- Como eu represento esses dados visualmente?
- Como o usuário interage com esses dados?

Se você quiser escrever um aplicativo de desenho de linha muito simples, por exemplo, você provavelmente quer que o usuário apenas desenhe na tela com seus dedos. É assim que o usuário interage com seu aplicativo. Ao fazê-lo, sua aplicação memorizaria as posições onde o dedo do usuário estava, de modo que você possa mais tarde traçar linhas entre essas posições. Assim, os pontos onde os dedos estavam serão seus dados e as linhas que você desenhar entre eles seria sua representação visual.

No Kivy, a interface do usuário de uma aplicação é composta de Widgets. Tudo que você vê na tela é de alguma forma desenhado por um widget. Muitas vezes você gostaria ser capaz de reutilizar o código que você já escreveu em um contexto diferente, razão pela qual widgets normalmente representam uma instância específica que respondem às três questões acima. Um widget encapsula dados, define a interação do usuário com esses dados e desenha sua representação visual. Você pode construir qualquer coisa, desde interfaces do usuário simples a complexas, aninhando widgets. Existem muitos widgets incorporados, como botões, controles deslizantes e outras coisas comuns. Em muitos casos, contudo, você necessita de um widget personalizado que esteja além do escopo do que é fornecido com Kivy (por exemplo, um widget de visualização médica).

Portanto, mantenha essas três perguntas em mente quando você projetar/desenhar seus widgets. Tente escrevê-los de forma mínima e reutilizável (ou seja, um widget faz exatamente o que é suposto fazer e nada mais. Se você precisar de mais, escrever mais widgets ou compor outros widgets de widgets menores. Tentamos aderir ao [Princípio de Responsabilidade Única](#)).

3.2.2 Widget Paint

Temos certeza de que um de seus sonhos de infância sempre foi criar o seu próprio programa de pintura multitouch. Permita-nos ajudá-lo a conseguir isso. Nas seções a seguir você aprenderá sucessivamente como escrever um programa como este usando o Kivy. Certifique-se de ter lido e compreendido: ref: *quickstart*. Você já fez isso? Ótimo! Vamos começar!

Estrutura Inicial

Vamos começar escrevendo a estrutura de código muito básica que precisamos. A propósito, todas as diferentes partes do código que são usadas nesta seção também es-

tão disponíveis no diretório `examples/guide/firstwidget` que vem com o Kivy, então você não precisa copiar e colar o tempo todo. Aqui está o esqueleto do código básico que iremos precisar:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3
4
5 class MyPaintWidget(Widget):
6     pass
7
8
9 class MyPaintApp(App):
10     def build(self):
11         return MyPaintWidget()
12
13
14 if __name__ == '__main__':
15     MyPaintApp().run()
```

Isso é realmente muito simples. Salve-o como `paint.py`. Se você executá-lo, você só deve ver uma tela preta. Como você pode ver, em vez de usar um widget embutido, como um Button (veja: ref: *quickstart*), vamos escrever nosso próprio widget para fazer o desenho. Fazemos isso criando uma classe que herda da classe: `~kivy.uix.widget.Widget` (linha 5-6) e embora essa classe ainda não faça nada, ainda podemos tratá-la como um widget Kivy normal (linha 11). A construção `if __name__ == '__main__'` (linha 14) é um mecanismo Python que impede que você execute o código na instrução `if` ao realizar a importação do arquivo, ou seja, se você escrever `import paint`, ele não vai fazer algo inesperado, mas apenas fornecer bem as classes definidas no arquivo.

Nota: Podes estar se perguntando por que tens que importar o App e o Widget separadamente, em vez de fazer algo como `from kivy import *`. Embora mais curto e prático, isso teria a desvantagem de **poluir seu namespace** e aumentaria o tempo da inicialização do aplicativo, pois mais simbolos precisariam ser importados o que aumentaria o tempo de carregamento. Ele também pode introduzir ambiguidade na classe e na nomenclatura de variável, por isso, essa é uma prática geralmente desaprovado pela comunidade Python. A forma como o fazemos é mais rápida e limpa.

Adicionando Comportamentos

Vamos agora adicionar algum comportamento ao Widget, ou seja, fazê-lo reagir à entrada do usuário. Altere o código da seguinte forma:

```

1  from kivy.app import App
2  from kivy.uix.widget import Widget
3
4
5  class MyPaintWidget(Widget):
6      def on_touch_down(self, touch):
7          print(touch)
8
9
10 class MyPaintApp(App):
11     def build(self):
12         return MyPaintWidget()
13
14
15 if __name__ == '__main__':
16     MyPaintApp().run()

```

Isso é apenas para mostrar como é fácil reagir à entrada do usuário. Quando a classe: `~kivy.input.motionevent.MotionEvent` (por exemplo, um toque, clique, etc.) ocorre, nós simplesmente imprimimos as informações sobre o objeto de toque para o console. Você não verá nada na tela, mas se observar a linha de comando a partir da qual você está executando o programa, você verá uma mensagem para cada toque. Isso também demonstra que um widget não precisa ter uma representação visual.

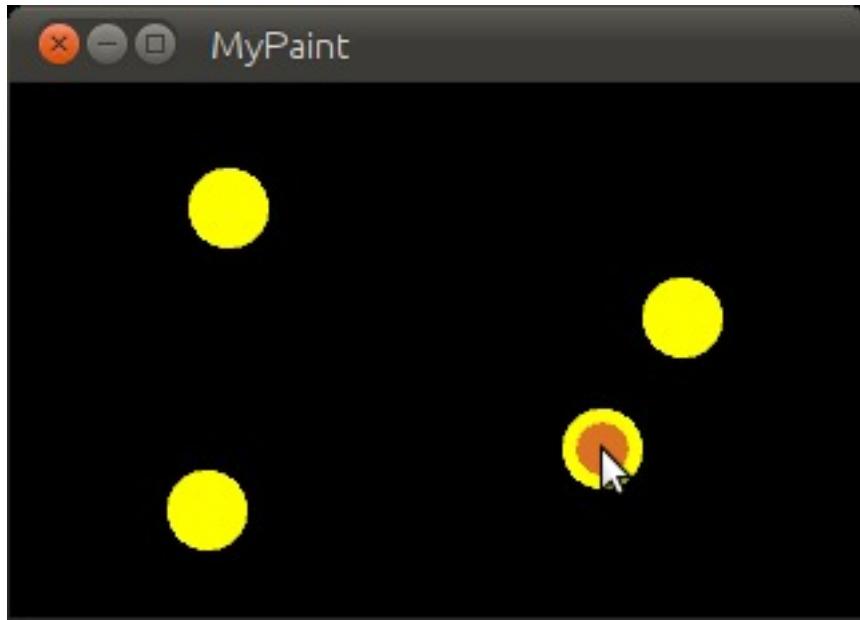
Agora que realmente não é uma experiência de usuário esmagadora. Vamos adicionar algum código que realmente desenhe algo em nossa janela:

```

1  from kivy.app import App
2  from kivy.uix.widget import Widget
3  from kivy.graphics import Color, Ellipse
4
5
6  class MyPaintWidget(Widget):
7
8      def on_touch_down(self, touch):
9          with self.canvas:
10              Color(1, 1, 0)
11              d = 30.
12              Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d,
13                ↳ d))
14
15 class MyPaintApp(App):
16
17     def build(self):
18         return MyPaintWidget()
19
20

```

```
21 if __name__ == '__main__':
22     MyPaintApp().run()
```



Se você executar seu código com estas modificações, você verá que cada vez que você tocar, haverá um pequeno círculo amarelo desenhado onde você tocou. Como funciona?

- Linha 9: Usamos a instrução `with` do Python com o objeto:`class:kivy.graphics.instructions.Canvas`. Isto é como uma área em que o Widget utiliza para fazer a sua representação gráfica. Usando a instrução `with`, todos os comandos de desenho sucessivos que estão devidamente indentados modificarão o Canvas. A instrução `with` também garante que, após o desenho, o estado interno possa ser limpo corretamente.
- Linha 10: Você já deve ter adivinhado: Isso define a: `classe:~kivy.graphics.context_instructions.Color` para operações de desenho sucessivas a amarelo (o formato de cor padrão é RGB, então $(1, 1, 0)$ é amarelo). Isso é verdade até que outra: `classe: ~kivy.graphics.context_instructions.Color` esteja definida. Pense nisso como mergulhar seus pincéis nessa cor, na qual você pode usar para desenhar em uma tela até que você mergulhe os pincéis em outra cor.
- Linha 11: Especificamos o diâmetro para o círculo que estamos prestes a desenhar. Usar uma variável para isso é preferível, uma vez que precisamos nos referir a esse valor várias vezes e não queremos ter que mudá-lo em vários lugares se quisermos que o círculo seja maior ou menor.
- Linha 12: Para desenhar um círculo, simplesmente desenhamos uma classe: `~kivy.graphics.vertex_instructions.Ellipse` com largura e altura iguais. Como queremos que o círculo seja desenhado onde o usuário toca, passamos a posição do toque para a elipse. Note que precisamos mudar a elipse por $-d/2$ nas direções x e y (exemplo: para esquerda e para baixo) porque a posição especifica o canto inferior esquerdo da caixa delimitadora da elipse, e queremos que ela fique cen-

trada em torno de nosso toque.

Foi fácil, não foi? Ele fica melhor! Atualize o código para ter esta aparência:

```
1 from kivy.app import App
2 from kivy.uix.widget import Widget
3 from kivy.graphics import Color, Ellipse, Line
4
5
6 class MyPaintWidget(Widget):
7
8     def on_touch_down(self, touch):
9         with self.canvas:
10             Color(1, 1, 0)
11             d = 30.
12             Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d,
13             → d))
14             touch.ud['line'] = Line(points=(touch.x, touch.y))
15
16     def on_touch_move(self, touch):
17         touch.ud['line'].points += [touch.x, touch.y]
18
19 class MyPaintApp(App):
20
21     def build(self):
22         return MyPaintWidget()
23
24
25 if __name__ == '__main__':
26     MyPaintApp().run()
```



Isso é o que foi modificado:

- Linha 3: Agora não importamos somente a instrução de desenho *Ellipse*, mas também a instrução de desenho *Line*. Se consultares a documentação de *Line*, verás que aceita um argumento *points* que deve ser uma lista de coordenadas de ponto 2D, como (*x₁* , *Y₁*, *x₂*, *y₂*, ..., *x_N*, *y_N*).
- Linha 13: Isto é onde fica interessante. *Touch.ud* é um dicionário Python (tipo <dict>) que nos permite armazenar *atributos personalizados* para um toque.
- Linha 13: Fazemos uso da instrução *Line* que importamos e definimos uma *Line* para desenho. Como isso é feito em *on_touch_down*, haverá uma nova linha para cada novo toque. Ao criar a linha dentro do bloco *with*, o Canvas automaticamente conhece a linha e irá desenhá-la. Nós só queremos modificar a linha mais tarde, então armazenamos uma referência dela no dicionário *touch.ud* sob a chave arbitrariamente escolhida, mas devidamente nomeada ‘line’. Passamos a linha que estamos criando a posição de toque inicial, porque é onde a nossa linha começará.
- Linhas 15: Adicionamos um novo método ao nosso Widget. Isso é semelhante ao método *on_touch_down*, mas ao invés de ser chamado quando um *novo* toque de tela ocorrer, este método será chamado quando um touch *existir* (para o qual “*on_touch_down*” já foi chamado) move, sua posição muda. Observe que este é o objeto **same MotionEvent** com atributos atualizados. Isso é algo que achamos incrivelmente útil e você verá logo por quê.
- Linha 16: Lembre-se: Este é o mesmo objeto de toque que temos em *on_touch_down*, então podemos simplesmente acessar os dados que armazenamos no dicionário *touch.ud*! Para a linha que configuramos para este toque anteriormente, agora adicionamos a posição atual do toque como um novo ponto. Sabemos que precisamos estender a linha porque isso acontece em *on_touch_move*, que só é invocado quando o toque foi movido, e é exatamente por isso que queremos atualizar a linha. Armazenar a linha em *touch.ud* torna muito mais fácil para nós, já que não precisamos manter nossa própria contabilidade de touch-to-line.

Por enquanto, tudo bem. No entanto, isto não está bonito ainda. Parece um pouco como spaghetti bolognese. Que tal dar cada toque sua própria cor. Ótimo, vamos fazer isso:

```

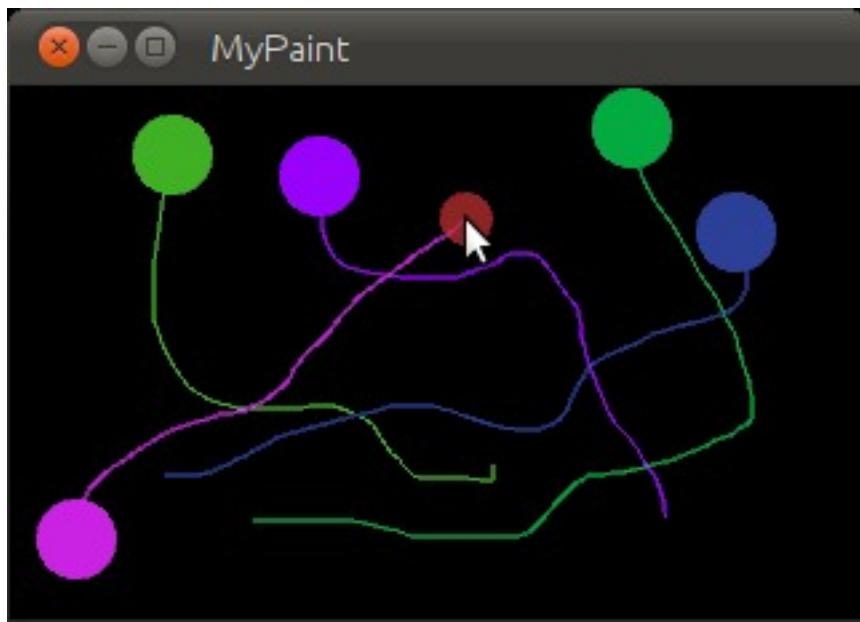
1 from random import random
2 from kivy.app import App
3 from kivy.uix.widget import Widget
4 from kivy.graphics import Color, Ellipse, Line
5
6
7 class MyPaintWidget(Widget):
8
9     def on_touch_down(self, touch):

```

```

10     color = (random(), random(), random())
11     with self.canvas:
12         Color(*color)
13         d = 30.
14         Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d,
15             ↵ d))
16         touch.ud['line'] = Line(points=(touch.x, touch.y))
17
18     def on_touch_move(self, touch):
19         touch.ud['line'].points += [touch.x, touch.y]
20
21 class MyPaintApp(App):
22
23     def build(self):
24         return MyPaintWidget()
25
26
27 if __name__ == '__main__':
28     MyPaintApp().run()

```



Aqui estão as modificações:

- Importamos a função random() do Python que nos dará valores aleatórios no intervalo de ([0., 1.]).
- Linha 10: Neste caso, simplesmente criamos uma nova tupla de 3 valores de ponto fluente aleatórios que representarão uma cor RGB aleatória. Como fazemos isso em on_touch_down, cada novo toque terá sua própria cor. Não se confunda com o uso de **tuplas**. Estamos apenas vinculando a tupla **color** para uso como um atalho dentro deste método porque estamos preguiçosos.

- Linha 12: Como antes, definimos a cor para a tela. Só que desta vez usamos valores aleatórios que geramos e os fornecemos para a classe de cores usando a sintaxe de descompactação de tuplas do Python (já que a classe Color espera três componentes individuais de cor, ao invés de apenas 1. Se passássemos diretamente a tupla, estaria sendo passado apenas 1 valor, independentemente do fato de que a própria tupla contém 3 valores).

Isso parece muito mais bonito já! Com muita habilidade e paciência, você pode até ser capaz de criar um pequeno desenho!

Nota: Como por padrão, as instruções `Color` assumem o modo RGB e estamos alimentando uma tupla com três valores aleatórios com ponto flutuante para ele, pode muito bem acontecer que nós acabemos com muita obscuridade. Ou até mesmo cores negras se tivermos azar. Isso seria ruim porque, por padrão, a cor de fundo também é escura, então você não seria capaz de (facilmente) ver as linhas que desenhamos. Há um bom truque para evitar isso: Em vez de criar uma tupla com três valores aleatórios, crie uma tupla como esta: `(random(), 1., 1.)`. Em seguida, ao passá-la para a instrução de cores, defina o modo para o espaço de cores HSV: `Color(*color, mode=' hsv ')`. Desta forma, terás um menor número de cores possíveis, mas as cores que receberes sempre serão igualmente brilhante: apenas a tonalidade é que muda.

Bonus Points

Neste ponto, poderíamos dizer que estamos prontos. O widget faz o que é esperado fazer: traça os toques e desenha linhas. Ele até desenha círculos nas posições em que uma linha começa.

Mas e se o usuário quiser iniciar um novo desenho? Com o código atual, a única maneira de limpar a janela seria reiniciar a aplicação inteira. Felizmente, podemos fazer melhor. Vamos adicionar um botão *Clear* que apaga todas as linhas e círculos que foram desenhados até agora. Agora há duas opções:

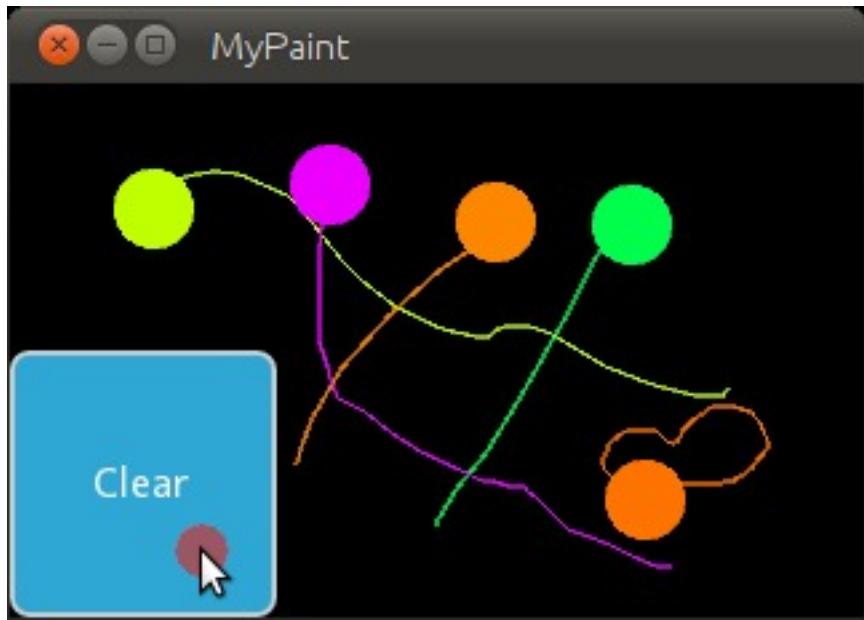
- Poderíamos criar o botão como um filho do nosso Widget. Isso implicaria que se criares mais de um Widget, cada Widget obterá i seu próprio botão. Se não tiveres cuidado, isso também permitirá que os usuários desenhem na parte superior do botão, que pode não ser o que desejas.
- Ou criamos o botão somente uma vez, inicialmente, em nossa classe de aplicativo e quando pressionado, limparemos o widget.

Para esse nosso simples exemplo, realmente não importa muito. Para aplicativos maiores, deverás pensar sobre quem faz o que no seu aplicativo. Iremos com a segunda opção aqui para ver como poderias criar a árvore de aplicativos em seu método de classe `build()`. Também mudaremos para o espaço de cores HSV (veja a nota anterior):

```

1  from random import random
2  from kivy.app import App
3  from kivy.uix.widget import Widget
4  from kivy.uix.button import Button
5  from kivy.graphics import Color, Ellipse, Line
6
7
8  class MyPaintWidget(Widget):
9
10     def on_touch_down(self, touch):
11         color = (random(), 1, 1)
12         with self.canvas:
13             Color(*color, mode='hsv')
14             d = 30.
15             Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d,
16             ↵ d))
17             touch.ud['line'] = Line(points=(touch.x, touch.y))
18
19     def on_touch_move(self, touch):
20         touch.ud['line'].points += [touch.x, touch.y]
21
22 class MyPaintApp(App):
23
24     def build(self):
25         parent = Widget()
26         self.painter = MyPaintWidget()
27         clearbtn = Button(text='Clear')
28         clearbtn.bind(on_release=self.clear_canvas)
29         parent.add_widget(self.painter)
30         parent.add_widget(clearbtn)
31         return parent
32
33     def clear_canvas(self, obj):
34         self.painter.canvas.clear()
35
36
37 if __name__ == '__main__':
38     MyPaintApp().run()

```



Eis o que acontece:

- Linha 4: Adicionamos uma instrução de importação para poder usar a classe: `class: ~kivy.uix.button.Button`.
- Linha 25: Criamos um objeto `Dummy Widget()` como um pai para o nosso Widget de pintura e o botão que estamos prestes a adicionar. Esta é apenas a abordagem de um pobre homem para configurar uma hierarquia de árvore de Widgets. Poderíamos também usar um layout ou fazer algumas coisas extravagantes. Mais uma vez: este Widget não faz absolutamente nada, exceto segurar os dois Widgets que agora vamos adicionar a ele como sendo filhos.
- Linha 26: Criamos nosso `MyPaintWidget()` como de costume, apenas desta vez não o retornamos diretamente, mas o atribuímos a um nome de variável.
- Linha 27: Nós criamos um widget `button`. Ele terá um label em sua exibição com o texto 'Clear' - 'Limpar'.
- Linha 28: Em seguida, ligamos o evento `on_release` do botão (que é disparado quando o botão é pressionado e depois liberado) para a função `callback clear_canvas` Definido nas linhas 33 e 34 abaixo.
- Linha 29 e 30: Criamos a hierarquia de widgets fazendo o `pintor` e o `clearbtn` filhos do widget-pai fictício. Isso significa que `pintor` e `clearbtn` agora são irmãos na terminologia usual da árvore de ciência da computação.
- Linha 33 e 34: Até agora, o botão não fez nada. Estava lá, visível, e você poderia pressioná-lo, mas nada aconteceria. Nós mudamos isso aqui: criamos uma função pequena e descartável que vai ser nossa função que irá responder quando o botão for pressionado <http://en.wikipedia.org/wiki/Callback_function#Python>. A função apenas limpa o conteúdo da tela, tornando-a preta novamente.

Nota: A classe Kivy Widget, por design, é mantida simples. Não há propriedades

gerais, como cor de fundo e cor de borda. Em vez disso, os exemplos e a documentação ilustram como lidar facilmente com essas coisas simples, como fizemos aqui, definindo a cor para a tela e desenhando a forma. A partir de um simples início, você pode ir para uma personalização mais elaborada. Os widgets internos de nível mais alto, derivados do Widget, como Button, possuem propriedades de conveniência, como background_color, mas variam de acordo com o widget. Use os documentos da API para ver o que é oferecido por um widget e subclasse se você precisar adicionar mais funcionalidade.

Parabéns! Você escreveu seu primeiro widget Kivy. Obviamente, esta foi apenas uma breve introdução. Há muito mais para ser descoberto. Sugerimos fazer uma pequena pausa para fixar o que você acabou de aprender. Talvez tirar algumas fotos agradáveis para relaxar? Se você sente que compreendeu tudo e está pronto para mais, recomendamos que você siga a diante.

3.3 Curso Rápido

O curso rápido de Kivy é uma série de vídeo tutoriais do Youtube do principal desenvolvedor do Kivy, [inclement](#). Estes vídeos fornecem um passo-a-passo em Kivy para usuários que sabem programar em Python e é amigável para iniciantes em Python. Após os tutoriais Pong e Paint, este conjunto de vídeos abrange as características e técnicas básicas a serem utilizadas na criação rápida de seu aplicativo, mantendo seu código elegante e atraente.

3.3.1 Informação Básica

O Crash Course consiste primariamente de uma série de vídeos no YouTube, cada um com aproximadamente 10 minutos. Há também artigos descrevendo alguns dos vídeos e os códigos usados.

Os tópicos cobertos pelo Crash Course incluem:

- Uso dos Widgets básicos do Kivy como o *Label*, *Button*, *Scatter* e *TextInput*
- Construindo uma aplicação para o Android com o [toolchain antigo do python-para-android](#)
- Vinculando funções aos eventos
- Usando mudanças em variáveis em movimento
- Interface de Usuário inteligente ([Kv language](#))
- Propriedades
- Canvas e Desenhos

- Label quando paginando
- Posicionamento e Layouts
- Animação e Clock
- Acessando a API do Android ([pyjnius](#), [plyer](#))
- Painel de Configurações (e construindo suas próprias opções)
- ScreenManager

Links:

- [Videos](#)
- [Artigos](#)
- [Código](#)

Referência API

Referência API é uma lista com o léxico das classes, métodos e funcionalidades que o Kivy oferece.

4.1 Framework Kivy

Kivy é uma biblioteca open source para o desenvolvimento de aplicações multi-touch. Funciona em diversas plataformas (Linux/OSX/Windows/Android/iOS) e é distribuída sob a [MIT License](#).

Vem com suporte nativo para muitos dispositivos de entrada, uma biblioteca em expansão que usa para desenhar widgets e aceleração de hardware OpenGL. Kivy é desenhado para permitir foco na construção de aplicações com interface configurável de maneira rápida e fácil.

Com Kivy, conseguirá tirar vantagem competitiva da natureza dinâmica do Python. Existem milhares de bibliotecas gratuitas que podem ser integradas em sua aplicação. Ao mesmo tempo, partes críticas para a performance, são implementadas usando [Cython](#).

Ver <http://kivy.org> para mais detalhes.

`kivy.require(version)`

Requerido pode ser usado para confirmar a versão mínima requerida para executar aplicação Kivy. Por exemplo, pode iniciar seu código de aplicação como:

```
import kivy  
kivy.require('1.0.1')
```

Se o usuário tentar executar a aplicação com a versão do Kivy que é anterior a versão especificada, uma Exceção é acionada.

A string com a versão Kivy pode ser assim:

```
X.Y.Z[-tag[-tagrevision]]
```

X **is** the major version
Y **is** the minor version
Z **is** the bugfixes revision

A tag é opcional, mas pode ser ‘dev’, ‘alpha’ ou ‘beta’. Tagrevision é revisão da tag.

Aviso: Não deve haver versão para uma tag, exceto -dev. Verificar uma versão ‘dev’ irá avisar o usuário que a versão corrente Kivy não é ‘-dev’, mas não irá acionar exceção. Não deve ser validado versão com uma tagrevision.

kivy.kivy_configure()

Chamada pós configuração do Kivy. Essa função deve ser chamada se janela foi criada por você mesmo.

kivy.kivy_register_post_configuration(callback)

Registrar a função que será chamada quando kivy_configure() for chamada.

Aviso: Uso interno.

kivy.kivy_options = {‘window’: (‘egl_rpi’, ‘sdl2’, ‘pygame’, ‘sdl’, ‘x11’), ‘camera’: (‘opencv’, ‘g
Configuração global opções kivy

kivy.kivy_base_dir = ‘/usr/local/lib/python2.7/dist-packages/kivy’
Diretório Kivy

kivy.kivy_modules_dir = ‘/usr/local/lib/python2.7/dist-packages/kivy/modules’
Diretório módulos Kivy

kivy.kivy_data_dir = ‘/usr/local/lib/python2.7/dist-packages/kivy/data’
Diretório dados Kivy

kivy.kivy_shader_dir = ‘/usr/local/lib/python2.7/dist-packages/kivy/data/glsł’
Diretório glsl shader Kivy

kivy.kivy_icons_dir = ‘/usr/local/lib/python2.7/dist-packages/kivy/data/icons/’
Config caminho ícones Kivy (Não remova as aspas finais)

kivy.kivy_home_dir = “
Diretório armazenamento user-home Kivy

```
kivy.kivy_config_fn = ""  
    Configuração nome arquivo Kivy  
kivy.kivy_usermodules_dir = ""  
    Diretório módulos usuário Kivy
```

4.1.1 Animação

class:*Animation* e *AnimationTransition* são usadas para animar as propriedades *Widget*. Você deve especificar pelo menos o nome da propriedade e o valor do objeto. Para usar uma animação siga esses passos:

- Configura um objeto da Animação
- Use o objeto da Animação em um Widget

Animação simples

Para animar uma posição x ou y do Widget, simplesmente especifique o valor do objeto x/y onde você deseja que esteja posicionado o widget no final da animação.

```
anim = Animation(x=100, y=100)  
anim.start(widget)
```

A animação irá durar 1 segundo a menos que *duration* seja especificado. Quando *anim.start()* é chamada, o Widget irá se mover suavemente da posição x/y atual para (100, 100)

Múltiplas propriedades e transições

Você pode animar várias propriedades e usar funções de transição internas ou personalizadas usando: attr: *transition* (ou o atalho 't = '). Por exemplo, para animar a posição e o tamanho usando a transição 'in_quad':

```
anim = Animation(x=50, size=(80, 80), t='in_quad')  
anim.start(widget)
```

Note que o parâmetro *t=* pode ter o nome da string de um método da classe :class:'AnimationTransition' ou da sua própria função de animação

Animação sequencial

Para juntar animações seqüencialmente, use o operador '+'. O exemplo a seguir animará para x = 50 durante 1 segundo, depois animará o tamanho para (80, 80) nos próximos dois segundos:

```
anim = Animation(x=50) + Animation(size=(80, 80), duration=2.)
anim.start(widget)
```

Animação paralela

Para juntar animações em paralelo, use o operador '&'. O exemplo a seguir animará a posição para (80,10) durante 1 segundo, enquanto em paralelo anima para o tamanho (800,800):

```
anim = Animation(pos=(80, 10))
anim &= Animation(size=(800, 800), duration=2.)
anim.start(widget)
```

Tenha em mente que a criação de animações sobrepostas na mesma propriedade pode ter resultados inesperados. Se você quiser aplicar várias animações à mesma propriedade, você deve programá-las seqüencialmente (através do operador '+' ou usando o * on_complete * callback) ou cancelar animações anteriores usando o método: attr: ~ *Animation.cancel_all*.

Repetindo animação

Novo na versão 1.8.0.

Nota: Atualmente, isso é apenas implementado para animações em 'Sequência'.

Para por uma animação em repetição, apenas coloque a propriedade `Sequence.repeat` como True

```
anim = Animation(...) + Animation(...)
anim.repeat = True
anim.start(widget)
```

Para controle do fluxo das animações como parar e cancelar, use os métodos já existentes no módulo da animação.

`class kivy.animation.Animation(**kw)`
Bases: `kivy.event.EventDispatcher`

Crie uma definição de animação que pode ser usada para animar um Widget.

Parameters

“Duracao” ou “d”: float, o padrão é 1.Duração de uma animação, em segundos.

transition or t: str ou func Função de transição para propriedades de animação. Pode ser o nome de um método de *AnimationTransition*.

step ou s: float Step em milissegundos da animação. O default é 0, o que significa que a animação é atualizada para cada quadro.

Para atualizar a animação com menos freqüência, defina o valor de step para float. Por exemplo, se você quiser animar com 30 FPS, use s=1/30

Events

on_start: animação, widget Disparado quando a animação é iniciada em um widget.

on_complete: animação, widget Disparada quando a animação é completada ou parada em um widget.

on_progress: animação, widget, progressão Disparada quando a progressão da animação esta mudando.

Alterado na versão 1.4.0: Adicionar parâmetro s/step

Alterado na versão 1.10.0: O valor padrão de um parâmetro step foi mudado de 1/60. para 0.

animated_properties

Retorna a propriedade usada para animar.

cancel(widget)

Cancaela a animação anteriormente aplicada a um widget. O mesmo efeito que: attr: stop, exceto que o evento 'on_complete' não será executado!

Novo na versão 1.4.0.

static cancel_all(widget, *largs)

Cancela todas as animações que afetam um widget específico / lista de propriedades. Veja *cancel*.

Exemplo:

```
anim = Animation(x=50)
anim.start(widget)

# and later
Animation.cancel_all(widget, 'x')
```

Novo na versão 1.4.0.

cancel_property(widget, prop)

Mesmo se uma animação estiver em execução, remove uma propriedade. Não será mais animado. Se for a única / última propriedade a ser animada, a animação será cancelada (ver: attr: *cancel*)

Novo na versão 1.4.0.

duration

Retorna a duração da animação

have_properties_to_animate(widget)

Retorna True se um widget ainda tem propriedades para animar.

Novo na versão 1.8.0.

start(widget)

Começa a animação de um widget.

stop(widget)

Para a animação aplicada provisoriamente a um widget, desencadeando o evento *on_complete*

static stop_all(widget, *largs)

Para todas as animações que afetam o widget específico / lista de propriedades.

Exemplo:

```
anim = Animation(x=50)
anim.start(widget)

# and later
Animation.stop_all(widget, 'x')
```

stop_property(widget, prop)

Mesmo que uma animação se encontre em execução, remove uma propriedade. A animação não será continuada. Se for a única/última propriedade a ser animada, a animação será interrompida (ver :attr:'stop').

transition

Retorna a transição da animação

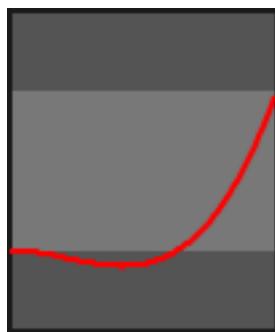
class kivy.animation.AnimationTransition

Bases: object

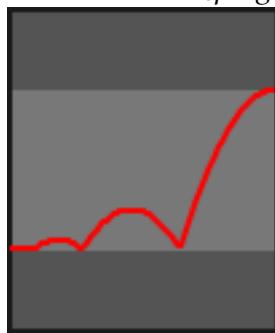
Coleção de funções de animações a serem usadas com o objecto Animação. Funções de Facilitação portadas para Kivy do Projeto Clutter. <https://developer.gnome.org/clutter/stable/ClutterAlpha.html>

O parâmetro ‘progresso’ em cada função da animação esta em um range de 0-1

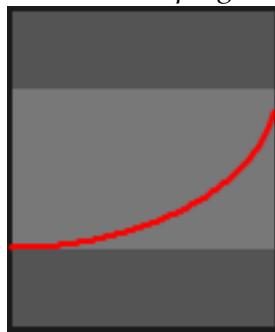
static in_back(progress)



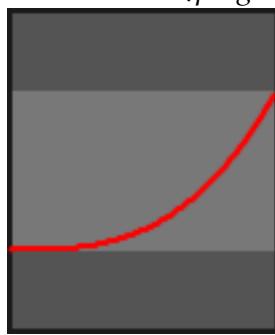
static in_bounce(*progress*)



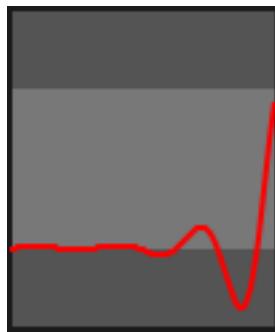
static in_circ(*progress*)



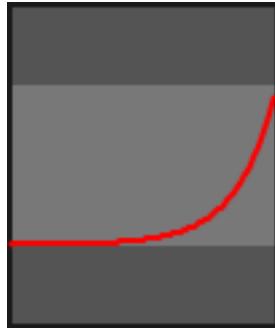
static in_cubic(*progress*)



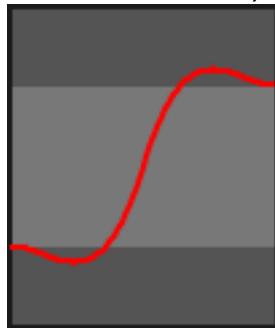
static in_elastic(*progress*)



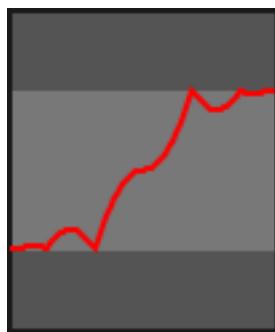
static `in_expo`(progress)



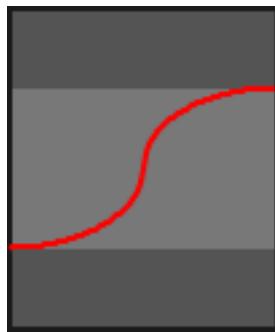
static `in_out_back`(progress)



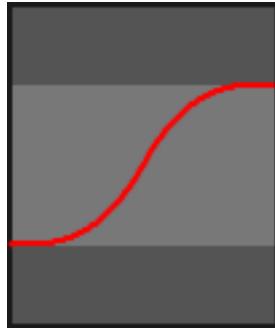
static `in_out_bounce`(progress)



static `in_out_circ`(progress)



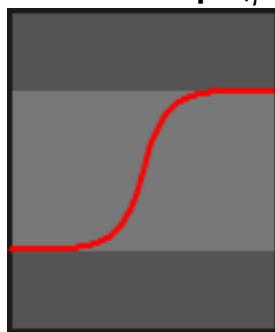
static in_out_cubic(progress)



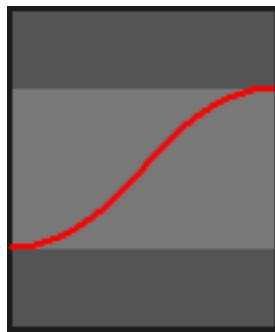
static in_out_elastic(progress)



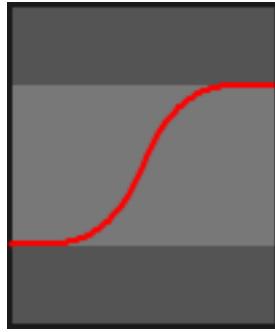
static in_out_expo(progress)



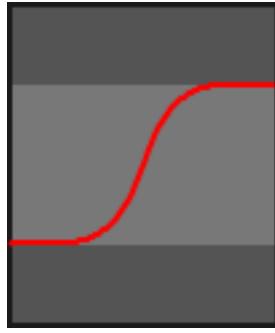
static in_out_quad(progress)



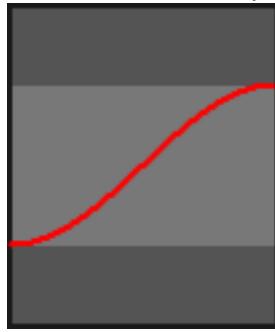
static in_out_quart(*progress*)



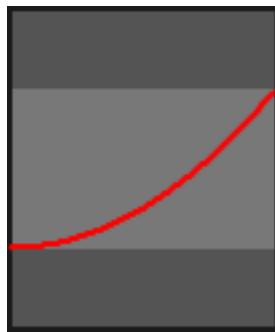
static in_out_quint(*progress*)



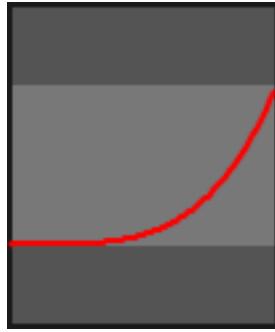
static in_out_sine(*progress*)



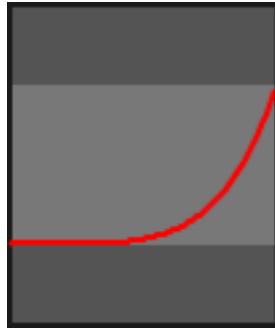
static in_quad(*progress*)



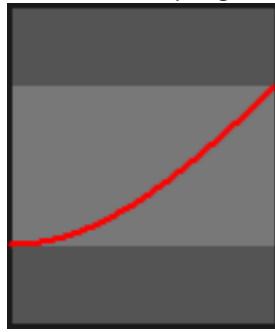
static `in_quart`(progress)



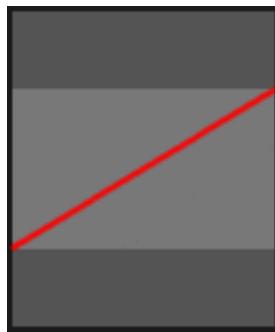
static `in_quint`(progress)



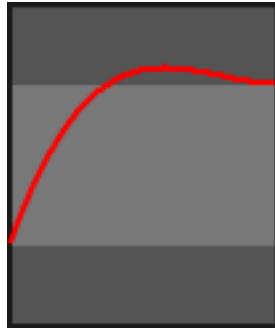
static `in_sine`(progress)



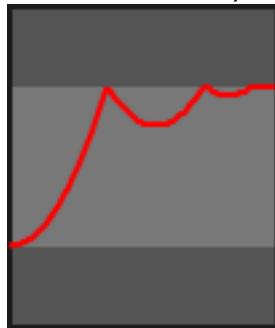
static `linear`(progress)



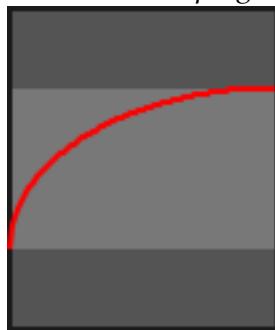
static out_back(*progress*)



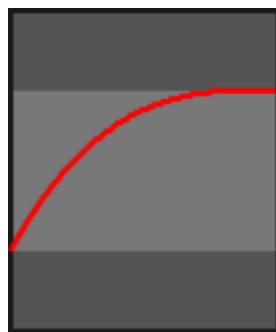
static out_bounce(*progress*)



static out_circ(*progress*)



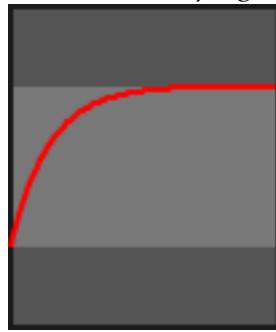
static out_cubic(*progress*)



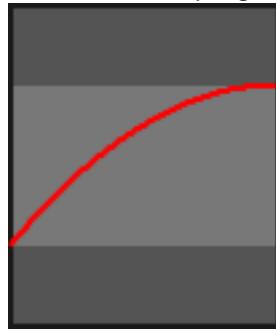
static out_elastic(*progress*)



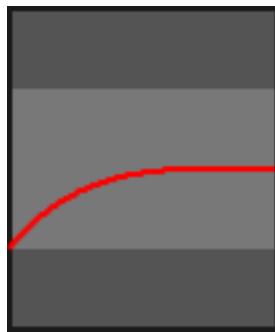
static out_expo(*progress*)



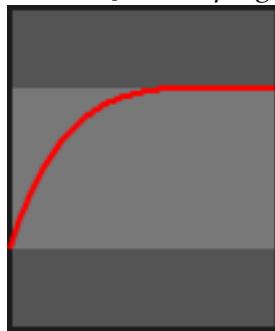
static out_quad(*progress*)



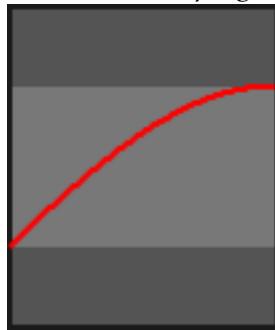
static out_quart(*progress*)



`static out_quint(progress)`



`static out_sine(progress)`



4.1.2 Aplicação

A classe `App` é a base para criar aplicações Kivy. Pense nisso como seu ponto de entrada principal no loop de execução do Kivy. Na maioria dos casos, você cria uma subclasse dessa classe e faz seu próprio aplicativo. Você cria uma instância de sua classe de aplicativo específica e, quando está pronto para iniciar o ciclo de vida do aplicativo, chama o método :meth: `App.run`.

Criando uma Aplicação

Método usando sobreposição `build()`

Para inicializar seu app com uma arvore de widgets, sobreponha o método `build()` na classe de seu app e retorne a árvore de widgets que você construiu.

Aqui está um exemplo de uma aplicação bastante simples que apenas mostra um botão:

```
...
Application example using build() + return
=====

An application can be built if you return a widget on build(), or_
if you set
self.root.
...

import kivy
kivy.require('1.0.7')

from kivy.app import App
from kivy.uix.button import Button

class TestApp(App):

    def build(self):
        # return a Button() as a root widget
        return Button(text='hello world')

if __name__ == '__main__':
    TestApp().run()
```

O arquivo também está disponível na pasta de exemplos em `kivy/examples/application/app_with_build.py`.

Aqui, nenhuma árvore de widget foi construída (ou se você quiser, uma árvore com um único nó principal).

Método usando arquivo kv

Você também pode utilizar o *Linguagem Kivy* para criar aplicações. O arquivo `.kv` pode conter regras e a definição do Widget principal (raiz) ao mesmo tempo. Aqui nós temos um exemplo de como um Botão pode ser definido num arquivo `*.kv`.

Conteúdo do ‘`test.kv`’:

```
#:kivy 1.0

Button:
    text: 'Hello from test.kv'
```

Conteúdo do 'main.py':

```
...
Application built from a .kv file
=====

This shows how to implicitly use a .kv file for your application. You
should see a full screen button labelled "Hello from test.kv".

After Kivy instantiates a subclass of App, it implicitly searches for a .kv
file. The file test.kv is selected because the name of the subclass of App is
TestApp, which implies that kivy should try to load "test.kv". That file
contains a root Widget.
...

import kivy
kivy.require('1.0.7')

from kivy.app import App

class TestApp(App):
    pass

if __name__ == '__main__':
    TestApp().run()
```

Veja `kivy/examples/application/app_with_kv.py`.

O relacionamento entre `main.py` e `test.kv` é explicado em [App.load_kv\(\)](#).

Configuração da aplicação

Usar o arquivo de configuração

Seu aplicativo pode precisar do seu próprio arquivo de configuração. A classe `App` lida com os arquivos 'ini' automaticamente se você adicionar o par chave-valor da seção ao método `meth: 'App.build_config'` usando o parâmetro `config` (uma instância de: `class: ~kivy.config.ConfigParser`)

```
class TestApp(App):
    def build_config(self, config):
        config.setdefault('section1', {
```

```

        'key1': 'value1',
        'key2': '42'
    })

```

Assim que você adiciona uma seção à configuração, um arquivo é criado no disco (veja: attr: `~App.get_application_config` para sua localização) e nomeado com base no nome da sua classe. “TestApp” dará um arquivo de configuração chamado “test.ini” com o conteúdo:

```

[section1]
key1 = value1
key2 = 42

```

O “test.ini” será automaticamente carregado em tempo de execução e você poderá acessar a configuração no seu método: meth: `App.build`:

```

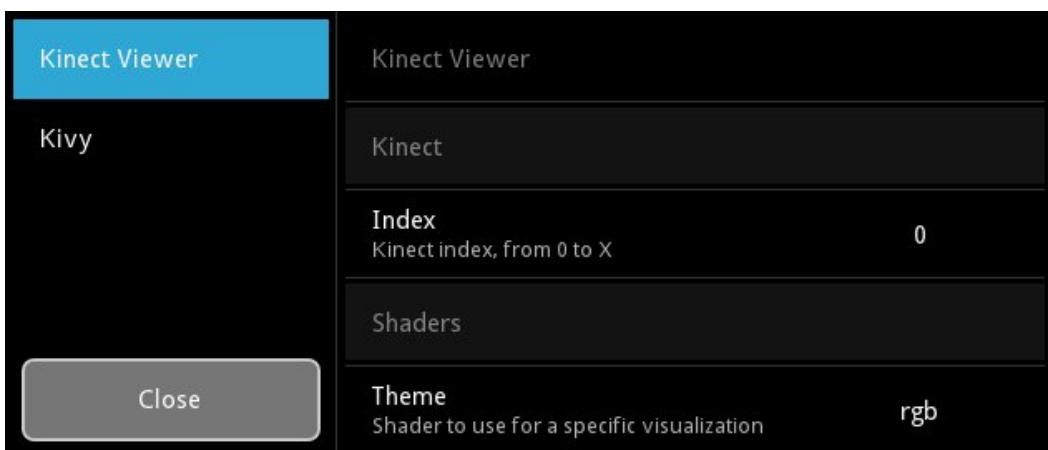
class TestApp(App):
    def build_config(self, config):
        config.setdefault('section1', {
            'key1': 'value1',
            'key2': '42'
        })

    def build(self):
        config = self.config
        return Label(text='key1 is %s and key2 is %d' % (
            config.get('section1', 'key1'),
            config.getint('section1', 'key2')))

```

Cria um painel de configurações

Seu aplicativo pode ter um painel de configurações para permitir que o usuário configure alguns de seus tokens. Aqui está um exemplo feito em KinectViewer (disponível no diretório de exemplos):



Você pode adicionar seus próprios painéis de configurações, estendendo o método: meth: *App.build_settings*. Verifique a classe: *~kivy.uix.settings.Settings* sobre como criar um painel, porque primeiramente, você precisa de um arquivo / dados JSON.

Vamos tomar como exemplo o snippet anterior do TestApp com configuração personalizada. Podemos criar um JSON como este:

```
[  
    { "type": "title",  
        "title": "Test application" },  
  
    { "type": "options",  
        "title": "My first key",  
        "desc": "Description of my first key",  
        "section": "section1",  
        "key": "key1",  
        "options": ["value1", "value2", "another value"] },  
  
    { "type": "numeric",  
        "title": "My second key",  
        "desc": "Description of my second key",  
        "section": "section1",  
        "key": "key2" }  
]
```

Então, podemos criar um painel usando este JSON para criar automaticamente todas as opções e vinculá-las ao nosso: attr: *App.config* Instância ConfigParser:

```
class TestApp(App):  
    # ...  
    def build_settings(self, settings):  
        jsondata = """... put the json data here ..."""  
        settings.add_json_panel('Test application',  
                               self.config, data=jsondata)
```

Isso é tudo! Agora você pode pressionar F1 (tecla pressionada) para alternar o painel de configurações ou pressione a tecla “configurações” no seu dispositivo Android. Você pode chamar manualmente: meth: *App.open_settings* e: meth: *App.close_settings* se você quiser lidar com isso manualmente. Todas as alterações no painel são salvas automaticamente no arquivo de configuração.

Você também pode usar: meth: *App.build_settings* para modificar as propriedades do painel de configurações. Por exemplo, o painel padrão possui uma barra lateral para alternar entre painéis json cuja largura padrão é 200dp. Se preferir que este seja mais estreito, pode adicionar:

```
settings.interface.menu.width = dp(100)
```

para o seu método *build_settings()*.

Você pode querer saber quando um valor de configuração foi alterado pelo usuário para adaptar ou recarregar sua interface do usuário. Você pode, então, sobrecarregar o método: meth: `on_config_change`:

```
class TestApp(App):
    # ...
    def on_config_change(self, config, section, key, value):
        if config is self.config:
            token = (section, key)
            if token == ('section1', 'key1'):
                print('Our key1 has been changed to', value)
            elif token == ('section1', 'key2'):
                print('Our key2 has been changed to', value)
```

O painel de configuração do Kivy é adicionado por padrão à instância de configurações. Se você não quiser este painel, você pode declarar sua aplicação da seguinte maneira:

```
class TestApp(App):
    use_kivy_settings = False
    # ...
```

Isso só remove o painel do Kivy, mas não impede que a instância de configurações apareça. Se você quiser impedir que a instância de configurações apareça completamente, você pode fazer isso:

```
class TestApp(App):
    def open_settings(self, *largs):
        pass
```

Novo na versão 1.0.7.

Perfis com `on_start` e `on_stop`

É frequentemente útil perfilar códigos python em ordem para descobrir localizações para otimizar. A biblioteca padrão de perfiladores (<http://docs.python.org/2/library/profile.html>) fornece múltiplas opções para códigos de perfilação. Para perfilar o programa inteiro, as abordagens naturais para perfilar como um módulo ou método de execução para perfilar não funciona com Kivy. É contudo, possível usar `App.on_start()` and `App.on_stop()` methods:

```
import cProfile

class MyApp(App):
    def on_start(self):
        self.profile = cProfile.Profile()
        self.profile.enable()
```

```

def on_stop(self):
    self.profile.disable()
    self.profile.dump_stats('myapp.profile')

```

Será criado um arquivo chamado 'myapp.profile' quando você sair da aplicação.

Customização de layout

Você pode escolher diferentes configurações para os layouts do widget definindo `App.settings_cls`. Por padrão, isto é uma `:classe:'~kivy.uix.settings.Settings'` classe que fornece o layout da barra lateral retratado, mas você poderia definir qualquer um dos layouts fornecidos em `kivy.uix.settings` ou criar o seu. Veja a documentação do modulo por `kivy.uix.settings` para mais informações.

Você pode personalizar como o painel de configurações é exibido sobrepondo `App.display_settings()` que é chamado antes de exibir o painel de configurações na tela. Por padrão, simplesmente desenha o painel no topo da janela, mas você poderia modificá-lo para (por exemplo) mostrar as configurações em uma `:class:'~kivy.uix.popup.Popup'` ou adicioná-lo para seu `ScreenManager` do app se você está utilizando um. Caso esteja, você deveria também modificar `App.close_settings()` para sair o painel apropriadamente. Por exemplo, para que painel de configurações apareça em um popup você pode fazer:

```

def display_settings(self, settings):
    try:
        p = self.settings_popup
    except AttributeError:
        self.settings_popup = Popup(content=settings,
                                      title='Settings',
                                      size_hint=(0.8, 0.8))
    p = self.settings_popup
    if p.content is not settings:
        p.content = settings
    p.open()

def close_settings(self, *args):
    try:
        p = self.settings_popup
        p.dismiss()
    except AttributeError:
        pass # Settings popup doesn't exist

```

Finalmente, se você quer substituir o widget do painel de configurações atual, você pode remover as referências internas a ele usando `App.destroy_settings()`. Se você tiver modificado `App.display_settings()`, você deveria ter cuidado para detectar se o painel de configurações foi substituído.

Modo pausado

Novo na versão 1.1.0.

Em tablets e celulares, o usuário pode alternar à qualquer momento para outra aplicação. Por padrão, sua aplicação fechará e o evento `App.on_stop()` será disparado.

Se você oferecer suporte ao modo de Pausa, ao alternar para outro aplicativo, o aplicativo irá aguardar indefinidamente até que o usuário retorne ao aplicativo. Há um problema com o OpenGL em dispositivos Android: não é garantido que o Contexto OpenGL ES será restaurado quando o seu aplicativo for reiniciado. O mecanismo para restaurar todos os dados do OpenGL ainda não está implementado no Kivy.

O mecanismo de pausa implementado atualmente é:

1. Kivy verifica cada Frame se o modo de Pausa é ativado pelo sistema operacional devido ao usuário alternar para outro aplicativo, o desligamento do telefone ou qualquer outro motivo.
2. `App.on_pause()` é chamado:
3. If False is returned, then `App.on_stop()` is called.
4. If True is returned (default case), the application will sleep until the OS resumes our App.
5. Quando o aplicativo é reiniciado, `App.on_resume()` é chamado.
6. Se a memória do aplicativo tiver sido recuperada pelo sistema operacional, então, nada será chamado.

Aqui está um simples exemplo de como `on_pause()` deve ser utilizado:

```
class TestApp(App):  
  
    def on_pause(self):  
        # Here you can save data if needed  
        return True  
  
    def on_resume(self):  
        # Here you can check if any data needs replacing (usually  
        # nothing)  
        pass
```

Aviso: Ambos `on_pause` e `on_stop` devem salvar dados importantes porque depois `on_pause` é chamado, `on_resume` pode não ser chamado.

`class kivy.app.App(**kwargs)`
Bases: `kivy.event.EventDispatcher`

Classe *Application*, veja o módulo da documentação para maiores informações.

Events

on_start: Disparado quando o aplicativo está sendo iniciado (antes da chamada `runTouchApp()`).

on_stop: Chamado quando a aplicação está parada.

on_pause: Chamado quando a aplicação é pausada pelo SO.

on_resume: Chamado quando a aplicação volta a ser executada pelo sistema operacional que antes havia pausado-a. CUIDADO: não há garantia de que este evento será chamado antes do evento *on_pause* ser invocado!

Alterado na versão 1.7.0: Parâmetro `kv_file` adicionado.

Alterado na versão 1.8.0: Parâmetros `kv_file` e `kv_directory` são agora propriedades de App.

build()

Inicializa a aplicação. Isso será chamado somente uma vez. Se o método retornar um Widget (árvore), está será utilizada como o Widget root e será adicionada à janela.

RetornaNone ou uma instância de root (Widget raiz) `Widget` caso o `self.root` não exista.

build_config(config)

Novo na versão 1.0.7.

Esse método é chamado antes que o aplicativo seja inicializado para construir o seu objeto `ConfigParser`. Este é onde você pode colocar qualquer seção/key/valor padrão para sua configuração. Se alguma coisa estiver definida, a configuração será automaticamente salva no arquivo retornado por `get_application_config()`.

Parameters

`config: ConfigParser` Utilize isso para adicionar seção / chave / itens de valores

build_settings(settings)

Novo na versão 1.0.7.

Este método é chamado quando o usuário (ou você) deseja mostrar as configurações do aplicativo. Ele é chamado uma vez quando o painel de configurações for aberto pela primeira vez, após o qual o painel será armazenado em cache. O método pode ser chamado novamente se o painel de configurações armazenado em cache for removido por `destroy_settings()`.

Você pode utilizar esse método para adicionar painéis de configuração e personalizar o widget de configurações, por exemplo, para alterar a largura da barra lateral. Consulte a documentação do módulo para obter mais detalhes

Parameters

settings: *Settings* Instância de Configuração para adicionar painéis

close_settings(*args)

Fecha a janela de configuração aberta anteriormente.

Retorna *True* se as configurações tiverem sido modificadas.

config = None

Retorna uma instância da *ConfigParser* para a configuração do aplicativo. Pode usar isso para consultar alguns tokens de configuração no método *build()*.

create_settings()

Crie o painel de configurações. Esse método normalmente será chamado apenas uma vez durante o tempo de vida da aplicação e o resultado é armazenado internamente no cache, mas o método pode ser chamado novamente se as configurações do painel armazenadas cache forem removido por *destroy_settings()*.

Por padrão, ele criará um painel de configurações de acordo com *settings_cls*, call *build_settings()*, adicione um painel Kivy se: attr: *use_kivy_settings* é True, e ligue a on_close/on_config_change.

Se você quiser conectar seu próprio jeito de fazer as configurações, sem o painel Kivy ou os eventos de mudança close/config, este é o método que você deseja sobrecarregar.

Novo na versão 1.8.0.

destroy_settings()

Novo na versão 1.8.0.

Dereferencia o painel de configurações atual, se houver um. Isto significa que quando :meth: *App.open_settings* é o próximo a executar, um novo painel será criado e exibido. Ele não afeta nenhum dos conteúdos do painel, mas permite que você (por exemplo) atualize o layout do painel de configurações se você alterou o widget de configurações em resposta a uma alteração no tamanho da tela.

Se você tiver modificado *open_settings()* ou *display_settings()*, você deve ter cuidado para detectar corretamente se o widget de configurações anteriores foi destruído.

directory

Novo na versão 1.0.7.

Retorna o diretório onde o aplicativo existe.

display_settings(settings)

Novo na versão 1.8.0.

Exibe o painel de configurações. Por padrão, o painel é desenhado diretamente na parte superior da janela. Você pode definir outro comportamento sobrescrevendo este método, tal como adicioná-lo a um Screenmanager ou Popup.

Retorne *True* se a exibição for bem-sucedida, senão, *False*.

Parameters

settings: *Settings* Modifique este objeto para modificar a exibição das configurações.

get_application_config(*defaultpath*='%(appdir)s%(appname)s.ini')

Novo na versão 1.0.7.

Alterado na versão 1.4.0: Personalizado o caminho padrão para plataformas iOS e Android. Adicionado um parâmetro defaultpath para sistemas operacionais desktop (não aplicável ao iOS e Android.)

Retorna o nome do arquivo de configuração do aplicativo. Dependendo da plataforma, o arquivo do aplicativo será armazenado em locais diferentes:

- sobre iOS: <appdir>/Documents/.<appname>.ini
- sobre Android: /sdcard/.<appname>.ini
- Caso contrário: <appdir>/<appname>.ini

Quando você estiver distribuindo sua aplicação para Desktop, por favor, note que se a aplicação precisa ser instalada no lado do cliente, o usuário não precisa ter acesso de escrita ao diretório da aplicação. Se você desejar armazenar as configurações de usuário, você deverá sobrescrever este método e modificar o comportamento padrão para salvar as configurações no disco do usuário.

```
class TestApp(App):
    def get_application_config(self):
        return super(TestApp, self).get_application_config(
            '~/.(appname)s.ini')
```

Algumas notas:

- O caractere til ‘~’ expandirá o diretório do usuário.
- %(appdir)s será sobreescrito com *directory* da aplicação.
- %(appname)s será sobreescrito com o *name* da aplicação

get_application_icon()

Retorna o ícone da aplicação.

get_application_name()

Retorna o nome da aplicação.

`static get_running_app()`

Retorna a instância atual da aplicação.

Novo na versão 1.1.0.

`icon`

Ícone da aplicação. O ícone pode estar localizado no mesmo diretório como também no arquivo principal. Você pode definir essa configuração setando:

```
class MyApp(App):
    def build(self):
        self.icon = 'myicon.png'
```

Novo na versão 1.0.5.

Alterado na versão 1.8.0: `icon` é agora um *StringProperty*. Não defina o ícone na classe como indicado anteriormente na documentação.

Nota: Versões da biblioteca Kivy anteriores a 1.8.0 era necessário setar da seguinte forma:

```
class MyApp(App):
    icon = 'customicon.png'
```

Recommended 256x256 or 1024x1024? for GNU/Linux and Mac OSX
32x32 for Windows7 or less. <= 256x256 for windows 8
256x256 does work (on Windows 8 at least), but is scaled
down and doesn't look as good as a 32x32 icon.

`kv_directory`

Path para o diretório onde o arquivo kv da aplicação está armazenado, o padrão é `None`.

Novo na versão 1.8.0.

Se um `kv_directory` estiver definido, ele será usado para obter o arquivo kv inicial. Por padrão, é assumido o arquivo estar no mesmo diretório do atual arquivo de definição de App.

`kv_file`

Nome do arquivo do kv a ser carregado, o padrão é `None`.

Novo na versão 1.8.0.

Se um `kv_file` é definido, ele será carregado quando a aplicação iniciar. O carregamento do `kv_file` “padrão” será evitado.

`load_config()`

(interno) Esta função é usada para retornar um ConfigParser com a configuração da aplicação. Está fazendo 3 coisas:

- 1.Criando uma instância de *ConfigParser*
- 2.Abre as configurações padrão chamando *build_config()*, então
- 3.Caso exista, será aberto o arquivo de configuração da aplicação, do contrário, o mesmo será criado.

Retornainstância *ConfigParser*

load_kv(filename=None)

Este método é invocado na primeira vez em que a aplicação está sendo executada se nenhuma árvore de widgets tiver sido construída anteriormente para esta aplicação. Este método então procura um arquivo kv correspondente no mesmo diretório que o arquivo que contém a classe do aplicativo.

Por exemplo, digamos que você tenha um arquivo chamado main.py que contém:

```
class ShowcaseApp(App):  
    pass
```

Este método irá procurar por um arquivo chamado *showcase.kv* no diretório que contém main.py. O nome do arquivo kv tem que ser o nome em letras minúsculas da classe, sem o sufixo 'App' no final se ele existir.

Você pode definir regras e quem será o Widget raiz no seu arquivo kv:

```
<ClassName>: # this is a rule  
    ...  
  
ClassName: # this is a root widget  
    ...
```

Deve haver apenas um widget raiz. Consulte a documentação :doc: *api-kivy.lang* para obter mais informações sobre como criar arquivos kv. Se seu arquivo kv contém um widget raiz, ele será usado como self.root, o widget raiz para o aplicativo.

Nota: Esta função é chamada de *run()*, portanto, qualquer widget cujo estilo é definido neste arquivo kv e é criado antes *run()* é chamado (e.g. em *__init__*), não terá seu estilo aplicado. Observe que *build()* é chamado depois de *load_kv* ter sido chamado.

name

Novo na versão 1.0.7.

Retorna o nome do aplicativo com base no nome da classe.

on_config_change(config, section, key, value)

Manipulador de eventos disparado quando um Token de configuração foi alterado pela página de configurações.

on_pause()

Event handler called when Pause mode is requested. You should return True if your app can go into Pause mode, otherwise return False and your application will be stopped.

Você não pode controlar quando o aplicativo vai entrar neste modo. É determinado pelo sistema operacional e usado principalmente para dispositivos móveis (android / ios) e para redimensionamento.

The default return value is True.

Novo na versão 1.1.0.

Alterado na versão 1.10.0: The default return value is now True.

on_resume()

Manipulador do evento disparado quando o aplicativo está voltando do modo Pausado.

Novo na versão 1.1.0.

Aviso: Ao retomar, o Contexto OpenGL pode ter sido danificado/liberado. Isto é onde você pode reconstruir alguns de seu estado OpenGL, conteúdo FBO.

on_start()

Manipulador de eventos para o evento *on_start* que é disparado após a inicialização (depois que o `build()` foi chamado) mas antes que o aplicativo tenha começado a ser executado.

on_stop()

Manipulador de eventos para o evento *on_stop* que é disparado quando o aplicativo terminou de ser executado (ou seja, a janela está prestes a ser fechada).

open_settings(*largs)

Abra o painel de configurações da aplicação. Ele será criado pela primeira vez, ou recriado se o painel anteriormente em cache foi removido por :meth: *destroy_settings*. O painel de configurações será exibido com o método :meth: *display_settings*, que por padrão adiciona o painel de configurações à Janela anexada à sua aplicação. Você deve sobrescrever esse método se desejar exibir o painel de configurações de forma diferente.

Retorna Verdadeiro se as configurações foram abertas.

options = None

Opções enviadas pelo `__init__` da aplicação

root = None

O widget `raiz` retornado pelo método `build()` ou pelo método :meth:`load_kv` se o arquivo kv contiver um widget raiz.

root_window

Novo na versão 1.9.0.

Retorna a instância da janela raiz usada por `run()`.

run()

Inicia o aplicativo no modo autônomo (standalone).

settings_cls

Novo na versão 1.8.0.

A classe usada para construir o painel de configurações e a instância passada para `build_config()`. Você deveria usar ou `Settings` ou uma das subclasses fornecidas com layouts diferentes (`SettingsWithSidebar`, `SettingsWithSpinner`, `SettingsWithTabbedPanel`, `SettingsWithNoMenu`). Você também pode criar sua própria subclasse Configurações. Consulte a documentação de `Settings` para obter mais informações.

`settings_cls` é uma `ObjectProperty` e o padrão é `SettingsWithSpinner` que exibe os painéis de configurações com um botão (spinner) para alternar entre eles. Se você definir uma string, uma `Factory` será usada para resolver a classe.

stop(*args)

Pára o aplicativo.

Se você usar este método, o aplicativo inteiro irá parar emitindo uma chamada para `stopTouchApp()`.

title

Título do seu aplicativo. Você pode definir isto como a seguir:

```
class MyApp(App):
    def build(self):
        self.title = 'Hello world'
```

Novo na versão 1.0.5.

Alterado na versão 1.8.0: `title` é agora uma `StringProperty`. Não defina o título na classe como indicado anteriormente na documentação.

Nota: Para Kivy < 1.8.0, você pode definir como a seguir:

```
class MyApp(App):
    title = 'Custom title'
```

Se você quiser modificar dinamicamente o título, você pode fazer:

```
from kivy.base import EventLoop
EventLoop.window.title = 'New title'
```

use_kivy_settings = True

Novo na versão 1.0.7.

Se True, as configurações do aplicativo incluirão as configurações do Kivy. Se você não quer que o usuário modifique as configurações do Kivy da interface do usuário, altere para False.

user_data_dir

Novo na versão 1.7.0.

Retorna o caminho para o diretório no sistema de arquivos do usuário que o aplicativo pode usar para armazenar dados adicionais.

Diferentes plataformas tem diferentes convenções em relação ao local onde o usuário pode armazenar dados, preferências, dados de jogos e configurações. Esta função implementa estas convenções. O diretório <app_name> é criado quando a propriedade é chamada, a menos que já exista.

No IOS, *~/Documents/<app_name>* é retornado (dentro da sandbox do aplicativo).

No Android, */sdcard/<app_name>* é retornado.

No Windows, *%APPDATA%/<app_name>* é retornado.

No OS X, *~/Library/Application Support/<app_name>* é retornado.

No Linux, *\$XDG_CONFIG_HOME/<app_name>* é retornado.

4.1.3 Abertura de Dados Assíncrona

Este é um carregador assíncrono. Pode ser usado para carregar uma imagem e usá-la, mesmo se os dados ainda não estiverem totalmente disponíveis. Você deve especificar um padrão de carregamento de imagem quando usar o ‘carregador’.:

```
from kivy.loader import Loader
image = Loader.image('mysprite.png')
```

Você pode inclusive abrir uma imagem desde uma URL:

```
image = Loader.image('http://mysite.com/test.png')
```

Se você quiser alterar a imagem de carregamento padrão, você pode fazer da seguinte forma:

```
Loader.loading_image = Image('another_loading.png')
```

Ajustando o Carregador Assíncrono

Novo na versão 1.6.0.

Você pode ajustar o *Loader* para fornecer uma melhor experiência de usuário ou um melhor desempenho, dependendo das imagens que irás carregar. Dê uma olhada nos parâmetros:

- `Loader.num_workers` - Define o número de threads a serem iniciadas iniciar para carregar imagens.
- `Loader.max_upload_per_frame` - define o máximo de uploads de imagem na GPU que pode ser feito por frame.

`class kivy.loader.LoaderBase`

Bases: `object`

Base comum para implementações específicas do *Loader*. Por padrão, o *Loader* será a melhor implementação de carregador disponível.

A função `_update()` é chamada a cada 1/25.s ou a cada Frame caso haja menos do que 25 FPS.

error_image

Imagem usada para erro. Você pode alterá-la da seguinte forma:

```
Loader.error_image = 'error.png'
```

Alterado na versão 1.6.0: Não é mais somente leitura.

image(*filename*, *load_callback=None*, *post_callback=None*, ***kwargs*)

Carregua uma imagem usando o *Loader*. Um `ProxyImage` é retornado com uma imagem do carregamento. Pode usá-la como no exemplo a seguir:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.loader import Loader

class TestApp(App):
    def _image_loaded(self, proxyImage):
        if proxyImage.image.texture:
            self.image.texture = proxyImage.image.texture
```

```

def build(self):
    proxyImage = Loader.image("myPic.jpg")
    proxyImage.bind(on_load=self._image_loaded)
    self.image = Image()
    return self.image

TestApp().run()

```

Para cancelar todo o carregamento do plano de fundo, invoque a função *Loader.stop()*.

loading_image

Imagen usada para carregar. Você pode alterá-la da seguinte forma:

```
Loader.loading_image = 'loading.png'
```

Alterado na versão 1.6.0: Não é mais somente leitura.

max_upload_per_frame

O número de imagens a ser carregada por Frame. Por padrão, enviaremos apenas 2 imagens para a GPU por Frame. Se estiveres carregando muitas imagens pequenas, podes facilmente aumentar esse parâmetro para 10 ou mais. Se estiveres carregando várias imagens Full HD, o tempo de upload pode ter consequências e acabar bloqueando o aplicativo. Se quiseres uma experiência suave, use o padrão.

A verdade de fato é que uma imagem Full-HD RGB terá utilizará aproximadamente ~6MB na memória, isso pode levar tempo. Se *mipmap* = *True* também estiver ativado, então a GPU também precisará calcular o *mipmap* dessas imagens grandes, em tempo real. Talvez, possa ser melhor reduzir o *max_upload_per_frame* para 1 ou 2. Se quiseres se livrar disso (ou reduzir muito), dê uma olhada no formato DDS.

Novo na versão 1.6.0.

num_workers

Número de Workers a ser utilizado durante o carregamento (utilizado apenas se a implementação do *Loader* suportar isso). Essa configuração afeta o *Loader* somente na inicialização. Uma vez que o *Loader* tenha sido iniciado, a configuração não terá impacto:

```

from kivy.loader import Loader
Loader.num_workers = 4

```

O valor padrão é 2 para dar uma experiência de usuário suave. Você poderia aumentar o número de Workers, dessa forma, todas as imagens serão carregadas mais rapidamente, porém, o usuário não será capaz de usar o aplicativo durante o carregamento. Antes da versão 1.6.0, o número padrão era 20, e carregar muitas imagens Full-HD acabar por bloquear totalmente

o aplicativo.

Novo na versão 1.6.0.

pause()

Pausar o carregador, pode ser útil durante as interações.

Novo na versão 1.6.0.

resume()

Retomar o carregador, depois de *pause()*.

Novo na versão 1.6.0.

run(*largs)

Loop principal para o carregador.

start()

Inicia o thread/processo de carregamento.

stop()

Para o thread/processo de carregamento.

class kivy.loader.ProxyImage(arg, **kwargs)

Bases: *kivy.core.image.Image*

Imagen retornada pela função *Loader.image()*.

Properties

loaded: bool, e o padrão é False Esse valor pode ser *True* se a imagem já estiver em cache.

Events

on_load Disparado quando a imagem é carregada ou alterada.

on_error Disparado quando a imagem não pode ser carregada. *Error:* dados da exceção que irá ocorrer

4.1.4 Atlas

Novo na versão 1.1.0.

Atlas gerencia mapeamento de imagens, organizando múltiplas imagens em um só lugar. Com isso, você reduz o número de imagens carregadas e acelera a inicialização do aplicativo. Este módulo contém a classe *Atlas* e processamento por linha de comando para criar um *Atlas* a partir de arquivos PNG. A função linha de comando requer a biblioteca Pillow, ou o obsoleto Python Imaging Library (PIL), instalado.

Um *Atlas* é composto por 2 ou mais arquivos:

- arquivo json (.atlas) que contém o nome do arquivo da imagem e a textura do atlas.

- um ou vários arquivos de imagem contendo texturas referenciadas pelo arquivo .atlas.

Definição de arquivos .atlas

Um arquivo com `<basename>.atlas` é um arquivo json formatado como este:

```
{
    "<basename>-<index>.png": {
        "id1": [ <x>, <y>, <width>, <height> ],
        "id2": [ <x>, <y>, <width>, <height> ],
        # ...
    },
    # ...
}
```

Exemplo de um Kivy `data/images/defaulttheme.atlas`:

```
{
    "defaulttheme-0.png": {
        "progressbar_background": [431, 224, 59, 24],
        "image-missing": [253, 344, 48, 48],
        "filechooser_selected": [1, 207, 118, 118],
        "bubble_btn": [83, 174, 32, 32],
        # ... and more ...
    }
}
```

Neste exemplo, “`defaulttheme-0.png`” é uma imagem grande, com os pixels no retângulo de (431, 224) para (431 + 59, 224 + 24) usado como `atlas://data/images/defaulttheme/progressbar_background` em qualquer parâmetro de imagem.

Como criar um Atlas

Aviso: Para criação de atlas é requerido a biblioteca Pillow (ou a biblioteca descontinuada Imaging/PIL). Esse requerimento será removido no futuro pois o Kivy core Image será capaz de suportar o carregamento, blitting, e salvamento de operações.

Esse módulo pode ser usado diretamente para criar arquivos atlas com o comando:

```
$ python -m kivy.atlas <basename> <size> <list of images...>
```

Supondo que você tenha uma lista de imagens e você queria colocar no Atlas. O diretório chamado `images` com lotes de arquivo png 64x64 dentro:

```
$ ls  
images  
$ cd images  
$ ls  
bubble.png bubble-red.png button.png button-down.png
```

Você pode combinar todos os png's em um e gerar um arquivo atlas com:

```
$ python -m kivy.atlas myatlas 256x256 *.png  
Atlas created at myatlas.atlas  
1 image has been created  
$ ls  
bubble.png bubble-red.png button.png button-down.png myatlas.atlas  
myatlas-0.png
```

Como você pode ver, nos temos 2 novos arquivos: `myatlas.atlas` e `myatlas-0.png`. `myatlas-0.png` é um novo arquivo 256x256 .png composto por todas as suas imagens.

Nota: Quando utilizamos o script, os ids referenciados no atlas são os nomes bases das imagens sem extenção. assim, se você for dar nome a um arquivo `../images/button.png`, o id para essa imagem será `button`.

Se você precisar de informações do caminho incluso, você deve incluir `use_path` como no exemplo a seguir:

```
$ python -m kivy.atlas use_path myatlas 256 *.png
```

Nesse caso, o id para `../images/button.png` será `images_button`

Como utilizar um Atlas

Normalmente é necessário especificar o caminho da imagem:

```
a = Button(background_normal='images/button.png',  
           background_down='images/button_down.png')
```

No exemplo anterior, nos tinhemos criado o atlas contendo imagens e colocamos em `images/myatlas.atlas`. Você precisa olhar a url de notação

```
a = Button(background_normal='atlas://images/myatlas/button',  
           background_down='atlas://images/myatlas/button_down')
```

Quer dizer, o caminho para a imagem é substituído por:

```
atlas://path/to/myatlas/id
# will search for the ``path/to/myatlas.atlas`` and get the image_
↳ ``id``
```

Nota: Na URL do *Atlas*, não há necessidade de adicionar a extensão `.atlas`. Ele será anexado automaticamente ao nome do arquivo.

Manual de uso dos Atlas

```
>>> from kivy.atlas import Atlas
>>> atlas = Atlas('path/to/myatlas.atlas')
>>> print(atlas.textures.keys())
['bubble', 'bubble-red', 'button', 'button-down']
>>> print(atlas['button'])
<kivy.graphics.texture.TextureRegion object at 0x2404d10>
```

class kivy.atlas.Atlas(*filename*)

Bases: `kivy.event.EventDispatcher`

Gerenciar a textura do atlas. Ver a documentação para maiores detalhes.

static create(*outname*, *filenames*, *size*, *padding*=2, *use_path*=False)

Esse método pode ser usado para manualmente criar um atlas a partir de um conjunto de imagens.

Parameters

***outname*: str** Nome de base usado na criação de `.atlas` e imagens associadas a -<*idx*>.png.

***filenames*: list** Lista de nomes de arquivos a serem adicionados num atlas.

***size*: int ou list (width, height)** Tamanho da imagem do Atlas.

***padding*: int, padrão é 2** Padding a ser posto ao redor de cada imagem.

Seja cuidadoso. Se você estiver usando um preenchimento < 2, você pode ter problemas com as bordas das imagens. Por causa da linearização OpenGL, ele pode usar os pixels da imagem adjacente

Se você estiver usando um preenchimento ≥ 2 , nós iremos gerar automaticamente uma “borda” de 1px em torno da sua imagem. Se você olhar para o resultado, não se assuste se a imagem de dentro não é exatamente a mesma que a sua :).

use_path: bool, o padrão é *False*Se *True*, o caminho relativo dos nomes de arquivos png de origem será incluído nos atlas ids em vez de apenas nos nomes dos arquivos. Os pontos e as barras principais serão excluídos e todas as outras barras no caminho serão substituídas por sublinhados. Por exemplo, se *use_path* for *False* (o padrão) e o nome do arquivo for' *../data/tiles/green_grass.png*', o id será' *green_grass*'. Se *use_path* for *True*, será' *data_tiles_green_grass*'.

Alterado na versão 1.8.0: Parâmetro *use_path* adicionado

filename

Nome do arquivo atual do Atlas.

filename é um *AliasProperty* e o padrão é *None*.

original_textures

Lista de texturas originais do Atlas (que contêm *textures*).

original_textures é um *ListProperty* e o padrão é *[]*.

Novo na versão 1.9.1.

textures

Lista das texturas disponibilizadas com o Atlas.

textures é um *DictProperty* e o padrão é *{}*.

4.1.5 Gerenciador de Cache

O gerenciador de cache pode ser usado para armazenar objeto Python anexados a uma chave única. O cache pode ser controlado de duas formas: como um objeto que determina o limite ou um timeout (tempo de vida).

Por exemplo, criamos um novo cache com um limite de 10 objetos e um timeout de 5 segundos.

```
# register a new Cache
Cache.register('mycache', limit=10, timeout=5)

# create an object + id
key = 'objectid'
instance = Label(text=text)
Cache.append('mycache', key, instance)

# retrieve the cached object
instance = Cache.get('mycache', key)
```

Se a instância é NULL, o cache pode ter sujeira porque você não usou o label para 5 segundos e você atingiu o limite.

class kivy.cache.Cache

Bases: object

Veja o módulo da documentação para maiores informações.

static append (category, key, obj, timeout=None)

Adicionar um novo objeto ao cache.

Parameters

'category': strIdentificador da categoria

'key': strIdentificador único de um objeto a ser armazenado.

'obj': objectObjeto pra armazenar em cache.

timeout: double (opcional)Tempo após o qual excluir o objeto se ele não tiver sido utilizado. Se *None*, nenhum timeout é aplicado.

static get (category, key, default=None)

Pega um objeto desde o cache.

Parameters

'category': strIdentificador da categoria

'key': strIdentificador único de um objeto a ser armazenado.

default: qualquer coisa, padrão é NoneValor padrão a ser retornado se a chave não for encontrada

static get_lastaccess (category, key, default=None)

Pega o a hora do último acesso ao objeto em cache.

Parameters

'category': strIdentificador da categoria

'key': strIdentificador único de um objeto a ser armazenado.

default: qualquer coisa, padrão é NoneValor padrão a ser retornado se a chave não for encontrada

static get_timestamp (category, key, default=None)

Pega a otimestamp do objeto em cache.

Parameters

'category': strIdentificador da categoria

'key': strIdentificador único de um objeto a ser armazenado.

default: qualquer coisa, padrão é NoneValor padrão a ser retornado se a chave não for encontrada

static print_usage()

Imprime o uso do cache no console.

static register(category, limit=None, timeout=None)

Registra uma nova categoria no cache com um limite específico.

Parameters

'category': str Identificador da categoria

'limit': Int (opcional) Número máximo de objetos permitidos no cache. Se *None*, nenhum limite é aplicado.

'timeout': double (opcional) Tempo após o qual excluir o objeto se ele não tiver sido utilizado. Se *None*, nenhum timeout é aplicado.

static remove(category, key=None)

Limpar o cache

Parameters

'category': str Identificador da categoria

'key': str (opcional) Identificador único do objeto a ser armazenado. Se este argumento não for fornecido, toda a categoria será removida.

4.1.6 Objeto Clock

O objeto **Clock** permite você agendar uma função e chama-la no futuro, uma ou várias vezes ou mesmo, repetidas vezes num intervalo determinado. Para obter o tempo decorrido entre o agendamento e a chamada através do argumento *dt*:

```
# dt means delta-time
def my_callback(dt):
    pass

# call my_callback every 0.5 seconds
Clock.schedule_interval(my_callback, 0.5)

# call my_callback in 5 seconds
Clock.schedule_once(my_callback, 5)

# call my_callback as soon as possible (usually next frame.)
Clock.schedule_once(my_callback)
```

Nota: Se o callback retornar *False*, a agendamento será cancelado e não repetirá.

Se você deseja agendar uma função pra ser invocada com um argumento padrão, pos usar o módulo do Python **functools.partial**.

```

from functools import partial

def my_callback(value, key, *largs):
    pass

Clock.schedule_interval(partial(my_callback, 'my value', 'my key'),_
    →0.5)

```

Por outro lado, caso quiras agendar uma função que não aceite o argumento *dt*, você pode usar uma expressão lambda **lambda** para escrever uma pequena função que não aceita *dt*. Por exemplo:

```

def no_args_func():
    print("I accept no arguments, so don't schedule me in the clock"
    →"")
Clock.schedule_once(lambda dt: no_args_func(), 0.5)

```

Nota: Você não pode agendar um função anônima a meno que você mantenha a referência dela. É melhor adicionar **args* à definição da sua função, dessa forma, é possível enviar uma quantidade arbitrária de parâmetros.

Importante: O callback é uma referência fraca: você é o responsável por manter a referência ao objeto/callback original. Se você não mantiver a referência, o ClockBase não executará o seu callback. Por exemplo:

```

class Foo(object):
    def start(self):
        Clock.schedule_interval(self.callback, 0.5)

    def callback(self, dt):
        print('In callback')

# A Foo object is created and the method start is called.
# Because no reference is kept to the instance returned from Foo(),
# the object will be collected by the Python Garbage Collector and
# your callback will be never called.
Foo().start()

# So you should do the following and keep a reference to the_
→instance
# of foo until you don't need it anymore!
foo = Foo()
foo.start()

```

Agenda Antes do Frame

Novo na versão 1.0.5.

Algumas vezes precisarás agendar um callback ANTES do próximo frame. A partir de 1.0.5, podes utilizar um timeout de -1:

```
Clock.schedule_once(my_callback, 0) # call after the next frame  
Clock.schedule_once(my_callback, -1) # call before the next frame
```

O Clock executará todos os callbacks com um timeout de -1 antes do próximo frame mesmo se adicionares novos callbacks com -1 desde a execução do callback. De qualquer forma, a classe **Clock** tem um limite de iteração para esses callbacks: por padrão é igual a 10.

Se agendares um callback que agende um callback que agende um.... etc... mais de 10 vezes, isso deixará o loop e enviará uma mensagem de aviso ao console, em seguida, continuará depois do próximo frame. Essa é uma implementação para prevenir bugs que perdurem ou colidam com a aplicação.

Se você precisar aumentar o limite, ajuste a propriedade `max_iteration` property:

```
from kivy.clock import Clock  
Clock.max_iteration = 20
```

Eventos disparados

Novo na versão 1.0.5.

Um evento acionado é uma forma de adiar a invocação de um callback. Isso funciona exatamente como `schedule_once()` e `schedule_interval()`, exceto que não há o agendamento imediatamente do callback. Em vez disso, o retorno da chamada é programado usando a **ClockEvent** retornado por ele. Isso garante a você poder invocar o evento várias vezes, sem que seja agendado mais do que uma vez. Isso não acontece com `Clock.schedule_once()`:

```
# will run the callback twice before the next frame  
Clock.schedule_once(my_callback)  
Clock.schedule_once(my_callback)  
  
# will run the callback once before the next frame  
event = Clock.create_trigger(my_callback)  
event()  
event()  
  
# will also run the callback only once before the next frame  
event = Clock.schedule_once(my_callback) # now it's already  
→scheduled
```

```
event() # won't be scheduled again  
event()
```

Além disso, é mais conveniente criar e vincular um evento por gatilho do que usar numa função `Clock.schedule_once()`:

```
from kivy.clock import Clock  
from kivy.uix.widget import Widget  
  
class Sample(Widget):  
    def __init__(self, **kwargs):  
        self._trigger = Clock.create_trigger(self.cb)  
        super(Sample, self).__init__(**kwargs)  
        self.bind(x=self._trigger, y=self._trigger)  
  
    def cb(self, *largs):  
        pass
```

Mesmo se x e y mudarem em um frame, o callback é executado somente uma única vez.

`CyClockBase.create_trigger()` tem um parâmetro de “limite de tempo” que se comporta exatamente como `CyClockBase.schedule_once()`.

Alterado na versão 1.10.0: `CyClockBase.create_trigger()` agora tem um parâmetro `interval`. Se `False`, o padrão, será criado um evento similar a `CyClockBase.schedule_once()`. Do contrário será criado um evento similar a `CyClockBase.schedule_interval()`.

Desagendamento

Um evento agendado com `CyClockBase.schedule_once()`, `CyClockBase.schedule_interval()`, ou com `CyClockBase.create_trigger()` e então invocado pode ser desagendado várias vezes. Por exemplo:

```
def my_callback(dt):  
    pass  
  
# call my_callback every 0.5 seconds  
event = Clock.schedule_interval(my_callback, 0.5)  
  
# call my_callback in 5 seconds  
event2 = Clock.schedule_once(my_callback, 5)  
  
event_trig = Clock.create_trigger(my_callback, 5)  
event_trig()  
  
# unschedule using cancel  
event.cancel()
```

```

# unschedule using Clock.unschedule
Clock.unschedule(event2)

# unschedule using Clock.unschedule with the callback
# NOT RECOMMENDED
Clock.unschedule(my_callback)

```

A melhor forma para desagendar um callback é com `ClockEvent.cancel()`. `CyClockBase.unschedule()` é principalmente um apelido para a função que faz isso. De qualquer, se a própria função de callback é passada por `CyClockBase.unschedule()`, ela terá todas as suas instâncias de callback desagendadas (provê `all` é True, o padrão, outras, além da primeira serão removidas).

Invocar `CyClockBase.unschedule()` sobre um evento original de callback é altamente desencorajado, porque isso é significantemente mais lento do que usar o evento.

Threading e ordem do Callback

Beginning with 1.10.0, all the events scheduled for the same frame, e.g. all the events scheduled in the same frame with a `timeout` of 0, will be executed in the order they were scheduled.

E também, todo os desagendamentos e chamadas a métodos são completamente thread safe e podem ser seguramente utilizados desde threads externos.

Como uma consequência, invocar `CyClockBase.unschedule()` com um callback original é agora significativamente mais lento e altamente desencorajado. Ao invés disso, o evento retornado deve ser usado para cancelar. Como um tradeoff, todos os outros métodos estão agora significativamente mais rápido do que anteriormente.

Detalhes Avançados de Clock

A seção seguinte entrará nos detalhes internos do Clock do Kivy como também as opções de Clock. Isso foi planejado somente para usuários avançados.

Fundamentalmente, o relógio Kivy tenta executar qualquer callback programado ritmicamente conforme determinado pelo fps especificado (quadro por segundo, veja “maxfps” em: mod: `~kivy.config`). Isto é, idealmente, suponha um exemplo com fps desejado de 30, o relógio irá executar os callbacks em intervalos de 1/30 segundos, ou a cada 33,33 ms. Todos os retornos de chamada em uma moldura recebem o mesmo carimbo de data / hora, ou seja, o “`dt`” passado para o retorno de chamada são todos iguais e é a diferença de tempo entre o início deste e o quadro anterior.

Devido ao indeterminismo inherente, os frames não ocorrem exatamente em intervalos de FPS e `dt` poder ficar acima ou abaixo do FPS desejado. Além disso, uma vez que

timeout está “suficientemente próximo” para o timeout desejado, conforme determinado internamente, Kivy executará os frames de callbacks atuais, mesmo quando o “tempo real” não tiver decorrido a quantidade de tempo limite.

Kivy offers now, since `1.10.0`, multiple clocks with different behaviors.

Padrão Clock

O Clock padrão (`default`) funciona como descrito acima. Quando um callback com um timeout igual ou não a zero é agendado, ele será executado no mesmo frame que esteja perto do timeout, e isso é uma função do FPS. Assim, um timeout igual a zero ainda resultaria num atraso de um frame ou aproximadamente $1/\text{FPS}$, geralmente, um pouco menos, porém, as vezes, dependendo do uso da CPU ou de outros eventos agendados para este quadro.

Num teste usando um FPS de 30, um callback com timeout de 0, 0.001 e 0.05 resultou num retardo em torno de 0.02487, 0.02488 e 0.05011 segundos respectivamente. Quando testado com um FPS de 600, o atraso para 0.05 foi semelhante, exceto que o desvio padrão foi reduzido resultando numa melhor exatidão global.

Clock Ininterrumpível

O Clock padrão sofre com o problema de quantização, uma vez que os Frames ocorrem somente em intervalos e qualquer tempo limite programado não será capaz de ocorrer durante um intervalo. Por exemplo, com o timeout de 0.05, enquanto a média foi 0.05011, os seus valores variaram entre 0,02548 - 0,07348 com um desvio padrão de 0,002. Além disso, há o timeout mínimo foi cerca de 0,02487.

O relógio interrompível (“interrupt”) executará tempos limite mesmo durante um frame. Assim, um tempo limite de zero será executado o mais rapidamente possível e, da mesma forma, um tempo limite não-zero será executado mesmo durante o intervalo

Este relógio e todos os relógios descritos a seguir têm uma opção: attr: `ClockBaseInterruptBehavior.interupt_next_only`. Quando True, qualquer comportamento - novo comportamento só se aplicará aos callbacks com um tempo limite de zero. Os tempos limite não-zero se comportarão como no relógio padrão, por exemplo, para este relógio quando True, somente zero timeouts serão executados durante o intervalo.

Em um teste usando um fps de 30, um retorno de chamada com um tempo limite de 0, 0,001 e 0,05, resultou em um retardo de retorno médio de 0,00013, 0,00013 e 0,04120 segundos, respectivamente quando: attr: `ClockBaseInterruptBehavior.interupt_next_only` foi False. Além disso, em comparação com o relógio padrão, o desvio padrão foi reduzido. Quando: attr: `ClockBaseInterruptBehavior.interupt_next_only` foi Verdadeiro, os valores foram 0.00010, 0.02414 e 0.05034, respectivamente.

Liberação do Clock

O relógio interrompível pode não ser ideal para todos os casos, porque todos os eventos são executados durante os intervalos e os eventos não são mais executados ritmicamente como múltiplos dos fps. Por exemplo, pode não haver qualquer benefício para os gráficos para atualizar em um sub-intervalo, portanto, a precisão adicional desperdiça CPU.

O relógio Livre (“ free_all”) resolve isso por ter “ Clock.xxx_free” versões de todos os métodos de programação Clock. Por ‘livre’, queremos dizer que o evento está livre do fps porque não é fps limitado. Por exemplo. : Meth: *CyClockBaseFree.create_trigger_free* corresponde a: meth:’ *CyClockBase.create_trigger*’. Somente quando um evento agendado usando os métodos “ Clock.xxx_free” estiver presente, o relógio irá interromper e executar os eventos durante o intervalo. Portanto, se nenhum evento “ free” estiver presente, o relógio se comporta como “ default”, caso contrário ele se comporta como “ interrupt”.

Em um teste usando um FPS de 30, um callback com tempo limite de 0s, 0.001s e 0.05s, resultou num retardo médio do callback de 0.00012s, 0.00017s e 0.04121s segundos, respectivamente, quando foi um evento livre e 0,02403s, 0,02405s e 0,04829s, respectivamente, quando não era.

Liberação somente do Clock

O relógio Livre executa todos os eventos quando um evento livre foi agendado. Isso resulta em eventos normais também sendo executados no meio do intervalo quando um evento livre é agendado. Por exemplo, acima, quando um evento livre estava ausente, um evento normal com um tempo limite de 0.001s foi adiado para 0.02405s. No entanto, se aconteceu de um evento livre também ser agendado, o evento normal foi apenas 0,00014s atrasado, o que pode ser indesejável.

O relógio apenas Livre (“ free_only”) resolve-o apenas executando eventos livres durante o intervalo e os eventos normais são sempre executados como com o relógio padrão. Por exemplo, na presença de um evento livre, um evento normal com um tempo limite de 0.001s ainda tinha um atraso de 0.02406. Assim, este relógio, trata eventos livres e normais independentemente, com eventos normais sempre sendo fps limitado, mas nunca os eventos livres.

Resumo

O tipo Clock do Kivy para usar pode ser definido com a opção `kivy_clock config`. Se `KIVY_CLOCK` está presente no ambiente que sobrescreve a seleção de configuração. Seus valores possíveis são os seguintes:

- Quando o `kivy_clock` é `default`, o clock normal, `ClockBase`, que limita os callbacks para à quantificação de `maxfps` - é usado.
- Quando `kivy_clock` é `interrupt`, um Clock sem interrupção `class:ClockBaseInterrupt`, que não limita quaisquer callbacks para o `maxfps` - é usado. Os callbacks serão executados a qualquer momento.
- Quando “`kivy_clock`“ é “`free_all`“, um relógio interrompível; `class: ClockBaseFreeInterruptAll`, que não limita quaisquer callbacks para o `maxfps` na presença de eventos livres, mas na sua ausência limita os eventos ao Fps intervalo de quantificação - é usado.
- Quando “`kivy_clock`“ é “`free_only`“, um relógio interrompível; `class: ClockBaseFreeInterruptAll`, que trata os eventos livres e normais independentemente; Eventos normais são fps limitados enquanto eventos livres não são - é usado.

`kivy.clock.Clock = None`

A instância do Clock do Kivy. Veja o módulo da documentação para maiores informações.

`class kivy.clock.ClockEvent`

Bases: `object`

Uma classe que descreve um callback programado com Kivy `Clock`. Essa classe nunca é criada pelo usuário; em vez disso, o Kivy cria e retorna uma instância dessa classe ao programar um callback.

Um evento pode ser disparado (agendado) chamando-o. Caso já esteja programado, nada acontecerá, caso contrário, o mesmo será agendado. Por exemplo:

```
event = Clock.schedule_once(my_callback, .5)
event() # nothing will happen since it's already scheduled.
event.cancel() # cancel it
event() # now it's scheduled again.
```

`cancel()`

Cancela o callback se ele tiver agendado para ser chamado. Se não tiver agendado, nada acontecerá.

`clock`

A classe `CyClockBase` é uma instância associada com o evento.

`get_callback()`

Retorna o callback associado com o evento. Os Callbacks são armazenados com uma referência indireta para que ele não tenha objetos vivos. Se o callback morrer, `None` será retornado.

`is_triggered`

Retorna se o evento está agendado para que o callback seja executado pela thread do Kivy.

loop

Se este evento se repete em intervalos de **timeout**.

next

A próxima **ClockEvent** na ordem que eles foram agendados.

prev

O anterior **ClockEvent** na ordem em que eles foram agendados.

release()

(método interno) Converte o callback numa referência indireta.

tick()

(Método interno) Processa o evento para o thread do kivy.

timeout

A duração após o agendamento quando o callback deve ser executado.

class kivy.clock.FreeClockEvent

Bases: `kivy._clock.ClockEvent`

O evento retornado pelo método `Clock.XXX_free` da **CyClockBaseFree**. Ele armazena se o evento foi agendado como um evento livre.

free

Se este evento foi agendado como um evento livre.

class kivy.clock.CyClockBase

Bases: `object`

Objeto de Clock base com suporte a eventos.

clock_resolution

Se o tempo restante até o tempo limite do evento for menor que **clock_resolution**, o `Clock` executará o callback mesmo que não tenha exatamente expirado.

Se -1, o padrão, a resolução será calculada a partir do `maxfps` da configuração. Caso contrário, o valor fornecido será usado. O padrão é -1.

create_trigger()

Cria um evento de gatilho. Veja a documentação do módulo para maiores informações.

RetornaUma instância de **ClockEvent**. Para agendar o callback dessa instância, você pode chama-la.

Novo na versão 1.0.5.

Alterado na versão 1.10.0: `interval` foi adicionado. Se `True`, ele cria um evento que é invocada cada <timeout> segundos similar a **schedule_interval()**. O padrão é `False`.

get_events()

Retorna a lista de instâncias de **ClockEvent** atualmente agendadas.

get_min_timeout()

Retorna o tempo restante desde o início do frame atual para o evento com o menor tempo limite.

get_resolution()

Retorna a resolução mínima do Clock. Isso é uma função de *clock_resolution* e *maxfps* é provido na configuração.

max_iteration

O número máximo de iterações de callback no final do Frame, antes do próximo Frame. Se ocorrerem mais iterações, será emitido um aviso.

on_schedule()

Função que é chamada internamente toda vez que um evento é disparado para este Clock. Ele toma o evento como um parâmetro.

schedule_interval()

Agenda um evento para ser chamado a cada <timeout> segundos.

RetornaUma instâncias de *ClockEvent*. Como oposição a *create_trigger()* que somente cria um evento de gatilho, este método também agenda-o.

schedule_once()

Agenda um evento em <timeout> segundos. Se <timeout> não foi definido ou é igual a 0, o callback será chamado após o próximo frame ser renderizado.

RetornaUma instâncias de *ClockEvent*. Como oposição a *create_trigger()* que somente cria um evento de gatilho, este método também agenda-o.

Alterado na versão 1.0.5: Se o timeout é igual a -1, o callback será chamado antes do próximo frame (no *tick_draw()*).

unschedule()

Remove um evento previamente agendado.

Parameters

callback: ***ClockEvent* ou um callable**. Se isso for uma instância de *ClockEvent*, então o callback associado a este evento será cancelado caso o mesmo seja agendado.

Se ele é um callable, então o callable será desagendado se ele estiver agendado.

Aviso: Passa a função de callback ao invés da retornada *ClockEvent* que resultará em um agendamento significativamente mais lento.

all: bool Se *True* e se o *callback* for chamado, todas as instâncias deste callable não serão agendadas (isto é, se este callable foi agendado várias vezes). O padrão é *True*.

Alterado na versão 1.9.0: O parâmetro *all* foi adicionado. Antes, ele se comportava como se *all* fosse *True*.

class kivy.clock.CyClockBaseFree

Bases: `kivy._clock.CyClockBase`

Uma classe de Clock que suporta o agendamento de eventos livres além de eventos normais.

Cada um dos métodos `create_trigger()`, `schedule_once()`, e `schedule_interval()`, que criam eventos normais, possuem um método correspondente para a criação de eventos livres.

create_trigger_free()

Similar ao `create_trigger()`, mas ao invés disso, criam um evento livre.

get_min_free_timeout()

Retorna o tempo restante desde o início de um frame atual para um evento livre com o menor tempo limite.

schedule_interval_free()

Similar ao `schedule_interval()`, mas ao invés disso, cria um evento livre.

schedule_once_free()

Similar ao `schedule_once()`, mas ao invés disso, cria um evento livre.

class kivy.clock.ClockBaseBehavior(kwargs)**

Bases: `object`

A base do Clock do Kivy.

MIN_SLEEP = 0.005

O tempo mínimo para dormir. Se o tempo restante for menor do que este, o event loop continuará.

frames

Número de frames internos (não necessariamente desenhados) desde o início do Clock.

Novo na versão 1.8.0.

frames_displayed

Número de frames exibidos desde o início do Clock.

frametime

Tempo gasto entre o último frame e o frame atual (em segundos).

Novo na versão 1.8.0.

get_boottime()

Pega o tempo em segundo desde o início da aplicação.

get_fps()

Pega a média atual em FPS calculada pelo Clock.

get_rfps()

Pega o atual “real” FPS calculado pelo Clock. Este contador reflete o frame-rate real mostrado na tela.

Em contraste ao `get_fps()`, essa função retorna um contador do número de frames, não a média de frames por segundo.

get_time()

Obtém o último tick feito pelo Clock.

idle()

(interno) espera aqui até o próximo frame.

tick()

Avança o Clock para o próximo passo. Deve ser chamado a cada novo frame. O padrão Clock tem uma função `tick()` chamada pelo núcleo do framework Kivy.

tick_draw()

Marca o contador de desenho.

time = <functools.partial object>**usleep(microseconds)**

Dorme por um número de micro segundos.

```
class kivy.clock.ClockBaseInterruptBehavior(interrupt_next_only=False,  
                                            **kwargs)
```

Bases: `kivy.clock.ClockBaseBehavior`

Um Clock Kivy que pode ser interrompido durante um frame de execução de eventos.

```
class kivy.clock.ClockBaseInterruptFreeBehavior(**kwargs)
```

Bases: `kivy.clock.ClockBaseBehavior`

Uma classe base para o Clock que interrompe o intervalo de inatividade para eventos livres.

```
class kivy.clock.ClockBase(**kwargs)
```

Bases: `kivy.clock.ClockBaseBehavior`, `kivy._clock.CyClockBase`

O Clock padrão do default. Veja o módulo para maiores informações.

```
class kivy.clock.ClockBaseInterrupt(interrupt_next_only=False,  
                                    **kwargs)
```

Bases: `kivy.clock.ClockBaseInterruptBehavior`, `kivy._clock.CyClockBase`

O Clock Kivy `interrupt`. Veja o módulo para maiores detalhes.

```
class kivy.clock.ClockBaseFreeInterruptAll(**kwargs)
    Bases:      kivy.clock.ClockBaseInterruptFreeBehavior,      kivy.
                _clock.CyClockBaseFree
```

O Kivy Clock `free_all`. Veja o módulo para maiores detalhes.

```
class kivy.clock.ClockBaseFreeInterruptOnly(**kwargs)
    Bases:      kivy.clock.ClockBaseInterruptFreeBehavior,      kivy.
                _clock.CyClockBaseFree
```

O Kivy Clock `free_only`. Veja o módulo para maiores detalhes.

`kivy.clock.mainthread(func)`

Decorador que agendará a chamada para a função para o próximo frame disponível na thread principal. Isso pode ser útil quando usares `UrlRequest` ou quando fizeres a programação de Thread: não precisas fazer nenhum trabalho relacionado a thread do OpenGL.

Observe que este método retornará diretamente e nenhum resultado pode ser retornado:

```
@mainthread
def callback(self, *args):
    print('The request succeeded!',
          'This callback is called in the main thread.')

self.req = UrlRequest(url='http://...', on_success=callback)
```

Novo na versão 1.8.0.

4.1.7 Módulo de Compatibilidade com Python 2.7 e > 3.3

Este módulo fornece um conjunto de vários tipos de utilitário e funcionalidades para otimização e auxilio na escrita de código compatível com as versões 2x e 3x.

`kivy.compat.PY2 = True`

True se esta versão do Python for 2.x

`kivy.compat.clock()` → floating point number

Um Clock com a resolução mais alta disponível no seu Sistema Operacional.

`kivy.compat.string_types`

Um tipo de utilitário para detectar String em Python 2/3 de um modo amigável.
Por exemplo:

```
if isinstance(s, string_types):
    print("It's a string or unicode type")
```

```
else:
    print("It's something else.")
```

apelido de `basestring`

`kivy.compat.isclose(a, b, rel_tol=1e-09, abs_tol=0.0)`

Me se dois floats estão “fechados” um ao outro. Identico a <https://docs.python.org/3.6/library/math.html#math.isclose>, para versões antigas do Python.

4.1.8 Objeto de Configuração

O objeto da `Config` é uma instância modificada do `ConfigParser` do Python . Veja a [documentação do ConfigParser](#) para maiores informações.

A biblioteca Kivy possui um arquivo de configuração que determina as configurações padrões. A fim de mudar estas definições, você pode alterar este arquivo manualmente ou utilizar o objecto `Config`. Por favor veja a secção :ref:`Configure Kivy` para mais informação.

Aplicando configurações

As opções de configuração controlam a inicialização da classe :Cass:`'~kivy.app.App'`. Por forma a evitar situações onde as definições não funcionem ou não estão aplicadas antes da criação da janela (como definir um tamanho inicial para a janela), :meth:`Config.set <kivy.config.ConfigParser.set>` deverá ser usado antes de importar quaisquer outros módulos Kivy. Idealmente, isto significa defini-los logo no inicio do seu script `main.py` .

Alternativamente, poderá guardar estas definições permanentemente utilizando :meth:`Config.set <ConfigParser.set>` e seguidamente `Config.write`. Neste caso, necessitará de reiniciar a aplicação, para que as alterações sejam assumidas. Tome nota de que esta abordagem terá efeito em todas as aplicações kivy no seu sistema.

Utilização do objecto `Config`

Para ler os símbolos de configuração de uma secção em particular:

```
>>> from kivy.config import Config
>>> Config.getint('kivy', 'show_fps')
0
```

Alterar a configuração e guardá-la:

```
>>> Config.set('postproc', 'retain_time', '50')
>>> Config.write()
```

Para informação na configuração da sua classe :class:'~kivy.app.App', por favor veja a secção [Configuração da aplicação](#).

Alterado na versão 1.7.1: O ConfigParser deverá funcionar correctamente com o utf-8. Os valores serão convertidos de ascii para unicode apenas quando necessário. O método get() retorna “Strings” no formato utf-8 .

Disponíveis símbolos de configuração

kivy

desktop: int, 0 or 1 Esta opção controla características específicas do sistema operativo desktop, tais como habilitar “scroll-bar” arrastáveis nas “scroll views”, e desabilitando as bolhas nos TextInput etc. 0 é desabilitado, 1 é habilitado.

exit_on_escape: int, 0 or 1 Habilita a saída o kivy, quando é premida a tecla escape. 0 é desabilitado, 1 é habilitado.

pause_on_minimize: int, 0 or 1 Se definido igual a 1, o main loop é pausado e o evento *on_pause* é enviado quando a janela está minimizada. Essa opção destina-se a ser usada somente em Desktop. O padrão é 0.

keyboard_layout: string Identificador do layout a utilizar.

'keyboard_mode': string Especifica o modo do teclado a usar. Pode ser um dos seguintes:

- “- Deixar o Kivy usar a melhor opção para a plataforma corrente.
- ‘system’ - teclado físico.
- ‘dock’ - um teclado virtual acoplado a um lado do ecran.
- ‘multi’ - um teclado virtual para todo o pedido a partir de qualquer widget.
- ‘systemanddock’ - teclado virtual acoplado, mais o teclado físico.
- ‘systemandmulti’ - analogamente.

log_dir: string Caminho para a directoria de log.

'log_enable': int, 0 or 1 Activar o logging em ficheiros. 0 está desabilitado, 1 está habilitado.

log_level: string, um dos ‘trace’, ‘debug’, ‘info’, ‘warning’, ‘error’ or ‘critical’
Define o nível mínimo de log a usar.

log_name: string A “String” formatada a usar para o nome de ficheiro do log.

log_maxfiles: int Keep log_maxfiles recent logfiles while purging the log directory. Set 'log_maxfiles' to -1 to disable logfile purging (eg keep all logfiles).

Nota: You end up with 'log_maxfiles + 1' logfiles because the logger adds a new one after purging.

window_icon: string Caminho para o icon de janela. Use isto, se preferir substituir o icon por omissão do pygame.

postproc

double_tap_distance: float Distancia máxima permitida para um duplo toque, normalizado no intervalo 0 - 1000.

double_tap_time: int Tempo permitido para a detecção de um duplo toque, em milisegundos.

'ignore': lista de tuplas Lista de regiões onde novos toques serão ignorados. Estes simbolos de configuração poderão ser utilizados para resolver problemas de hotspot com hardware DIY. O formato da lista deverá ser:

```
ignore = [(xmin, ymin, xmax, ymax), ...]
```

Todos os valores deverão estar compreendidos no intervalo 0 - 1.

'jitter_distance': int Máxima distancia para a detecção de click's rápidos (jitter), normalizado no intervalo de 0 - 1000.

jitter_ignore_devices: string, separados com virgulas Lista de dispositivos a ignorar para a detecção de click's rápidos.

retain_distance: int Se o toque se mover mais do que está indicado em retain_distance, este não será retido. O argumento deverá ser um inteiro no intervalo 0 - 1000.

retain_time: int Tempo permitido para reter o toque, em milisegundos.

triple_tap_distance: float Distância máxima permitida para um toque triplo, normalizado dentro do intervalo de 0 - 1000.

'triple_tap_time': int Tempo permitido para a detecção de um toque triplo, em milisegundos.

graphics

borderless: int, one of 0 or 1 Se definido como 1, remove a borda/decoração da janela. O redimensionamento da janela

também deve ser desativado para ocultar o redimensionamento da borda.

window_state: string, um denre ‘visible’, ‘hidden’, ‘maximized’ ou ‘minimized’

Define o estado da janela, o padrão é ‘visible’. Esta opção está disponível somente para provedores de janelas SDL2 e somente deve ser utilizado em Desktop OSes.

fbo: string, um dentre ‘hardware’, ‘software’ ou ‘force-hardware’

Seleciona o backend FBO para usar.

fullscreen: int ou string, um de 0, 1, ‘fake’ ou ‘auto’ Ativa fullscreen.

Se definido como 1, será usado uma resolução de *width* vezes *height* pixels. Se definido como *auto*, sua resolução do seu monitor atual será utilizada em vez disso. Isso é o mais provável que você queira. Se quiseres colocar a janela em outra tela, use *fake*, ou defina a opção ‘borderless’ na seção de gráficos, então ajuste *width*, ‘*height*’, *top* e ‘*left*’.

height: int Altura da:`class:~kivy.core.window.Window`, não será se *fullscreen* estiver definido como *auto*.

left: int Posição esquerda da `Window`.

maxfps: int, padrão é 60 Máximo de FPS permitido.

Aviso: Definir *maxfps* como 0 levará ao uso máximo da CPU.

‘multisamples’: int, o padrão é 2 Define o nível *MultiSample Anti-Aliasing* (MSAA) <http://en.wikipedia.org/wiki/Multisample_antialiasing>. Aumentando este valor resulta em um gráfico mais suave mas ao custo de tempo de processamento.

Nota: Este recurso é limitado pelo suporte de hardware do dispositivo e não terá efeito em dispositivo que não suportem o nível de MSAA solicitado.

position: string, uma das duas opções ‘auto’ ou ‘custom’ Posição da janela no visor. Se *auto* for usado, você não terá controle da posição inicial: *top* e *left* são ignorados.

show_cursor: int, um de 0 ou 1 Define se o cursor é ou não exibido sobre a janela.

top: int Posição superior da `Window`.

resizable: int, um de 0 ou 1 Se 0, a janela terá um tamanho fixo. Se 1, a janela será redimensionável.

rotation: int, um de 0, 90, 180 ou 270 Rotação da Window.

width: int Largura da Window, não será se *fullscreen* for definido como *auto*.

minimum_width: int Largura mínima para restringir a janela a (somente SDL2)

minimum_height: int Altura mínima para restringir a janela a (SDL2 somente)

min_state_time: float, o padrão é .035 Tempo mínimo para os Widgets exibirem um determinado estado visual. Este atributo é atualmente usado pelos Widgets como *DropDown* & *ButtonBehavior* para se certificarem que eles exibem seu estado visual atual para um tempo determinado.

kivy_clock: um dentre default, interrupt, free_all, free_only O tipo Clock a ser usado com Kivy. Veja o *kivy.clock*.

default_font: list, defaults to ['Roboto', 'data/fonts/Roboto-Regular.ttf', 'data/fonts/Roboto-Italic.ttf', 'data/fonts/Roboto-Bold.ttf', 'data/fonts/Roboto-BoldItalic.ttf']

Default font used for widgets displaying any text.

allow_screensaver: int, one of 0 or 1, defaults to 1 Allow the device to show a screen saver, or to go to sleep on mobile devices. Only works for the sdl2 window provider.

input Você pode criar um novo dispositivo de entrada usando esta sintaxe:

```
# example of input provider instance
yourid = providerid,parameters

# example for tuio provider
default = tuio,127.0.0.1:3333
mytable = tuio,192.168.0.1:3334
```

Ver também:

Verifique o provedor em *kivy.input.providers* para a sintaxe a ser usada dentro do arquivo de configuração.

widgets

scroll_distance: int Valor padrão da propriedade *scroll_distance* usado pelo Widget *ScrollView*. Verifique a documentação do Widget para maiores informações.

scroll_friction: float Valor padrão para a propriedade *scroll_friction* usada pelo Widget *ScrollView*. Verifique a documentação para maiores informações.

Obsoleto desde a versão 1.7.0: Por favor, ao invés, utilize `effect_cls`.

`scroll_timeout: int` Valor padrão da propriedade `scroll_timeout` usada pelo Widget `ScrollView`. Verifique a documentação do Widget para maiores informações.

`scroll_stoptime: int` Valor padrão da propriedade `scroll_stoptime` usada pelo Widget `ScrollView`. Veja a documentação para maiores informações.

Obsoleto desde a versão 1.7.0: Por favor, ao invés, utilize `effect_cls`.

`scroll_moves: int` Valor da propriedade `scroll_moves` usada pelo Widget `ScrollView`. Veja a documentação para maiores informações.

Obsoleto desde a versão 1.7.0: Por favor, ao invés, utilize `effect_cls`.

modules Você pode ativar módulos com esta sintaxe:

```
modulename =
```

Qualquer coisa após o = será passado para o módulo como argumentos. Veja a documentação específica do módulo para uma lista de argumentos aceitos.

Alterado na versão 1.10.0: `min_state_time` and `allow_screensaver` have been added to the `graphics` section. `kivy_clock` has been added to the `kivy` section. `default_font` has been added to the `kivy` section.

Alterado na versão 1.9.0: `borderless` e `window_state` foram adicionados a seção de gráficos. A configuração `fake` da opção `fullscreen` entrou em desuso, utilize agora a opção `borderless`. `pause_on_minimize` foi adicionado para a seção Kivy.

Alterado na versão 1.8.0: `systemanddock` e `systemandmulti` foram adicionados como valores possíveis para o `keyboard_mode` na seção Kivy. `exit_on_escape` foi adicionado à seção Kivy.

Alterado na versão 1.2.0: `resizable` foi adicionada para a seção gráficos.

Alterado na versão 1.1.0: TUO não escuta mais o padrão. Os ícone da janela não são copiados para o diretório do usuário. Você ainda pode definir um novo ícone de janela usando a configuração `window_icon`.

Alterado na versão 1.0.8: `scroll_timeout`, `scroll_distance` e `scroll_friction` foram adicionados. `list_friction`, `list_trigger_distance` e `list_friction_bound` foram removidos. `keyboard_type` e `keyboard_layout` foram removidos dos Widgets. `keyboard_mode` e `keyboard_layout` foram adicionados à seção Kivy.

`kivy.config.Config = None`

O objeto de configuração padrão do Kivy. Esta é uma instância de `ConfigParser` com o `name` definido para o ‘kivy’.

```
Config = ConfigParser(name='kivy')
```

`class kivy.config.ConfigParser(name= '')`

Bases: `ConfigParser`, `ConfigParser`, `object`

Classe `ConfigParser` melhorada que suporta a adição de seção padrão e valor padrão.

Por padrão a instância de kivy `ConfigParser`, `Config`, é chamada de ‘*kivy*’ e a instância de `ConfigParser` é usada pelo `App.build_settings` método chamado ‘*app*’.

Parameters

`name: string` O nome da instância. Veja o `name`. O padrão é ‘’.

Alterado na versão 1.9.0: Cada `ConfigParser` pode agora ser um `named`. Você pode obter o `ConfigParser` associado com o nome usando o `get_configparser()`. Além disso, você pode agora controlar os valores de configuração com a `ConfigParserProperty`.

Novo na versão 1.0.7.

`add_callback(callback, section=None, key=None)`

Adiciona um callback para ser chamado quando uma seção específica ou chave tiver sido modificada. Se você não especificar um seção ou chave, será chamado o callback para todas as seções/chaves modificadas.

Callbacks recebem 3 argumentos: a seção, a chave e o valor.

Novo na versão 1.4.1.

`adddefaultsection(section)`

Adicione uma seção caso a seção esteja ausente.

`static get_configparser(name)`

Retorna a instância de `ConfigParser` cujo nome é `name`, ou `None` se não for encontrada.

Parameters

`name: string` O nome da instância `ConfigParser` pra retornar.

`getdefault(section, option, defaultvalue)`

Obtém o valor de uma opção numa seção específica. Se não for encontrado, isso retornará o valor padrão.

`getdefaultint(section, option, defaultvalue)`

Obtém o valor de uma opção numa seção específica. Se não for encontrado,

isso retornará o valor padrão. O valor sempre será retornado como um integer.

Novo na versão 1.6.0.

name

O nome associado com esta instância de ConfigParser, se não''. O padrão é ''. Pode ser seguro modificar dinamicamente ou definir para ''.

Quando um ConfigParser está dado um nome, esse objeto de configuração pode ser recuperado usando `get_configparser()`. Além disso, essa instância de configuração também pode ser usada com uma instância `ConfigParserProperty` que define seu valor `config` pra este nome.

Definir mais do que um ConfigParser com o mesmo nome levantará uma exceção `ValueError`.

read(filename)

Um nome de arquivo somente leitura. Em contraste com o ConfigParser original do Python, este é capaz de ler apenas um arquivo por vez. A último arquivo de leitura será usado pelo método `write()`.

Alterado na versão 1.9.0: `read()` agora chama o callback se `read` alterou qualquer valor

remove_callback(callback, section=None, key=None)

Remove o callback adicionado com `add_callback()`. `remove_callback()` deve ser chamado com o mesmo parâmetro como em `add_callback()`.

Levanta uma exceção `ValueError` se não for encontrado.

Novo na versão 1.9.0.

set(section, option, value)

Funciona de maneira similar ao método se PythonConfigParser's, exceto que o valor é implicitamente convertido numa String.

setall(section, keyvalues)

Define vários pares key-value numa seção. Keyvalues deve ser um dicionário contendo os pares key-value a serem definido.

setdefault(section, option, value)

Define o valor padrão para um opção num seção específica.

setdefaults(section, keyvalues)

Define vários pares key-value numa seção. Keyvalues deve ser um dicionário contendo os novos key-values padrão.

update_config(filename, overwrite=False)

Atualiza a configuração baseada em um novo arquivo de configuração. Substitua os valores existente se o `overwrite` for `True`.

write()

Escreve a configuração para o último arquivo aberto usando o método **read()**.

Retorna *True* se a escrita finalizar com sucesso, senão, retorna *False*.

4.1.9 Contexto

Novo na versão 1.8.0.

Aviso: Isto é experimental e está sujeito a alterações todas as vezes que esta advertência esteja presente.

O Kivy tem algumas pequenas instâncias “globais” que são utilizadas diretamente por várias peças do framework: *Cache*, *Builder*, *Clock*.

TODO: documentar este módulo.

kivy.context.register_context(name, cls, *args, **kwargs)
Registra um novo contexto.

kivy.context.get_current_context()
Retorna o contexto atual.

4.1.10 Distribuidor de Eventos

Todas as classes que implementam eventos no Kivy utilizam o **EventDispatcher** que provê uma interface consistente para o registro e manipulação de handle de eventos.

Alterado na versão 1.0.9: A descoberta de propriedades e os métodos foram movidos de **Widget** para **EventDispatcher**.

class kivy.event.EventDispatcher
Bases: **kivy.event.ObjectWithUid**

Interface de distribuição de eventos genéricos.

Veja o módulo docstring para utilizar.

apply_property()

Adiciona propriedades a classes em tempo de execução. A função aceita argumentos com palavras chaves na forma de *prop_name=prop*, onde *prop* é uma instância de **Property** e *prop_name* é o nome do atributo da propriedade.

Novo na versão 1.9.1.

Aviso: Este método não é recomendado para uso comum porque você deve declarar as propriedades em sua classe ao invés de utilizar este método.

Por exemplo:

```
>>> print(wid.property('sticks', quiet=True))
None
>>> wid.apply_property(sticks=ObjectProperty(55, max=10))
>>> print(wid.property('sticks', quiet=True))
<kivy.properties.ObjectProperty object at 0x04303130>
```

bind()

Vincula um tipo de evento ou um callback de alguma propriedade.

Uso:

```
# With properties
def my_x_callback(obj, value):
    print('on object', obj, 'x changed to', value)
def my_width_callback(obj, value):
    print('on object', obj, 'width changed to', value)
self.bind(x=my_x_callback, width=my_width_callback)

# With event
def my_press_callback(obj):
    print('event on object', obj)
self.bind(on_press=my_press_callback)
```

Geralmente, callbacks de propriedades são invocados com a passagem de 2 argumentos (o objeto e o novo valor da propriedade) e um evento de callbacks com um argumento (o objeto). O exemplo abaixo ilustra o que foi dito.

O exemplo a seguir demonstra as várias formas de utilizar a função de vinculação numa aplicação completa:

```
from kivy.uix.boxlayout import BoxLayout
from kivy.app import App
from kivy.uix.button import Button
from functools import partial

class DemoBox(BoxLayout):
    """
        This class demonstrates various techniques that can be
        used for binding to
        events. Although parts could be made more optimal,
        advanced Python concepts
```

```

are avoided for the sake of readability and clarity.
"""

def __init__(self, **kwargs):
    super(DemoBox, self).__init__(**kwargs)
    self.orientation = "vertical"

    # We start with binding to a normal event. The only_
    ↪argument
        # passed to the callback is the object which we_
        ↪have bound to.
        btn = Button(text="Normal binding to event")
        btn.bind(on_press=self.on_event)

    # Next, we bind to a standard property change event.
    ↪ This typically
        # passes 2 arguments: the object and the value
        btn2 = Button(text="Normal binding to a property_
        ↪change")
        btn2.bind(state=self.on_property)

    # Here we use anonymous functions (a.k.a lambdas)_
    ↪to perform binding.
        # Their advantage is that you can avoid declaring_
        ↪new functions i.e.
        # they offer a concise way to "redirect" callbacks.
        btn3 = Button(text="Using anonymous functions.")
        btn3.bind(on_press=lambda x: self.on_event(None))

    # You can also declare a function that accepts a_
    ↪variable number of
        # positional and keyword arguments and use_
        ↪introspection to determine
            # what is being passed in. This is very handy for_
            ↪debugging as well
                # as function re-use. Here, we use standard event_
                ↪binding to a function
                    # that accepts optional positional and keyword_
                    ↪arguments.
        btn4 = Button(text="Use a flexible function")
        btn4.bind(on_press=self.on_anything)

    # Lastly, we show how to use partial functions._
    ↪They are sometimes
        # difficult to grasp, but provide a very flexible_
        ↪and powerful way to
            # reuse functions.
        btn5 = Button(text="Using partial functions. For_
        ↪hardcores.")

```

```

        btn5.bind(on_press=partial(self.on_anything, "1", "2
→", monthy="python"))

    for but in [btn, btn2, btn3, btn4, btn5]:
        self.add_widget(but)

    def on_event(self, obj):
        print("Typical event from", obj)

    def on_property(self, obj, value):
        print("Typical property change from", obj, "to",
→value)

    def on_anything(self, *args, **kwargs):
        print('The flexible function has *args of',
→str(args),
        "and **kwargs of", str(kwargs))

class DemoApp(App):
    def build(self):
        return DemoBox()

if __name__ == "__main__":
    DemoApp().run()

```

Quando uma função é vinculada a um evento ou propriedade, um callback `kivy.weakmethod.WeakMethod` é salvo, e ao despachar o callback de chamada será removido caso a referência de retorno seja inválida.

Se o callback já tiver sido vinculado a um evento ou a uma função, o mesmo não será adicionado novamente.

`create_property()`

Cria uma nova propriedade em tempo de execução.

Novo na versão 1.0.9.

Alterado na versão 1.8.0: parâmetro *value* adicionado, pode ser utilizado para definir o valor padrão da propriedade. Além disso, o tipo do valor é usado para especializar a propriedade criada.

Alterado na versão 1.9.0: No passado, se *value* não fosse do tipo *bool*, um *NumericProperty* seria criado, agora é criado um *BooleanProperty*.

Além disso, agora argumentos posicionais ou keyword arguments são passados para a propriedade quando criada.

Aviso: Essa função é desenhada para a linguagem Kivy, não utilize-a em seu código. Você deve declarar uma propriedade em sua classe ao invés de utilizar este método.

Parameters

name: string Nome da propriedade

value: object, opcional Valor padrão da propriedade. O tipo também é usado para criar tipo de propriedades mais apropriados. O padrão é *None*.

```
>>> mywidget = Widget()
>>> mywidget.create_property('custom')
>>> mywidget.custom = True
>>> print(mywidget.custom)
True
```

dispatch()

Emite um evento sobre todos os handlers adicionado em bind/fbind(). Assim que algum handle retornar True, o despacho (chamada do evento) será finalizada.

A função coleta todos os argumentos posicionais e os keyword arguments e envia-os aos manipuladores (handlers).

Nota: Os manipuladores (handlers) são chamados em ordem reversas na qual foram registrados com *bind()*.

Parameters

event_type: basestring nome do evento para despacho.

Alterado na versão 1.9.0: Coleta de palavras-chave e o encaminhamento foram adicionados. Depois, somente argumentos posicionais serão coletados e encaminhados.

events()

Retorna todos os eventos da classe. Pode ser utilizado introspecção.

Novo na versão 1.8.0.

fbind()

Um método para vinculação avançada e geralmente mais rápido. Este método é diferente do *bind()* e destina-se ao uso interno e a usuários avançados. Ele pode ser utilizado enquanto os seguinte pontos forem atendidos.

1. Ao contrário de `bind()`, ele não verifica se esta função e largs/kwags não foram antes ligados pelos seus nomes. Então ligar o mesmo callback várias vezes continuará adicionando-o.

2. Embora: meth: `bind` cria uma classe: ' WeakMethod' do retorno de chamada quando se vincula a um evento ou propriedade, este método armazena o retorno de chamada diretamente, a menos que um argumento de palavra-chave `ref` com valor True seja fornecido e então uma classe: `WeakMethod` é salvo. Isso é útil quando não há risco de um vazamento de memória armazenando o retorno de chamada diretamente.

3. Este método retorna um número positivo único se `name` foi encontrado e vinculado, e 0, caso contrário. Ele não gera uma exceção, como: meth: `bind` se a propriedade `name` não for encontrada. Se não for zero, o uid retornado é exclusivo para este `name` e a chamada de volta pode ser usado com: meth: `unbind_uid` para desvincular.

Ao vincular uma chamada de volta com args e / ou kwargs.: meth: `funbind` deve ser usado para desvincular. Se args e kwargs não são fornecidos, meth: `unbind` pode ser usado também. :meth: `unbind_uid` pode ser usado em qualquer caso.

Este método transmite quaisquer argumentos posicionais e/ou palavra-chave capturados para o retorno de chamada, removendo a necessidade da chamada parcial. Ao chamar a chamada de retorno os args gastados são transmitidos seguidos por instâncias/valores (apenas instâncias para kwargs) seguido por kwargs gastados.

A seguir é um exemplo de uso similar ao exemplo em `bind()`:

```
class DemoBox(BoxLayout):

    def __init__(self, **kwargs):
        super(DemoBox, self).__init__(**kwargs)
        self.orientation = "vertical"

        btn = Button(text="Normal binding to event")
        btn.fbind('on_press', self.on_event)

        btn2 = Button(text="Normal binding to a property_"
                     ↪change")
        btn2.fbind('state', self.on_property)

        btn3 = Button(text="A: Using function with args.")
        btn3.fbind('on_press', self.on_event_with_args,
                     ↪'right',
                                tree='birch', food='apple')

        btn4 = Button(text="Unbind A.")
        btn4.fbind('on_press', self.unbind_a, btn3)
```

```

        btn5 = Button(text="Use a flexible function")
        btn5.fbind('on_press', self.on_anything)

        btn6 = Button(text="B: Using flexible functions_
→with args. For hardcores.")
        btn6.fbind('on_press', self.on_anything, "1", "2",
→monthly="python")

        btn7 = Button(text="Force dispatch B with different_
→params")
        btn7.fbind('on_press', btn6.dispatch, 'on_press', 6,
→7, monthly="other python")

    for but in [btn, btn2, btn3, btn4, btn5, btn6,
→btn7]:
        self.add_widget(but)

    def on_event(self, obj):
        print("Typical event from", obj)

    def on_event_with_args(self, side, obj, tree=None,
→food=None):
        print("Event with args", obj, side, tree, food)

    def on_property(self, obj, value):
        print("Typical property change from", obj, "to",
→value)

    def on_anything(self, *args, **kwargs):
        print('The flexible function has *args of',
→str(args),
        "and **kwargs of", str(kwargs))
        return True

    def unbind_a(self, btn, event):
        btn.unbind('on_press', self.on_event_with_args,
→'right',
                    tree='birch', food='apple')

```

Nota: Uma vez que o kv lang usa este método para ligar, é preciso implementar este método, em vez de: meth: *bind* ao criar uma não :classe:, class: *EventDispatcher* é usada com o kv lang. Veja: class: *Observable* para um exemplo.

Novo na versão 1.9.0.

Alterado na versão 1.9.1: Um keyword argumento *ref* que já foi adicionado.

funbind()

Similar a *fbind()*.

Quando unbinding *unbind()* irá desvincular todos os callbacks que correspondam ao callback, enquanto este método só desvinculará o primeiro.

Para desvincular, os mesmos argumentos posicionais e de keywords passados a :meth: *fbind* devem ser enviado por *funbind()*.

Nota: É seguro usar *funbind()* para desvincular uma função vinculada com *bind()*, desde que nenhuma keyword e argumentos posicionais sejam passados para *funbind()*.

Novo na versão 1.9.0.

get_property_observers()

Retorna uma lista de métodos vinculados aos eventos/propriedades como o * nome* do argumento:

```
widget_instance.get_property_observers('on_release')
```

Parameters

name: strO nome do evento ou propriedade.

args: boolSe retornar os argumentos vinculados. Para manter a compatibilidade, apenas as funções de retorno de chamada e não os args fornecidos serão retornadas na lista quando *args* for igual a *False*.

Se True, cada elemento na lista é um tupla de 5 (*callback*, *args*, *kwargs*, *is_ref*, *uid*), onde *is_ref* indica se *callback* é uma referência fraca, e *uid* é o uid dado por :meth: *fbind*, ou None se: meth: *bind* foi usado. O padrão é *False*.

ReturnsA lista de callbacks ligados. Veja *args* para detalhes.

Novo na versão 1.8.0.

Alterado na versão 1.9.0: *args* foram adicionadas.

getter()

Retorna o getter da propriedade.

Novo na versão 1.0.9.

is_event_type()

Retorna True se o event_type já estiver registrado.

Novo na versão 1.0.4.

properties()

Retorna todas as propriedades da classe numa classe de dicionário key/propriedade. Pode ser utilizado para instropecção.

Novo na versão 1.0.9.

property()

Pega a instância da propriedade de um nome de propriedade. Se quiet for *True*, *None* será retornado ao invés de uma exceção ser levantada quando o *nome* não for um propriedade. O padrão é *False*.

Novo na versão 1.0.9.

RetornaUma instância derivada de *Property* correspondente ao nome.

Alterado na versão 1.9.0: silencioso foi adicionado.

proxy_ref

Implementação padrão do proxy_ref, retorna self. .. versionadded:: 1.9.0

register_event_type()

Registra um tipo de evento com o dispatcher.

O registro de tipos de eventos permite que o dispatcher valide os nomes dos manipuladores de eventos à medida que eles são anexados e busca objetos anexados para manipuladores adequados. Cada declaração de tipo de evento deve:

- 1.inicia com o prefixo *on_*.
- 2.possui um handle padrão na classe.

Exemplo de criação de um evento padrão:

```
class MyWidget(Widget):
    def __init__(self, **kwargs):
        super(MyWidget, self).__init__(**kwargs)
        self.register_event_type('on_swipe')

    def on_swipe(self):
        pass

    def on_swipe_callback(*largs):
        print('my swipe is called', largs)
w = MyWidget()
w.dispatch('on_swipe')
```

setter()

Retornar o Setter de uma propriedade. Use: *instance.setter ('nome')*. O setter é um função de callback conveniente e útil se quiseres ligar diretamente uma propriedade a outra. Ele retorna uma *partial function* que aceitará (obj,

valor) args e resultados na propriedade ‘nome’ da instância sendo definida como valor.

Novo na versão 1.0.9.

Por exemplo, para vincular o número 2 com o número em Python você faria:

```
class ExampleWidget(Widget):
    number1 = NumericProperty(None)
    number2 = NumericProperty(None)

    def __init__(self, **kwargs):
        super(ExampleWidget, self).__init__(**kwargs)
        self.bind(number1=self.setter('number2'))
```

Isso é o equivalente a forma que ocorre a ligação/vinculação com a linguagem kv.

```
<ExampleWidget>:
    number2: self.number1
```

unbind()

Desvincular propriedades de funções de retorno de chamada com uso semelhante como: `meth: bind`.

Se um callback foi vinculado a um determinado evento ou propriedade várias vezes, somente a primeira ocorrência será desvinculada.

Nota: É seguro usar :meth: `unbind` em uma função vinculada com :meth: `fbind`, desde que essa função tenha sido originalmente vinculada sem qualquer palavra-chave e argumentos posicionais. Caso contrário, a função não será desvinculada e você deve usar :meth: `funbind` em vez disso.

unbind_uid()

Use o `uid` retornado por `fbind()` para desvincular o callback.

Este método é muito mais eficiente do que :meth: `funbind`. Se `uid` avalia False (por exemplo, 0) um `ValueError` uma exceção é levantada. Além disso, apenas callbacks vinculados com: `meth: fbind` podem ser desvinculadas com este método.

Uma vez que cada chamada para `fbind()` irá gerar um único `uid`, apenas um callback será removido. Se `uid` não for encontrado entre os callbacks, nenhum erro será gerado.

Por exemplo.:

```
btn6 = Button(text="B: Using flexible functions with args. ↴For hardcores.")
```

```

uid = btn6.fbind('on_press', self.on_anything, "1", "2",
    ↪monthly="python")
if not uid:
    raise Exception('Binding failed').
...
btn6.unbind_uid('on_press', uid)

```

Novo na versão 1.9.0.

unregister_event_types()

Cancelar o registro de um tipo de evento no dispatcher.

class kivy.event.ObjectWithUid

Bases: `object`

(internal) This class assists in providing unique identifiers for class instances. It is not intended for direct usage.

class kivy.event.Observable

Bases: `kivy.event.ObjectWithUid`

`Observable` é uma classe de stub e define os métodos necessários para ligar. `EventDispatcher` é (o) um exemplo de classe que implementa o interface de ligação. Veja a classe `EventDispatcher` para maiores informações.

Novo na versão 1.9.0.

fbind()

Veja `EventDispatcher.fbind()`.

Nota: Para manter a compatibilidade com classes derivadas que podem ter herdado de `Observable` antes, o método `fbind()` foi adicionado. A implementação padrão de `fbind()` é criar uma função parcial que passa para ligar enquanto salva o uid e largs/kwags. No entanto, meth:`funbind` (e `unbind_uid()`) são bastante ineficientes, uma vez que temos de primeiro pesquisar esta função parcial usando o largs/kwags ou uid e, em seguida, chamar `unbind()` na função retornada. É recomendável substituir esses métodos em classes derivadas para vincular diretamente para um melhor desempenho.

Similarmente a `EventDispatcher.fbind()`, este método retorna 0 em caso de falha e um uid único e positivo no caso de sucesso. Este uid pode ser usado com `unbind_uid()`.

funbind()

Veja `fbind()` and `EventDispatcher.funbind()`.

unbind_uid()

Veja `fbind()` e `EventDispatcher.unbind_uid()`.

4.1.11 Fábrica de Objetos

A fábrica pode ser usada para registrar automaticamente qualquer classe ou módulo e instanciar classe dele em qualquer lugar em seu projeto. Essa é uma implementação do Padrão de Projeto **Factory Pattern**.

A lista de classes e os módulos disponíveis são automaticamente gerados pelo *setup.py*.

Exemplo para registrar uma classe/módulo:

```
>>> from kivy.factory import Factory  
>>> Factory.register('Widget', module='kivy.uix.widget')  
>>> Factory.register('Vector', module='kivy.vector')
```

Exemplo de uso de uma Factory:

```
>>> from kivy.factory import Factory  
>>> widget = Factory.Widget(pos=(456, 456))  
>>> vector = Factory.Vector(9, 2)
```

Exemplo de uso de um nome de classe:

```
>>> from kivy.factory import Factory  
>>> Factory.register('MyWidget', cls=MyWidget)
```

Por padrão, o primeiro classname registrado pelo Factory é permanente. Se desejares modificar uma classe registrada, será necessário cancelar o registro do classname primeiro e refazer o processo.

```
>>> from kivy.factory import Factory  
>>> Factory.register('MyWidget', cls=MyWidget)  
>>> widget = Factory.MyWidget()  
>>> Factory.unregister('MyWidget')  
>>> Factory.register('MyWidget', cls=CustomWidget)  
>>> customWidget = Factory.MyWidget()
```

`kivy.factory.Factory = <kivy.factory.FactoryBase object>`

Instância de Factory que será utilizada para obter novas classes

4.1.12 Utilitários geométricos

Este módulo contem algumas funções de ajuda para o calculo geométrico.

`kivy.geometry.circumcircle(a, b, c)`

Calcula a circunferência de um triângulo definido por *a*, *b*, *c*. http://en.wikipedia.org/wiki/Circumscribed_circle

Parameters

a: iterável contendo pelo menos 2 valores (para *x* e *y*) O 1º ponto do triângulo.

b: iterável contendo pelo menos 2 valores (para *x* e *y*) O 2º ponto do triângulo.

c: iterável contendo pelo menos 2 valores (para *x* e *y*) O 3º ponto do triângulo.

Return

Uma tupla que define o círculo:

- O primeiro elemento na tupla retornada é o centro como (*x*, *y*)
- O segundo é o raio (float)

`kivy.geometry.minimum_bounding_circle(points)`

Retorna o círculo mínimo para um conjunto de pontos.

Para uma descrição do problema que está sendo resolvido, veja o [Smallest Circle Problem](#).

A função usa o Algoritmo de Applet, o tempo de execução é $O(h^3 * n)$, onde h é o número de pontos no casco convexo do conjunto de pontos. Mas isso roda em tempo linear em quase todos os casos do mundo real. Veja: <http://tinyurl.com/6e4n5yb>

Parameters

points: iterable Uma lista de pontos (2 tuplas com coordenadas *x*, *y*)

Return

Uma tupla que define o círculo:

- O primeiro elemento na tupla retornada é o centro (*x*, *y*)
- O segundo o raio (float)

4.1.13 Reconhecimento de Gestos

Esta classe permite que você facilmente crie novos gestos e compare-os:

```
from kivy.gesture import Gesture, GestureDatabase

# Create a gesture
g = Gesture()
g.add_stroke(point_list=[(1,1), (3,4), (2,1)])
g.normalize()
```

```

# Add it to the database
gdb = GestureDatabase()
gdb.add_gesture(g)

# And for the next gesture, try to find it!
g2 = Gesture()
# ...
gdb.find(g2)

```

Aviso: Você não quer realmente fazer isso: é mais como um exemplo de como construir gestos dinamicamente. Normalmente, você precisaria de muito mais pontos, então é melhor gravar gestos em um arquivo e recarrega-los para comparar mais tarde. Procure no diretório exemplos/gestos um exemplo de como fazer isso.

class kivy.gesture.Gesture(*tolerance=None*)

Uma implementação em Python de um algoritmo de reconhecimento de gesto por Oleg Dopertchouk: <http://www.gamedev.net/reference/articles/article2039.asp>

Implementado por Jeiel Aranal (chemikhazi@gmail.com), disponibilizado em domínio público.

add_stroke(*point_list=None*)

Adiciona um traço ao gesto e retorna a instância do *Stroke*. O argumento *point_list* opcional é uma lista dos pontos do mouse para o traço.

dot_product(*comparison_gesture*)

Calcula produto do ponto do gesto com outro gesto.

get_rigid_rotation(*dstpts*)

Extrai a rotação para aplicar a um grupo de pontos para minimizar a distância para um segundo grupo de pontos. Considera-se que os dois grupos de pontos estão centrados. Esta é uma versão simples que apenas escolhe um ângulo com base no primeiro ponto do gesto.

get_score(*comparison_gesture, rotation_invariant=True*)

Retorna a pontuação correspondente do gesto contra outro gesto.

normalize(*stroke_samples=32*)

Executa o algoritmo de normalização gestual e calcula o produto do ponto com self (auto).

class kivy.gesture.GestureDatabase

Bases: object

Classe para lidar com um banco de dados de gestos.

add_gesture(*gesture*)

Adiciona um novo gesto ao banco de dados.

```
find(gesture, minscore=0.9, rotation_invariant=True)
    Encontra um gesto correspondente no banco de dados.

gesture_to_str(gesture)
    Converte um gesto numa String.

str_to_gesture(data)
    Converte uma String única num gesto.

class kivy.gesture.GestureStroke
    Os gestos podem ser feitos de múltiplos traços.

    add_point(x=x_pos, y=y_pos)
        Adiciona um ponto ao traço.

    center_stroke(offset_x, offset_y)
        Centra o traçado compensando os pontos.

    normalize_stroke(sample_points=32)
        Normaliza traços de modo que cada traço tenha um número padrão de pontos. Retorna True se o Stroke for normalizado, False se ele não puder ser normalizado. sample_points controla a resolução do Stroke.

    points_distance(point1=GesturePoint, point2=GesturePoint)
        Retorna a distância entre dois GesturePoints.

    scale_stroke(scale_factor=float)
        Dimensiona o traçado para baixo por scale_factor.

    stroke_length(point_list=None)
        Calcula o comprimento do Stroke. Se for fornecida uma lista de pontos, localiza o comprimento dessa lista.
```

4.1.14 Interactive launcher

Novo na versão 1.3.0.

Alterado na versão 1.10.0: O lançador interativo está obsoleto.

A *InteractiveLauncher* fornece uma interface amigável do Shell do Python para que uma App possa ser interativamente prototipada e depurada'.

Nota: A API do Kivy planeja que algumas funções sejam executadas apenas uma vez ou antes do início do *EventLoop* principal. Métodos que normalmente podem ser chamados durante a execução do aplicativo funcionará como esperado, mas especificamente os métodos anônimos como `on_touch()` quando executados dinamicamente podem desencadear alguns problemas.

Craindo um *InteractiveLauncher*

Pegue a sua subclasse existente de App (isto pode ser código de produção) e passe uma instância ao construtor *InteractiveLauncher*:

```
from kivy.interactive import InteractiveLauncher
from kivy.app import App
from kivy.uix.button import Button

class MyApp(App):
    def build(self):
        return Button(text='Hello Shell')

launcher = InteractiveLauncher(MyApp())
launcher.run()
```

Depois de pressionar *Enter*, o script retornará. Isso permite que o interpretador continue executando. A inspeção ou modificação do :class: *App* pode ser feita com segurança através da instância do *InteractiveLauncher* ou das instâncias de classe fornecidas por *SafeMembrane*.

Nota: Se quiseres testar este exemplo, inicie o Python sem que qualquer arquivo já tenha sido interpretado e copie/cole todas as linhas. Você ainda terá o intérprete no final + o aplicativo Kivy em execução.

Desenvolvimento Interativo

O IPython fornece uma maneira rápida de aprender a API do Kivy. A instância de :class: 'App' e todos os seus atributos, incluindo métodos e sua árvore inteira de Widgets, podem ser listados rapidamente usando o operador '.' e pressionando a tecla 'Tab'. Experimente este código num shell do Ipython:

```
from kivy.interactive import InteractiveLauncher
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.graphics import Color, Ellipse

class MyPaintWidget(Widget):
    def on_touch_down(self, touch):
        with self.canvas:
            Color(1, 1, 0)
            d = 30.
            Ellipse(pos=(touch.x - d/2, touch.y - d/2), size=(d, d))

class TestApp(App):
```

```

def build(self):
    return Widget()

i = InteractiveLauncher(TestApp())
i.run()
i.      # press 'tab' to list attributes of the app
i.root. # press 'tab' to list attributes of the root widget

# App is boring. Attach a new widget!
i.root.add_widget(MyPaintWidget())

i.safeIn()
# The application is now blocked.
# Click on the screen several times.
i.safeOut()
# The clicks will show up now

# Erase artwork and start over
i.root.canvas.clear()

```

Nota: Todos os Proxies usados no módulo armazenam sua referência no atributo :attr:`_ref`, que pode ser acessado diretamente se necessário, tal como para obter DocStrings. `help()` e `type()` acessará o proxy, não seu referente.

Pausando a Aplicação Diretamente

Ambas as `InteractiveLauncher` e `SafeMembrane` mantém referências internas para objetos EventLoop's 'safe' e 'confirmed' `threading.Event`. Pode usar seus métodos de segurança para controlar o aplicativo manualmente.

`SafeMembrane.safeIn()` fará com que o aplicativo seja pausado e `SafeMembrane.safeOut()` permitirá a um aplicativo pausado continuar a sua execução. Isso é potencialmente útil para ações de Scripts em funções que precisam que a tela atualizada etc.

Nota: A pausa é implementada através do método `Clocks' schedule_once()` e ocorre antes do início de cada Frame.

Adicionando Atributos Dinamicamente

Nota: Este módulo usa proxies de subprocessos e objetos para encapsular a execução do App. Deadlocks e Memory Corruption podem ocorrer ao estabelecer referências diretas dentro do Thread sem passar pelo proxy fornecido (s).

A class:*InteractiveLauncher* pode ter atributos adicionados igualmente a qualquer objeto normal e se estes foram criados fora da membrana, eles não serão Threadafe porque as referências externas a eles no intérprete Python não passarão pelo comportamento da membrana do *InteractiveLauncher*, herdado de *SafeMembrane*.

Para obter ThreadSafe com segurança para as referências externas, basta atribuí-las a instâncias de si mesmas de *Safe Membrane*, como por exemplo:

```
from kivy.interactive import SafeMembrane

interactiveLauncher.attribute = myNewObject
# myNewObject is unsafe
myNewObject = SafeMembrane(myNewObject)
# myNewObject is now safe. Call at will.
myNewObject.method()
```

TODO

Testes unitários, exemplos e uma melhor explicação de quais métodos são seguros numa aplicação em execução seria bom. Todos os três seriam excelente.

Poderia ser reescrita com um estilo de gerenciador de contexto, por exemplo:

```
with safe:
    foo()
```

Quaisquer casos de uso além de compactar código?

```
class kivy.interactive.SafeMembrane(ob, *args, **kwargs)
    Bases: object
```

Esta ajuda é para um objeto Proxy. Você queria ajuda no referente do Proxy?
Tente usar help(<instance>._ref)

O *SafeMembrane* é um Proxy Threadafe que também retorna atributos como novos objetos Thread-safe e faz chamadas de método Thread-safe, impedindo o vazamento de objetos thread-unsafe no ambiente do usuário.

safeIn()

Fornece um ponto de entrada Thread-safe ao *InteractiveLauncher*.

safeOut()

Fornece um ponto de saída thread-safe ao *InteractiveLauncher*.

```
class kivy.interactive.InteractiveLauncher(*args, **kwargs)
    Bases: kivy.interactive.SafeMembrane
```

Proxy para uma instância da aplicação que lança num thread e, em seguida, retorna e atua como um Proxy para o aplicativo no Thread.

4.1.15 Kivy Base

Este módulo contém as funcionalidades principais do Kivy e não é destinado a usuários finais. Sinta-se livre para olhar e estudar, porém, invocar algum desses métodos diretamente pode produzir comportamentos inesperados.

Gerenciamento do Event Loop

```
kivy.base.EventLoop = <kivy.base.EventLoopBase object>
```

Instância do EventLoop

```
class kivy.base.EventLoopBase
```

Bases: *kivy.event.EventDispatcher*

Event loop principal. Este loop gerencia a atualização dos eventos de entrada e de despacho.

add_event_listener(*listener*)

Adiciona um novo “event listener” para capturar eventos de toque.

add_input_provider(*provider*, *auto_remove=False*)

Adiciona um novo provedor de entrada para “escutar” os eventos de toque.

add_postproc_module(*mod*)

Adiciona um módulo de entrada postproc (DoubleTap, TripleTap, DeJitter RetainTouch são os padrões).

close()

Sai do “main loop” e para todos os provedores de entradas configurados.

dispatch_input()

Chamado pelo idle() para ler eventos do provedor de entrada, passa o evento para o postproc e envia os eventos finais.

ensure_window()

Certifique-se de que temos uma janela.

exit()

Feche o “main loop” e feche a aplicação.

idle()

Esta função é invocada depois de cada frame. Por padrão:

- Ele “marca” o relógio para o próximo frame.

- Ele lê todas as entradas e os eventos de despacho.
- Ele envia os eventos *on_update*, *on_draw* and *on_flip* para a janela.

on_pause()

Event Handler para *on_pause* que será disparado quando o loop de evento estiver pausado.

on_start()

Event handler para *on_start* que será disparado logo após todos os provedores terem sido inicializados.

on_stop()

Event handler para o evento *on_stop* que será disparado logo após todos os provedores terem sido parados.

post_dispatch_input(etype, me)

Essa função é chamada por *dispatch_input()* que nós queremos enviar um evento de entrada. O evento é enviado para todos os ouvintes e, se agarrado, é despachado para agarrar widgets.

remove_android_splash(*args)

Remove android presplash in SDL2 bootstrap.

remove_event_listener(listener)

Remove um event listener da lista.

remove_input_provider(provider)

Remove um provedor de entrada.

remove_postproc_module(mod)

Remove um módulo postproc.

run()

Main Loop

set_window(window)

Define a janela usada para o event loop.

start()

Deve ser chamado somente uma vez depois de *run()*. Isso inicia todas as configurações dos provedores de entrada.

stop()

Pare todos os provedores de entrada e chame o callbacks registrados usando o *EventLoop.add_stop_callback()*.

touches

Retorna a lista de todos os toques atuais pressionados ou move o estado.

class kivy.base.ExceptionHandler

Bases: object

Base handle que pega exceções em `runTouchApp()`. Você pode criar uma sub-classe e estender como o exemplo a seguir:

```
class E(ExceptionHandler):
    def handle_exception(self, inst):
        Logger.exception('Exception catched by ExceptionHandler
        ↴')
        return ExceptionManager.PASS

ExceptionManager.add_handler(E())
```

Todas exceções serão enviadas para o PASS e logadas no console!

handle_exception(exception)

Lidar com uma exceção, padrão para retornar `ExceptionManager.STOP`.

class kivy.base.ExceptionManagerBase

A classe `ExceptionManager` gerencia os manipuladores de exceções.

add_handler(cls)

Adiciona uma novo manipulador de exceção para a pilha.

handle_exception(inst)

Chamado quando um exceção ocorrer na função `runTouchApp()` do main loop.

remove_handler(cls)

Remove uma manipulador de exceção da pilha.

kivy.base.ExceptionManager = <kivy.base.ExceptionManagerBase instance>

Implementação de uma instância da classe `ExceptionManagerBase`.

kivy.base.runTouchApp(widget=None, slave=False)

Função estática principal que inicia o loop da aplicação. Você pode acessar alguns magics através dos seguintes argumentos:

Parameters

*<empty>*Para fazer o trabalho de despacho, precisarás de pelo menos um input listener (ouvinte de entrada). Se não, a aplicação finalizará. (MTWindow age como um input listener (ouvinte de entrada))

*widget*Se você passar somente um Widget, um MTWindow será criado e seus Widget será adicionado a janela como o Widget principal.

*slave*Nenhum despacho de evento foi feito. Este será o seu trabalho.

*widget + slave*Nenhum despacho de eventos é feito. Este será o seu trabalho, mas tentamos obter a janela(deve ser criado por você de antemão) e adicione o widget para ele. Muito útil

para incorporar o Kivy em outro kit de ferramentas. (como Qt, check kivy-designed)

kivy.base.stopTouchApp()

Para a aplicação atual saindo do main loop.

4.1.16 Objeto Logger

Diferentes níveis de logs estão disponíveis: *trace*, *debug*, *info*, *warning*, *error* e *critical*.

Exemplos de uso:

```
from kivy.logger import Logger

Logger.info('title: This is a info message.')
Logger.debug('title: This is a debug message.')

try:
    raise Exception('bleh')
except Exception:
    Logger.exception('Something happened!')
```

A mensagem passada para o *Logger* é dividida em duas partes, separadas por dois pontos (:). A primeira parte é usada como um título, e a segunda parte é usada como a mensagem. Desta forma, você pode “categorizar” a sua mensagem facilmente:

```
Logger.info('Application: This is a test')

# will appear as

[INFO    ] [Application] This is a test
```

Configuração do Logger

O Logger pode ser controlado através do arquivo de configuração Kivy:

```
[kivy]
log_level = info
log_enable = 1
log_dir = logs
log_name = kivy_%y-%m-%d_.txt
log_maxfile = 100
```

Mais informações sobre os valores permitidos estão descritos no módulo *kivy.config*.

História do Logger

Mesmo que o *Logger* não esteja ativado, você ainda tem acesso às últimas 100 mensagens:

```
from kivy.logger import LoggerHistory  
print(LoggerHistory.history)
```

`kivy.logger.Logger = <logging.Logger object>`

Instância do *Logger* padrão do Kivy

`class kivy.logger.LoggerHistory(level=0)`

Bases: `logging.Handler`

Manipulador de história Kivy

4.1.17 Métricas

Novo na versão 1.5.0.

Uma tela é definida pelo seu tamanho físico, desnsidade de pixels e definição. Tais fatores são essenciais para criar interfaces de usuários (UI) com o tamanho correto em qualquer dispositivo, em qualquer lugar.

Em Kivy, todos os pipelines gráficos trabalham com pixels. Mas usar pixels como uma unidade de medida é problemático porque os tamanhos mudam de acordo com a tela.

Dimensões

Se quiseres projetar sua interface de usuário para diferentes tamanhos de tela, irás querer utilizar unidades de medição mais eficientes no desenvolvimento. A biblioteca Kivy fornece algumas alternativas escaláveis.

Units

pt Pontos - 1/72 de polegada com base no tamanho físico da tela. Prefera utilizar *sp* ao invés de *pt*.

mm Milímetros - Com base no tamanho físico da tela.

cm Centímetros - Com base no tamanho físico da tela.

in Polegadas - Com base no tamanho físico da tela.

dp Pixels independentes de densidade - é uma unidade abstrata que se baseia na densidade física da tela. Com uma *density* de 1, *1dp* é igual a *1px*. Ao executar em uma tela de maior densidade, o número de pixels usados para desenhar *1dp* é escalonado um fator apropriado para o DPI da tela, e o inverso para um menor DPI. A

proporção de *dp* para pixels mudará com a densidade de tela, mas não necessariamente numa proporção direta. Usar a unidade *dp* é uma solução simples para tornar as dimensões de exibição em seu layout redimensionável corretamente para diferentes densidades de tela. Em outras palavras, isso fornece consistência para o tamanho do mundo real de sua interface com diferentes dispositivos.

sp Pixels independentes da escala - a mesma é semelhante a unidade *dp*, mas também é dimensionada conforme as preferências do usuário em relação ao tamanho da letra em que os textos devem ser exibidos. Recomendamos que utilize esta unidade ao especificar tamanhos de fonte, de modo a permitir que o tamanho da fonte seja ajustado tanto para a diferentes densidade de tela quanto para usuários especiais que necessitam definir a exibição das letras em tamanhos maiores.

Exemplos

Aqui está um exemplo de criação de um Label com *sp* *sp_size* e a definição da altura manualmente com uma margem de *10dp*:

```
#:kivy 1.5.0
<MyWidget>:
    Label:
        text: 'Hello world'
        font_size: '15sp'
        size_hint_y: None
        height: self.texture_size[1] + dp(10)
```

Controle Manual de Métricas

As métricas não podem ser alteradas em tempo de execução. Uma vez que um valor tenha sido convertido em pixels, você não poderá mais recuperar o valor original. Isso decorre do fato de que o DPI e a densidade de um dispositivo não podem ser alterados durante o tempo de execução.

Fornecemos algumas variáveis de ambiente para controlar métricas:

- *KIVY_METRICS_DENSITY*: se definido, este valor será utilizado para *density* em vez de utilizar o valor do sistema. No Android, o valor varia entre 0,75, 1, 1,5 e 2.
- *KIVY_METRICS_FONTSIZE*: se definido, este valor será utilizado para *fontscale* ao invés de utilizar o do sistema. No Android, o valor varia entre 0,8 e 1,2.

- **KIVY_DPI**: se definido, este valor será utilizado para **dpi**. Observe que a configuração do DPI não afetará a notação *dp/sp* porque elas são baseadas na densidade da tela.

Por exemplo, se você quiser simular uma tela de alta densidade (como o HTC One X):

```
KIVY_DPI=320 KIVY_METRICS_DENSITY=2 python main.py --size 1280x720
```

Ou uma de densidade média (como um Motorola Droid 2):

```
KIVY_DPI=240 KIVY_METRICS_DENSITY=1.5 python main.py --size 854x480
```

Você também pode simular uma preferência de usuário alternativa para *fontscale*, por exemplo:

```
KIVY_METRICS_FONTSCALE=1.2 python main.py
```

kivy.metrics.Metrics = <kivy.metrics.MetricsBase object>
Instância padrão de **MetricsBase**, usado em todo o código... versionadded ::
1.7.0

class kivy.metrics.MetricsBase
Bases: **object**

Classe que contém os atributos padrão para *Metrics*. Não use essa classe diretamente, utilize a instância *Metrics*.

density()

Retorna a densidade da tela. Esse valor é 1 por padrão em Desktops, mas variará de acordo com o Android, dependendo da tela.

dpi()

Retorna o DPI da tela. Dependendo da plataforma, o DPI pode ser pego do provedor Window (geralmente em Desktop) ou de um módulo específico da plataforma (como no caso do Android/iOS).

dpi_rounded()

Retornar o DPI da tela, arredondado para o mais próximo de 120, 160, 240 ou 320.

fontscale()

Retorna a preferência do usuário *fontscale*. Esse valor é 1 por padrão, mas pode variar entre 0,8 e 1,2.

kivy.metrics.pt(value)

Converte de ponto para pixels.

kivy.metrics.inch(value)

Converte de polegadas para pixels.

kivy.metrics.cm(value)

Converte de centímetros para pixels.

`kivy.metrics.mm(value)`

Converte de milímetros para pixels.

`kivy.metrics.dp(value)`

Converte de pixels independentes de densidade para pixels

`kivy.metrics.sp(value)`

Converter de escala independentes de pixels para pixels

4.1.18 Reconhecedor de gestos Multistroke

Novo na versão 1.9.0.

Aviso: Isto é experimental e está sujeito a alterações todas as vezes que esta advertência esteja presente.

Veja `kivy/examples/demo/multistroke/main.py` para uma aplicação completa de exemplo.

Visão Geral Conceitual

Este módulo implementa os algoritmos de reconhecimento de gestos Protractor.

`Recognizer` e a pesquisa/database API semelhante a `GestureDatabase`. Ele mantém uma lista de objetos `MultistrokeGesture` e permite que você procure por um gesto feito pelo usuário.

`ProgressTracker` tracks the progress of a `Recognizer.recognize()` call. It can be used to interact with the running recognizer task, for example forcing it to stop half-way, or analyzing results as they arrive.

`MultistrokeGesture` representa um gesto no banco de dados de gestos (`Recognizer.db`). É um container para o objeto `UnistrokeTemplate`, e implementa o algoritmo de permuta de heap para gerar automaticamente todas as possíveis ordens de toques (caso seja desejado).

`UnistrokeTemplate` represents a single stroke path. It's typically instantiated automatically by `MultistrokeGesture`, but sometimes you may need to create them manually.

`Candidate` represents a user-input gesture that is used to search the gesture database for matches. It is normally instantiated automatically by calling `Recognizer.recognize()`.

Exemplo de uso

Veja `kivy/examples/demo/multistroke/main.py` para uma aplicação completa de exemplo.

Você pode ligar a eventos na: classe: `Recognizer` para rastrear o estado de todas as chamadas para: meth: `Recognizer.recognize`. A função callback receberá uma instância da: classe: `ProgressTracker` que pode ser usado para analisar e controlar vários aspectos do processo de reconhecimento:

```
from kivy.vector import Vector
from kivy.multistroke import Recognizer

gdb = Recognizer()

def search_start(gdb, pt):
    print("A search is starting with %d tasks" % (pt.tasks))

def search_stop(gdb, pt):
    # This will call max() on the result dictionary, so it's best_
    ↵to store
    # it instead of calling it 3 times consecutively
    best = pt.best
    print("Search ended (%s). Best is %s (score %f, distance %f)" %
    ↵(
        pt.status, best['name'], best['score'], best['dist']))

# Bind your callbacks to track all matching operations
gdb.bind(on_search_start=search_start)
gdb.bind(on_search_complete=search_stop)

# The format below is referred to as `strokes`, a list of stroke_
    ↵paths.
# Note that each path shown here consists of two points, ie a_
    ↵straight
# line; if you plot them it looks like a T, hence the name.
gdb.add_gesture('T', [
    [Vector(30, 7), Vector(103, 7)],
    [Vector(66, 7), Vector(66, 87)]])

# Now you can search for the 'T' gesture using similar data (user_
    ↵input).
# This will trigger both of the callbacks bound above.
gdb.recognize([
    [Vector(45, 8), Vector(110, 12)],
    [Vector(88, 9), Vector(85, 95)])
```

No próximo passo: classe: `~kivy.clock.Clock`, o processo de correspondência inicia (e, nesse caso, é concluído).

Para localizar chamadas individuais para: meth: *Recognizer.recognize*, use o valor de retorno (também uma instância da: classe: *ProgressTracker*)

```
# Same as above, but keep track of progress using returned value
progress = gdb.recognize([
    [Vector(45, 8), Vector(110, 12)],
    [Vector(88, 9), Vector(85, 95)])]

progress.bind(on_progress=my_other_callback)
print(progress.progress) # = 0

# [ assuming a kivy.clock.Clock.tick() here ]

print(result.progress) # = 1
```

Detalhes do Algoritmo

Para maiores informações sobre o algoritmos correspondente, veja:

“Protractor: Um rápido e preciso reconhecedor de gesto” por Yang Li <http://yangl.org/pdf/protractor-chi2010.pdf>

“\$N-Protractor” by Lisa Anthony e Jacob O. Wobbrock <http://depts.washington.edu/aimgroup/proj/dollar/ndollar-protractor.pdf>

Parte do código é derivado da implementação de JavaScript: <http://depts.washington.edu/aimgroup/proj/dollar/ndollar.html>

class kivy.multistroke.Recognizer(kwargs)**
Bases: *kivy.event.EventDispatcher*

Recognizer fornece um banco de dados de gestos com facilidades de correspondência.

Events

on_search_start Disparado quando uma nova busca é iniciada usando este Reconhecedor.

on_search_complete Acionado quando uma pesquisa em execução termina, por qualquer motivo. (Use: data: *ProgressTracker.status* para descobrir)

Properties

db Uma *ListProperty* que contém os objetos disponíveis *MultistrokeGesture*.

db é um *ListProperty* e o padrão é *[]*.

add_gesture(*name*, *strokes*, ***kwargs*)

Adicione uma nova ação ao banco de dados. Isso instanciará uma nova classe: *MultistrokeGesture* com *strokes* e adicioná-lo ao *self.db*.

Nota: Se você já tiver instanciado um objeto da classe: *MultistrokeGesture* e deseja adicioná-lo, anexe-o para: attr: *Recognizer.db* manualmente.

export_gesture(*filename=None*, ***kwargs*)

Exporta uma lista de objetos *MultistrokeGesture*. Emite uma sequência codificada em base64 que pode ser decodificada em uma lista Python com a função *parse_gesture()* ou importada diretamente para *self.db* usando *Recognizer.import_gesture()*. Se *filename* for especificado, a saída será gravada no disco, caso contrário será retornada.

Este método aceita argumentos opcionais *Recognizer.filter()*.

filter(***kwargs*)

filter() retorna um subconjunto de objetos em *self.db*, de acordo com os critérios fornecidos. Isso é usado por muitos outros métodos da *Recognizer*; Os argumentos abaixo podem, por exemplo, ser usados ao invocar *Recognizer.recognize()* ou *Recognizer.export_gesture()*. Você normalmente não precisa invocar isso diretamente.

Arguments

*name*Limita a lista retornada aos gestos em que *MultistrokeGesture.name* corresponde à expressão regular (*s*). Se *re.match(name, MultistrokeGesture.name)* testar True, o gesto é incluído na lista retornada. Pode ser uma string ou uma matriz de strings:

```
gdb = Recognizer()

# Will match all names that start with a_
# capital N
# (ie Next, New, N, Nebraska etc, but not "n"
# or "next")
gdb.filter(name='N')

# exactly 'N'
gdb.filter(name='N$')

# Nebraska, teletubbies, France, fraggle, N, n,
# etc
gdb.filter(name=['[Nn]', '(?i)T', '(?i)F'])
```

*priority*Limita a lista retornada à ações com certos valores de

attr: *MultistrokeGesture.priority*. Se especificado como um número inteiro, somente os gestos com prioridade menor serão retornados. Se especificado como uma lista (min / max):

```
# Max priority 50
gdb.filter(priority=50)

# Max priority 50 (same result as above)
gdb.filter(priority=[0, 50])

# Min priority 50, max 100
gdb.filter(priority=[50, 100])
```

Quando essa opção é usada, `Recognizer.db` é automaticamente ordenado de acordo com a prioridade, incorrendo em custo extra de processamento. Podes usar `force_priority_sort` para substituir esse comportamento se seus gestos já estiverem classificados de acordo com a prioridade.

`orientation_sensitive` Limita a lista retornada a gestos sensíveis à orientação (True - Verdadeiro), gestos que não são sensíveis à orientação (False - Falso) ou Nenhum-None (ignore a sensibilidade do modelo, este é o padrão).

`numstrokes` Limita a lista retornada aos gestos que têm o número especificado de traços (em `MultistrokeGesture.strokes`). Pode ser um único inteiro ou uma lista de inteiros.

`numpoints` Limita a lista retornada às ações que possuem valores específicos: attr: *MultistrokeGesture.numpoints*. Isso é fornecido para flexibilidade, não use-o, a menos que você entenda o que ele faz. Pode ser um único inteiro ou uma lista de inteiros.

`force_priority_sort` Pode ser usado para substituir o comportamento de classificação padrão. Normalmente os objetos `MultistrokeGesture` são retornados em ordem de prioridade se a opção `priority` estiver sendo usada. Configurando isso para `True` retornará os gestos ordenados em ordem de prioridade, `False` retornará na ordem em que os gestos foram adicionados. `None` significa que será decidido automaticamente (o padrão).

Nota: Para melhorar o desempenho, podes carregar seu banco de dados de gesto em ordem de prioridade e definir este como `False` ao invocar meth:`Recognizer.recognize`

`db` Pode ser definido se desejas filtrar uma lista diferente

de objetos do que `Recognizer.db`. Você provavelmente não desejaras fazer isso; Ele é usado internamente por `import_gesture()`.

`import_gesture(data=None, filename=None, **kwargs)`

Importa uma lista de gestos como formatado por `export_gesture()`. Um destes valores deve ser especificado `data` ou `filename`.

Este método aceita os argumentos opcionais `Recognizer.filter()`, se for especificado 'None', todos os gestos em dados especificados serão importados.

`parse_gesture(data)`

Analise os dados formatados pelo `export_gesture()`. Retorna uma lista de objetos `MultistrokeGesture`. Isso é usado internamente por:meth:`import_gesture`, normalmente não precisas chamar isso diretamente.

`prepare_templates(**kwargs)`

Este método é usado para preparar objetos `UnistrokeTemplate` dentro dos gestos em `self.db`. Isso é útil se quiseres minimizar a punimento da montagem preguiçosa, preparando todos os vetores com antecedência. Se fizeres isso antes de uma chamada para `Recognizer.export_gesture()`, terás os vetores calculados quando carregares os dados mais tarde.

Este método aceita argumentos opcionais `Recognizer.filter()`.

`force_numpoints`, se especificado, irá preparar todos os modelos para o dado número de pontos (em vez de cada modelo preferido `n`, ie `UnistrokeTemplate.numpoints`). Normalmente não irás querer fazer isso.

`recognize(strokes, goodscore=None, timeout=0, delay=0, **kwargs)`

Pesquisa por uma ação correspondente 'strokes' - 'traços'. Retorna uma instância da classe: `ProgressTracker`.

Este método aceita argumentos opcionais `Recognizer.filter()`.

Arguments

`strokes`Uma lista de caminhos traçados (lista de listas de objetos `Vector`) que serão comparados contra os gestos no banco de dados. Também pode ser uma instância de `Candidate`.

Aviso: Se forneceres manualmente uma `Candidate` que tem um skip-flag, certifique-se de que os argumentos de filtro corretos foram definidos. Caso contrário, o sistema tentará carregar vetores que não foram computados. Por

exemplo, se você definir `skip_bounded` e não definir `orientation_sensitive` para `False`, ele criará uma exceção se uma `orientation_sensitive` :class:'UnistrokeTemplate' for encontrada.

goodscoreSe esta opção for definida (entre 0,0 e 1,0) e uma pontuação de gesto igual ou superior ao valor especificado, a pesquisa é imediatamente interrompida e o evento `on_search_complete` será disparado (+ o evento `on_complete` da classe associada `ProgressTracker`). O padrão é `None` (desativado).

timeoutEspecifica um tempo limite (em segundos) para quando a pesquisa é abortada e os resultados retornados. Esta opção aplica-se somente quando `max_gpf` não for igual a 0. O valor padrão é 0, o que significa que todos os gestos na base de dados serão testados, não importa quanto tempo demore.

max_gpfEspecifica o número máximo de objetos `MultistrokeGesture` que podem ser processados por Frame. Quando ultrapassado, fará com que a pesquisa pare e retome o trabalho no Frame seguinte. A definição para 0 completará a pesquisa imediatamente (e bloqueará a UI).

Aviso: Isso não limita o número de `UnistrokeTemplate` objetos combinados! Se um único gesto tiver um milhão de modelos, todos eles serão processados em um único Frame com `max_gpf = 1`!

delaySets an optional delay between each run of the recognizer loop. Normally, a run is scheduled for the next frame until the tasklist is exhausted. If you set this, there will be an additional delay between each run (specified in seconds). Default is 0, resume in the next frame.

force_numpointsForça todos os modelos (e candidatos) a serem preparados para um certo número de pontos. Isso pode ser útil, por exemplo, se você estiver avaliando modelos ótimos para n (não use isso a menos que você entenda o que faz).

transfer_gesture(*tgt*, ***kwargs*)

Transferências de objetos `MultistrokeGesture` de `Recognizer.db` para outra instância :class: `Recognizer` de 'tgt'.

Este método aceita argumentos opcionais `Recognizer.filter()`.

class kivy.multistroke.ProgressTracker(*candidate*, *tasks*, ***kwargs*)

Bases: `kivy.event.EventDispatcher`

Representa uma operação de pesquisa em andamento (ou concluída). Instanciado e retornado pelo método `Recognizer.recognize()` quando o mesmo for invocado. O atributo `results` é um dicionário que é atualizado à medida que a operação de reconhecimento avança.

Nota: Você não pode precisar instanciar esta classe.

Arguments

`candidateCandidate` objeto a ser avaliado

`tasks`Número total de gestos na lista de tarefas (teste contra).

Events

`on_progress`Disparado por cada gesto que é processado

`on_result`Acionado quando um novo resultado é adicionado, e é o primeiro resultado para o `nome/'name'` até agora, ou um resultado consecutivo com a melhor pontuação.

`on_complete`Disparado quando a pesquisa é completada, por qualquer motivo (utilize `ProgressTracker.status` para descobrir).

Attributes

`results`Um dicionário de todos os resultados (até agora). A chave é o nome do gesto (ou seja `UnistrokeTemplate.name` normalmente herdado de `MultistrokeGesture`). Cada item no dicionário é um dict com as seguintes entradas:

`name`Nome do template correspondente (redundante)

`score`Calculado os pontos de 1.0 (correspondência perfeita) para 0.0

`dist`Distância do cosseno do candidato para modelo (low=closer)

`gesture`O objeto `MultistrokeGesture` que foi correspondido

`best_template`Índice da melhor template correspondente (em `MultistrokeGesture.templates`)

`template_results`Lista da distância de todos os templates. O índice da lista corresponde a um índice `UnistrokeTemplate` em `gesture.templates..`

`status`

search Atualmente Trabalhando

stop Foi interrompido pelo usuário (chamado *stop()*)

timeout Um timeout ocorreu (especificado como um *timeout=* para *recognize()*)

goodscore A pesquisa foi logo interrompida porque um gesto com uma pontuação suficientemente alta foi encontrado (especificado como *goodscore=* para *recognize()*)

complete A busca está completa (todos os gestos filtrados correspondentes foram testados)

best

Retorna a melhor correspondência encontrada por *recognize()* até agora. Ele retorna um dicionário com três teclas, 'name', 'dist' e 'score' representando o nome do modelo, a distância (do caminho do candidato) e o valor do escore calculado. Esta é uma propriedade do Python.

progress

Retorna o progresso como um número float, onde 0 é 0% e 1 é igual a 100%. Esta é uma propriedade Python.

stop()

Levanta um sinalizador de parada que é verificado pelo processo de pesquisa. Ele será interrompido no próximo tick do Clock (se ele ainda estiver em execução).

```
class kivy.multistroke.MultistrokeGesture(name,      strokes=None,
                                            **kwargs)
```

Bases: object

MultistrokeGesture representa um gesto. Ele mantém um conjunto de *strokes* e gera permutas de unistroke (isto é *UnistrokeMemory*) que são usadas para avaliar candidatos depois deste gesto.

Arguments

name Identifica o nome do gesto - ele será retornado para você nos resultados de uma pesquisa *Recognizer.recognize()*.

Poderás ter qualquer número de objetos MultistrokeGesture com o mesmo nome; muitas definições de um gesto. O mesmo nome é dado a todas as permutações unistroke geradas. Obligatório, sem padrão.

strokes Uma lista de caminhos que representam o gesto. Um caminho é uma lista de objetos *Vector*.

```
gesture = MultistrokeGesture('my_gesture',_
    ↪strokes=[_
        [Vector(x1, y1), Vector(x2, y2), ..... ], #_
    ↪stroke 1
```

```
[Vector(), Vector(), Vector(), Vector() ] #_
→stroke 2
#, [stroke 3], [stroke 4], ...
])
```

Para fins de correspondência de modelos, todos os traços são combinados em uma única lista (unistroke). Ainda deverás especificar os traços individualmente e definir *stroke_sensitive* igual a True (sempre que possível).

Depois de fazer isso, as permutas de unistroke serão geradas e armazenadas imediatamente em *self.templates* para mais tarde, a menos que você defina o sinalizador *permute* para False.

prioritydetermina quando *Recognizer.recognize()* tentará corresponder a este modelo, as prioridades mais baixas serão avaliadas primeiro (apenas se for utilizado um filtro de prioridade *filter*). Deves usar a prioridade mais baixa em gestos que são mais propensos a corresponder. Por exemplo, define modelos de usuário com um número menor do que os modelos genéricos. O padrão é 100.

numpointsDetermina o número de pontos que este gesto deve ser redimensionado para (para fins de correspondência). O padrão é 16.

stroke_sensitiveDetermina se o número de Strokes (caminhos) neste gesto são necessário para ser o mesmo no gesto candidato (entrada do usuário) durante a correspondência. Se isso for Falso, os candidatos serão sempre avaliados, desconsiderando o número de traços. O padrão é True.

orientation_sensitiveDetermina se este gesto é sensível à orientação. Se True, alinha a orientação indicativa com a de oito orientações de base que requerem menos rotação. O padrão é True.

angle_similarityIsso é usado pela função: func: *Recognizer.recognize* quando um candidato é avaliado contra esse gesto. Se os ângulos entre eles estão muito longe, o modelo é considerado uma não-correspondência. O padrão é 30.0 (graus)

permuteSe False, não use o algoritmo Heap Permute para gerar diferentes ordens de percursos quando instanciado. Se você definir isso como False, um único UnistrokeTemplate construído a partir de *strokes* é usado.

add_stroke(stroke, permute=False)

Adicione um traço à lista *self.strokes*. Se *permute* for True, o método: meth: *permute* é chamado para gerar novos modelos unistroke

get_distance(*cand, tpl, numpoints=None*)

Calcula a distância deste Candidato para um UnistrokeTemplate. Retorna a distância do Cosseno entre o caminho percorrido.

numpoints irá preparar tanto o UnistrokeTemplate como o caminho Candidate para n pontos (quando necessário), você provavelmente não quer fazer isso.

match_candidate(*cand, **kwargs*)

Combine um determinado candidato com este objeto MultistrokeGesture. Testará contra todos os modelos e relatará resultados como uma lista de quatro itens:

*index 0*Melhor índice de template correspondente (em *self.templates*)

*index 1*Distância calculada desde o template até o caminho do candidato

*index 2*Lista de distâncias para todos os modelos. O índice de lista corresponde a: classe: índice *UnistrokeTemplate* em *self.templates*.

*index 3*Contador para o número de operações de correspondência executadas, isto é, modelos correspondentes ao candidato

permute()

Gere todas as possíveis permutações unistroke de *self.strokes* e salve a lista resultante de objetos *UnistrokeTemplate* em *self.templates*.

Citação de <http://faculty.washington.edu/wobbrock/pubs/gi-10.2.pdf>

```
We use Heap Permute [16] (p. 179) to generate all stroke_
orders
in a multistroke gesture. Then, to generate stroke_
directions for
each order, we treat each component stroke as a dichotomous
[0,1] variable. There are  $2^N$  combinations for N strokes,
so we
convert the decimal values 0 to  $2^{N-1}$ , inclusive, to binary
representations and regard each bit as indicating forward_
(0) or
reverse (1). This algorithm is often used to generate truth_
tables
in propositional logic.
```

Veja a seção 4.1: “\$N Algorithm” do paper vinculado para maiores informações.

Aviso: Usando permuta de heap para gestos com mais de 3 cursos pode resultar em um número muito grande de modelos (um gesto de 9 tempos = 38 milhões de modelos). Se você estiver lidando com esses tipos de gestos, você deve compor manualmente todas as ordens de curso desejado.

```
class kivy.multistroke.UnistrokeTemplate(name,          points=None,
                                           **kwargs)
```

Bases: object

Representa um caminho de (uni)traçado como uma lista de vetores. Normalmente, essa classe é instanciada por MultistrokeGesture e não pelo programador diretamente. No entanto, é possível compor manualmente objetos UnistrokeTemplate.

Arguments

*name*Identifica o nome da ação. Isso normalmente é herdado do objeto-pai MultistrokeGesture quando um modelo é criado.

*points*Uma lista de pontos que representa um caminho unistroke. Esta é normalmente uma das possíveis permutações de ordem de curso de um MultistrokeGesture.

*numpoints*O número de pontos que esse modelo deve (idealmente) ser redimensionado antes do processo coincidido. O padrão é 16, mas você pode usar um modelo de configurações específicas, se isso melhorar os resultados.

*orientation_sensitive*Determina se este modelo é sensível à orientação (True) ou totalmente invariante à rotação (False). O padrão é True.

Nota: Você obterá uma exceção se definir um skip-flag e tentar recuperar esses vetores.

add_point(*p*)

Adicione um ponto ao unistroke/path. Isso invalida todos os vetores previamente computados.

prepare(*numpoints=None*)

Esta função prepara o UnistrokeTemplate para corresponder a um dado número alvo de pontos (para redimensionar). 16 é ótimo.

```
class kivy.multistroke.Candidate(strokes=None,           numpoints=16,
                                   **kwargs)
```

Bases: object

Representa um conjunto de caminhos unistroke de entrada do usuário, ou seja, os dados a serem comparados contra um objeto da classe: *UnistrokeTemplate* usando o algoritmo Protractor. Por padrão, os dados são pré-computados para corresponder tanto à rotação limitada como totalmente invariante: objeto da classe: *UnistrokeTemplate*.

Arguments

strokes Veja: data: *MultistrokeGesture.strokes* para o exemplo de formato. Os traços do Candidato são simplesmente combinados a um unistroke na ordem dada. A ideia é que isso corresponderá a uma das permutações unistroke em *MultistrokeGesture.templates*.

numpoints O Padrão do Candidato N; este é apenas para um fallback, não é normalmente utilizado uma vez que n é conduzido pelo *UnistrokeTemplate* que estamos comparando.

skip_bounded Se True, não gere/armazene vetores que são limitados de rotação

skip_invariant Se True, não gere/armazene vetores que são invariantes de rotação

Observe que você IRÁ obter erros se você definir um skip-flag e, em seguida, tentar recuperar os dados.

add_stroke(stroke)

Adicione um traço ao candidato; isso invalidará todos os vetores previamente calculados

get_angle_similarity(*tpl*, *kargs*)**

(Apenas para uso interno) Calcule a similaridade de ângulo entre este Candidato e um objeto *UnistrokeTemplate*. Retorna um número que representa a similaridade de ângulo (menor é mais semelhante).

get_protractor_vector(*numpoints*, *orientation_sens*)

(Apenas para uso interno) Vetor de retorno para comparação com um *UnistrokeMemplate* com o Protractor

get_start_unit_vector(*numpoints*, *orientation_sens*)

(Apenas para uso interno) Obtenha o vetor de início para este Candidato, com o caminho redimensionado para pontos *numpoints*. Este é o primeiro passo no processo de correspondência. É comparado com um vetor de início do objeto *UnistrokeMemplate* para determinar a similaridade de ângulo.

prepare(*numpoints=None*)

Prepare os vetores Candidatos. self.strokes é combinado para um único unistroke (conectado de ponta a ponta), redimensionado para: attr: *numpoints* pontos, e então os vetores são calculados e armazenados em self.db (para uso de 'get_distance' e 'get_angle_similarity')

4.1.19 Utilitários Parser

Funções auxiliares usadas para análise de CSS.

kivy.parser.parse_color(*text*)

Analisa uma String para uma Cor Kivy. Formatos suportados:

- rgb(r, g, b)
- rgba(r, g, b, a)
- rgb
- rgba
- rrggb
- rrggbbaa

Para valores hexadecimais, você pode também usar:

- #rgb
- #rgba
- #rrggb
- #rrggbbaa

kivy.parser.parse_int

apelido de int

kivy.parser.parse_float

apelido de float

kivy.parser.parse_string(*text*)

Analisa uma String para um String (removendo aspas simples e duplas)

kivy.parser.parse_bool(*text*)

Analisa uma String para um Boolean, ignorado caso. “true”/“1” é *True*, “false”/“0” é *False*. Qualquer outra coisa será levantado uma exceção.

kivy.parser.parse_int2(*text*)

Analisa uma String para uma lista de exatamente 2 inteiros.

```
>>> print(parse_int2("12 54"))
12, 54
```

kivy.parser.parse_float4(*text*)

Analisa uma String para uma lista de exatamente 4 floats.

```
>>> parse_float4('54 87. 35 0')
54, 87., 35, 0
```

`kivy.parser.parse_filename(filename)`

Analisa uma nome de arquivo e busca por ele usando `resource_find()`. Se encontrado, o caminho do recurso é retornado, caso contrário, retorna o nome do arquivo não modificado (conforme especificado pelo caller).

4.1.20 Propriedades

A classe `Properties` são usadas quando você cria um `EventDispatcher`.

Aviso: Propriedades Kivy não podem ser confundidas com a definição de propriedade do Python (por exemplo o decorador `@property` e o tipo `<property>`).

As classes de propriedade do Kivy suportam:

Checagem de Valor/Validação Quando você atribui um novo valor a uma propriedade, o valor é checado pelas restrições de validação. Por exemplo, a validação para uma `OptionProperty` assegurará que o valor está dentro de uma lista de possibilidades pré-definidas. Validação para um `NumericProperty` assegurará que o valor é do tipo numérico. Isso previne muitos erros iniciais.

Observer Pattern Podes especificar o que deve acontecer quando o valor de uma propriedade for alterada. Podes vincular sua própria função como um callback para alterações de uma `Property`. Se, por exemplo, quiseres que um pedaço de código seja chamado quando uma propriedade do Widget `pos` mudar, podes vincular `bind` uma função pra ele.

Melhor Gerenciador de Memória A mesma instância de uma propriedade é compartilhada entre várias instâncias de Widget.

Comparação entre Python vs Kivy

Simples exemplo

Vamos comparar as propriedades Python e Kivy criando uma classe Python com a propriedade de nome 'a' como sendo um float:

```
class MyClass(object):
    def __init__(self, a=1.0):
        super(MyClass, self).__init__()
        self.a = a
```

Com Kivy, você pode fazer:

```
class MyClass(EventDispatcher):
    a = NumericProperty(1.0)
```

Profundidade sendo rastreada

Somente o “top level” de um objeto aninhado está sendo rastreado. Por exemplo:

```
my_list_prop = ListProperty([1, {'hi': 0}])
# Changing a top level element will trigger all `on_my_list_prop`_
# callbacks
my_list_prop[0] = 4
# Changing a deeper element will be ignored by all `on_my_list_`_
# `prop` callbacks
my_list_prop[1]['hi'] = 4
```

O mesmo vale para todas as propriedades Kivy do tipo container.

Checagem de Valores

Se desejas adicionar um validação para um valor mínimo/máximo permitido por uma propriedade, aqui está um implementação possível em Python:

```
class MyClass(object):
    def __init__(self, a=1):
        super(MyClass, self).__init__()
        self.a_min = 0
        self.a_max = 100
        self.a = a

    def _get_a(self):
        return self._a
    def _set_a(self, value):
        if value < self.a_min or value > self.a_max:
            raise ValueError('a out of bounds')
        self._a = value
    a = property(_get_a, _set_a)
```

A desvantagem é que você terá que fazer todo o trabalho. E isso torna-se trabalhoso e complexo se tiveres muitas propriedades. Como o Kivy, podes simplificar o processo:

```
class MyClass(EventDispatcher):
    a = BoundedNumericProperty(1, min=0, max=100)
```

Isso é tudo!

Tratamento de Erros

Se o valor configurado

```
# simply returns 0 if the value exceeds the bounds
bnp = BoundedNumericProperty(0, min=-500, max=500, errorvalue=0)
```

A segunda opção é para usar um parâmetro *errorhandler*. Um *errorhandler* é um invocável (simples argumento de função ou lambda) que pode retornar um valor substituto:

```
# returns the boundary value when exceeded
bnp = BoundedNumericProperty(0, min=-500, max=500,
    errorhandler=lambda x: 500 if x > 500 else -500)
```

Conclusão

As propriedades Kivy são mais fáceis de serem usadas do que a forma padrão de definir propriedades. Veja o próximo capítulo de como utiliza-las.

Modificação no Observe Property

Como dissemos, as propriedades de Kivy implementam o padrão Observer **Observer pattern**. Isso significa que você pode *bind()* para uma propriedade e ter sua própria função invocada quando o valor mudar.

Existem muitas formas de observar as mudanças.

Using Observe *bind()*

Podes observar uma alteração de propriedade usando o método *bind()* fora da classe

```
class MyClass(EventDispatcher):
    a = NumericProperty(1)

    def callback(instance, value):
        print('My callback is call from', instance)
        print('and the a value changed to', value)

ins = MyClass()
ins.bind(a=callback)

# At this point, any change to the a property will call your
# callback.
ins.a = 5    # callback called
```

```
ins.a = 5      # callback not called, because the value did not change  
ins.a = -1    # callback called
```

Nota: Os objetos de propriedade vivem ao nível da classe e gerenciam os valores anexados às instâncias. A reatribuição a nível de classe removerá a propriedade. Por exemplo, continuando com o código acima, `MyClass.a = 5` substitui o objeto de propriedade por um simples `int`.

Observe usando ‘on_<propname>’

Se você definir uma classe você mesmo, podes usar o callback ‘on_<propname>’

```
class MyClass(EventDispatcher):  
    a = NumericProperty(1)  
  
    def on_a(self, instance, value):  
        print('My property a changed to', value)
```

Aviso: Tenha cuidado com ‘on_<propname>’. Se estiveres criando um callback numa propriedade que está herdando, não deve esquecer de também chamar a função da superclasse.

Vinculando a propriedade de propriedades.

Quando se vincula a uma propriedade de uma propriedade, por exemplo, vinculando a uma propriedade numérica de um objeto salvo em uma propriedade de objeto, a atualização da propriedade de objeto que aponta para um novo objeto não voltará a vincular a propriedade numérica ao novo objeto. Por exemplo:

```
<MyWidget>:  
    Label:  
        id: first  
        text: 'First label'  
    Label:  
        id: second  
        text: 'Second label'  
    Button:  
        label: first  
        text: self.label.text  
        on_press: self.label = second
```

Ao clicar no botão, embora a propriedade do objeto de Label tenha sido alterada para o segundo Widget, o texto do botão não será alterado porque está vinculado diretamente à propriedade de texto do primeiro rótulo.

Na versão 1.9.0 foi introduzida a opção `rebind` que permitirá a atualização automática do `text` quando o `Label` for alterado, desde que habilitado. Veja [ObjectProperty](#).

```
class kivy.properties.Property  
Bases: object
```

Classe base para construção de propriedades mais complexas.

Esta classe lida com todos os setters e getters básicos, manipulação de tipo `None`, a lista de observadores e a inicialização de armazenamento. Esta classe não deve ser instanciada diretamente.

Por padrão, uma `Property` sempre terá um valor padrão:

```
class MyObject(Widget):  
  
    hello = Property('Hello world')
```

O valor padrão deve ser um valor que concorde com o tipo da `Property`. Por exemplo, você não pode definir uma lista para um `StringProperty` porque o `StringProperty` verificará o valor padrão.

`None` é um caso especial: você pode definir o valor `default` da propriedade para `None`, mas não podes definir `None` para uma propriedade posteriormente. Se você realmente desejar fazer isso, deves declarar a `Property` como `allownone=True`:

```
class MyObject(Widget):  
  
    hello = ObjectProperty(None, allownone=True)  
  
    # then later  
    a = MyObject()  
    a.hello = 'bleh' # working  
    a.hello = None # working too, because allownone is True.
```

Parameters

`default`: Especifica o valor padrão para a propriedade.

`**kwargs`: Se o parâmetro incluir um `errorhandler`, este deve ser um callable que deve ter um único argumento e retornar um valor substituto válido.

Se o parâmetro incluiem `errorvalue`, este deve ser um objeto. Se

definido, substituirá um valor de propriedade inválido (substitui o *errorhandler*).

Se os parâmetros incluem *force_dispatch*, os mesmos deverão ser um booleano. Caso seja *True*, nenhuma comparação de valores será feita, então o evento de propriedade será despachado mesmo se o novo valor corresponder ao valor antigo (por padrão, valores idênticos não são despachados para evitar a recursão infinita em bindings bidirecionais). Tenha cuidado, isso é apenas para uso avançado.

Alterado na versão 1.4.2: Parâmetro *errorhandler* e *errorvalue* adicionados.

Alterado na versão 1.9.0: Parâmetro *force_dispatch* adicionado

bind()

Adicionado um novo observador pra ser chamado somente quando o valor é alterado.

dispatch()

Despacha o valor alterado para todos os observadores.

Alterado na versão 1.1.0: O método é agora acessível desde o Python;.

Isso pode ser usado para forçar o despacho da propriedade, mesmo se o valor não for alterado:

```
button = Button()  
# get the Property class instance  
prop = button.property('text')  
# dispatch this property on the button instance  
prop.dispatch(button)
```

fbind()

Similar ao *bind*, exceto que ele não verifica se o observador já existe. Também expande e envia *largs* e *kwargs* para o callback. *funbind()* ou *unbind_uid* deverá ser invocado quando *unbinding()*. Será retornado um *uid* positivo exclusivo pra ser usado com *unbind_uid*.

funbind()

Remove o observador da nossa lista de observadores de Widget vinculados com *fbind()*. Remove a primeira correspondência encontrada, ao contrário de *unbind()*, que procura todas as correspondências.

get()

Retorna o valor da propriedade.

link()

Vincula a instância com o seu valor real.

Aviso: Uso somente interno.

Quando um Widget é definido e usa uma classe *Property*, a criação do objeto propriedade acontece, mas a instância não conhece nada a respeito do nome na classe Widget:

```
class MyWidget(Widget):
    uid = NumericProperty(0)
```

Neste exemplo, o *uid* será uma instância *NumericProperty()*, mas a instância da propriedade não conhece o seu nome. É por isso que *link* é usado em `Widget.__new__(). A função de link é usada para criar o espaço de armazenamento da propriedade para essa instância em específico.

set()

Define um novo para para a propriedade.

unbind()

Remove o observador da nossa lista de observadores de Widget.

unbind_uid()

Remove o observador da nossa lista de observadores vinculados com *fbind()* usando o *uid*.

class kivy.properties.NumericProperty

Bases: *kivy.properties.Property*

Propriedade que representa um valor numérico.

Parameters

defaultvalue: int ou float, e o padrão é 0 Especifica o valor padrão da propriedade.

```
>>> wid = Widget()
>>> wid.x = 42
>>> print(wid.x)
42
>>> wid.x = "plop"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "properties.pyx", line 93, in kivy.properties.Property._set_
    File "properties.pyx", line 111, in kivy.properties.Property._set_
    File "properties.pyx", line 159, in kivy.properties.NumericProperty.check
ValueError: NumericProperty accept only int/float
```

Alterado na versão 1.4.1: NumericProperty pode agora aceitar um texto personalizado e valor em de tupla para indicar um tipo, como "in", "pt", "px", "cm", "mm", no formato: '10pt' ou (10, 'pt').

get_format()

Retorna o formato usado para calculo numérico. O padrão é *px* (significa que o valor não foi alterado). Caso contrário, não pode ser um de 'in', 'pt', 'cm', 'mm'.

class kivy.properties.StringProperty

Bases: *kivy.properties.Property*

Propriedade que representa um valor String.

Parameters

defaultvalue: string, o padrão é ""Especifica o valor padrão da propriedade.

class kivy.properties.ListProperty

Bases: *kivy.properties.Property*

Propriedade que representa uma lista.

Parameters

defaultvalue: list, o padrão é []Especifica o valor padrão da propriedade.

Aviso: Ao atribuir uma lista a *ListProperty*, a lista armazenada na propriedade é uma cópia rasa da lista e não a lista original. Isto pode ser demonstrado com o seguinte exemplo:

```
>>> class MyWidget(Widget):
>>>     my_list = ListProperty([])

>>> widget = MyWidget()
>>> my_list = [1, 5, {'hi': 'hello'}]
>>> widget.my_list = my_list
>>> print(my_list is widget.my_list)
False
>>> my_list.append(10)
>>> print(my_list, widget.my_list)
[1, 5, {'hi': 'hello'}, 10] [1, 5, {'hi': 'hello'}]
```

No entanto, as alterações nos níveis internos também afetarão a propriedade, uma vez que a esta utiliza uma cópia superficial de *my_list*.

```
>>> my_list[2]['hi'] = 'bye'
>>> print(my_list, widget.my_list)
[1, 5, {'hi': 'bye'}, 10] [1, 5, {'hi': 'bye'}]
```

`class kivy.properties.ObjectProperty`

Bases: `kivy.properties.Property`

Propriedade que representa um objeto Python.

Parameters

`defaultvalue: tipo do objeto` Especifica o valor padrão da propriedade.

`rebind: bool, o valor padrão é False` Se as regras kv que usam este objeto como um atributo intermediário numa regra kv, atualizá-loão a propriedade ligada quando esse objeto mudar.

Esse comportamento padrão é para caso haja uma regra kv `text: self.a.b.c.d`, onde `a`, `b`, e `c` são propriedade com `rebind False` e `d` é uma `StringProperty`. Então, quando a regra é aplicada, `text` torna-se apenas `d`. Se `a`, `b`, ou `c` mudar, `text` ainda permanecerá vinculado a `d`. Além disso, se algum deles for `None` quando a regra foi inicialmente avaliada, por exemplo, `b` for “`None`”; então `text` é obrigado a `b` e não se tornará obrigado a `d` mesmo quando `b` for alterado e seja igual a `None`.

Ao definir “`rebind`” com sendo `True`, no entanto, a regra será reavaliada e todas as propriedades serão revertidas quando essa propriedade intermédia for alterada. Por exemplo, no exemplo acima, sempre que `b` mudar ou se tornar `None` se for `None` antes, `text` será novamente avaliado e se tornará `d`. O resultado geral é que `text` está agora vinculado a todas as propriedades entre `a`, `b`, ou `c` que tiverem `rebind` definido como sendo igual a `True`.

`**kwargs: uma lista de argumentos keyword`

`baseclass` Se `kwargs` incluir um argumento `baseclass`, este valor será usado para validação: `isinstance(value, kwargs['baseclass'])`.

Aviso: Para marcar a propriedade como alterado, você deve reatribuir um novo objeto Python.

Alterado na versão 1.9.0: `rebind` foi introduzido.

Alterado na versão 1.7.0: `baseclass` parâmetro adicionado.

`class kivy.properties.BooleanProperty`

Bases: `kivy.properties.Property`

Propriedade que representa somente um valor Booleano.

Parameters

defaultvalue: boolean Especifica o valor padrão da propriedade.

class kivy.properties.BoundedNumericProperty

Bases: *kivy.properties.Property*

Propriedade que representa um valor numérico dentro de um limite mínimo e máximo - dentro de um intervalo numérico.

Parameters

default: numeric Especifica o valor padrão da propriedade.

***kwargs*: uma lista de argumentos keyword Se um parâmetro *min* estiver incluído, este especifica o valor mínimo numérico que será aceito. Se o parâmetro *max* é incluso, este especifica o valor numérico máixmo que será aceito.

bounds

Retorna o valor mínimo/máximo.

Novo na versão 1.0.9.

get_max()

Retorna o valor máximo aceito para o BoundedNumericProperty em *obj*.
Retorna *None* se o valor máximo está definido. Veja [get_min](#) para um exemplo de uso.

Novo na versão 1.1.0.

get_min()

Retorna o valor mínimo aceitável para o BoundedNumericProperty em *obj*.
Retorna *None* se nenhum valor mínimo está definido:

```
class MyWidget(Widget):
    number = BoundedNumericProperty(0, min=-5, max=5)

widget = MyWidget()
print(widget.property('number').get_min(widget))
# will output -5
```

Novo na versão 1.1.0.

set_max()

Altera o valor máximo aceitável para o BoundedNumericProperty, somente para instância de *obj*. Defina como *None* se você desejar desativar isso. Veja [set_min](#) para um ver um exemplo de utilização.

Aviso: Alterar os limites não revalida o valor atual.

Novo na versão 1.1.0.

`set_min()`

Altera o valor mínimo aceitável para o BoundedNumericProperty, apenas para instâncias de *obj*. Defina como *None* se você desejar desativar isso:

```
class MyWidget(Widget):
    number = BoundedNumericProperty(0, min=-5, max=5)

    widget = MyWidget()
    # change the minimum to -10
    widget.property('number').set_min(widget, -10)
    # or disable the minimum check
    widget.property('number').set_min(widget, None)
```

Aviso: Alterar os limites não revalida o valor atual.

Novo na versão 1.1.0.

`class kivy.properties.OptionProperty`

Bases: *kivy.properties.Property*

Propriedade que representa uma String de um lista predefinida de opções válidas.

Se a String definida na propriedade não está na lista de opções válidas (passado no momento da criação), uma exceção *ValueError* será levantada.

Parameters

`default:` qualquer tipo válido numa lista de opçõesEspecifica o valor padrão da propriedade.

`kwargs:`** uma lista de argumentos keywordDeve incluir um parâmetro *options* especificando um lista (não tupla) de opções válidas.

Por exemplo:

```
class MyWidget(Widget):
    state = OptionProperty("None", options=[ "On", "Off", "None
    ↴" ])
```

options

Retorna as opções disponíveis.

Novo na versão 1.0.9.

`class kivy.properties.ReferenceListProperty`

Bases: *kivy.properties.Property*

Propriedade que permite a criação de uma tupla de outras propriedades.

Por exemplo, se `x` e `y` forem `NumericProperty`s podemos criar uma `ReferenceListProperty` para o `pos`. Se você alterar o valor de `pos`, ele mudará automaticamente os valores de `x` e `y` em conformidade. Se leres o valor de `pos`, ele retornará uma tupla com os valores de `x` e `y`.

Por exemplo:

```
class MyWidget(Dispatcher):
    x = NumericProperty(0)
    y = NumericProperty(0)
    pos = ReferenceListProperty(x, y)
```

class kivy.properties.AliasProperty
Bases: `kivy.properties.Property`

Cria uma propriedade com um getter e setter personalizado.

Se você não encontrar uma classe de `Property` que complete suas necessidades, podes fazer sua própria criando um método getter e setter personalizado.

Exemplo de `kivy/uix/widget.py`:

```
def get_right(self):
    return self.x + self.width
def set_right(self, value):
    self.x = value - self.width
right = AliasProperty(get_right, set_right, bind=['x', 'width'])
```

Parameters

getter: functionFunção para usar como uma propriedade getter

setter: functionFunção para usar como um `setter` de propriedade.

As propriedades que escutam apelidos de propriedade não serão atualizadas quando a propriedade for definida (por exemplo, `right = 10`), a menos que `setter` devolva `True`.

bind: list/tuplePropriedades para observar alterações, como propriedade de nome String

cache: booleanSe `True`, o valor será armazenado em cache, até que um dos elementos vinculados seja alterado

rebind: bool, o valor padrão é FalseVeja `ObjectProperty` para maiores detalhes.

Alterado na versão 1.9.0: `rebind` foi introduzido.

Alterado na versão 1.4.0: Parâmetro `cache` adicionado.

class kivy.properties.DictProperty
Bases: `kivy.properties.Property`

Propriedade que representa um dict.

Parameters

defaultvalue: dict, o padrão é *None*Especifica o valor padrão da propriedade.

rebind: bool, o valor padrão é *False*Veja [ObjectProperty](#) para maiores detalhes.

Alterado na versão 1.9.0: *rebind* foi introduzido.

Aviso: Semelhante a classe [ListProperty](#), quando um dict foi atribuído a um [DictProperty](#), o dict armazenado na propriedade será uma cópia rasa do dict e não o dict original. Veja a [ListProperty](#) para maios informações.

class kivy.properties.VariableListProperty

Bases: [kivy.properties.Property](#)

Uma ListProperty que permite você trabalhar com uma quantidade variável de itens de lista e expandi-los para o tamanho de lista que desejas.

Por exemplo, o padding de um GridLayout's usado para somente aceitar um valor numérico que foi aplicado igualmente ao *left*, *top*, *right* e *bottom* do GridLayout. Agora o padding pode ser dado um, dois ou quatro valores, que são expandidos num comprimento quatro lista [*left*, *top*, *right* e *bottom*] e armazenados na propriedade.

Parameters

default: uma lista padrão de valoresEspecifica o valor padrão para a lista.

length: int, um de 2 ou 4.Especifica o comprimento da lista final.
A lista *default* será expandida para corresponder a uma lista deste comprimento.

****kwargs:** uma lista de argumentos keywordAtualmente não utilizado.

Tenha em mente que a lista *default* é expandida para uma lista de comprimento 4, aqui temos alguns exemplos de como um VariabelListProperty's é manipulado.

- VariableListProperty([1]) representa [1, 1, 1, 1].
- VariableListProperty([1, 2]) representa [1, 2, 1, 2].
- VariableListProperty(['1px', (2, 'px'), 3, 4.0]) representa [1, 2, 3, 4.0].
- VariableListProperty(5) representa [5, 5, 5, 5].
- VariableListProperty(3, length=2) representa [3, 3].

Novo na versão 1.7.0.

class kivy.properties.ConfigParserProperty
Bases: *kivy.properties.Property*

Propriedade que permite vincular as alterações aos valores das configurações de um *ConfigParser* como também para vincular o valor do ConfigParser a outras propriedades.

Um ConfigParser é composto de seções, onde cada seção tem um número de chaves e valores associados a esta chave. ConfigParserProperty permite que você automaticamente escute e altere os valores de chaves especificadas com base em outras propriedades Kivy.

Por exemplo, digamos que queremos que um TextInput escreva automaticamente seu valor, representado como um *int*, na seção *info* de um *ConfigParser*. Além disso, os TextInput devem atualizar os valores dos campos do *ConfigParser*. Finalmente, seus valores devem ser exibidos em um Label. Em py:

```
class Info(Label):  
  
    number = ConfigParserProperty(0, 'info', 'number', 'example'  
        ↴ ,  
                    val_type=int, errorvalue=41)  
  
    def __init__(self, **kw):  
        super(Info, self).__init__(**kw)  
        config = ConfigParser(name='example')
```

O código acima cria uma propriedade que está conectada à chave *number* na seção 'info' do *ConfigParser* chamado *example*. Inicialmente, este *ConfigParser* não existe. Então, em *__init__*, um *ConfigParser* é criado com o nome *example*, que é automaticamente vinculado a esta propriedade. Temos em kv:

```
BoxLayout:  
    TextInput:  
        id: number  
        text: str(info.number)  
    Info:  
        id: info  
        number: number.text  
        text: 'Number: {}'.format(self.number)
```

Você notará que precisamos fazer *text: str(info.number)*, porque o valor dessa propriedade é sempre um *int*, porque especificamos *int* como *val_type*. No entanto, podemos atribuir qualquer coisa à propriedade, por exemplo, *number: number.text* que atribui uma String, porque é convertida instantaneamente com a chamada de retorno *val_type*.

Nota: Se um arquivo tiver sido aberto para este `ConfigParser` usando `read()`, então `write()` será chamado a cada alteração de propriedade, mantendo o arquivo atualizado.

Aviso: Recomenda-se que o objeto `ConfigParser` seja atribuído à propriedade após a árvore `kv` ter sido construída (por exemplo, agendar no próximo quadro do `init`). Isso ocorre porque a árvore `kv` e suas propriedades, quando construídas, são avaliadas em sua própria ordem, portanto, quaisquer valores iniciais no analisador podem ser substituídos por objetos aos quais está vinculado. Assim, no exemplo acima, o `TextInput` pode estar inicialmente vazio e se `number: number.text` for avaliado antes de `text: str (info.number)`, o valor config será sobreescrito com o valor de texto (vazio).

Parameters

default: tipo de objeto Especifica o valor padrão para a chave. Se o analisador associado com esta propriedade não tiver essa seção ou chave, ela será criada com o valor atual, que é o valor padrão inicial.

section: tipo String A seção no `ConfigParser` onde a chave/valor será gravada. Deve ser providenciado. Se a seção não existir, ela será criada.

key: tipo String A chave na seção `section` onde o valor será gravado. Deve ser providenciado. Se a chave não existir, ela será criada e o valor atual será gravado, caso contrário seu valor será usado.

config: String ou instância de `ConfigParser`. A instância do `ConfigParser` para associar a esta propriedade se não for `None`. Se for uma `String`, será usada a instância `ConfigParser` cujo `name` é o valor de `config`. Caso nenhum analisador exista, sempre que um `ConfigParser` com esse nome for criado, ele será automaticamente vinculado a esta propriedade.

Sempre que um `ConfigParser` se torna vinculado a uma propriedade, se a seção ou chave não existir, o valor da propriedade atual será usado para criar essa chave, caso contrário, o valor da chave existente será usado para o valor da propriedade; substituindo seu valor atual. Você pode alterar o `ConfigParser` associado a esta propriedade se uma string foi usada aqui, alterando o: `attr: ~kivy.config.ConfigParser.name` de uma instância do `ConfigParser` existente ou novo. Ou através de: méthod: `set_config`.

****kwargs: uma lista de argumentos keyword**

val_type: um objeto callable Os key values são salvos no *ConfigParser* como sequências de caracteres. Quando o valor *ConfigParser* for lido internamente e atribuído à propriedade ou quando o usuário mudar o valor da propriedade diretamente, se *val_type* não for *None*, ele será chamado com o novo valor como entrada e ele deverá retornar o valor convertido para o apropriado tipo aceito na esta propriedade. Por exemplo, se a propriedade representa ints, *val_type* pode ser simplesmente *int*.

Se o callback' gerar um *ValueError*, *errorvalue* ou *errorhandler* serão usados se fornecidos. Dica: a função *getboolean* do *ConfigParser* também pode ser útil aqui para converter num tipo booleano.

verify: um objeto callable Pode ser usado para restringir os valores permitidos da propriedade. Para cada valor atribuído à propriedade, se esta for especificada, *verify* é invocado com um novo valor, e se ele retornar *True* o valor é aceito, caso contrário, *errorvalue* ou *errorhandler* será usado se for fornecido ou um *ValueError* é aumentado.

Novo na versão 1.9.0.

set_config()

Define o objeto *ConfigParser* a ser usado por essa propriedade. Normalmente, o *ConfigParser* é definido ao inicializar o *Property* usando o parâmetro *config*.

Parameters

config: uma Instância de *ConfigParser*. A instância a ser usada para ouvir e salvar o valor da propriedade muda. Se *None*, ele desconecta o *ConfigParser* atualmente usado.

```
class MyWidget(Widget):
    username = ConfigParserProperty(' ', 'info', 'name', ↵
        None)

    widget = MyWidget()
    widget.property('username').set_config(ConfigParser())
```

4.1.21 Gerenciador de Recursos

Gerenciadores de recursos podem ser um problema se tiveres vários caminhos e projetos. O Kivy oferece 2 funções para procurar por um recurso específico através de uma

lista de caminhos.

Pesquisa de Recursos

Quando o Kivy procurar um recurso, por exemplo, uma imagem ou um arquivo kv, ele procura através de um conjunto predeterminado de pastas. Você pode modificar esta lista de pastas usando as funções `resource_add_path()` e `resource_remove_path()`.

Personalizando o Kivy

Essas funções também podem ser úteis se quiseres substituir os recursos padrão do Kivy pelos seus próprios recursos. Por exemplo, se desejas personalizar ou re-estilizar o Kivy, podes forçar seus arquivos `style.kv` * ou `*data/defaulttheme-0.png` a serem usados em preferência aos padrões simplesmente adicionando o caminho para a sua alternativa preferida através do método `resource_add_path()`.

Como quase todos os recursos do Kivy são pesquisados usando o `resource_find()`, então podes usar esta abordagem para adicionar fontes e layouts de teclado e para substituir imagens e ícones.

`kivy.resources.resource_find(filename)`

Procura por um resource na lista de caminhos. Utilize `resource_add_path` para adicionar um caminho personalizado para a procura.

`kivy.resources.resource_add_path(path)`

Adicione um caminho personalizado para pesquisar.

`kivy.resources.resource_remove_path(path)`

Remova um caminho de busca.

Novo na versão 1.0.8.

4.1.22 Suporte

Ativa outros frameworks/toolkits dentro do event loop do Kivy.

`kivy.support.install_gobject_iteration()`

Importe e instale

`kivy.support.install_twisted_reactor(**kwargs)`

Instala uma Threaded Twisted reactor, que agendará uma iteração do reactor antes do próximo Frame somente quando o Twisted precisar fazer algum trabalho.

Quaisquer argumentos ou argumentos palavra-chave passados para esta função serão passados na função de intercalação de reatores threadedselect. Esses são os argumentos que normalmente se passariam ao reator twisted.

Ao contrário do Twisted reactor padrão, o reactor instalado não manipulará quaisquer sinais a menos que você defina o argumento de palavra-chave ‘installSignalHandlers’ como 1 explicitamente. Isto é feito para permitir que o Kivy manipule os sinais como de costume, a menos que deseje especificamente que o Twisted reactor manipule os sinais (por exemplo SIGINT).

Nota: O Twisted não está incluído na compilação do iOS por padrão. Para usá-lo no iOS, coloque a distribuição do Twisted (e a dependência de *zope.interface*) no diretório do seu aplicativo.

kivy.support.uninstall_twisted_reactor()

Desinstala o Twisted Reactor do Kivy. Mais nenhum Twisted reactor serão executadas após isso ser invocado. Use isso para limpar o *twisted.internet.reactor*.

Novo na versão 1.9.0.

kivy.support.install_android()

Instale hooks para a plataforma Android.

- Dorme automaticamente quando o dispositivo Android estiver pausado.
- Fecha automaticamente a aplicação quando a tecla return é pressionada.

4.1.23 Util

O módulo Utils fornece utilidades em funções e classes gerais que podem ser úteis em várias aplicações. Inclui matemática, cor, álgebra e plataforma.

Alterado na versão 1.6.0: Classe OrderDict foi removida. Usar collections.OrderedDict.

kivy.utils.intersection(set1, set2)

Retorna a interseção de 2 listas.

kivy.utils.difference(set1, set2)

Retorna a diferença entre 2 listas.

kivy.utils.strtotuple(s)

Converte string par em par com verificação de segurança. Desenhado para ser usado com função eval():

```
a = (12, 54, 68)
b = str(a)           # return '(12, 54, 68)'
c = strtotuple(b)   # return (12, 54, 68)
```

kivy.utils.get_color_from_hex(s)

Transforma string cor hexa para *Color*.

`kivy.utils.get_hex_from_color(color)`

Transforma `Color` para valor hexa:

```
>>> get_hex_from_color((0, 1, 0))
'#00ff00'
>>> get_hex_from_color((.25, .77, .90, .5))
'#3fc4e57f'
```

Novo na versão 1.5.0.

`kivy.utils.get_random_color(alpha=1.0)`

Retorna cor randômica (4 pares).

Parameters

`alpha: float, padrão 1.0` Se alpha = 'random', um valor aleatório de alpha é gerado.

`kivy.utils.is_color_transparent(c)`

Retorna True se canal alpha é 0.

`kivy.utils.boundary(value, minvalue, maxvalue)`

Limita valor entre mínimo e máximo.

`kivy.utils.deprecated(func)`

Esse decorador pode ser usado para marcar funções como obsoletas. Irá apresentar um aviso emitido na primeira vez que a função for usada.

`class kivy.utils.SafeList`

Bases: `list`

Listar com método `clear()`.

Aviso: Uso função `iterate()` irá diminuir sua performance

`kivy.utils.interpolate(value_from, value_to, step=10)`

Interpolar entre dois valores. Pode ser útil para melhorar transições. Por exemplo:

```
# instead of setting directly
self.pos = pos

# use interpolate, and you'll have a nicer transition
self.pos = interpolate(self.pos, new_pos)
```

Aviso: Essas interpolações funcionam somente em listas/pares/duplas com a mesma dimensão. Nenhum teste é feito para ver se as dimensões são a mesma.

```
class kivy.utils.QueryDict
```

Bases: dict

QueryDict é dict() que pode ser consultado.

```
d = QueryDict()  
# create a key named toto, with the value 1  
d.toto = 1  
# it's the same as  
d['toto'] = 1
```

Novo na versão 1.0.4.

```
kivy.utils.platform = 'linux'
```

String identificando sistema operacional corrente. Pode ser um desses: 'win', 'linux', 'android', 'macosx', 'ios' ou 'unknown'. Pode ser usado assim:

```
from kivy.utils import platform  
if platform == 'linux':  
    do_linux_things()
```

Novo na versão 1.3.0.

Alterado na versão 1.8.0: plataforma é uma variável em vez de função.

```
kivy.utils.escape_markup(text)
```

Marcação de escape encontrada em um texto. Para ser usado quando um texto marcado é acionado em um Label:

```
untrusted_text = escape_markup('Look at the example [1]')  
text = '[color=ff0000]' + untrusted_text + '[/color]'  
w = Label(text=text, markup=True)
```

Novo na versão 1.3.0.

```
class kivy.utils.reify(func)
```

Bases: object

Coloca o resultado do método o qual usa descriptor (non-data) decorador na instância dict após a primeira chamada, efetivamente substitui o decorador com uma instância variável.

Atua como @property, exceto que a função é chamada única; após isso o valor é colocado em cache como atributo regular. Isso habilita criação de atributo em objetos que parecem ser imutáveis.

Derivado de [Pyramid project](#).

Para usar como um decorador:

```
@reify  
def lazy(self):
```

```

    ...
    return hard_to_compute_int
first_time = self.lazy # lazy is reify obj, reify.__get__()
→runs
second_time = self.lazy # lazy is hard_to_compute_int

```

kivy.utils.rgb(a, *args)

Retorna cor Kivy (4 valores intervalo 0-1) a partir de string hexa ou lista de valores 0-255.

Novo na versão 1.10.0.

4.1.24 Vetor

A classe `:Vector` representa um vetor 2D (x, y). Nossa implementação é construída em cima de um objeto list do Python.

Exemplo de construção de um Vector:

```

>>> # Construct a point at 82,34
>>> v = Vector(82, 34)
>>> v[0]
82
>>> v.x
82
>>> v[1]
34
>>> v.y
34

>>> # Construct by giving a list of 2 values
>>> pos = (93, 45)
>>> v = Vector(pos)
>>> v[0]
93
>>> v.x
93
>>> v[1]
45
>>> v.y
45

```

Uso otimizado

Na maioria das vezes, podes usar uma lista com argumentos ao invés de usar um Vetor. Por exemplo, se você quiser calcular a distância entre 2 pontos:

```

a = (10, 10)
b = (87, 34)

# optimized method
print('distance between a and b:', Vector(a).distance(b))

# non-optimized method
va = Vector(a)
vb = Vector(b)
print('distance between a and b:', va.distance(vb))

```

Operadores de Vetores

A classe `Vector` suporta alguns operadores numéricos tais como `+, - , /

```

>>> Vector(1, 1) + Vector(9, 5)
[10, 6]

>>> Vector(9, 5) - Vector(5, 5)
[4, 0]

>>> Vector(10, 10) / Vector(2., 4.)
[5.0, 2.5]

>>> Vector(10, 10) / 5.
[2.0, 2.0]

```

Você também pode usar operadores in-place:

```

>>> v = Vector(1, 1)
>>> v += 2
>>> v
[3, 3]
>>> v *= 5
[15, 15]
>>> v /= 2.
[7.5, 7.5]

```

`class kivy.vector.Vector(*largs)`

Bases: `list`

Classe `Vector`. Consulte a documentação do módulo para obter mais informações.

`angle(a)`

Calcula o ângulo entre A e B e retorna o ângulo em graus.

```
>>> Vector(100, 0).angle((0, 100))
-90.0
>>> Vector(87, 23).angle((-77, 10))
-157.7920283010705
```

distance(*to*)

Retorna a distância entre dois pontos.

```
>>> Vector(10, 10).distance((5, 10))
5.
>>> a = (90, 33)
>>> b = (76, 34)
>>> Vector(a).distance(b)
14.035668847618199
```

distance2(*to*)

Retorna a distância entre dois pontos ao quadrado.

```
>>> Vector(10, 10).distance2((5, 10))
25
```

dot(*a*)

Calcula o produto do ponto A e B.

```
>>> Vector(2, 4).dot((2, 2))
12
```

static in_bbox(*point, a, b*)

Retorna True se *point* estiver na caixa delimitadora definida por *a* e *b*.

```
>>> bmin = (0, 0)
>>> bmax = (100, 100)
>>> Vector.in_bbox((50, 50), bmin, bmax)
True
>>> Vector.in_bbox((647, -10), bmin, bmax)
False
```

length()

Retorna o comprimento do Vetor.

```
>>> Vector(10, 10).length()
14.142135623730951
>>> pos = (10, 10)
>>> Vector(pos).length()
14.142135623730951
```

length2()

Retorna o comprimento do vetor ao quadrado.

```

>>> Vector(10, 10).length2()
200
>>> pos = (10, 10)
>>> Vector(pos).length2()
200

```

static line_intersection(*v1, v2, v3, v4*)

Localiza o ponto de intersecção entre as linhas (1) $v1 \rightarrow v2$ e (2) $v3 \rightarrow v4$ e retorna-o como sendo um objeto vetorial.

```

>>> a = (98, 28)
>>> b = (72, 33)
>>> c = (10, -5)
>>> d = (20, 88)
>>> Vector.line_intersection(a, b, c, d)
[15.25931928687196, 43.911669367909241]

```

Aviso: Este é um método de interseção de linha, não uma interseção de segmento.

Para matemática veja: http://en.wikipedia.org/wiki/Line-line_intersection

normalize()

Retorna um novo Vetor que tem a mesma direção que *vec*, mas tem um comprimento de um.

```

>>> v = Vector(88, 33).normalize()
>>> v
[0.93632917756904444, 0.3511234415883917]
>>> v.length()
1.0

```

rotate(*angle*)

Gira o Vetor com um ângulo em graus.

```

>>> v = Vector(100, 0)
>>> v.rotate(45)
>>> v
[70.710678118654755, 70.710678118654741]

```

static segment_intersection(*v1, v2, v3, v4*)

Localiza o ponto de intersecção entre os segmentos (1) $v1 \rightarrow v2$ e (2) $v3 \rightarrow v4$ e retorna-o como sendo um objeto vetorial.

```

>>> a = (98, 28)
>>> b = (72, 33)
>>> c = (10, -5)
>>> d = (20, 88)
>>> Vector.segment_intersection(a, b, c, d)
None

```

```

>>> a = (0, 0)
>>> b = (10, 10)
>>> c = (0, 10)
>>> d = (10, 0)
>>> Vector.segment_intersection(a, b, c, d)
[5, 5]

```

x

x representa o primeiro elemento na lista.

```

>>> v = Vector(12, 23)
>>> v[0]
12
>>> v.x
12

```

y

y representa o segundo elemento da lista.

```

>>> v = Vector(12, 23)
>>> v[1]
23
>>> v.y
23

```

4.1.25 Método fraco

A **WeakMethod** é usado pela classe **Clock** para permitir referências a um método vinculado que permite objetos que os objetos associados sejam coletados pelo “Garbage Collector”. Veja a documentação para maiores informações *examples/core/clock_method.py*.

Esta classe **WeakMethod** foi retirada da receita <http://code.activestate.com/recipes/81253/>, baseada na versão nicodemus. Muito obrigado nicodemus!

```
class kivy.weakmethod.WeakMethod(method)
    Bases: object
```

Implementação de um **weakref** para função e métodos associados.

is_dead()

Retorna *True* se o callable referenciado for um método vinculado a uma instância que não existe mais. Do contrário, retorna *False*.

4.1.26 Proxy fraco

Para permitir a coleta de lixo “Garbage Collector”, o Weak Proxy fornece ([referências fracas para objetos](#)) [weak references to objects](#). Isso efetivamente melhora o [weakref.proxy](#) pela adição de suporte a comparação.

class kivy.weakproxy.WeakProxy

Bases: `object`

Substituição de `weakref.proxy` para permitir comparações.

4.2 Adaptadores

Novo na versão 1.5.0.

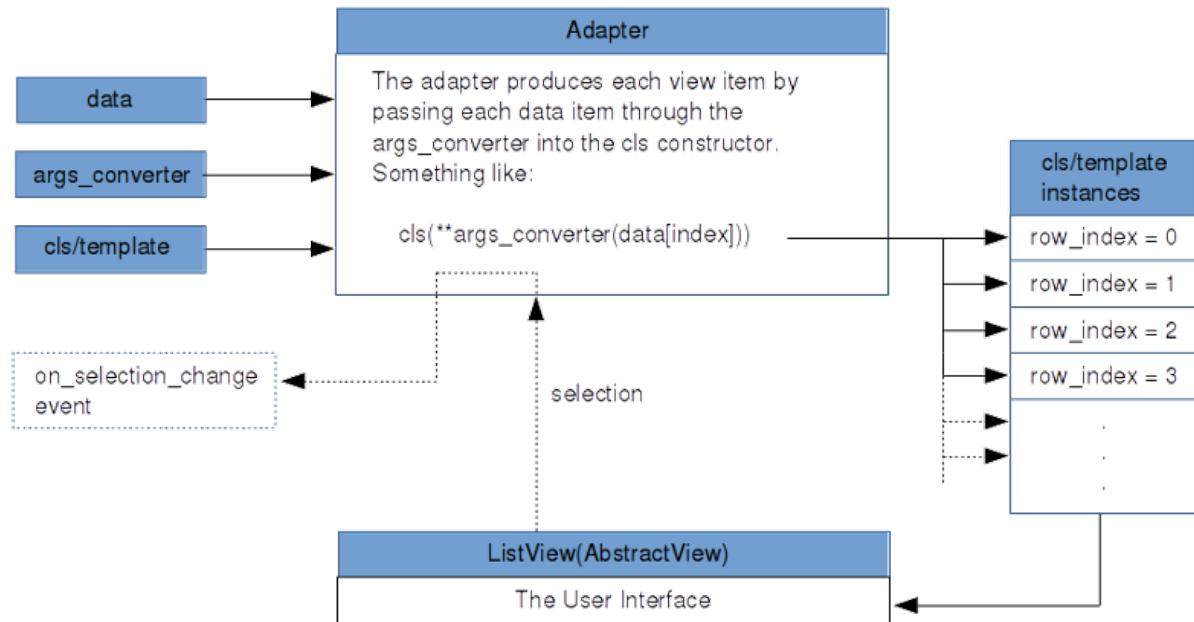
Nota: The feature has been deprecated.

Um adaptador é um tipo de classe de controlador que intermediá o processamento e disponibiliza os dados para serem exibidos. Isso é feito pela geração de um modelo, geralmente uma lista de itens do tipo [SelectableView](#), que são consumidas e apresentadas pelas View. Views são top-left widget, tais como [ListView](#), que permitem o usuário rolar os itens e interagir com as informações.

4.2.1 O Conceito

Adaptadores Kivy são modelados sob o [Adapter design pattern](#). Conceitualmente, desempenham o papel de um “controlador” entre os dados e a visualizações num sistema hierárquico MVC Model-View-Controller <<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>>’.

O papel de um adaptador pode ser representado da seguinte forma:



4.2.2 Os componentes

Os componentes envolvidos no processo são:

- **Adaptadores:** O adaptador tem o papel de mediador entre a interface de usuário e os seus dados. Ele gerencia a criação dos elementos da view para o model usando o args_converter para preparar os argumentos do construtor para os seus itens da view cls/template.

A classe base: `class:Adapter` é extendida para as classes: `class:SimpleListAdapter` e para classe: `class:ListAdapter`. A classe: `class:DictAdapter` é mais avançada e flexível do que a subclasse `ListAdapter`

Adapter, SimpleListAdapter, ListAdapter, DictAdapter.

- **Models:** Os dados nos quais um adaptador faz o papel de ponte para os views podem ser qualquer tipo de dado. Entretanto, por conveniência, as classes model mixin podem facilitar a preparação ou formatação dos dados para uso no sistema. Para operações de seleção, a `:classe:'SelectableDataItem'` pode opcionalmente preparar dados de itens para receber e enviar seleções de informação (dados de itens não necessitam ser “selection-aware”, mas em alguns casos pode ser desejado).

`:doc:`api-kivy.adapters.models``

- Os conversores de argumentos são feitos pelo programador de aplicativo para fazer o trabalho de converter itens de dados para dicionários de argumento adequados para instanciar Views. Com efeito, eles tomam cada linha de seus dados e criam dicionários que são passados para os construtores do seu `cls / template` que são então usados preencher sua View.

Conversor de Lista de itens de Argumentos.

- **Views:** O Models de seus dados são apresentados ao usuário através da views. Cada um dos items de dados criados corresponde a um subitem (o cls ou template) mostrado numa lista pela View. A classe base: *AbstractView* atualmente possui uma implementação concreta: a classe: *ListView*.

View abstrata, ListView.

4.2.3 Adapter

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: This code is still experimental, and its API is subject to change in a future version.

An *Adapter* is a bridge between data and an *AbstractView* or one of its subclasses, such as a *ListView*.

The following arguments can be passed to the constructor to initialise the corresponding properties:

- *data*: for any sort of data to be used in a view. For an *Adapter*, data can be an object as well as a list, dict, etc. For a *ListAdapter*, data should be a list. For a *DictAdapter*, data should be a dict.
- *cls*: the class used to instantiate each list item view instance (Use this or the template argument).
- *template*: a kv template to use to instantiate each list item view instance (Use this or the cls argument).
- *args_converter*: a function used to transform the data items in preparation for either a cls instantiation or a kv template invocation. If no args_converter is provided, the data items are assumed to be simple strings.

Please refer to the *adapters* documentation for an overview of how adapters are used.

```
class kivy.adapters.adapter.Adapter(*args, **kwargs)
    Bases: kivy.event.EventDispatcher
```

An *Adapter* is a bridge between data and an *AbstractView* or one of its subclasses, such as a *ListView*.

args_converter

A function that prepares an args dict for the cls or kv template to build a view from a data item.

If an args_converter is not provided, a default one is set that assumes simple content in the form of a list of strings.

args_converter is an *ObjectProperty* and defaults to None.

cls

A class for instantiating a given view item (Use this or template). If this is not set and neither is the template, a *Label* is used for the view item.

cls is an *ObjectProperty* and defaults to None.

data

The data for which a view is to be constructed using either the cls or template provided, together with the args_converter provided or the default args_converter.

In this base class, data is an ObjectProperty, so it could be used for a wide variety of single-view needs.

Subclasses may override it in order to use another data type, such as a *ListProperty* or *DictProperty* as appropriate. For example, in a *ListAdapter*, data is a *ListProperty*.

data is an *ObjectProperty* and defaults to None.

get_cls()

Novo na versão 1.9.0.

Returns the widget type specified by self.cls. If it is a string, the *Factory* is queried to retrieve the widget class with the given name, otherwise it is returned directly.

template

A kv template for instantiating a given view item (Use this or cls).

template is an *ObjectProperty* and defaults to None.

4.2.4 DictAdapter

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

Uma classe: ***DictAdapter*** é um adaptador em torno do dicionario de registros Python. Ele amplia as capacidades de lista da classe: ***ListAdapter***.

Se você deseja ter um adaptador de lista bare-bones, sem seleção, use a classe : kivy.adapters.simplelistadapter.SimpleListAdapter

class kivy.adapters.dictadapter.DictAdapter(kwargs)**

Bases: ***kivy.adapters.listadapter.ListAdapter***

Uma classe: ***DictAdapter*** é um adaptador em torno do dicionario de registros Python. Ele amplia as capacidades de lista da classe: ***ListAdapter***.

cut_to_sel(*args)

O mesmo de trim_to_sel, mas os itens intervenientes na lista no intervalo serão também cortados, deixam apenas na lista os itens que estejam selecionados.

sorted_keys será atualizado pelo update_for_new_data().

data

Um dicionário dict que indexa regtos por chaves que são equivalentes às chaves nas sorted_keys, ou que sejam um superconjunto das chaves em sorted_keys.

Os valores podem ser strings, classes instanciadas, dicts, etc.

data é uma ***DictProperty*** e definida por padrão como None

sorted_keys

A propriedade sorted_keys contém a lista dos objetos que podem ter hash (devem ser strings) isso será usado diretamente se nenhuma função args_converter for informada. Se existe um args_converter, o registro recebido de um lookup nos dados, usando as chaves de sorted_keys, será passada para instânciar a lista do item.

sorted_keys é uma ***ListProperty*** e o padrão é [].

trim_left_of_sel(*args)

Lista de Itens Recortar com índices em sorted_keys que são menos que o primeiro item selecionado, se existir uma seleção.

sorted_keys será atualizado pelo update_for_new_data().

trim_right_of_sel(*args)

Corte lista de itens com índices de sorted_keys que são maiores do que o índice do último item selecionado, se houver uma seleção.

sorted_keys será atualizado pelo update_for_new_data().

trim_to_sel(*args)

Corte lista de itens com índices de sorted_keys que são maiores do que o índice do último item selecionado, se houver uma seleção. Isso preserva a lista de itens intervenientes dentro do intervalo selecionado.

`sorted_keys` será atualizado pelo `update_for_new_data()`.

4.2.5 Conversor de Lista de itens de Argumentos

Novo na versão 1.5.

Nota: The feature has been deprecated.

A lista do adaptador de conversor padrão é uma função (exibida abaixo) que pega o índice de uma linha e uma string. E retorna um dict com a string como um item de `text`, em conjunto com as 2 propriedades adaptadas para itens de texto simples com uma altura de 25.

Uso simples

Conversores de argumentos podem ser funções normais, ou como nesse caso, conversor de argumento padrão lambdas:

```
list_item_args_converter = lambda row_index, x: {'text': x,
                                                'size_hint_y': None,
                                                'height': 25}
```

Uso avançando

Typically, having the argument converter perform a simple mapping suffices. There are times, however, when more complex manipulation is required. When using `CompositeListItem`, it is possible to specify a list of cls dictionaries. This allows you to compose a single view item out of multiple classes, each of which can receive their own class constructor arguments via the `kwargs` keyword:

```
args_converter = lambda row_index, rec: \
    {'text': rec['text'],
     'size_hint_y': None,
     'height': 25,
     'cls_dicts': [{ 'cls': ListItemButton,
                     'kwargs': { 'text': rec['text']}},
                   { 'cls': ListItemLabel,
                     'kwargs': { 'text': rec['text'],
                                 'is_representing_cls': True}},
                   { 'cls': ListItemButton,
                     'kwargs': { 'text': rec['text']} } ] }
```

Por favor, veja a lista `list_composite.py` para um exemplo completo.

4.2.6ListAdapter

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

Uma [ListAdapter](#) é um adaptador que envolve uma lista python e adiciona suporte para operações de seleção. Se você desejar um list adapter simples, sem seleção, use a [SimpleListAdapter](#).

De uma :class:`~kivy.adapters.Adapter`, uma [ListAdapter](#) herda cls, template e propriedades args_converter e incorpora outros comportamentos de controle de seleção:

- [*selection*](#): é uma lista de itens selecionados.
- [*selection_mode*](#): é um de 'single', 'multiple' ou 'none'.
- [*allow_empty_selection*](#): é do tipo boolean. Se for False, uma seleção é forcada. Se for True, e somente o usuário ou ação de programação irá mudar a seleção, pode ser vazia.

UM [DictAdapter](#) é uma subclasse de um [ListAdapter](#). Ambos despacham o evento [*on_selection_change*](#) quando a seleção muda.

Alterado na versão 1.6.0: Adicionado data = ListProperty([]), que foi provável e inadvertidamente eliminado em algum momento. Isto significa que sempre que os dados mudarem uma atualização irá disparar, em vez de ter que redefinir o objeto de dados (Adapter tem os dados definidos como um ObjectProperty, portanto nós precisamos redefini-lo aqui para ListProperty). Consulte também DictAdapter e seu conjunto de dados = DictProperty().

```
class kivy.adapters.listadapter.ListAdapter(**kwargs)
    Bases:           kivy.adapters.adapter.Adapter,      kivy.event.
                    EventDispatcher
```

Uma classe base para adaptadores que interagem com listas, dicionários ou outros dados de tipo de coleção, adicionando seleção, criação de exibição e funcionalidade de gerenciamento.

allow_empty_selection

O allow_empty_selection pode ser usado para a seleção em cascata entre várias visualizações de lista, ou entre uma exibição de lista e uma observação. Essa manutenção automática da seleção é importante para todos, exceto para exibições de lista simples. Definindo allow_empty_selection como False, a seleção é inicializada automaticamente e sempre mantida, portanto,

as visualizações de observação também podem ser atualizadas para permanecerem sincronizadas.

`allow_empty_selection` é um *BooleanProperty* e o padrão é *True*.

cached_views

As instâncias de exibição de itens de dados são instanciadas e gerenciadas pelo adaptador. Aqui nós mantemos um dicionário contendo as instâncias de visualização digitadas nos índices nos dados.

Este dicionário funciona como um cache. `get_view()` solicita somente uma exibição do adaptador se uma já não estiver armazenada para o índice solicitado.

`cached_views` é um *DictProperty* e o padrão é `{}`.

create_view(index)

Este método é mais complicado das classes: `:class: ~kivy.adapters.adapter.Adapter` e `:class: ~kivy.adapters.simplelistadapter.SimpleListAdapter'` porque aqui criamos ligações para os itens de dados e seus filhos voltam para o evento `*self.handle_selection () *`. Também realizamos outras tarefas relacionadas à seleção para manter as visualizações do item em sincronia com os dados.

cut_to_sel(*args)

O mesmo de `trim_to_sel`, mas os itens intervenientes na lista no intervalo serão também cortados, deixam apenas na lista os itens que estejam selecionados.

data

A propriedade lista de dados é redefinida aqui, sobrepondo sua definição como um *ObjectProperty* na classe Adapter. Nós vinculamos ao dado de modo que qualquer mudança irá disparar atualizações. Consulte também como `DictAdapter` redefine os dados como uma *DictProperty*.

`data` é um *ListProperty* e o padrão é `[]`.

on_selection_change(*args)

`on_selection_change()` é o manipulador padrão para o evento `on_selection_change`. Você pode vincular a este evento para ser notificado das alterações de seleção.

Parameters

Adaptador: `ListAdapter` ou subclasseA instância do adaptador de lista onde a seleção foi alterada. Use a propriedade dos adaptadores: `attr: selection` para ver o que foi selecionado.

propagate_selection_to_data

Normalmente, os itens de dados não são selecionados/desmarcados porque os itens de dados podem não ter uma propriedade booleana `is_selected`

– somente a exibição do item para um dado item de dados é selecionada/desmarcada como parte da lista de seleção mantida. No entanto, se os itens de dados têm uma propriedade `is_selected`, ou se eles se misturam em `SelectableDataItem`, o mecanismo de seleção pode propagar a seleção para itens de dados. Isso pode ser útil para armazenar o estado de seleção em um banco de dados local ou banco de dados back-end para manter o estado no jogo ou outros cenários semelhantes. É uma função de conveniência.

É para propagar a seleção ou não?

Considere uma aplicação de lista de compras para compras de frutas no mercado. O aplicativo permite a seleção de frutas para comprar para cada dia da semana, apresentando sete listas: uma para cada dia da semana. Cada lista é carregada com todos os frutas disponíveis, mas a seleção para cada é um subconjunto. Há apenas um conjunto de dados de frutas compartilhado entre as listas, por isso não faria sentido propagar a seleção para os dados porque a seleção em qualquer uma das sete listas entraria em choque e se misturariam com as dos outros.

Entretanto, considere um jogo que usa os mesmos dados de frutas para selecionar as disponíveis para arremesso. Uma dada rodada de jogo poderia ter uma lista completa de frutas, com as que estiverem disponíveis para lançamento exibindo a selecionada. Se o jogo é salvo e recomeça, a lista completa de frutas, com a seleção marcada em cada item, seria recarregada corretamente se a seleção for sempre propagada para os dados. Você poderia completar a mesma funcionalidade escrevendo código para operar na seleção de lista, mas ter a seleção armazenada nos dados `ListProperty` pode ser conveniente em alguns casos.

Nota: Esta definição deve ser definida como `True` se você quiser inicializar a view com itens de views já selecionados.

`propagate_selection_to_data` é um `BooleanProperty` e o padrão é `False`.

`select_list`(*view_list*, *extend=True*)

A chamada selecionada é feita para os itens na lista providas por *view_list*.

Argumentos:

view_list: A lista de visualizações do item para se tornar a nova seleção ou para adicionar à seleção existente

extend: boolean para estender ou não a lista existente

`selection`

A propriedade lista de seleção é o recipiente para os itens selecionados.

`selection` é uma *ListProperty* e o padrão é [].

selection_limit

Quando o `selection_mode` é ‘multiple’ e o `selection_limit` não é negativo, este número limitará o número de itens selecionados. Ele pode ser definido como 1, o que equivale a uma única seleção. Se `selection_limit` não estiver definido, o valor padrão é -1, o que significa que nenhum limite será aplicado.

`selection_limit` é uma *NumericProperty* e o padrão é -1 (sem limites).

selection_mode

O `selection_mode` é uma String e pode ser definida como um dos seguintes valores:

- ‘none’: usa a lista como uma lista simples (nenhuma ação de seleção). Esta opção está aqui para que a seleção possa ser desligada, momentânea ou permanentemente, para um adaptador de lista existente. A *ListAdapter* não se destina a ser usado como um adaptador de lista principal sem seleção. Use a *SimpleListAdapter* para isso.
- ‘single’: multi-touch/clique ignorado. Apenas seleção de item único.
- ‘multiple’: multi-touch / adição incremental à seleção permitida; pode ser limitado a uma contagem definindo o `selection_limit`.

`selection_mode` é um *OptionProperty* e o padrão é ‘single’.

trim_left_of_sel(*args)

Corta itens da lista com índices em `sorted_keys` que são menores do que o índice do primeiro item selecionado se houver uma seleção.

trim_right_of_sel(*args)

Corta itens da lista com índices em `sorted_keys` que são maiores do que o índice do último item selecionado se houver uma seleção.

trim_to_sel(*args)

Corta itens da lista com índices em `sorted_keys` que são menores ou maiores do que o índice do último item selecionado se houver uma seleção. Isso preserva os itens da lista intermediária dentro da faixa selecionada.

4.2.7 SelectableDataItem

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

Data Models

O Kivy está aberto a novos tipo de dados utilizados na construção de aplicações. No entanto, às vezes são necessárias classes básicas para garantir a conformidade dos dados com os requisitos de algumas partes do sistema.

Uma [SelectableDataItem](#) é modelo de classe básico do Python que pode ser mixado na construção de objetos de dados que são compatíveis com [Adapter](#) do Kivy e selecionam o sistema e que trabalhem com Views como as [ListView](#). A propriedade booleana *is_selected* é requerida.

A acção por omissão do sistema de selecção é a de não propagar a selecção em vistas como a ListView, até aos dados subjacentes: a selecção é por omissão, uma operação de uma só vista. No entanto, em alguns casos, é útil propagar a selecção até aos dados subjacentes.

Você pode, é claro, construir seu próprio sistema de modelos de dados com o backend em Python para um aplicativo Kivy. Por exemplo, você pode utilizar o sistema do [Google App Engine Data Modeling](#) com Kivy, no caso, você definiria suas classes como a seguir:

```
from google.appengine.ext import db

class MySelectableDataItem(db.Model):
    # ... other properties
    is_selected = db.BooleanProperty()
```

É fácil construir essa classe utilizando somente a linguagem Python.

`class kivy.adapters.models.SelectableDataItem(*args, **kwargs)`
Bases: object

Uma classe Mixin contendo os requisitos para operações de seleção.

is_selected

Uma propriedade booleana indicando se o item de dados está selecionado ou não.

4.2.8 SimpleListAdapter

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: Este código ainda é experimental e essa API está sujeita a mudanças em versões futuras.

A classe: `~kivy.adapters.simplelistadapter.SimpleListAdapter` é utilizada em listas básicas. Por exemplo, pode ser utilizado para exibir um lista com strings de apenas leitura que não requerem uma interação do usuário.

`class kivy.adapters.simplelistadapter.SimpleListAdapter(**kwargs)`
Bases: `kivy.adapters.adapter.Adapter`

A `SimpleListAdapter` é um adaptador em torno de uma lista Python.

De `Adapter`, the `ListAdapter` obtém as propriedades `cls`, `template` e `args_converter`.

data

A propriedade `lista` de dados contém uma lista de objetos (que podem ser strings) que serão usados diretamente se nenhuma função `args_converter` for fornecida. Se houver um `args_converter`, os objetos de dados serão passados para ele para instanciar as instâncias de classe do item view.

`data` é um `ListProperty` e o padrão é `[]`.

4.3 Núcleo de Abstração

Este módulo define a camada de abstração para os nossos núcleos de provedores e suas implementações. Para maiores informações, por favor, veja [Visão geral da Arquitetura](#) e a seção [Provedores Principais e Provedores de Entrada](#) da documentação.

Na maioria das vezes, não deverias utilizar diretamente uma biblioteca que já está coberta/implementada pela abstração principal. Tente sempre utilizar nossos provedores primeiro. Caso falte um recurso ou método, informe-nos abrindo um novo Bug Report (relatório de erro) ao invés de depender de sua biblioteca.

Aviso: Estas classes **não são** Widgets! Estas implementações são abstrações das respectivas funcionalidades. Por exemplo, você não pode adicionar um Núcleo de Imagem à sua janela. Ao invés, utilize a classe do Widget Image. Caso estejas procurando os Widgets, por favor, consulte `kivy.uix`.

4.3.1 Áudio

Abra um áudio som/música toque o mesmo com:

```
from kivy.core.audio import SoundLoader

sound = SoundLoader.load('mytest.wav')
if sound:
    print("Sound found at %s" % sound.source)
    print("Sound is %.3f seconds" % sound.length)
    sound.play()
```

Você não deve usar diretamente a classe Sound. A classe retornada por `SoundLoader.load()` irá fornecer o melhor som para o tipo de arquivo em particular, então ele deve retornar diferentes classe de Sound dependendo do tipo de arquivo.

Despacho de eventos e alterações de estado

O áudio é muitas vezes processado em paralelo ao seu código. Isso significa que geralmente precisarás entrar em Kivy `eventloop` para permitir que eventos e alterações de estado sejam despachados corretamente.

Você raramente precisa se preocupar com isso, pois os aplicativos Kivy sempre exigem que esse loop de eventos para a GUI permaneça responsivo, mas é bom manter isso em mente ao depurar ou executar em um `REPL` (ciclo Read-eval-print).

Alterado na versão 1.10.0: The pygst and gi providers have been removed.

Alterado na versão 1.8.0: Existe neste momento 2 implementações diferentes utilizando o Gstreamer: a primeira utilizando Gl/Gst que funciona no Python 2+3 com Gstreamer 1.0, e outra utilizando o PyGST que funciona somente para a versão 2 + Gstreamer 0.10.

Nota: A biblioteca de áudio principal não suporta a gravação de áudio. Se você precisar dessa funcionalidade, consulte a extensão `audiostream`

class kivy.core.audio.Sound
Bases: `kivy.event.EventDispatcher`

Representa um som pra ser reproduzido. Esta classe é abstrata e não pode ser usada diretamente.

Use o SoundLoader para carregar um som

Events

`on_play: None` Disparado quando o som é tocado.

on_stop: `None` Disparado quando o som é pausado.

filename

Obsoleto desde a versão 1.3.0: Ao invés, usa `source`.

get_pos()

Retorna a posição atual do arquivo de áudio. Retorna `0` se não estiver tocando.

Novo na versão 1.4.1.

length

Obtém o tamanho do som (em segundos).

load()

Abre o arquivo para a memória;

loop

Defina como `True` se o som deve repetir automaticamente quando terminar.

Novo na versão 1.8.0.

`loop` é um `BooleanProperty` e o padrão é `False`.

pitch

Pitch de um som. `2` é uma oitava maior, `0,5` uma oitava abaixo. Isto é implementado ainda apenas para o provedor de áudio SDL2.

Novo na versão 1.10.0.

`pitch` é uma `NumericProperty` e o padrão é `1`.

play()

Toca o arquivo.

seek(*position*)

Vá para a <*position*> (em segundos).

Nota: Most sound providers cannot seek when the audio is stopped. Play then seek.

source

Nome do arquivo/source do seu arquivo de áudio.

Novo na versão 1.3.0.

`source` é um `StringProperty` e o padrão é `None` e é somente leitura. Utilize o `SoundLoader.load()` para abrir arquivos de áudio.

state

Estado do som, uma das seguintes opções, 'stop' ou 'play'.

Novo na versão 1.3.0.

state é um *OptionProperty* e é somente leitura.

status

Obsoleto desde a versão 1.3.0: Ao invés, use **state**.

stop()

Para o reprodução.

unload()

Remove o arquivo da memória.

volume

Volume, intervalo entre 0-1. 1 significa o volume máximo, 0 significa o volume mínimo.

Novo na versão 1.3.0.

volume é uma *NumericProperty* e o padrão é 1.

class kivy.core.audio.SoundLoader

Carregar um som, usando o melhor carregador para o tipo de arquivo determinado.

static load(filename)

Carrega um som, e retorna uma instância de *Sound()*.

static register(classobj)

Registra uma nova classe para abrir o som.

4.3.2 Câmera

Classe (básica) principal para adquirir a câmera e converter sua entrada em um *Texture*.

Alterado na versão 1.10.0: The pygst and videocapture providers have been removed.

Alterado na versão 1.8.0: Existe neste momento 2 implementações diferentes utilizando o Gstreamer: a primeira utilizando Gl/Gst que funciona no Python 2+3 com Gstreamer 1.0, e outra utilizando o PyGST que funciona somente para a versão 2 + Gstreamer 0.10.

class kivy.core.camera.CameraBase(kwargs)**

Bases: *kivy.event.EventDispatcher*

Dispositivo Câmera de classe abstrata.

As classes de câmera concretas devem implementar a inicialização e a captura de Frames em um Buffer que pode ser carregado para o GPU.

Parameters

index: int Índice de origem da câmera.

size: tuple (int, int) Tamanho no qual a imagem é desenhada. Se nenhum tamanho for especificado, o padrão será a resolução da imagem da câmera.

resolution: tuple (int, int) Resolução para tentar solicitar da câmera. Usado no gstreamer pipeline forçando as tampas do appsink para esta resolução. Se a câmera não suportar a resolução, um erro de negociação pode ser lançado.

Events

on_load Disparado quando a câmera é carregada e a textura ficou disponível.

on_texture Disparado cada vez que a textura da câmera é atualizada.

index

Índice de origem da câmera

init_camera()

Inicializar a câmera (interna)

resolution

Resolução da captura da câmera (largura, altura)

start()

Inicia a aquisição da câmera

stop()

Solta a câmera

texture

Retorna a textura da câmera com a captura mais recente

4.3.3 Área de Transferência

Classe principal para o acesso à Área de Transferência (Clipboard). Caso não seja possível acessar a área de transferência do sistema, será criado e utilizado uma área de transferência falsa.

Exemplo de uso:

```
#:import Clipboard kivy.core.clipboard.Clipboard

Button:
    on_release:
        self.text = Clipboard.paste()
        Clipboard.copy('Data')
```

4.3.4 OpenGL

Selecione e utilize a melhor biblioteca OpenGL disponível. Dependendo do seu sistema operacional, o provedor de núcleo poderá escolher um OpenGL ES ou a biblioteca OpenGL ‘clássica’ para desktop.

4.3.5 Imagem

Classes principais para carregamento de imagens e conversão delas para uma **Texture**. Os dados da imagem original podem ser mantidos em memória por longos acessos.

Carregamento de imagem pra memória

Novo na versão 1.9.0: Suporte oficial para carregamento In-memory. não é compatível com todos os provedores, mas atualmente tem compatibilidade com SDL2, pygame, pil and imageio work

Para carregar uma imagem com um nome de um arquivo, você pode fazer isto, normalmente:

```
from kivy.core.image import Image as CoreImage
im = CoreImage("image.png")
```

Você também pode carregar os dados da imagem diretamente de um bloco da memória. Em vez de passar o nome do arquivo, você precisará passar os dados como um objeto BytesIO jnstantemente com um parêmtro “ext”. Ambos são indispensáveis.

```
import io
from kivy.core.image import Image as CoreImage
data = io.BytesIO(open("image.png", "rb").read())
im = CoreImage(data, ext="png")
```

Por padrão, a imagem não ficará armazenada em cache, pois o cache interno exige um nome de arquivo. Se você quiser armazenar em cache, adicione o nome correspondente ao seu arquivo (ele será usado somente para cache)

```
import io
from kivy.core.image import Image as CoreImage
data = io.BytesIO(open("image.png", "rb").read())
im = CoreImage(data, ext="png", filename="image.png")
```

```
class kivy.core.image.Image(arg, **kwargs)
    Bases: kivy.event.EventDispatcher
```

Carrega uma imagem e armazena o tamanho e a textura.

Alterado na versão 1.0.7: Atributo “mipmap” foi adicionado. As propriedades: “texture_mipmap” e “texture_rectangle” foram deletadas.

Alterado na versão 1.0.8: Um widget de imagem pode ter sua textura mudada. Um novo evento “on_texture” foi invocado. Novos métodos de manipulação de animações em sequencia estão sendo adicionados.

Parameters

‘arg’: pode ser uma string (str), Textura, BytesIO ou objeto Imagem.

Uma string com o caminho (path) do arquivo de imagem ou a URL que os dados serão carregados; ou o objeto Textura que será aplicada à imagem; ou o objeto BytesIO contendo a imagem original; ou uma imagem já existente, em cada caso, uma cópia da imagem usada será retornada.

keep_data: bool, e o padrão é False Mantém os dados da imagem quando a textura é criada.

scale: float, o padrão é 1.0 Escala da imagem.

mipmap: bool, padrão é False Cria um mipmap para a textura.

anim_delay: float, o padrão é .25 Atraso em segundos entre cada quadro da animação. Valores menores implicam em uma animação mais rápida.

ext: str, somente com BytesIO arg Extensão de arquivo a ser usada na determinação de como carregar dados de imagem bruta.

filename: str, somente com BytesIO arg Nome do arquivo a ser usado no cache de imagem para dados de imagem bruta.

anim_available

Retorna True se esta instância de imagem tiver uma animação disponível.

Novo na versão 1.0.8.

anim_delay

Delay entre cada Frame da animação. Um valor menor significa animação mais rápida.

Novo na versão 1.0.8.

anim_index

Retorna o número de índice da imagem atualmente na textura.

Novo na versão 1.0.8.

anim_reset (allow_anim)

Redefine uma animação, se disponível.

Novo na versão 1.0.8.

Parameters

allow_anim: boolIndica se a animação deve ser, ou não, reiniciada.

Uso:

```
# start/reset animation  
image.anim_reset(True)  
  
# or stop the animation  
image.anim_reset(False)
```

Você pode alterar velocidade da animação durante a execução:

```
# Set to 20 FPS  
image.anim_delay = 1 / 20.
```

filename

Obtém/define o nome do arquivo da imagem

height

Tamanho da imagem

image

Obtém/define os dados do objeto imagem

static load(filename, **kwargs)

Abre uma imagem

Parameters

filename: strNome do arquivo da imagem.

keep_data: bool, e o padrão é FalseMantém os dados da imagem quando a textura é criada.

load_memory(data, ext, filename='__inline__')

(interno) método para abrir dados brutos de uma imagem.

nocache

Indica se a textura será armazenada ou não armazenada em cache.

Novo na versão 1.6.0.

on_texture(*args)

Este evento é disparado quando a referência de textura ou o conteúdo é alterado. Ele é normalmente usado por sequências de imagem.

Novo na versão 1.0.8.

read_pixel(x, y)

Para uma determinada posição local x/y, devolve a cor do pixel nessa posição.

Aviso: Essa função só pode ser usada com imagem carregadas com a palavra-chave `keep_data=True`. Para um exemplo veja:

```
m = Image.load('image.png', keep_data=True)
color = m.read_pixel(150, 150)
```

Parameters

`x: int` Coordenada local X do pixel em questão.

`y: int` Local da coordenada no eixo Y do pixel em questão.

`remove_from_cache()`

Remove a imagem do cache. Isso facilita novos carregamentos de imagens do disco, caso o conteúdo da imagem seja modificado.

Novo na versão 1.3.0.

Uso:

```
im = CoreImage('1.jpg')
# -- do something --
im.remove_from_cache()
im = CoreImage('1.jpg')
# this time image will be re-loaded from disk
```

`save(filename, flipped=False)`

Salva a textura da imagem no arquivo.

O nome de arquivo deve ter a extensão ".png" devido ao fato de os dados da textura lidos da placa de vídeo (GPU) seguirem o padrão RGBA. ".jpg" pode funcionar, mas ainda não foi testado de forma consistente, portanto, alguns provedores podem enfrentar problemas no uso de arquivos ".jpg". Quaisquer outras extensões não são oficialmente suportadas

O parâmetro "flipped" gira verticalmente a imagem salva, e tem por 'False' por padrão

Exemplo:

```
# Save an core image object
from kivy.core.image import Image
img = Image('hello.png')
img.save('hello2.png')

# Save a texture
texture = Texture.create(. . .)
img = Image(texture)
img.save('hello3.png')
```

Novo na versão 1.7.0.

Alterado na versão 1.8.0: Parâmetro ‘flipped’ adicionado para girar a imagem antes de salvá-la. Tem ‘False’ por padrão.

size

Tamanho da imagem (largura, altura)

texture

Textura da Imagem

width

Largura da Imagem

```
class kivy.core.image.ImageData(width, height, fmt, data, source=None,
                                flip_vertical=True, source_image=None,
                                rowlength=0)
```

Bases: `object`

Contêiner para imagens e imagens mipmap. O contêiner terá sempre, no mínimo o nível de mipmap 0.

add_mipmap (level, width, height, data, rowlength)

Adiciona uma imagem para um nível mipmap específico.

Novo na versão 1.0.7.

data

Dados da imagem. (Se a imagem é mipmapped, usará o nível 0)

flip_vertical

Indica se a textura deverá ser girada verticalmente.

fmt

Formato da imagem decodificado, um formato de textura disponível

get_mipmap (level)

Obtenha a imagem mipmap num nível específico se ela existir

Novo na versão 1.0.7.

height

Altura da imagem em pixels (se for uma imagem mipmap, será usado o nível 0).

iterate_mipmaps ()

Itera sobre todas as imagens mipmap disponíveis.

Novo na versão 1.0.7.

mipmaps

Dados para cada mipmap.

rowlength

Comprimento da linha da imagem (se for uma imagem mipmap, será usado

o nível 0).

Novo na versão 1.9.0.

size

Imagen (largura, altura) em pixels. (Se a imagem for mipmapped, usará o nível 0)

source

Origem da imagem, se disponível

width

Largura da imagem em pixels. (Se a imagem for mipmapped, usará o nível 0)

4.3.6 Spelling

Fornece acesso abstruído a uma série de backends ortográficos, bem como sugestões de palavras. A API é inspirada no pacote *Enchant*, mas outros backends que implementam a mesma API podem ser adicionados.

A ortografia atualmente requer o pacote *python-enchant* em todas as plataformas, exceto o OSX, onde existe uma implementação nativa.

```
>>> from kivy.core.spelling import Spelling
>>> s = Spelling()
>>> s.list_languages()
['en', 'en_CA', 'en_GB', 'en_US']
>>> s.select_language('en_US')
>>> s.suggest('heло')
[u'hole', u'help', u'helot', u'hello', u'halo', u'hero', u'hell', u
↳'held',
u'helm', u'he-lo']
```

class kivy.core.spelling.SpellingBase(*language=None*)

Bases: object

Classe base para todos provedores ortográficos. Suporta alguns métodos abstratos de verificação de palavras e obtenção de sugestões.

check(*word*)

Se *word* é uma palavra válida em *self._language* (a linguagem atualmente ativa), retorna *True*.; Se a palavra não deve ser verificada, retorna *None* (por exemplo para ""). Se não for uma palavra válida *self._language*, retorna *False*.

Parameters

word: strA palavra pra verificar.

list_languages()

Retorna uma lista de todas as linguagens suportadas. Por exemplo: ['en',

‘en_GB’, ‘en_US’, ‘de’, ...]

select_language(*language*)

A partir do conjunto de idiomas registados, selecione o primeiro idioma para *language*.

Parameters

language: strIdentificador de Linguagem. Precisa ser uma das opções retornadas por *list_languages()*. Define a linguagem usada para verificação ortográfica e para as sugestões de palavras.

suggest(*fragment*)

Para um dado *fragment* (ou seja, parte de uma palavra ou uma palavra propriamente dita), forneça correções (*fragment* pode estar mal escrito) ou complementos como uma lista de strings.

Parameters

fragment: strA palavra *fragment* obtém sugestões/correções por exemplo, ‘foo’ pode tornar-se ‘of’, ‘food’ ou ‘foot’.

class kivy.core.spelling.NoSuchLangError

Bases: exceptions.Exception

Exceção para ser levantada quando uma palavra específica não puder ser encontrada.

class kivy.core.spelling.NoLanguageSelectedError

Bases: exceptions.Exception

Exceção a ser gerada quando um método de uso da linguagem é chamado, mas nenhum idioma foi selecionado antes da chamada.

4.3.7 Texto

Um abstração da criação de texto. Dependendo do backend selecionado, a precisão da renderização de texto pode variar.

Alterado na versão 1.5.0: *LabelBase.line_height* adicionado.

Alterado na versão 1.0.7: A *LabelBase* não gera nenhuma textura se a largura (width) do texto for ≤ 1 .

Esta é a camada de backend para pegar o texto de diferentes provedores, só utilize isso diretamente quando as suas necessidades não forem atendidas pela classe *Label*.

Exemplo de uso:

```
from kivy.core.text import Label as CoreLabel
```

```
...
```

```

...
my_label = CoreLabel()
my_label.text = 'hello'
# the label is usually not drawn until needed, so force it to draw.
my_label.refresh()
# Now access the texture of the label and use it wherever and
# however you may please.
hello_texture = my_label.texture

```

```

class kivy.core.text.LabelBase(text='', font_size=12, font_name=None,
                               bold=False, italic=False, underline=False,
                               strikethrough=False, halign='left', valign='bottom',
                               shorten=False, text_size=None, mipmap=False,
                               color=None, line_height=1.0, strip=False,
                               strip_reflow=True, shorten_from='center',
                               split_str=' ', unicode_errors='replace',
                               font_hinting='normal', font_kerning=True,
                               font_blended=True, outline_width=None,
                               outline_color=None, **kwargs)

```

Bases: `object`

Núcleo do texto do Label. Esta é a classe abstrata usada por backends diferentes para renderizar texto.

Aviso: O núcleo de texto do Label não pode ser alterada em tempo de execução. Você deve recria-lo.

Parameters

font_size: int, o padrão é 12Tamanho da fonte do texto

font_name: str, padrão é DEFAULT_FONTNome da fonte do texto

bold: bool, padrão é FalseAtiva o estilo de texto “negrito” (bold)

italic: bool, o padrão é FalseAtiva o estilo de texto “itálico” (italic)

text_size: tupla, padrão é (None, None)Adiciona restrições para renderizar o texto (dentro de uma caixa delimitadora). Se nenhum tamanho é dado, o tamanho do Label será definido pelo tamanho do texto.

padding: float, padrão é NoneCaso isso seja do tipo float, será definido padding_x e padding_y

padding_x: float, padrão é 0.0Padding dos lados Left/right

padding_y: float, padrão é 0.0Padding do Top/bottom

halign: str, padrão é “left” Alinhamento horizontal do texto dentro dos limites definidos

valign: str, padrão é “bottom” Alinhamento vertical do texto dentro dos limites definidos

shorten: bool, o padrão é False Indique se o Label deve tentar reduzir seu conteúdo textual tanto quanto possível se um *size* é fornecido. Definir isso como *True* sem um tamanho apropriadamente levará a resultados inesperados.

shorten_from: str, padrão é center O lado do qual deveríamos encurtar o texto pode ser esquerdo, direito ou central. Por exemplo, se for o esquerdo, a elipse aparecerá para o lado esquerdo e ele irá exibir tanto texto a partir da direita como possível.

split_str: string, padrão é ‘ ’ (espaço) A sequência de caracteres para usar para dividir as palavras quando encurtar. Se estiver vazio, nós podemos dividir depois que cada caracter preencher a linha, tanto quanto possível.

max_lines: int, padrão é 0 (sem limites) Caso definido, esta propriedade definirá a linha máxima permitida na renderização do texto. Funciona somente se houver uma limitação definida em *text_size*.

mipmap: bool, padrão é False Cria uma textura para mipmap

strip: bool, padrão é False Se cada linha do texto tem seus espaços à esquerda e à direita despojado. Se ‘halign’ for ‘justify’ é implicitamente *True*.

strip_reflow: bool, padrão é True Se tiver sido transferido para uma segunda linha, ele deve ser removido, mesmo que *strip* seja *False*. Isso só terá efeito quando *size_hint_x* não for *None*, porque de outra forma as linhas nunca são divididas.

unicode_errors: str, padrão é ‘replace’ Como lidar com erros de decodificação Unicode. Pode ser ‘strict’, ‘replace’ ou ‘ignore’.

outline_width: int, padrão é None Largura em pixels do contorno.

outline_color: tupla, padrão é (0, 0, 0) Cor do contorno.

Alterado na versão 1.10.0: *outline_width* e *outline_color* foram adicionadas.

Alterado na versão 1.9.0: *strip*, *strip_reflow*, *shorten_from*, *split_str*, e *unicode_errors* foram adicionadas.

Alterado na versão 1.9.0: *padding_x* e *padding_y* foram corrigidas para funcionar como o esperado. No passado, o padding do texto era estabelecido utilizando valores negativos.

Alterado na versão 1.8.0: *max_lines* parâmetro foi adicionado.

Alterado na versão 1.0.8: *size* entrou em desuso e foi substituído por *text_size*.

Alterado na versão 1.0.7: O *valign* é agora respeitado. Este não era o caso anteriormente para que você possa ter um problema em seu aplicativo se você não considerou isso.

content_height

Retorna a altura (height) do contexto; Ou seja, a altura (height) do texto sem qualquer preenchimento.

content_size

Retorno o tamanho do contexto (width, height)

content_width

Retorna a largura (width) do contexto; Ou seja, a largura do texto sem o valor do padding.

fontid

Retornar um valor único (id) para todos os parâmetros de fonte

get_cached_extents()

Retorna a versão do cache da função *get_extents()*.

```
>>> func = self._get_cached_extents()
>>> func
<built-in method size of pygame.font.Font object at
    0x01E45650>
>>> func('a line')
(36, 18)
```

Aviso: Este método retorna uma função de medição de tamanho que é válida para as configurações de fonte usadas no momento que *get_cached_extents()* foi chamada. Alteração nas configurações de fonte tornará a função retornada incorreta. Só altere isso se souberes o que estás fazendo.

Novo na versão 1.9.0.

get_extents(*text*)

Retorna uma tupla (width, height) indicando o tamanho do texto especificado.

static get_system_fonts_dir()

Retorna os diretórios utilizados pelo sistema de fontes.

label

Get/Set o texto

refresh()

Força o re-renderização do texto

static register(name, fn_regular, fn_italic=None, fn_bold=None, fn_bolditalic=None)

Registra um apelido (alias) para a Font.

Novo na versão 1.1.0.

Se utilizares um diretório ttf, talvez não seja possível definir as propriedades bold/italic para as versões do ttf. Caso a fonte seja distribuída em vários arquivos (um normal, um itálico e um negrito), então, registre estes arquivos e utilize o apelido (alias).

Todos os parâmetros fn_regular/fn_italic/fn_bold são resolvidos com [kivy.resources.resource_find\(\)](#). Se fn_italic/fn_bold for None, fn_regular será utilizado.

render(real=False)

Retorna uma tupla (width, height) para criar a imagem com as restrições do usuário. O (width, height) inclui o valor do padding.

shorten(text, margin=2)

Reduz o texto para caber em uma única linha pela largura especificada por: attr: *text_size* [0]. Se: attr: *text_size* [0] for None, ele retorna texto sem alteração.

`split_text` e `shorten_from` determina como o texto é encurtado.

Params *text* str, o texto a ser encurtado. *margin* int, a quantidade de espaço a sair entre as margens e o texto. Isso é além de: attr: *padding_x*.

Retorna O texto encurtado para caber em uma única linha.

text

Get/Set o texto

text_size

Obtém/define a (largura, altura) da caixa de renderização restrita

usersize

(Em desuso) utilize ao invés *text_size*.

Text Layout

Um módulo interno para definir o layout de acordo com as opções e restrições. Isso não é parte da API e pode ser modificada a qualquer momento.

kivy.core.text.text_layout.layout_text()

Mostra o texto dentro de uma série de instâncias [LayoutWord](#) e [LayoutLine](#) de acordo com as opções definidas.

A função é desenhada para ser chamada várias vezes, cada vez um novo texto é adicionado a última linha (ou a primeira linha, caso seja anexada para cima), a menos que uma nova linha esteja presente no texto. Cada texto adicionado é descrito por suas próprias opções que podem ser alteradas em chamadas sucessivas. Se o texto estiver restrito, paramos assim que a restrição é alcançada.

Parameters

***text*: string or bytes**o texto a ser dividido em linhas. Se a linha não estiver vazia, o texto é adicionado à última linha (ou a primeira linha se *append_down* for *False*) até uma nova linha ser alcançada, o que cria uma nova linha em *lines*. Veja [LayoutLine](#).

***lines*: list**um lista de instâncias [LayoutLine](#), cada uma descrevendo uma linha do texto. Chamadas para [layout_text\(\)](#) adiciona ou criar uma nova instância de [LayoutLine](#) em *lines*.

***size*: 2-tuple of int**o tamanho do texto apresentado até agora. Na primeira chamada provavelmente deve ser (0, 0), depois deve ser o (w, h) retornado por esta função numa chamada anterior. Quando o tamanho atinge o tamanho restritivo, *text_size*, paramos de adicionar linhas e retornamos *True* para o parâmetro cortado. Tamanho inclui o padding *x* e *y*.

***text_size*: 2-tuple of ints or None**. A restrição de tamanho no texto apresentado. Se qualquer elemento é *None*, o texto não é restrinido nessa dimensão. Por exemplo, (*None*, 200) irá restringir a altura, incluindo preenchimento para 200, enquanto a largura é livre. A primeira linha e o primeiro caractere de uma linha é sempre retornado, mesmo se exceder a restrição. O valor deve ser alterado entre diferentes chamadas.

***options*: dict**as opções *Label* deste 'texto'. As opções são salvas com cada palavra permitindo que palavras diferentes tenham opções diferentes de chamadas sucessivas.

Note que *options* deve incluir uma chave *space_width* com um valor indicando a largura de um espaço para esse conjunto de opções.

***get_extents*: callable**um função chamada com um *String*, que retorna uma tupla contendo a largura, altura de um *String*.

***append_down*: bool**Durante sucessivas chamadas para a função acrescenta linhas antes ou depois das linhas existentes. Se *True*, eles são acrescentados à última linha e abaixo dele. Se *False*, é acrescentado na primeira linha e acima. Por exemplo, se *False*, tudo após a última nova linha em *text* é acrescentado à primeira linha nas linhas. Tudo antes da última nova linha é inserido no início das linhas na mesma ordem que o texto; Ou seja,

não invertemos a ordem de linha.

Isso permite estabelecer de cima para baixo até que o restrin-gido seja alcançado, ou de baixo para cima até que o restrin-gido seja alcançado.

complete: bool Se este texto completa linhas. Ele usa o que normalmente é strip-faixa em *options* é True, assim, todos os espaços à esquerda e à direita são removidos de cada linha, exceto a partir da última linha (ou a primeira linha se 'append_down' é False) que só remove espaços à esquerda. Isso é porque mais texto ainda pode ser acrescentado à última linha e então não nós não podemos removê-los. Se *complete* for True, indica que não há mais texto e todas as linhas serão removidas.

A função pode também ser chamada com *text* definido como uma String vazia e *complete* definido como sendo True em or-dem para o última (primeiro) linha seja removida.

Retorna Tupla de 3, (w, h, cortada). W e h é a largura e altura do texto em linhas até agora e inclui preenchimento. Isto pode ser maior que *text_size*, exemplo, se nem mesmo um único fixado, a primeira linha ainda seria devolvida. *Clipped* é True se nem todo o texto tiver sido adicionado às linhas porque w, h atingiu o tamanho li-mitado.

A seguir um simples exemplo sem paddind e sem stripping:

```
>>> from kivy.core.text import Label
>>> from kivy.core.text.text_layout import layout_text

>>> l = Label()
>>> lines = []
>>> # layout text with width constraint by 50, but no height_
>>> constraint
>>> w, h, clipped = layout_text('heres some text\nnah, another_
>>> line',
... lines, (0, 0), (50, None), l.options, l.get_cached_
>>> extents(), True,
... False)
>>> w, h, clipped
(46, 90, False)
# now add text from bottom up, and constrain width only be 100
>>> w, h, clipped = layout_text('\nyay, more text\n', lines, (w,
>>> h),
... (100, None), l.options, l.get_cached_extents(), False, True)
>>> w, h, clipped
(77, 120, 0)
>>> for line in lines:
...     print('line w: {}, line h: {}'.format(line.w, line.h))
```

```

...
    for word in line.words:
...
        print('w: {}, h: {}, text: {}'.format(word.lw, word.
→lh,
...
        [word.text]))
line w: 0, line h: 15
line w: 77, line h: 15
w: 77, h: 15, text: ['yay, more text']
line w: 31, line h: 15
w: 31, h: 15, text: ['heres']
line w: 34, line h: 15
w: 34, h: 15, text: [' some']
line w: 24, line h: 15
w: 24, h: 15, text: [' text']
line w: 17, line h: 15
w: 17, h: 15, text: ['ah,']
line w: 46, line h: 15
w: 46, h: 15, text: [' another']
line w: 23, line h: 15
w: 23, h: 15, text: [' line']

```

class kivy.core.text.text_layout.LayoutWord

Bases: object

Formalmente descreve uma palavra contida em uma linha. A palavra de nome significa simplesmente um pedaço de texto e pode ser usado para descrever qualquer texto.

Um para tem alguma largura, altura e é renderizada de acordo com as opções salvas em [options](#). Veja [LayoutLine](#) para maiores informações.

Parameters

options: dicto dicionário de opções do label para essa palavra.

lw: inta largura de um texto em pixels.

lh: inta altura de um texto em pixels.

text: stringo texto da palavra.

class kivy.core.text.text_layout.LayoutLine

Bases: object

Formalmente descreve uma linha de texto. Uma linha de texto é composta por muitas instâncias de [LayoutWord](#), cada um com o seu próprio texto, tamanho e opções.

A: class: *LayoutLine* instância nem sempre implica que as palavras contidas na linha terminou com uma nova linha. Esse é apenas o caso se: attr: *is_last_line* é True. Por exemplo, uma única linha real de texto pode ser dividida em várias instâncias de classe: *LayoutLine* se a linha inteira não se encaixa na largura restrita.

Parameters

- x:** int local na textura de onde o lado esquerdo dessas linha inicia o desenho.
- y:** int a localização na textura de onde o fundo da linha é desenhado.
- w:** int a largura da linha. Isso é a soma da largura individual das instâncias de **LayoutWord**. Não inclui nenhum padding.
- h:** int altura da linha. Este é o máximo das alturas individuais de suas instâncias **LayoutWord** multiplicadas pelo *line_height* dessas instâncias. Então isso é maior que a altura da palavra.
- is_last_line:** bool Se esta linha foi a última linha de um parágrafo. Quando True, isso implica que a linha foi seguida por uma nova linha. As linhas novas não devem ser incluídas no texto das palavras, mas estão implícitas ao definir isso como True.
- line_wrap:** bool se essa linha é continuada de uma linha anterior que não se encaixava em uma largura restrita e, portanto, foi dividida em várias instâncias **LayoutLine**. *Line_wrap* pode ser *True* ou *False* independentemente de *is_last_line*.
- words:** lista python Uma lista que contém apenas instâncias de **LayoutWord** descrevendo o texto da linha.

Marcação de Texto

Novo na versão 1.1.0.

É fornecido uma linguagem de marcação de texto para estilização de texto em linha (inline). A sintaxe é igual a utilizada pelo **BBCode**.

Uma TAG é definida como **[tag]**, e sempre deve ter uma TAG de fechamento correspondente **[/tag]**. Por exemplo:

```
[b]Hello [color=ff0000]world[/color][/b]
```

As TAGs a seguir estão disponíveis:

- [b][/b]** Define o texto em negrito
- [i][/i]** Define o texto em itálico
- [u][/u]** Define o texto como sublinhado
- [s][/s]** Define o texto como riscado
- [font=<str>][/font]** Altera a fonte do texto

[size=<size>] [/size] Altera a fonte do texto. <size> deve ser do tipo integer, utilizar a unidade de medida é opcional (por exemplo 16sp)

[color=#<color>] [/color] Altera a cor do texto

[ref=<str>] [/ref] Adiciona uma zona interativa. A referência + define que todo o texto dentro da caixa de referência será disponibilizada em *MarkupLabel.refs*

[anchor=<str>] Coloca uma âncora no texto. Você pode pegar a posição da sua âncora com o texto *MarkupLabel.anchors*

[sub] [/sub] Exibir o texto numa posição de subíndice em relação ao texto principal.

[sup] [/sup] Exibir o texto numa posição acima em relação ao texto principal.

Caso seja necessário “escapar” a marcação de algum texto, utilize *kivy.utils.escape_markup()*.

```
class kivy.core.text.markup.MarkupLabel(*args, **kwargs)
Bases: kivy.core.text.LabelBase
```

Texto de marcação do Label.

Veja o módulo da documentação para maiores informações.

anchors

Pega a posição de todas as [anchor=...]:

```
{ 'anchorA': (x, y), 'anchorB': (x, y), ... }
```

markup

Retorna o texto com todas marcações divididas (splitted):

```
>>> MarkupLabel('[b]Hello world[/b]').markup
>>> ([b], 'Hello world', [b])
```

refs

Obtém todos os limites da caixa em que está contido [ref=...]:

```
{ 'refA': ((x1, y1, x2, y2), (x1, y1, x2, y2)), ... }
```

shorten_post(*lines, w, h, margin=2*)

Reduz o texto a uma única linha de acordo com as opções do Label.

Esta função opera num texto que já foi estabelecido pela marcação, partes do texto podem ter tamanho e opções diferentes.

Se *text_size* [0] é *None*, as linhas são retornadas sem modificações. Do contrário, as linhas são convertidas para um único encaixe de linha dentro da largura restrita, *text_size* [0].

Params*lines*: lista de instâncias ‘LayoutLine’ que descrevem o texto.

w: int, a largura do texto em linhas, incluindo preenchimento.

h: int, a altura do texto em linhas, incluindo preenchimento.

margin int, o espaço adicional deixado nos lados. Isso é além de: attr: *padding_x*.

Retornatupla de 3 (xw, h, lines), onde w, e h são similares à entrada e contém a largura / altura resultante do texto, incluindo preenchimento; lines, é uma lista contendo um único ‘LayoutLine’, que contém as palavras para a linha.

4.3.8 Vídeo

Core class for reading video files and managing the video *Texture*.

Alterado na versão 1.10.0: The pyglet, pygst and gi providers have been removed.

Alterado na versão 1.8.0: There are now 2 distinct Gstreamer implementations: one using Gi/Gst working for both Python 2+3 with Gstreamer 1.0, and one using PyGST working only for Python 2 + Gstreamer 0.10.

Nota: A gravação não é suportada.

class kivy.core.video.VideoBase(kwargs)**

Bases: *kivy.event.EventDispatcher*

A classe VideoBase é utilizada para implementar o reproduutor de vídeo.

Parameters

filename: strNome do arquivos de vídeo. Pode ser um arquivo ou uma URL.

eos: str, padrão para ‘pause’Ação a ser tomada quando um EOS for atingido. Pode ser um de “pause”, “stop” ou “loop”.

Alterado na versão unknown: Adicionado ‘pause’

async: bool, o padrão é TrueCarregar o vídeo de forma assíncrona (não é suportado por todos os provedores).

autoplay: bool, padrão é FalseIniciar o vídeo no início (Auto play the video on init.).

Events

on_eosAcionado quando EOS for atingido.

on_loadDisparado quando o vídeo está carregado e a textura está disponível.

on_frameAcionado quando um novo frame é escrito na textura.

duration

Retorna a duração do vídeo (em segundo)

filename

Retorna ou define o nome/uri do vídeo atual

load()

Abre o vídeo do nome do arquivo definido

pause()

Para o vídeo

Novo na versão 1.4.0.

play()

Inicia o vídeo

position

Get/set a posição do vídeo (em segundos)

seek(*percent*)

Se move sob a posição percentual

state

Obtém o status de reprodução de vídeo

stop()

Para/interrompe a reprodução do vídeo

texture

Obtém a textura do vídeo

unload()

Descarrega/fecha o arquivo atual

volume

Obtém/define o volume do vídeo (1.0 = 100%)

4.3.9 Janela

Classe principal para a criação da janela padrão do Kivy. O Kivy suporta somente uma janela por aplicação: por favor, não tente criar mais do que uma.

class kivy.core.window.Keyboard(kwargs)**
Bases: *kivy.event.EventDispatcher*

Interface de teclado que é retornada por *WindowBase.request_keyboard()*. Quando você requerer o teclado, você ganhará uma instância dessa classe. Seja qual for a entrada do teclado (teclado ou sistema virtual), você receberá eventos através desta instância.

Events

`on_key_down: keycode, text, modifiers` Disparado quando uma nova tecla é pressionada

`on_key_up: keycode` Disparado quando uma tecla é solta (up)

Aqui está um exemplo de como solicitar o teclado de acordo com a configuração atual:

```
import kivy
kivy.require('1.0.8')

from kivy.core.window import Window
from kivy.uix.widget import Widget


class MyKeyboardListener(Widget):

    def __init__(self, **kwargs):
        super(MyKeyboardListener, self).__init__(**kwargs)
        self._keyboard = Window.request_keyboard(
            self._keyboard_closed, self, 'text')
        if self._keyboard.widget:
            # If it exists, this widget is a VKeyboard object,
            # which you can use
            # to change the keyboard layout.
            pass
        self._keyboard.bind(on_key_down=self._on_keyboard_down)

    def _keyboard_closed(self):
        print('My keyboard have been closed!')
        self._keyboard.unbind(on_key_down=self._on_keyboard_
                               down)
        self._keyboard = None

    def _on_keyboard_down(self, keyboard, keycode, text,_
                         modifiers):
        print('The key', keycode, 'have been pressed')
        print(' - text is %r' % text)
        print(' - modifiers are %r' % modifiers)

        # Keycode is composed of an integer + a string
        # If we hit escape, release the keyboard
        if keycode[1] == 'escape':
            keyboard.release()

        # Return True to accept the key. Otherwise, it will be_
        # used by
        # the system.
```

```

    return True

if __name__ == '__main__':
    from kivy.base import runTouchApp
    runTouchApp(MyKeyboardListener())

```

callback = None

Callback que será chamado quando o teclado for liberado

keycode_to_string(value)

Converte um número keycode para String de acordo com o Keyboard.keycodes. Se o valor não for encontrado no keycodes, será retornado ''.

release()

Chame este método para liberar o teclado atual. Isso assegurará que o teclado não está mais vinculado a sua função de callback.

string_to_keycode(value)

Converte uma String para um número keycode de acordo com Keyboard.keycodes. Se o valor não for encontrado no keycodes, será retornado -1.

target = None

Alvo que solicitou o teclado

widget = None

Widget VKeyboard, se permitido pela configuração

window = None

Janela que o teclado também está anexado

class kivy.core.window.WindowBase(kwargs)**

Bases: *kivy.event.EventDispatcher*

WindowBase é um Widget de janela abstrata para qualquer implementação de janela.

Parameters

borderless: str, um de ('0', '1') Define o estado da borda da janela.

Verifica a documentação de *config* para obter uma explicação mais detalhada sobre os valores.

fullscreen: str, um de ('0', '1', 'auto', 'fake') Faz a janela fullscreen.

Verifique a documentação *config* para obter uma explicação mais detalhada sobre os valores.

width: int Largura da janela.

height: int Altura da janela.

minimum_width: int Largura mínima da janela (somente funciona para o provedor de janela sdl2).

minimum_height: intAltura mínima da janela (somente funciona para o provedor de janela sdl2).

allow_screensaver: boolAllow the device to show a screen saver, or to go to sleep on mobile devices. Defaults to True. Only works for sdl2 window provider.

Events

on_motion: etype, motioneventDisparado quando uma nova *MotionEvent* é despachada

on_touch_down:Chamado quando um novo evento de toque é instantaneamente iniciado.

on_touch_move:Disparado quando um evento de toque existente muda de local.

on_touch_up:Disparado quando um evento de toque existente é encerrado.

on_draw:Disparado quando a *Window* está sendo desenhada.

on_flip:Disparado quando a *Window* GL surface está invertida.

on_rotate: rotaçãoDisparado quando *Window* está sendo girada.

on_close:Disparado quando a *Window* é fechada.

on_request_close:Disparado quando o event loop desejar fechar a janela, ou se a tecla de escape 'ESC' é pressionada e *exit_on_escape* for igual a *True*. Se a função vinculada a esse evento retornar *True*, a janela não será fechada. Se o evento é disparado por causa da tecla de escape do teclado, o keyword argumento *source* será despachado juntamente com o valor do *keyboard* para as funções vinculadas.

Novo na versão 1.9.0.

on_cursor_enter:Disparado quando o cursor entra na janela.

Novo na versão 1.9.1.

on_cursor_leave:Disparado quando o cursor deixa a janela.

Novo na versão 1.9.1.

on_minimize:Disparado quando a janela é minimizada.

Novo na versão 1.10.0.

on_maximize:Disparado quando a janela é maximizada

Novo na versão 1.10.0.

on_restore:Disparado quando a janela é restaurada

Novo na versão 1.10.0.

on_hide:Disparado quando a janela é escondida.

Novo na versão 1.10.0.

on_show:Disparado quando a janela é exibida.

Novo na versão 1.10.0.

on_keyboard: **key, scancode, codepoint, modifier**Disparado quando o teclado é usado para entrada.

Alterado na versão 1.3.0: O parâmetro *unicode* será descontinuado em favor do codepoint e será removido completamente em versões futuras.

on_key_down: **key, scancode, codepoint, modifier**Disparado quando uma tecla é pressionada.

Alterado na versão 1.3.0: O parâmetro *unicode* será descontinuado em favor do codepoint e será removido completamente em versões futuras.

on_key_up: **key, scancode, codepoint**Disparado quando uma tecla é liberada.

Alterado na versão 1.3.0: O parâmetro *unicode* ficou obsoleto em favor do codepoint e será removido completamente em versões futuras.

on_dropfile: **str**Disparado quando um arquivo é arrastado pra cima da aplicação.

Nota: This event doesn't work for apps with elevated permissions, because the OS API calls are filtered. Check issue [#4999](#) for pointers to workarounds.

on_memorywarning:Disparado quando a plataforma tem problema de memória (iOS / Android principalmente). Você pode ouvir e limpar/remover tudo o que puder.

Novo na versão 1.9.0.

`add_widget(widget, canvas=None)`

Adiciona um Widget para a janela

`allow_screensaver`

Whether the screen saver is enabled, or on mobile devices whether the device is allowed to go to sleep while the app is open.

Novo na versão 1.10.0.

allow_screensaver is a *BooleanProperty* and defaults to True.

borderless

Quando definido igual a *True*, essa propriedade remove a borda/decoração da janela. Veja a documentação *config* para maiores detalhes e para uma explicação detalhada dos valores.

Novo na versão 1.9.0.

borderless é um *BooleanProperty* e o padrão é *False*.

center

Centro da janela girada.

Novo na versão 1.0.9.

center is an *AliasProperty*.

children

Lista dos filhos desta janela.

children é uma instância de *ListProperty* e o padrão é uma lista vazia.

Usa *add_widget()* e *remove_widget()* para manipular a lista de filhos. Não manipule a lista diretamente a menos que você saiba o que estás fazendo.

clear()

Limpala a janela com a cor de background.

clearcolor

Cor usada para limpar a janela.

```
from kivy.core.window import Window

# red background color
Window.clearcolor = (1, 0, 0, 1)

# don't clear background at all
Window.clearcolor = None
```

Alterado na versão 1.7.2: O valor de clearcolor padrão agora é: (0, 0, 0, 1)

Novo na versão 1.0.9.

clearcolor is an *AliasProperty* and defaults to (0, 0, 0, 1).

close()

Fechá a janela

create_window(*largs)

Criará a janela principal e configurará ela

Aviso: Este método é chamado automaticamente no momento de execução. Se você chama-lo, ele irá recrar o RenderContext e Canvas. Isso significa que você terá uma nova árvore gráfica, e a antiga será inutilizada.

Este método existe para permitir a criação de um novo contexto OpenGL DEPOIS do primeiro. (Como utilizar `runTouchApp()` e `stopTouchApp()`).

Este método só foi testado em um ambiente unittest e não é adequado para Aplicações.

Novamente, não utilize este método a menos que você saiba exatamente o que esteja fazendo!

dpi()

Retorna o DPI da tela. Se a implementação não suportar qualquer procura DPI, será retornará apenas 96.

Aviso: Este valor não é multiplataforma, ao invés disso, utilize `kivy.base.EventLoop.dpi`.

flip()

Alternar entre os buffers

focus

Verifica se a janela atual possui ou não foco.

Novo na versão 1.9.1.

`focus` é somente leitura *AliasProperty* e o padrão é *True*.

fullscreen

Essa propriedade define o modo fullscreen da janela. Opções disponíveis são: `True`, `False`, `'auto'` e `'fake'`. Verifique a documentação `config` para maiores detalhes e explicação sobre esses valores.

`fullscreen` é um *OptionProperty* e o padrão é `False`.

Novo na versão 1.2.0.

Nota: A opção `'fake'` entrou em desuso, ao invés disso, utilize a propriedade `borderless`.

grab_mouse()

Grab mouse - so won't leave window

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

height

Altura da janela girada.

height é somente leitura *AliasProperty*.

hide()

Esconde a janela. Este método deve ser usado somente em plataformas Desktop.

Novo na versão 1.9.0.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

icon

A path to the window icon.

Novo na versão 1.1.2.

icon is a *StringProperty*.

keyboard_anim_args = {'t': 'in_out_quart', 'd': 0.5}

O atributo para animação softkeyboard/IME. *t* = *transition*, *d* = *duration*.

Não terá efeito em Desktops.

Novo na versão 1.10.0.

keyboard_anim_args é um dict com valores 't': 'in_out_quart', 'd': .5.

keyboard_height

Retorna a altura do softkeyboard/IME em plataformas mobile. Retornará 0 caso não seja uma plataforma mobile ou se o IME não estiver ativo.

Nota: Essa propriedade retorna 0 com SDL2 no Android, mas a configuração *Window.softinput_mode* não funciona.

Novo na versão 1.9.0.

keyboard_height é somente leitura *AliasProperty* e o padrão é 0.

keyboard_padding

O padding pra ter entre o softkeyboard/IME & target ou a parte inferior da janela. Não terá efeito em Desktops.

Novo na versão 1.10.0.

keyboard_padding é uma *NumericProperty* e o padrão 0.

left

Left position of the window.

Nota: It's an SDL2 property with [0, 0] in the top-left corner.

Alterado na versão 1.10.0: *left* is now an *AliasProperty*

Novo na versão 1.9.1.

left is an *AliasProperty* and defaults to position set in *Config*.

maximize()

Maximiza a janela. Este método deve ser usado somente em plataformas Desktop.

Novo na versão 1.9.0.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

minimize()

Minimiza a janela. Este método deve ser usado somente em plataformas Desktop.

Novo na versão 1.9.0.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

minimum_height

A altura mínima para restringir a janela a.

Novo na versão 1.9.1.

minimum_height é uma *NumericProperty* e o padrão é 0.

minimum_width

A largura mínima para restringir a janela a.

Novo na versão 1.9.1.

minimum_width é uma *NumericProperty* e o padrão é 0.

modifiers

Lista dos modificadores de teclado atualmente ativos.

Novo na versão 1.0.9.

modifiers is an *AliasProperty*.

mouse_pos

Posição 2D do mouse dentro da janela.

Novo na versão 1.2.0.

mouse_pos is an *ObjectProperty* and defaults to [0, 0].

on_close(*largs)

Evento chamado quando a janela está fechada

on_cursor_enter(*largs)

Evento chamado quando o cursor entra na janela.

Novo na versão 1.9.1.

Nota: Este recurso requer o provedor de janelas SDL2.

on_cursor_leave(*largs)

Evento chamado quando o cursor deixa a janela.

Novo na versão 1.9.1.

Nota: Este recurso requer o provedor de janelas SDL2.

on_dropfile(filename)

Evento chamado quando um arquivo é arrastado pra cima do aplicativo.

Aviso: Este evento atualmente funciona com provedores de janela SDL2, no provedor de janela do PyGame e OS X com um path da versão do PyGame. Este evento é deixado neste lugar para evoluções futuras (iOS, Android e etc).

Novo na versão 1.2.0.

on_flip()

Alternar entre os buffers (evento)

on_hide(*largs)

Evento chamado quando a janela é escondida.

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

on_joy_axis(stickid, axisid, value)

Evento chamado quando um Joystick tem um Stick ou outros eixos movidos

Novo na versão 1.9.0.

on_joy_ball(*stickid*, *ballid*, *value*)

Evento chamado quando um Joystick tem uma bola movida

Novo na versão 1.9.0.

on_joy_button_down(*stickid*, *buttonid*)

Evento chamado quando um Joystick tem um botão pressionado

Novo na versão 1.9.0.

on_joy_button_up(*stickid*, *buttonid*)

Evento chamado quando um Joystick tem um botão liberado

Novo na versão 1.9.0.

on_joy_hat(*stickid*, *hatid*, *value*)

Evento chamado quando um Joystick tem um hat/dpad movido

Novo na versão 1.9.0.

on_key_down(*key*, *scancode=None*, *codepoint=None*, *modifier=None*,
 ***kwargs*)

Evento chamado quando uma tecla é pressionada (mesmos argumentos como *on_keyboard*)

on_key_up(*key*, *scancode=None*, *codepoint=None*, *modifier=None*, ***kwargs*)

Evento chamado quando uma tecla é solta (mesmos argumento como *on_keyboard*)

on_keyboard(*key*, *scancode=None*, *codepoint=None*, *modifier=None*,
 ***kwargs*)

Evento chamado quando o teclado é usado.

Aviso: Alguns provedores podem omitir *scancode*, *codepoint* e/ou *modifier*.

on_maximize(**largs*)

Evento chamado quando a janela é maximizada.

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

on_memorywarning()

Evento chamado quando a plataforma tem problema de memória. Seu objetivo é limpar o cache do seu aplicativo o máximo possível, liberar Widget não utilizado, e quaisquer outro recurso que não venha mais ser utilizado.

Atualmente, este evento é disparado somente desde um provedor SDL2, para iOS e Android.

Novo na versão 1.9.0.

on_minimize(*largs)

Evento chamado quando a janela é minimizada.

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

on_motion(etype, me)

Evento chamado quando um MotionEvent é recebido.

Parameters

etype: strUm de 'begin', 'update', 'end'

me: **MotionEvent**O Evento de Movimento atualmente despacchado.

on_mouse_down(x, y, button, modifiers)

Evento chamado quando o mouse é usado (pressionado/solto)

on_mouse_move(x, y, modifiers)

Evento chamado quando o mouse é movido com botões pressionados

on_mouse_up(x, y, button, modifiers)

Evento chamado quando o mouse é movido com botões pressionados

on_request_close(*largs, **kwargs)

Evento chamado antes de nós fecharmos a janela. Se um função associada retornar *True*, a janela não será fechada. Se um evento é disparado porque a tecla de escape do teclado, o keyword argumento *source* é despachado junto com o valor do *keyboard* para as funções associadas.

Aviso: Quando a função associada retornar *True* a janela não será fechada, então, use com cuidado porque o usuário não será capaz de fechar o programa, mesmo se o botão vermelho (botão de fechar da janela) for pressionado.

on_resize(width, height)

Evento chamado quando a janela é redimensionada.

on_restore(*largs)

Evento chamado quando a janela é restaurada.

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

on_rotate(*rotation*)

Evento chamado quando a tela tiver sido girada.

on_show(**args*)

Evento chamado quando a janela é exibida.

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

on_textinput(*text*)

Evento chamado quando texto: ou seja, alfanumérico que não sejam teclas de controle ou conjunto de teclas é inserido. Como não é garantido se recebemos um ou mais caracteres, este evento suporta o tratamento de vários caracteres.

Novo na versão 1.9.0.

on_touch_down(*touch*)

Evento chamado quando um evento de toque (pressionamento) é inicializado.

Alterado na versão 1.9.0: O toque *pos* agora é transformado em coordenadas da janela antes que este método seja chamado. Antes, a coordenada touch *pos* seria' (0, 0) 'quando esse método fosse chamado.

on_touch_move(*touch*)

Evento chamado quando um evento de toque se move (muda de local).

Alterado na versão 1.9.0: O toque *pos* agora é transformado em coordenadas da janela antes que este método seja chamado. Antes, a coordenada touch *pos* seria' (0, 0) 'quando esse método fosse chamado.

on_touch_up(*touch*)

Evento chamado quando um evento de toque é solto (terminado).

Alterado na versão 1.9.0: O toque *pos* agora é transformado em coordenadas da janela antes que este método seja chamado. Antes, a coordenada touch *pos* seria' (0, 0) 'quando esse método fosse chamado.

parent

Pai desta janela.

parent é uma instância *ObjectProperty* e o padrão é *None*. Quando criado, o pai é definido para a própria janela. Você deve cuidar dele caso estejas realizando uma verificação recursiva.

raise_window()

Levanta a janela. Este método deve ser usado somente em plataformas Desktop.

Novo na versão 1.9.1.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

release_all_keyboards()

Novo na versão 1.0.8.

Isso assegurará que o teclado virtual/sistema de tecla é requisitado. Todas as instâncias serão fechadas.

release_keyboard(*target=None*)

Novo na versão 1.0.4.

Método interno para o Widget liberar o teclado real. Veja a documentação [request_keyboard\(\)](#) para entender melhor o funcionamento.

remove_widget(*widget*)

Remove o Widget de uma janela

request_keyboard(*callback, target, input_type='text'*)

Novo na versão 1.0.4.

Método interno do Widget para solicitar o teclado. Este método raramente é requerido pelo usuário final, pois é tratado automaticamente por [TextInput](#). Nós o expomos para o caso de ser necessário manipular o teclado manualmente em cenários de entrada exclusivas.

Um Widget pode solicitar o teclado, indicando um callback que será chamado quando o teclado for liberado (ou utilizado por outro Widget).

Parameters

callback: funcCallback que será chamado quando o teclado está fechado. Isso pode ser porque alguém requisitou o teclado ou o usuário o fechou.

target: WidgetAnexa o teclado a um *target* específico. Este deve ser o Widget que solicitou o teclado. Certifique-se de que há um destino diferente anexado a cada teclado caso estejas trabalhando em modo multiusuário.

Novo na versão 1.0.8.

input_type: stringEscolha o tipo de teclado virtual a ser solicitado. Pode ser um ‘text’, ‘number’, ‘url’, ‘mail’, ‘datetime’, ‘tel’, ‘address’.

Nota: *input_type* atualmente é contemplado somente em dispositivos móveis.

Novo na versão 1.8.0.

ReturnUma instância da **Keyboard** contendo o callback, target, e se a configuração permitir isso, uma instância de **VKeyboard** anexada como uma propriedade do *.widget*.

Nota: O comportamento desta função é fortemente influenciada pelo *keyboard_mode* atual. Por favor, veja a seção de configuração *configuration tokens* para maiores informações.

restore()

Restaura o tamanho e a posição de uma janela maximizada ou minimizada. Este método só deve ser usado em plataformas Desktop.

Novo na versão 1.9.0.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

rotation

Obtém/define a rotação do conteúdo da janela. Pode ser um de 0, 90, 180 ou 270 graus.

Novo na versão 1.0.9.

rotation is an *AliasProperty*.

screenshot(*name*=‘screenshot{:04d}.png’)

Salva a imagem atualmente exibida em um arquivo.

set_icon(*filename*)

Define o ícone da janela.

Novo na versão 1.0.5.

set_title(*title*)

Define o título da janela.

Novo na versão 1.0.5.

set_vkeyboard_class(*cls*)

Novo na versão 1.0.8.

Define a classe VKeyboard a ser usada. Caso definido como *None*, será usado o *kivy.uix.vkeyboard.VKeyboard*.

show()

Mostra a janela. Este método deve ser somente usado em plataformas Desktop.

Novo na versão 1.9.0.

Nota: Este recurso requer o provedor de janela SDL2 e atualmente só é suportado em plataformas Desktop.

show_cursor

Define se o cursor é ou não exibido sobre a janela.

Novo na versão 1.9.1.

show_cursor é uma *BooleanProperty* e o padrão é *True*.

size

Obtém o tamanho da rotação da janela. Se *rotation* é definido, então o tamanho será alterado para refletir a rotação.

Novo na versão 1.0.9.

size is an *AliasProperty*.

softinput_mode

Especifica o comportamento do conteúdo da janela na exibição do teclado virtual em plataformas móveis. Isso pode ser '', 'pan', 'scale', 'resize' or 'below_target'. Seus efeitos são listados abaixo.

Valor	Efeito
''	A janela principal é deixada como está, permitindo a você usar o <i>keyboard_height</i> para gerenciar o conteúdo da janela manualmente.
'pan'	A janela principal é projetada, movendo a parte inferior da janela para ficar sempre acima do teclado.
're-size'	A janela é redimensionada e o conteúdo escalado para completar o espaço restante.
'be-low_target'	A janela é projetada de modo que o atual Widget alvo de <i>TextInput</i> que solicita o teclado é apresentado logo acima do teclado virtual.

softinput_mode é um *OptionProperty* e o padrão é *None*.

Nota: A opção de *resize* não está atualmente funcionando no Android com SDL2.

Novo na versão 1.9.0.

Alterado na versão 1.9.1: A opção 'below_target' foi adicionada.

system_size

Tamanho real da janela ignorando a rotação.

Novo na versão 1.0.9.

system_size is an [AliasProperty](#).

toggleFullscreen(*args, **kwargs)

Altera entre modo de tela cheia e janela normal.

Obsoleto desde a versão 1.9.0: Utilize ao invés a propriedade [fullscreen](#).

top

Top position of the window.

Nota: It's an SDL2 property with [0, 0] in the top-left corner.

Alterado na versão 1.10.0: *top* is now an [AliasProperty](#)

Novo na versão 1.9.1.

top is an [AliasProperty](#) and defaults to position set in [Config](#).

ungrabMouse()

Ungrab mouse

Novo na versão 1.10.0.

Nota: Este recurso requer o provedor de janelas SDL2.

width

Largura da janela rotacionada.

width é somente leitura [AliasProperty](#).

4.4 Módulo Kivy para dependências binárias

Dependências binárias como o Gstreamer são instalados como um módulo de namespace de kivy.deps. Estes módulos são responsáveis por garantir que os binários estejam disponível para o Kivy.

4.5 Efeitos

Novo na versão 1.7.0.

Tudo comeca com a [KineticEffect](#), a classe base para calcular a velocidade de um movimento.

Esta classe base é usada para implementar o *ScrollEffect*, uma classe base usada por nós *ScrollView* widget effect. Nós temos várias implementações:

- A classe base usada para implementar um efeito *ScrollEffect*: . Isso só calcula o scrolling e o overscroll.
- *DampedScrollEffect*: usa as informações do overscroll para permitir que o usuário arraste mais do que o esperado. Uma vez que o usuário pára o movimento de arrastar, a posição é retornada a um dos limites.
- *OpacityScrollEffect*: usa informações do overscroll para reduzir a opacidade do Widget *ScrollView*. Quando o usuário pára o arrastamento, a opacidade é definida novamente como sendo igual a 1.

4.5.1 Efeito de rolagem amortecido

Novo na versão 1.7.0.

Esse efeito rolagem usa *overscroll* para calcular valor scroll e diminui retornando limite superior ou inferior.

`class kivy.effects.dampedscroll.DampedScrollEffect(**kwargs)`
Tem como Base: *kivy.effects.scroll.ScrollEffect*

Classe DampedScrollEffect. Ver documentação módulo para maior informação.

edge_damping

Despejar Borda.

edge_damping é uma *NumericProperty* e tem valor de 0.25

min_overscroll

Sobreposição menor que o valor será normalizada para 0.

Novo na versão 1.8.0.

min_overscroll é uma *NumericProperty* e padrão é .5.

round_value

Se True, quando movimento parar, *value* será arredondado para próximo inteiro.

Novo na versão 1.8.0.

round_value é uma *BooleanProperty* e padrão é True.

spring_constant

Constante Spring.

spring_constant é uma *NumericProperty* e padrão é 2.0

4.5.2 Efeito Kinetic

Novo na versão 1.7.0.

A **KineticEffect** é a classe base que é usada para manipular a velocidade de um movimento. Quando o movimento é finalizado, o resultado será computado na posição final de acordo com a velocidade do movimento, e sua redução de velocidade considerando a fricção. O movimento para até a velocidade ser 0.

Conceitualmente, o uso poderia ser:

```
>>> effect = KineticEffect()
>>> effect.start(10)
>>> effect.update(15)
>>> effect.update(30)
>>> effect.stop(48)
```

Ao longo do tempo, você iniciará um movimento de um valor, atualize-o, e pare o movimento. Nesse momento, você terá o valor do movimento em **KineticEffect.value**. No exemplo que eu digitei manualmente, a velocidade calculado será:

```
>>> effect.velocity
3.1619100231163046
```

Depois da interação do relógio múltiplo, a velocidade irá diminuir de acordo com **KineticEffect.friction**. O valor calculado será armazenado em :attr:`KineticEffect.value`. A saída desse valor poderia ser:

```
46.30038145219605
54.58302451968686
61.9229016256196
# ...
```

class kivy.effects.kinetic.KineticEffect(kwargs)**
Bases: **kivy.event.EventDispatcher**

Classe Kinetic Effect. Veja o módulo da documentação para maiores informações.

cancel()

Cancelar um movimento. Isso pode ser usado no caso **stop()** não possa ser chamado. Irá redefinir **is_manual** para False, e calcular o movimento se a velocidade é >0

friction

Atrito para aplicar na velocidade

velocity é uma **NumericProperty** e o padrão é 0.05.

is_manual

Indica se o movimento está em progresso (True) ou não (False).

velocity é uma *BooleanProperty* e o padrão é *False*.

max_history

Salva até o valor *max_history* de movimentos no histórico. Isto é usado para o cálculo correto da velocidade de acordo com o movimento.

max_history é uma *NumericProperty* e o padrão é 5.

min_distance

A distância mínima para um movimento ter velocidade não zero.

Novo na versão 1.8.0.

min_distance é uma *NumericProperty* e o padrão é 0.1.

min_velocity

A velocidade abaixo dessa quantidade é normalizado para 0. Em ordem alfabética, qualquer movimento que a velocidade cair abaixo desse número será parado.

Novo na versão 1.8.0.

min_velocity é uma *NumericProperty* e o padrão é 0.5.

start(*val*, *t=None*)

Inicia o movimento.

Parameters

val: float ou intValor do movimento

t: float, e o padrão é *None*Momento (Time) em que o movimento acontece. Se não houver tempo definido, usará *time.time()*

stop(*val*, *t=None*)

Para o movimento.

Veja *start()* para mais argumentos.

update(*val*, *t=None*)

Atualiza o movimento.

Veja *start()* para mais argumentos.

update_velocity(*dt*)

(interno) atualiza a velocidade de acordo ao frametime e friction.

value

Valor(durante o movimento e calculado) do efeito.

velocity é uma *NumericProperty* e o padrão é 0.

velocity

Velocidade do movimento.

velocity é uma *NumericProperty* e o padrão é 0.

4.5.3

Com base no `DampedScrollEffect`, este também diminuirá a opacidade do Widget de destino durante o overscroll.

`class kivy.effects.opacityscroll.OpacityScrollEffect(**kwargs)`
Bases: `kivy.effects.dampedscroll.DampedScrollEffect`

Classe `OpacityScrollEffect`. Usa a informação do overscroll para reduzir a opacidade do Widget ScrollView. Quando o usuário interrompe o arrastamento (ação de arrastar), a opacidade é definida como 1.

4.5.4 Scroll effect

Novo na versão 1.7.0.

Baseada em: class:`~kivy.effects.kinetic` effect, the `ScrollEffect` Limitará os movimentos até os limites determinados por `min` and `max` properties. Se o movimento exceder estes limites, será calculada a quantidade excedida através de `overscroll` e tentar retornar o valor de um dos limites.

Isso é muito útil para implementar uma lista de rolagem. Atualmente usamos esse classe como a efeito base para nosso widget class:`~kivy.uix.scrollview.ScrollView`.

`class kivy.effects.scroll.ScrollEffect(**kwargs)`
Bases: `kivy.effects.kinetic.KineticEffect`

Classe `ScrollEffect`. Veja o módulo da documentação para maiores informações.

`displacement`

Distância acumulada do movimento durante a interação. Isso é usado para determinar se o movimento é um arrastar (drag) (ou mais do que `drag_threshold`).

`displacement` é uma `NumericProperty` e o padrão é 0.

`drag_threshold`

A distância mínima para viajar antes do movimento é considerada como um arrasto.

`velocity` é uma `NumericProperty` e o padrão é 20sp.

`max`

Límite máximo a ser usado como rolagem.

`max` é uma `NumericProperty` e o padrão é 0.

`min`

Límite mínimo a ser usado como rolagem.

`min` é uma `NumericProperty` e o padrão é 0.

overscroll

Valor computado quando usuário ultrapassar limites páginação. ex. fora de limites.

overscroll é uma *NumericProperty* e o padrão é 0.

reset(pos)

(Interno) Redefinir o valor e a velocidade para o *pos*. Usado principalmente quando os limites são verificados.

scroll

Valor calculado para rolagem. Esse valor é diferente de *kivy.effects.kinetic.KineticEffect.value* em que ele retornará para um dos limites min/max.

scroll é uma *NumericProperty* e o padrão é 0.

target_widget

Widget pra ser anexado a este efeito. Mesmo que essa classe não faça alterações no *target_widget* por padrão, as subclasses podem usá-lo para alterar os gráficos ou aplicar transformações personalizadas.

target_widget é uma *ObjectProperty* e o padrão é *None*.

4.6 Garden

Novo na versão 1.7.0.

Alterado na versão 1.8.0.

Garden é um projeto para centralizar Addon para o Kivy e mantido pelos usuários. Você pode obter mais informações em [Kivy Garden](#). Todos os pacotes do Garden são centralizados no repositório [kivy-garden Github](#).

Garden é agora distribuído como um módulo do Python separado, kivy-garden. Para instala-lo utilize o pip:

```
pip install kivy-garden
```

O módulo Garden não inclui inicialmente nenhum pacote. Você pode baixa-lo com a ferramenta Garden instalada pelo gerenciador de pacotes pip:

```
# Installing a garden package
garden install graph

# Upgrade a garden package
garden install --upgrade graph

# Uninstall a garden package
garden uninstall graph
```

```
# List all the garden packages installed
garden list

# Search new packages
garden search

# Search all the packages that contain "graph"
garden search graph

# Show the help
garden --help
```

Todos os pacotes do Garden são instalados por padrão em `~/.kivy/garden`.

Nota: Em versões anteriores do Kivy, o Garden foi uma ferramenta contida em `kivy/tools/garden`. Isso não existe mais, mas o módulo `kivy-garden` disponibiliza exatamente a mesma funcionalidade.

4.6.1 Empacotamento

Se desejas incluir um pacote do Garden em sua aplicação, podes `--app` ao comando `install`. Isso criará um diretório `libs/garden` em seu diretório atual que será usado pelo `kivy.garden`.

Por exemplo:

```
cd myapp
garden install --app graph
```

kivy.garden.garden_system_dir = 'garden'

Sistema de arquivo onde os módulos do Garden podem ser instalados

4.7 Gráficos

Este pacote tem a capacidade de montar muitas funções de baixo nível usadas para desenhar. Todos os pacotes gráficos são compatíveis com OpenGL ES 2.0 e apresenta muitas opções de renderização.

4.7.1 O básico

Para desenhar na tela, você precisará:

1. uma objeto de *Canvas*.

2. *Instruction* objetos.

Cada *Widget* no Kivy já tem uma *Canvas* por padrão. Ao criar um widget, você pode criar todas as instruções necessárias para o desenho. Se *self* for o seu widget atual, você pode fazer:

```
from kivy.graphics import *
with self.canvas:
    # Add a red color
    Color(1., 0, 0)

    # Add a rectangle
    Rectangle(pos=(10, 10), size=(500, 500))
```

As instruções *Color* e *Rectangle* são automaticamente adicionadas ao objeto *Canvas* e serão usadas quando a janela for desenhada.

Nota: As instruções de desenho do Kivy não são automaticamente relativas à posição ou ao tamanho dos widgets. Você, portanto, precisa considerar esses fatores ao desenhar. Para tornar suas instruções de desenho relativas ao widget, as instruções precisam ser declaradas no: mod: *KvLang* <kivy.lang> ou ligadas às mudanças de posição e tamanho. Consulte: ref: *adding_widget_background* para obter mais detalhes.

4.7.2 Mecanismo de Recarga GL

Novo na versão 1.2.0.

Durante o tempo de vida do aplicativo, o contexto do OpenGL pode ser perdido. Isto acontece:

- Quando a janela é redimensionada no OS X ou na plataforma Windows e você está usando pygame como um provedor de janela. Isto é devido a SDL 1.2. No design SDL 1.2, ele precisa recriar um contexto GL sempre que a janela é redimensionada. Isso foi corrigido no SDL 1.3 mas o pygame ainda não está disponível nele por padrão
- Quando o Android lançar os recursos do aplicativo: quando o aplicativo for para o plano de fundo, o Android poderá recuperar seu contexto opengl para fornecer o recurso a outro aplicativo. Quando o usuário retorna ao aplicativo, um novo contexto criado para o seu aplicativo é fornecido.

A partir de 1.2.0, introduzimos um mecanismo para recarregar todos os recursos gráficos usando a GPU: *Canvas*, *FBO*, *Shader*, *Texture*, *VBO* e *VertexBatch*:

- *VBO* e *VertexBatch* são construídos por nossas instruções gráficas. Temos todos os dados necessários para reconstruir quando formos recarregar.

- *Shader*: o mesmo que *VBO*, é onde armazenamos a fonte (source) e os valores usados no *Shader* para que possamos recriar o Vertex/fragment/programa.
- *Texture*: se a textura tiver uma origem (um arquivo de imagem ou atlas), a imagem é recarregada da fonte (arquivo) e recarregada para a GPU.

Você deve cobrir estes casos pessoalmente:

- Texturas sem uma fonte: se você criou manualmente uma textura e manualmente blit dados / um buffer para ele, então, você que deverá lidar com o recarregamento. Verifique o: doc: *api-kivy.graphics.texture* para saber como gerenciar esse caso. (A renderização de texto já gera a textura e controla o recarregamento. Você não precisa recarregar o texto.)
- FBO: se você adicionou / removeu / desenhou coisas várias vezes no FBO, não podemos recarregá-lo. Nós não mantemos um histórico das instruções colocadas sobre ele. Quanto às texturas sem fonte, verifique: doc: *api-kivy.graphics.fbo* para saber como gerenciar esse caso.

`class kivy.graphics.Bezier`

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma curva Bezier 2D.

Novo na versão 1.0.8.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2...)

segments: int, o padrão é 180Define quantos segmentos são necessários para desenhar uma curva. O desenho será mais suave quanto maior for a quantidade de segmentos.

loop: bool, o padrão é FalseDefina a curva Bezier para juntar o último ponto ao primeiro.

dash_length: intTamanho de um segmento (se tracejado), o padrão é 1.

dash_offset: intDistância entre o final do segmento e o início do próximo, o padrão é 0. Alterando isso torna-o tracejado.

dash_length

Propriedade para obter/configurar o tamanho do tracejado na curva.

dash_offset

Propriedade para obter/configurar o deslocamento do tracejado na curva.

points

Propriedade para obter/configurar os pontos do Triângulo.

Aviso: Isso sempre reconstruirá todo o gráfico da nova lista de pontos. Pode ter uso bastante intensivo de CPU.

segments

Propriedade para obter/configurar o número de segmentos numa curva.

class kivy.graphics.BindTexture

Bases: *kivy.graphics.instructions.ContextInstruction*

InSTRUÇÕES Gráficas BindTexture . A instrução *BindTexture* vinculará uma textura e ativará *GL_TEXTURE_2D* para desenho subsequente.

Parameters

texture: Texture Especifica a textura a ser vinculada ao índice fornecido.

source

Define/Obtém a origem (nome do arquivo) pra carregar a textura.

class kivy.graphics.BorderImage

Bases: *kivy.graphics.vertex_instructions.Rectangle*

Uma 2D *BorderImage*. O comportamento do *BorderImage* é semelhante ao conceito de uma imagem de borda CSS3.

Parameters

border: list Border information in the format (bottom, right, top, left). Each value is in pixels.

auto_scale: bool Novo na versão 1.9.1.

If the *BorderImage*'s size is less than the sum of it's borders, horizontally or vertically, and this property is set to True, the borders will be rescaled to accommodate for the smaller size.

auto_scale

Propriedade para definir se os cantos são automaticamente dimensionados quando o *BorderImage* é muito pequeno.

border

Propriedade para obter/configurar a borda da classe.

display_border

Propriedade para obter/configurar o tamanho da exibição da borda.

class kivy.graphics.Callback

Bases: *kivy.graphics.instructions.Instruction*

Novo na versão 1.0.4.

Um callback é uma instrução que será chamada quando a operação de desenho for executada. Ao adicionar instruções ao Canvas, poderás fazer isso:

```
with self.canvas:  
    Color(1, 1, 1)  
    Rectangle(pos=self.pos, size=self.size)  
    Callback(self.my_callback)
```

A definição do callback deve ser:

```
def my_callback(self, instr):  
    print('I have been called!')
```

Aviso: Observe que, se você executar muitas e/ou chamadas custosas por callbacks, podes potencialmente diminuir significativamente o desempenho de renderização.

A atualização da sua tela não ocorrerá até que algo novo aconteça. Desde o seu callback, podes pedir por uma atualização:

```
with self.canvas:  
    self.cb = Callback(self.my_callback)  
# then later in the code  
self.cb.ask_update()
```

Se utilizar a classe *Callback* para invocar métodos de renderização de outro toolkit, terás problemas com o contexto OpenGL. O estado do OpenGL pode ter sido manipulado pelo outro conjunto de ferramentas e assim que o fluxo do programa retornar para o Kivy, ele será quebrado. Poderá haver quebras, falhas, buracos negros podem ocorrer, etc. Para evitar isso, podes ativar a opção *reset_context*. Ela redefinirá o estado do contexto do OpenGL para tornar a renderização do Kivy correta após a chamada para o callback.

Aviso: O *reset_context* não é um reset completo do OpenGL. Se tiveres problemas com isso, entre em contato conosco.

ask_update()

Informe o Canvas-pai que gostarias de atualizar no próximo Frame. Isso é útil quando precisares disparar um redesenho devido a algum valor alterado, por exemplo.

Novo na versão 1.0.4.

reset_context

Defina isso como *True* se quiseres redefinir o contexto OpenGL do Kivy após

o callback ter sido invocado.

```
class kivy.graphics.Canvas
    Bases: kivy.graphics.instructions.CanvasBase
```

A importante classe *Canvas*. Use esta classe para adicionar gráficos ou instruções de contexto que desejas usar para desenhar.

Nota: O Canvas suporta a instrução `with` do Python e sua semântica de Entrada & Saída.

Uso do Canvas sem a instrução `with`:

```
self.canvas.add(Color(1., 1., 0))
self.canvas.add(Rectangle(size=(50, 50)))
```

Uso do Canvas sem a instrução `with` do Python:

```
with self.canvas:
    Color(1., 1., 0)
    Rectangle(size=(50, 50))
```

after

Propriedade para obter o ‘after’ grupo.

ask_update()

Informe o Canvas que gostarias de atualizar no próximo Frame. Isso é útil quando precisas disparar um redesenho devido a alteração de algum valor, por exemplo.

before

Propriedade para obter o grupo ‘before’.

clear()

Limpa cada *Instruction* na tela, deixando-a limpa.

draw()

Aplica a instrução à nossa janela.

has_after

Propriedade para ver se o grupo *after* já foi criado.

Novo na versão 1.7.0.

has_before

Propriedade para ver se o grupo *before* já foi criado.

Novo na versão 1.7.0.

opacity

Propriedade para obter/definir o valor de opacidade da tela.

Novo na versão 1.4.1.

O atributo *opacity* controla a opacidade da tela e a de seus filhos. Tenha cuidado, é um atributo cumulativo: o valor é multiplicado pela opacidade global atual e o resultado é aplicado à cor do contexto atual.

Por exemplo: se o pai tiver uma opacidade de 0,5 e um filho tiver uma opacidade de 0,2, a opacidade final do filho será de $0,5 * 0,2 = 0,1$.

Em seguida, a opacidade é aplicada no *Shader* como por exemplo:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

class kivy.graphics.CanvasBase

Bases: *kivy.graphics.instructions.InstructionGroup*

O *CanvasBase* fornece os métodos do gerenciador de contexto para *Canvas*.

class kivy.graphics.Color

Bases: *kivy.graphics.instructions.ContextInstruction*

Instrução para definir o estado da cor para quaisquer vértices que será desenhado após ele.

Isso representa uma cor entre 0 e 1, mas é aplicado como *multiplicador* à textura de quaisquer instruções de vértice que o seguem em uma tela. Se nenhuma textura for definida, a instrução de vértice assumirá a cor precisa da instrução Color.

Por exemplo, se um retângulo tem uma textura com cor uniforme (0.5, 0.5, 0.5, 1.0) e a cor anterior tem *rgba* = (1, 0.5, 2, 1), a cor atual visível será “(0.5, 0.25, 1.0, 1.0)” uma vez que a instrução Color é aplicada como um multiplicador para cada componente do *rgba*. Neste caso, um componente de Cor fora da faixa 0-1 dá um resultado visível à medida que a intensidade do componente azul é duplicada.

Para declarar um Color em Python, podes fazer:

```
from kivy.graphics import Color

# create red v
c = Color(1, 0, 0)
# create blue color
c = Color(0, 1, 0)
# create blue color with 50% alpha
c = Color(0, 1, 0, .5)

# using hsv mode
c = Color(0, 1, 1, mode='hsv')
# using hsv mode + alpha
c = Color(0, 1, 1, .2, mode='hsv')
```

Você também pode definir componentes de cores que estarão disponíveis como propriedades, passando-as como argumentos de palavra-chave:

```
c = Color(b=0.5) # sets the blue component only
```

Em kv lang você pode definir as propriedades de cor diretamente:

```
<Rule>:  
    canvas:  
        # red color  
        Color:  
            rgb: 1, 0, 0  
        # blue color  
        Color:  
            rgb: 0, 1, 0  
        # blue color with 50% alpha  
        Color:  
            rgba: 0, 1, 0, .5  
  
        # using hsv mode  
        Color:  
            hsv: 0, 1, 1  
        # using hsv mode + alpha  
        Color:  
            hsv: 0, 1, 1  
            a: .5
```

a

Componente Alfa, entre 0 e 1.

b

Componente Azul, entre 0 e 1.

g

Componente Verde, entre 0 e 1.

h

Componente Hue, entre 0 e 1.

hsv

Cor HSV, lista de 3 valores no intervalo entre 0-1, alpha será 1.

r

Componente Vermelho, entre 0 e 1.

rgb

Cor RGB, lista de 3 valores no intervalo 0-1. O alfa será 1.

rgba

Cor RGBA, lista de 4 valores no intervalo de 0-1.

s

Componente de Saturação, entre 0 e 1.

v

Componente valor, entre 0 e 1.

class kivy.graphics.ContextInstruction

Bases: *kivy.graphics.instructions.Instruction*

A classe ContextInstruction é a base para a criação de instruções que não têm uma representação visual direta, mas em vez disso modifica o estado de atual de Canvas, por exemplo, vinculação de textura, configuração dos parâmetros de cor, manipulação de matriz e assim por diante.

class kivy.graphics.Ellipse

Bases: *kivy.graphics.vertex_instructions.Rectangle*

Uma elipse 2D.

Alterado na versão 1.0.7: Adicionado *angle_start* e *angle_end*.

Parameters

segments: int, o padrão é 180 Define quantos segmentos são necessários para desenhar a Elipse. O desenho será mais suave se você tiver muitos segmentos.

angle_start: int, o padrão é 0 Especifica o ângulo inicial, em graus, da porção do disco.

angle_end: int, o padrão é 360 Especifica o ângulo final, em graus, da porção do disco.

angle_end

Ângulo final da elipse em graus, o padrão é 360.

angle_start

Inicia o ângulo da elipse em graus, o padrão é 0.

segments

Propriedade para obter/configurar o número de segmentos de uma Elipse.

class kivy.graphics.Fbo

Bases: *kivy.graphics.instructions.RenderContext*

FBO para envolver a extensão OpenGL Framebuffer. O suporte FBO “com” declaração.

Parameters

clear_color: tuple, o padrão é (0, 0, 0, 0) Define a cor padrão para limpar a framebuffer

size: tuple, o padrão é (1024, 1024) Tamanho padrão do FrameBuffer

push_viewport: bool, o padrão é *True*Se *True*, o Viewport do OpenGL será definido para o tamanho do framebuffer, e será automaticamente restaurado quando o framebuffer for liberado.

with_depthbuffer: bool, o padrão é *False*Se *True*, o framebuffer será alocado com um buffer Z.

with_stencilbuffer: bool, o padrão é *False*Novo na versão 1.9.0.

Se *True*, o framebuffer será alocado com um buffer de Stencil.

texture: Texture, o padrão é *None*Se *None*, a textura padrão será criada.

Nota: Usar o **with_stencilbuffer** e **with_depthbuffer** não é suportado no Kivy 1.9.0

add_reload_observer()

Adiciona um callback pra ser chamado depois que o todo o contexto gráfico for recarregado. Este é o lugar onde podes reupload seus dados personalizados na GPU.

Novo na versão 1.2.0.

Parameters

callback: func(context) -> return NoneO primeiro parâmetro será o contexto propriamente dito

bind()

Vincula o FBO ao contexto OpenGL atual. *Bind* significa que você habilita o FrameBuffer, e todas as operações de desenho atuarão dentro do Framebuffer, **release()** seja invocado.

As operações de vinculação/liberação são chamadas automaticamente quando você adiciona objetos gráficos a ele. Se quiseres manipular um FrameBuffer você mesmo, podes usa-lo dessa forma:

```
self.fbo = FBO()  
self.fbo.bind()  
# do any drawing command  
self.fbo.release()  
  
# then, your fbo texture is available at  
print(self.fbo.texture)
```

clear_buffer()

Limpa o FrameBuffer com o **clear_color**.

Precisas vincular o FrameBuffer você mesmo antes de invocar este método:

```
fbo.bind()  
fbo.clear_buffer()  
fbo.release()
```

clear_color

Clear color no formato (red, green, blue, alpha).

get_pixel_color()

Obtém a cor do pixel com as coordenadas de janela especificadas em *wx*, *wy*. O resultado será retornado no formato RGBA.

Novo na versão 1.8.0.

pixels

Obtém a textura dos pixels, apenas no formato RGBA, Unsigned Byte. A origem da imagem está na parte inferior esquerda.

Novo na versão 1.7.0.

release()

Libera o FrameBuffer (unbind).

remove_reload_observer()

Remove o callback da lista de observadores, adicionado anteriormente por [*add_reload_observer\(\)*](#).

Novo na versão 1.2.0.

size

Tamanho do FrameBuffer, no formato (width, height).

Se você alterar o tamanho, o conteúdo do FrameBuffer será perdido.

texture

Retorna a textura do FrameBuffer

class kivy.graphics.GraphicException

Bases: [exceptions.Exception](#)

Exceção gerada quando um erro gráfico é disparado.

class kivy.graphics.Instruction

Bases: [kivy.event.ObjectWithUid](#)

Representa a menor instrução disponível. Esta classe é apenas para uso interno, não use-a diretamente.

proxy_ref

Retornar uma referência de proxy para a instrução, ou seja, sem criar uma referência do widget. Veja [weakref.proxy](#) para obter maiores informações.

Novo na versão 1.7.2.

```
class kivy.graphics.InstructionGroup
```

Bases: *kivy.graphics.instructions.Instruction*

Grupo de *Instructions*. Permite a adição e remoção de instruções gráficas. Pode ser usado diretamente como segue:

```
blue = InstructionGroup()
blue.add(Color(0, 0, 1, 0.2))
blue.add(Rectangle(pos=self.pos, size=(100, 100)))

green = InstructionGroup()
green.add(Color(0, 1, 0, 0.4))
green.add(Rectangle(pos=(100, 100), size=(100, 100)))

# Here, self should be a Widget or subclass
[self.canvas.add(group) for group in [blue, green]]
```

add()

Adiciona uma nova instrução *Instruction* para a nossa lista.

clear()

Remove todas as *Instructions*.

get_group()

Retorna um iterável para todos as *Instructions* com um nome de grupo específico.

insert()

Insira uma nova *Instruction* em nossa lista no índice.

remove()

Remove uma existente *Instruction* da nossa lista.

remove_group()

Remove todas as *Instructions* com um nome de grupo específico.

```
class kivy.graphics.Line
```

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma linha 2D.

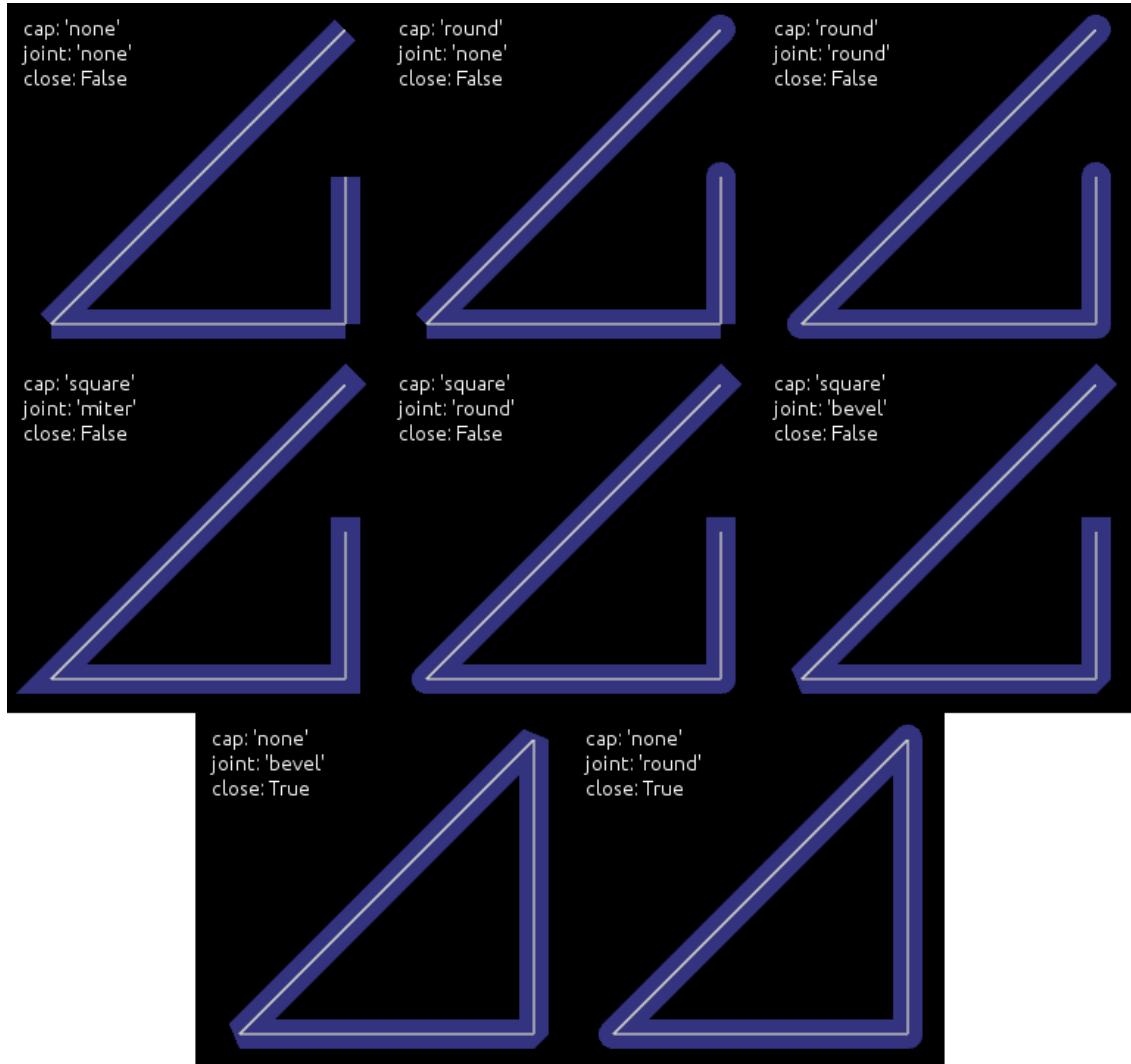
O desenho de uma linha pode ser feito facilmente:

```
with self.canvas:
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

A linha tem 3 modos de desenho interno que você deve estar ciente para obter os melhores resultados:

1.Se o *width* for 1.0, então o padrão de desenho *GL_LINE* do OpenGL será utilizado. *dash_length* e *dash_offset* funcionarão, enquanto as propriedades para o cap e a junção não tiverem nenhum significado aqui.

2.Se o: attr: *width* for maior que 1.0, então um método de desenho personalizado, baseado em triangulação, será usado. :attr: *dash_length* e: attr: *dash_offset* não funcionam neste modo. Além disso, se a cor atual tiver um alpha menor que 1,0, um estêncil será usado internamente para desenhar a linha.



Parameters

points: listLista de pontos no formato (x1, y1, x2, y2...)

dash_length: intTamanho de um segmento (se tracejado), o padrão é 1.

dash_offset: intOffset entre o final de um segmento e o início do próximo, o padrão é 0. Alterando isso torna-o tracejado.

width: floatLargura da linha, o padrão é 1.0.

cap: str, e o padrão é 'round'Veja o *cap* para maiores informações.

joint: str, e o padrão é 'round'Veja o *joint* para maiores informações.

cap_precision: int, o padrão é 10
Veja o *cap_precision* para maiores informações

joint_precision: int, o padrão é 10
Veja *joint_precision* para maiores informações. Veja *cap_precision* para maiores informações.

joint_precision: int, o padrão é 10
Veja *joint_precision* para maiores informações.

close: bool, defaults to False Se *True*, a linha será fechada.

circle: list Se definido, o *points* será definido para construir um círculo. Veja *circle* para maiores informações.

ellipse: list Se definido, o *points* será definido para construir uma ellipse. Veja *ellipse* para maiores informações.

rectangle: list Se definido, o *points* será definido para construir um retângulo. Veja *rectangle* para maiores informações.

bezier: list Se definido, o *points* será definido para construir uma linha bezier. Veja *bezier* para maiores informações.

bezier_precision: int, e o padrão é 180 Precisão do desenho Bezier.

Alterado na versão 1.0.8: *dash_offset* e *dash_length* foram adicionados.

Alterado na versão 1.4.1: *width, cap, joint, cap_precision, joint_precision, close, ellipse, rectangle* foram adicionados.

Alterado na versão 1.4.1: *bezier, bezier_precision* foram adicionados.

bezier

Utilize essa propriedade para construir uma linha Bezier, sem calcular os *points*. Você pode definir esta propriedade, não obtê-la.

O argumento deve ser uma tupla de $2n$ elementos, n sendo o número de pontos.

Uso:

```
Line(bezier=(x1, y1, x2, y2, x3, y3))
```

Novo na versão 1.4.2.

Nota: Os cálculos das linhas de Bezier usam poucos recursos para um número baixo de pontos, mas a complexidade é quadrática, então as linhas com muitos pontos podem ser muito demoradas para serem construídas, use com cuidado!

bezier_precision

Número de iteração para desenhar o Bezier entre 2 segmentos, o padrão é 180. O *bezier_precision* deve ser pelo menos '1'q.

Novo na versão 1.4.2.

cap

Determine o limite da linha, o padrão é 'round'. Pode ser uma das seguintes opções 'none', 'square' ou 'round'.

Novo na versão 1.4.1.

cap_precision

Número de iteração para desenhar a cápsula “redonda”, o padrão é 10. A *cap_precision* deve ser pelo menos 1.

Novo na versão 1.4.1.

circle

Use this property to build a circle, without calculating the **points**. You can only set this property, not get it.

O argumento deve ser uma tupla de (center_x, center_y, radius, angle_start, angle_end, segments):

- center_x* e *center_y* representam o centro do círculo
- radius* representa o raio do círculo
- (optional) *angle_start* and *angle_end* are in degree. The default value is 0 and 360.
- (optional) *segments* is the precision of the ellipse. The default value is calculated from the range between angle.

Observe que cabe a você fechar **close** ou não o círculo.

Por exemplo, para construir uma simples elipse em Python:

```
# simple circle
Line(circle=(150, 150, 50))

# only from 90 to 180 degrees
Line(circle=(150, 150, 50, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(circle=(150, 150, 50, 90, 180, 20))
```

Novo na versão 1.4.1.

close

Se *True*, a linha será fechada.

Novo na versão 1.4.1.

dash_length

Propriedade para obter/configurar o tamanho dos traços na curva

Novo na versão 1.0.8.

dash_offset

Propriedade para obter/configurar o deslocamento entre os traço curva

Novo na versão 1.0.8.

ellipse

Use esta propriedade para construir uma Elipse, sem calcular os *points*. Você somente pode definir essa propriedade, não obtê-la.

O argumento deve ser uma tupla de (*x*, *y*, *width*, *height*, *angle_start*, *angle_end*, *segments*,):

- *x* e *y* representam a parte inferior esquerda da Elipse
- *width* e *height* representam o tamanho da Elipse
- (optional) *angle_start* and *angle_end* are in degree. The default value is 0 and 360.
- (optional) *segments* is the precision of the ellipse. The default value is calculated from the range between angle.

Observe que cabe a você fechar *close* ou não a Elipse.

Por exemplo, para construir uma simples elipse em Python:

```
# simple ellipse
Line(ellipse=(0, 0, 150, 150))

# only from 90 to 180 degrees
Line(ellipse=(0, 0, 150, 150, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(ellipse=(0, 0, 150, 150, 90, 180, 20))
```

Novo na versão 1.4.1.

joint

Determine a junção da linha, o padrão é ‘round’. Pode ser um das opções ‘none’, ‘round’, ‘bevel’, ‘miter’.

Novo na versão 1.4.1.

joint_precision

Número de iteração para desenhar a junção “redonda”, o padrão é 10. A *joint_precision* deve ser pelo menos 1.

Novo na versão 1.4.1.

points

Propriedade para obter/configurar pontos de uma linha

Aviso: Isso sempre reconstruirá os gráficos completamente desde a nova lista de pontos. Pode utilizar muito CPU.

rectangle

Use essa propriedade para construir um retângulo, sem calcular o **points**. Você só pode definir esta propriedade, não obtê-lo.

O argumento deve ser uma tupla de (x, y, width, height):

- *x* e *y* representam a posição bottom-left (inferior esquerda) do retângulo.
- *width* e *height* representam o tamanho

A linha é automaticamente fechada.

Uso:

```
Line(rectangle=(0, 0, 200, 200))
```

Novo na versão 1.4.1.

rounded_rectangle

Use essa propriedade para construir um retângulo, sem calcular o **points**. Você só pode definir esta propriedade, não obtê-lo.

O argumento deve ser uma tupla de uma das seguintes formas:

- (x, y, width, height, corner_radius)
- (x, y, width, height, corner_radius, resolution)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4, resolution)
- x* e *y* representam a posição bottom-left (inferior esquerda) do retângulo.
- width* e *height* representam o tamanho
- corner_radius* é o número de pixels entre as duas bordas e o centro do arco do círculo juntando-os.
- resolução é o número de segmento de linha que será usado para desenhar o arco de círculo em cada canto (o padrão é 30)

A linha é automaticamente fechada.

Uso:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40,  
→100))
```

Novo na versão 1.9.0.

width

Determine a largura da linha, o padrão 1.0.

Novo na versão 1.4.1.

class kivy.graphics.SmoothLine

Bases: *kivy.graphics.vertex_instructions.Line*

Linha experimental usando métodos de over-draw para obter melhores resultados anti-aliasing. Possui algumas desvantagens:

- Desenhar uma linha com alfa provavelmente não terá o resultado desejado da linha cruzando a si mesma.
- A propriedade *cap*, *joint* e *dash* não são suportadas.
- Ele usa uma textura personalizada com um alfa premultiplied.
- Linhas com menos de 1px de largura não são suportadas: elas se parecem com as mesmas.

Aviso: Isso é uma trabalho não finalizado, experimental e sujeito a falhas.

Novo na versão 1.9.0.

overdraw_width

Determine a largura de overdraw da linha, o padrão é 1.2.

class kivy.graphics.MatrixInstruction

Bases: *kivy.graphics.instructions.ContextInstruction*

Classe base para instrução Matrix no Canvas.

matrix

Matrix propriedade. Matriz do módulo de transformação. Define a matriz usando esta propriedade quando uma alteração é feita é importante porque ele irá notificar o contexto sobre a atualização.

stack

Nome da pilha de matriz a ser usada. Pode ser ‘modelview_mat’ ou ‘projection_mat’.

Novo na versão 1.6.0.

class kivy.graphics.Mesh

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma Malha 2D.

Em OpenGL ES 2.0 e em nossa implementação gráfica, você não pode ter mais do que 65535 índices.

A lista de vértices é descrita como:

```
vertices = [x1, y1, u1, v1, x2, y2, u2, v2, ...]
           |     |   |     |
           +---- i1 ----+ +---- i2 ----+
```

Se desejas desenhar um triângulo, adicione 3 vértices. Você pode então fazer uma lista de índice da seguinte maneira:

```
indices = [0, 1, 2]
```

Novo na versão 1.1.0.

Parameters

vertices: iterableLista de vértices no formato (x1, y1, u1, v1, x2, y2, u2, v2...).

indices: iterableLista de índices no formato (i1, i2, i3...).

mode: strModo de VBO. Veja *mode* para maiores informações. O padrão é 'points'.

fmt: listO formato para vértices, por padrão, cada vértice é descrito por coordenadas 2D (x, y) e uma textura 2D coordenada (u, v). Cada elemento da lista deve ser uma tupla ou lista, do formulário

(variable_name, size, type)

que permitirá mapear dados de vértices para as instruções glsl.

[(b'v_pos', 2, b'float'), (b'v_tc', 2, b'float'),]

Permitirá usar

atributo *vec2 v_pos*; atributo *vec2 v_tc*;

No sombreador de vértices do glsl.

Alterado na versão 1.8.1: Antes, *vertices* e 'indices' sempre eram convertidos numa lista, agora, eles só são convertidos numa lista se eles não implementarem a interface de buffer. Então, por exemplo, Numpy Arrays, Arrays Python e etc são usados no lugar, sem criar cópias adicionais. No entanto, os buffers não podem ser readonly (mesmo que eles não sejam alterados, devido a uma limitação do Cython) e devem ser contíguos na memória.

Nota: Ao passar um memoryview ou uma instância que implementa a interface do buffer, *vertices* deverá ser um buffer de floats ("f" Código em Python de

Array) e *indices* deve ser um buffer de unsigned short ("H" Código em Python de Array). Arrays em outros formatos ainda terão de ser convertidos internamente, negando qualquer ganho potencial.

indices

Índices Vertex usados para especificar a ordem ao desenhar a malha.

mode

Modo VBO usado para desenhar vértices / índices. Pode ser uma das seguintes opções: 'points', 'line_strip', 'line_loop', 'lines', 'triangles', 'triangle_strip' or 'triangle_fan'.

vertices

Lista de coordenadas *x*, *y*, *u*, *v* usadas para construir a Malha. Agora, a instrução *Mesh* não permite que você altere o formato dos vértices, o que significa que é apenas *x*, *y* + uma coordenada de textura.

class kivy.graphics.Point

Bases: [kivy.graphics.instructions.VertexInstruction](#)

Uma lista de 2 pontos. Cada ponto é representado como um quadrado com uma largura/altura de 2 vezes o: attr:*pointsize*.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2 ...), onde cada par de coordenadas especifica o centro de um novo ponto.

pointsize: float, e o padrão é 1O tamanho do ponto, medido a partir do centro para a borda. Um valor de 1,0 significa que o tamanho real será de 2.0 x 2.0.

Aviso: A partir da versão 1.0.7, a instrução de vértice tem um limite de 65535 vértices (índices de vértice para ser exato). 2 entradas na lista (x, y) serão convertidas em 4 vértices. Assim, o limite dentro da classe Point() será de $2^{15}-2$.

add_point()

Adicione um ponto à lista atua **points**.

Se pretender adicionar vários pontos, prefira utilizar este método em vez de reatribuir uma nova lista de **points**. A atribuição de uma nova lista de **points** irá recalcular e recarregará todo o buffer para a GPU. Se usares *add_point*, o mesmo só enviará as alterações.

points

Propriedade para obter/configurar o ponto central na lista de pontos. Cada par de coordenadas especifica o centro de um novo ponto.

pointsize

Propriedade para obter/configurar o tamanho do ponto. O tamanho é medido a partir do centro para a borda, então o valor igual a 1.0 significa que o tamanho real será 2.0 x 2.0.

class kivy.graphics.PopMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Coloca a matriz da pilha de matrizes do contexto na vista do modelo. (Pop the matrix from the context's matrix stack onto the model view.)

stack

Nome da pilha de matriz a ser usada. Pode ser 'modelview_mat' ou 'projection_mat'.

Novo na versão 1.6.0.

class kivy.graphics.PushMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Empurra a matriz para a pilha de matrizes do contexto.

stack

Nome da pilha de matriz a ser usada. Pode ser 'modelview_mat' ou 'projection_mat'.

Novo na versão 1.6.0.

class kivy.graphics.Quad

Bases: *kivy.graphics.instructions.VertexInstruction*

Um quad 2D.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2, x3, y3, x4, y4).

points

Propriedade para obter/configurar ponto no quad.

class kivy.graphics.Rectangle

Bases: *kivy.graphics.instructions.VertexInstruction*

Um retângulo 2D.

Parameters

pos: listPosição do retângulo, no formato (x, y).

size: listTamanho do retângulo, no formato (largura, altura);

pos

Propriedade para obter/configurar a posição do retângulo.

size

Propriedade para obter/configurar o tamanho do retângulo.

class kivy.graphics.RenderContext

Bases: *kivy.graphics.instructions.Canvas*

O contexto do Render armazena todas as informações necessárias para o desenho, ou seja:

- O vértice Shader
- O fragmento Shader
- A textura padrão
- A pilha de estado (cor, textura, matriz ...)

shader

Retorna o Shader anexado ao contexto de renderização.

use_parent_modelview

Se True, a matriz de ModelView pai será usada.

Novo na versão 1.7.0.

Antes:

```
rc['modelview_mat'] = Window.render_context['modelview_mat']
```

Agora:

```
rc = RenderContext(use_parent_modelview=True)
```

use_parent_projection

Se True, a matriz de projeção pai será usada.

Novo na versão 1.7.0.

Antes:

```
rc['projection_mat'] = Window.render_context['projection_mat'  
    ↵']
```

Agora:

```
rc = RenderContext(use_parent_projection=True)
```

class kivy.graphics.Rotate

Bases: *kivy.graphics.context_instructions.Transform*

Gire o espaço de coordenadas aplicando uma transformação de rotação na matriz modelview. Você pode definir as propriedades das instruções posteriormente com, por exemplo:

```
rot.angle = 90  
rot.axis = (0, 0, 1)
```

angle

Propriedade para obter/configurar o ângulo da rotação.

axis

Propriedade para obter/configurar o eixo da rotação.

O formato do eixo é (x, y, z).

origin

Origem da rotação.

Novo na versão 1.7.0.

O formato da origem pode ser (x, y) ou (x, y, z).

set()

Defina o ângulo do eixo de rotação.

```
>>> rotationobject.set(90, 0, 0, 1)
```

Obsoleto desde a versão 1.7.0: O método set() não usa a nova propriedade **origin**.

class kivy.graphics.Scale

Bases: kivy.graphics.context_instructions.Transform

Instrução para criar uma transformação de escala não uniforme.

Cria usando um ou três argumentos:

```
Scale(s)      # scale all three axes the same  
Scale(x, y, z)  # scale the axes independently
```

Obsoleto desde a versão 1.6.0: Propriedade de escala simples desprezada em favor de fatores escalonados independentes de eixo x, y, z, xyz.

origin

Origem da escala.

Novo na versão 1.9.0.

O formato da origem pode ser (x, y) ou (x, y, z).

scale

Propriedade paga obter/configurar a escala.

Obsoleto desde a versão 1.6.0: Desprezado em favor de propriedades de escala de eixo x, y, z, xyz, etc.

x

Propriedade paga obter/configurar a escala sobre o eixo X.

Alterado na versão 1.6.0.

xyz

3 vetores de escala de tupla em 3D no eixo x, y e z.

Alterado na versão 1.6.0.

y

Propriedade paga obter/configurar a escala sobre o eixo y.

Alterado na versão 1.6.0.

z

Propriedade paga obter/configurar a escala sobre o eixo Z.

Alterado na versão 1.6.0.

class kivy.graphics.StencilPop

Bases: *kivy.graphics.instructions.Instruction*

Pop a pilha de Stencil. Consulte a documentação do módulo para obter mais informações.

class kivy.graphics.StencilPush

Bases: *kivy.graphics.instructions.Instruction*

Empurre a pilha de Stencil. Consulte a documentação do módulo para obter mais informações.

class kivy.graphics.StencilUse

Bases: *kivy.graphics.instructions.Instruction*

Use o atual buffer *Stencil* como uma máscara. Verifique a documentação do módulo para obter mais informações.

func_op

Determina a operação de Stencil que sera usada por glStencilFunc(). Pode ser um das seguintes opções ‘never’, ‘less’, ‘equal’, ‘lequal’, ‘greater’, ‘notequal’, ‘gequal’ ou ‘always’.

Por padrão, o operador é definido como ‘equal’.

Novo na versão 1.5.0.

class kivy.graphics.StencilUnUse

Bases: *kivy.graphics.instructions.Instruction*

Use buffer *Stencil* atual para desmontar a máscara.

class kivy.graphics.Translate

Bases: *kivy.graphics.context_instructions.Transform*

Instrução para criar uma tradução da coordenada espacial da view de modelo

Construa por:

```

Translate(x, y)           # translate in just the two axes
Translate(x, y, z)        # translate in all three axes

```

x

Propriedade paga obter/configurar a translação sobre o eixo X.

xy

2 tupla com vetor de translação em 2D para o eixo x e y.

xyz

3 vetor de translação da tupla em 3D no eixo x, y, e z.

y

Propriedade paga obter/configurar a translação sobre o eixo Y.

z

Propriedade paga obter/configurar a translação sobre o eixo Z.

class kivy.graphics.Triangle

Bases: *kivy.graphics.instructions.VertexInstruction*

Um triângulo 2D.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2, x3, y3).

points

Propriedade para obtenção/definição de pontos no triângulo.

class kivy.graphics.VertexInstruction

Bases: *kivy.graphics.instructions.Instruction*

A classe VertexInstruction é a base para todas as instruções gráficas que têm uma representação visual direta no Canvas, como Retângulos, Triângulos, Linhas, Elipses e assim por diante.

source

Esta propriedade representa o nome do arquivo pra carregar a textura desde. Se quiseres usar uma imagem como fonte, faça da seguinte forma:

```

with self.canvas:
    Rectangle(source='mylogo.png', pos=self.pos, size=self.
              size)

```

Aqui o equivalente com a linguagem Kivy:

```

<MyWidget>:
    canvas:
        Rectangle:
            source: 'mylogo.png'
            pos: self.pos
            size: self.size

```

Nota: O nome do arquivo será pesquisado usando a função `kivy.resources.resource_find()`.

tex_coords

Esta propriedade representa as coordenadas de textura usadas para desenhar a instrução de vértice. O valor deve ser uma lista de 8 valores.

Uma coordenada de textura tem uma posição (u, v) e um tamanho (w, h). O tamanho pode ser negativo e representaria a textura “invertida”. Por padrão, os *tex_coords* são:

```
[u, v, u + w, v, u + w, v + h, u, v + h]
```

Você pode passar suas próprias coordenadas da textura se quiseres conseguir efeitos extravagantes.

Aviso: Os valores padrão mencionados acima podem ser negativos. Dependendo da imagem e dos provedores de Labels, as coordenadas são invertidas verticalmente devido à ordem na qual a imagem é armazenada internamente. Em vez de passar os dados de imagem, estamos apenas a lançando as coordenadas de textura para que seja mais rápido.

texture

Propriedade que representa a textura usada para desenhar esta Instrução. Podes definir uma nova textura como esta:

```
from kivy.core.image import Image

texture = Image('logo.png').texture
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=self.size)
```

Geralmente, você usará o atributo `source` em vez da textura.

class kivy.graphics.ClearColor

Bases: `kivy.graphics.instructions.Instruction`

Instruções Gráficas ClearColor

Novo na versão 1.3.0.

Define o ClearColor usado para limpar o buffer com a função glClear ou as instruções gráficas `ClearBuffers`.

a

Componente Alfa, entre 0 e 1.

b

Componente Azul, entre 0 e 1.

g

Componente Verde, entre 0 e 1.

r

Componente Vermelho, entre 0 e 1.

rgb

Cor RGB, uma lista com 3 valores no range entre 0-1 onde o Alfa será 1.

rgba

Cor RGBA usada para o ClearColor, uma lista com 4 valores no intervalo entre 0-1.

class kivy.graphics.ClearBuffers

Bases: *kivy.graphics.instructions.Instruction*

InSTRUÇÕES Gráficas ClearBuffer

Novo na versão 1.3.0.

Limpe os buffers especificados pela propriedade de máscara de buffer de instruções. Por padrão, apenas o buffer de coloc é limpo/desmarcado.

clear_color

Se *True*, a cor do buffer será limpa.

clear_depth

Se *True*, o buffer de profundidade será apagado/límpio.

clear_stencil

Se *True*, o buffer do Stencil será limpo/removido.

class kivy.graphics.PushState

Bases: *kivy.graphics.instructions.ContextInstruction*

InSTRUÇÃO que empurram estados/uniformes arbitrários para a pilha de estado de contexto.

Novo na versão 1.6.0.

class kivy.graphics.ChangeState

Bases: *kivy.graphics.instructions.ContextInstruction*

InSTRUÇÃO que altera os valores de estados / uniformes arbitrários no contexto de renderização atual.

Novo na versão 1.6.0.

class kivy.graphics.PopState

Bases: *kivy.graphics.instructions.ContextInstruction*

InSTRUÇÃO que exibe estados/uniformes arbitrários fora da pilha de estado de contexto.

Novo na versão 1.6.0.

class kivy.graphics.ApplyContextMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Pré-multiplique a matriz no topo da pilha especificada por *target_stack* pela matriz no topo de 'source_stack'

Novo na versão 1.6.0.

source_stack

Nome da pilha de matriz a ser usada como fonte. Pode ser 'modelview_mat' ou 'projection_mat'.

Novo na versão 1.6.0.

target_stack

Nome da pilha de matriz a ser usada como destino. Pode ser 'modelview_mat' ou 'projection_mat'.

Novo na versão 1.6.0.

class kivy.graphics.UpdateNormalMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Atualize a matriz normal 'normal_mat' com base na matriz atual de Model-View. Isso irá calcular o uniformemente 'normal_mat' como *inverse(transpose(mat3(mvm)))*

Novo na versão 1.6.0.

class kivy.graphics.LoadIdentity

Bases: *kivy.graphics.instructions.ContextInstruction*

Carregar a matriz de identidade na pilha de matriz especificada pela propriedade de pilha da instruções (default='modelview_mat')

Novo na versão 1.6.0.

stack

Nome da pilha de matriz a ser usada. Pode ser 'modelview_mat' ou 'projection_mat'.

4.7.3 Canvas

A classe **Canvas** é o objeto raiz usado para desenhar com a classe: **Widget**. Para mais informações sobre o uso da 'Canvas', leia a documentação relacionada a ela.

class kivy.graphics.instructions.Instruction

Bases: *kivy.event.ObjectWithUid*

Representa a menor instrução disponível. Esta classe é apenas para uso interno, não use-a diretamente.

proxy_ref

Retornar uma referência de proxy para a instrução, ou seja, sem criar uma referência do widget. Veja [weakref.proxy](#) para obter maiores informações.

Novo na versão 1.7.2.

class kivy.graphics.instructions.InstructionGroup

Bases: [kivy.graphics.instructions.Instruction](#)

Grupo de [Instructions](#). Permite a adição e remoção de instruções gráficas. Pode ser usado diretamente como segue:

```
blue = InstructionGroup()
blue.add(Color(0, 0, 1, 0.2))
blue.add(Rectangle(pos=self.pos, size=(100, 100)))

green = InstructionGroup()
green.add(Color(0, 1, 0, 0.4))
green.add(Rectangle(pos=(100, 100), size=(100, 100)))

# Here, self should be a Widget or subclass
[self.canvas.add(group) for group in [blue, green]]
```

add()

Adiciona uma nova instrução [Instruction](#) para a nossa lista.

clear()

Remove todas as [Instructions](#).

get_group()

Retorna um iterável para todos as [Instructions](#) com um nome de grupo específico.

insert()

Insira uma nova [Instruction](#) em nossa lista no índice.

remove()

Remove uma existente [Instruction](#) da nossa lista.

remove_group()

Remove todas as [Instructions](#) com um nome de grupo específico.

class kivy.graphics.instructions.ContextInstruction

Bases: [kivy.graphics.instructions.Instruction](#)

A classe ContextInstruction é a base para a criação de instruções que não têm uma representação visual direta, mas em vez disso, modifica o atual estado de Canvas, por exemplo, vincula a textura, configura os parâmetros de cor, manipula a matriz e assim por diante.

class kivy.graphics.instructions.VertexInstruction

Bases: [kivy.graphics.instructions.Instruction](#)

A classe VertexInstruction é a base para todas as instruções gráficas que têm uma representação visual direta no Canvas, como Retângulos, Triângulos, Linhas, Elipses e assim por diante.

source

Esta propriedade representa o nome do arquivo pra carregar a textura desde. Se quiseres usar uma imagem como fonte, faça da seguinte forma:

```
with self.canvas:  
    Rectangle(source='mylogo.png', pos=self.pos, size=self.  
              size)
```

Aqui o equivalente com a linguagem Kivy:

```
<MyWidget>:  
    canvas:  
        Rectangle:  
            source: 'mylogo.png'  
            pos: self.pos  
            size: self.size
```

Nota: O nome do arquivo será pesquisado usando a função `kivy.resources.resource_find()`.

tex_coords

Esta propriedade representa as coordenadas de textura usadas para desenhar a instrução de vértice. O valor deve ser uma lista de 8 valores.

Uma coordenada de textura tem uma posição (u, v) e um tamanho (w, h). O tamanho pode ser negativo e representaria a textura “invertida”. Por padrão, os *tex_coords* são:

```
[u, v, u + w, v, u + w, v + h, u, v + h]
```

Você pode passar suas próprias coordenadas da textura se quiseres conseguir efeitos extravagantes.

Aviso: Os valores padrão mencionados acima podem ser negativos. Dependendo da imagem e dos provedores de Labels, as coordenadas são invertidas verticalmente devido à ordem na qual a imagem é armazenada internamente. Em vez de passar os dados de imagem, estamos apenas a lançando as coordenadas de textura para que seja mais rápido.

texture

Propriedade que representa a textura usada para desenhar esta Instrução. Podes definir uma nova textura como esta:

```
from kivy.core.image import Image

texture = Image('logo.png').texture
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=self.size)
```

Geralmente, você usará o atributo *source* em vez da textura.

class kivy.graphics.instructions.Canvas
Bases: *kivy.graphics.instructions.CanvasBase*

A importante classe *Canvas*. Use esta classe para adicionar gráficos ou instruções de contexto que desejas usar para desenhar.

Nota: O Canvas suporta a instrução *with* do Python e sua semântica de Entrada & Saída.

Uso do Canvas sem a instrução *with*:

```
self.canvas.add(Color(1., 1., 0))
self.canvas.add(Rectangle(size=(50, 50)))
```

Uso do Canvas sem a instrução *with* do Python:

```
with self.canvas:
    Color(1., 1., 0)
    Rectangle(size=(50, 50))
```

after

Propriedade para obter o ‘próximo’ grupo.

ask_update()

Informe o Canvas que gostarias de atualizar no próximo Frame. Isso é útil quando precisas desencadear um redesenho devido a algum valor alterado, por exemplo.

before

Propriedade para obter o grupos ‘anterior’.q

clear()

Limpa cada *Instruction* na tela, deixando-a limpa.

draw()

Aplica a instrução à nossa janela.

has_after

Propriedade para ver se o grupo *after* já foi criado.

Novo na versão 1.7.0.

has_before

Propriedade para ver se o grupo *before* já foi criado.

Novo na versão 1.7.0.

opacity

Propriedade para obter/definir o valor de opacidade da tela.

Novo na versão 1.4.1.

O atributo *opacity* controla a opacidade da tela e a de seus filhos. Tenha cuidado, é um atributo cumulativo: o valor é multiplicado pela opacidade global atual e o resultado é aplicado à cor do contexto atual.

Por exemplo: se o pai tiver uma opacidade de 0,5 e um filho tiver uma opacidade de 0,2, a opacidade final do filho será de $0,5 * 0,2 = 0,1$.

Em seguida, a opacidade é aplicada no *Shader* como por exemplo:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

class kivy.graphics.instructions.CanvasBase

Bases: *kivy.graphics.instructions.InstructionGroup*

O *CanvasBase* fornece os métodos do gerenciador de contexto para *Canvas*.

class kivy.graphics.instructions.RenderContext

Bases: *kivy.graphics.instructions.Canvas*

O contexto do Render armazena todas as informações necessárias para o desenho, ou seja:

- O vértice Shader
- O fragmento Shader
- A textura padrão
- A pilha de estado (cor, textura, matriz ...)

shader

Retorna o Shader anexado ao contexto de renderização.

use_parent_modelview

Se True, a matriz de ModelView pai será usada.

Novo na versão 1.7.0.

Antes:

```
rc['modelview_mat'] = Window.render_context['modelview_mat']
```

Agora:

```
rc = RenderContext(use_parent_modelview=True)
```

use_parent_projection

Se True, a matriz de projeção pai será usada.

Novo na versão 1.7.0.

Antes:

```
rc['projection_mat'] = Window.render_context['projection_mat'  
    ↵']
```

Agora:

```
rc = RenderContext(use_parent_projection=True)
```

class kivy.graphics.instructions.Callback

Bases: *kivy.graphics.instructions.Instruction*

Novo na versão 1.0.4.

Um callback é uma instrução que será chamada quando a operação de desenho for executada. Ao adicionar instruções ao Canvas, poderás fazer isso:

```
with self.canvas:  
    Color(1, 1, 1)  
    Rectangle(pos=self.pos, size=self.size)  
    Callback(self.my_callback)
```

A definição do callback deve ser:

```
def my_callback(self, instr):  
    print('I have been called!')
```

Aviso: Observe que, se você executar muitas e/ou chamadas custosas por callbacks, podes potencialmente diminuir significativamente o desempenho de renderização.

A atualização da sua tela não ocorrerá até que algo novo aconteça. Desde o seu callback, podes pedir por uma atualização:

```
with self.canvas:  
    self.cb = Callback(self.my_callback)  
    # then later in the code  
    self.cb.ask_update()
```

Se utilizar a classe *Callback* para invocar métodos de renderização de outro toolkit, terás problemas com o contexto OpenGL. O estado do OpenGL pode ter

sido manipulado pelo outro conjunto de ferramentas e assim que o fluxo do programa retornar para o Kivy, ele será quebrado. Poderá haver quebras, falhas, buracos negros podem ocorrer, etc. Para evitar isso, podes ativar a opção ***reset_context***. Ela redefinirá o estado do contexto do OpenGL para tornar a renderização do Kivy correta após a chamada para o callback.

Aviso: O ***reset_context*** não é um reset completo do *OpenGL*. Se tiveres problemas com isso, entre em contato conosco.

ask_update()

Informe o Canvas-pai que gostarias de atualizar no próximo Frame. Isso é útil quando precisares disparar um redesenho devido a algum valor alterado, por exemplo.

Novo na versão 1.0.4.

reset_context

Defina isso como *True* se quiseres redefinir o contexto *OpenGL* do Kivy após o callback ter sido invocado.

4.7.4 CGL: standard C interface for OpenGL

Kivy uses OpenGL and therefore requires a backend that provides it. The backend used is controlled through the **USE_OPENGL_MOCK** and **USE_SDL2** compile-time variables and through the **KIVY_GL_BACKEND** runtime environmental variable.

Currently, OpenGL is used through direct linking (`gl/glew`), `sdl2`, or by mocking it. Setting **USE_OPENGL_MOCK** disables `gl/glew`. Similarly, setting **USE_SDL2** to `0` will disable `sdl2`. Mocking is always available.

At runtime the following backends are available and can be set using **KIVY_GL_BACKEND**:

- **gl** – Available on unix (the default backend). Unavailable when **USE_OPENGL_MOCK=0**. Requires `gl` be installed.
- **glew** – Available on Windows (the default backend). Unavailable when **USE_OPENGL_MOCK=0**. Requires `glew` be installed.
- **sdl2** – Available on Windows/unix (the default when `gl/glew` is disabled). Unavailable when **USE_SDL2=0**. Requires `kivy.deps.sdl2` be installed.
- **angle_sdl2** – Available on Windows with Python 3.5+. Unavailable when **USE_SDL2=0**. Requires `kivy.deps.sdl2` and `kivy.deps.angle` be installed.
- **mock** – Always available. Doesn't actually do anything.

Additionally, the following environmental runtime variables control the graphics system:

- **KIVY_GL_DEBUG** – Logs all gl calls when 1.
- **KIVY_GRAPHICS** – Forces OpenGL ES2 when it is `gles`. OpenGL ES2 is always used on the android, ios, rpi, and mali OSs.

4.7.5 Instrução de Contexto

As instruções de contexto representam elementos não gráficos, tais como:

- Manipulação de matrizes (PushMatrix, PopMatrix, Rotate, Translate, Scale, MatrixInstruction)
- Manipulações de cor (cor)
- Texture bindings (BindTexture)

Alterado na versão 1.0.8: A instrução LineWidth foi removida. Ela não estava funcionando antes e nós realmente não temos nenhuma implementação funcional. Precisamos fazer mais experimentação para acertar. Verifique o erro #207 <<https://github.com/kivy/kivy/issues/207>> _ para obter mais informações.

class kivy.graphics.context_instructions.Color

Bases: [kivy.graphics.instructions.ContextInstruction](#)

Instrução para definir o estado da cor para quaisquer vértices que será desenhado após ele.

Isso representa uma cor entre 0 e 1, mas é aplicado como *multiplicador* à textura de quaisquer instruções de vértice que o seguem em uma tela. Se nenhuma textura for definida, a instrução de vértice assumirá a cor precisa da instrução Color.

Por exemplo, se um retângulo tem uma textura com cor uniforme (0.5, 0.5, 0.5, 1.0) e a cor anterior tem `rgba` = (1, 0.5, 2, 1), a cor atual visível será “(0.5, 0.25, 1.0, 1.0)” uma vez que a instrução Color é aplicada como um multiplicador para cada componente do `rgba`. Neste caso, um componente de Cor fora da faixa 0-1 dá um resultado visível à medida que a intensidade do componente azul é duplicada.

Para declarar um Color em Python, podes fazer:

```
from kivy.graphics import Color

# create red v
c = Color(1, 0, 0)
# create blue color
c = Color(0, 1, 0)
# create blue color with 50% alpha
c = Color(0, 1, 0, .5)
```

```
# using hsv mode
c = Color(0, 1, 1, mode='hsv')
# using hsv mode + alpha
c = Color(0, 1, 1, .2, mode='hsv')
```

Você também pode definir componentes de cores que estarão disponíveis como propriedades, passando-as como argumentos de palavra-chave:

```
c = Color(b=0.5) # sets the blue component only
```

Em kv lang você pode definir as propriedades de cor diretamente:

```
<Rule>:
    canvas:
        # red color
        Color:
            rgb: 1, 0, 0
        # blue color
        Color:
            rgb: 0, 1, 0
        # blue color with 50% alpha
        Color:
            rgba: 0, 1, 0, .5

        # using hsv mode
        Color:
            hsv: 0, 1, 1
        # using hsv mode + alpha
        Color:
            hsv: 0, 1, 1
            a: .5
```

a

Componente Alfa, entre 0 e 1.

b

Componente Azul, entre 0 e 1.

g

Componente Verde, entre 0 e 1.

h

Componente Hue, entre 0 e 1.

hsv

Cor HSV, lista de 3 valores no intervalo entre 0-1, alpha será 1.

r

Componente Vermelho, entre 0 e 1.

rgb

Cor RGB, lista de 3 valores no intervalo 0-1. O alfa será 1.

rgba

Cor RGBA, lista de 4 valores no intervalo de 0-1.

s

Componente de Saturação, entre 0 e 1

v

Componente valor, entre 0 e 1.

class kivy.graphics.context_instructions.BindTexture

Bases: *kivy.graphics.instructions.ContextInstruction*

InSTRUÇÕES Gráficas BindTexture . A instrução *BindTexture* vinculará uma textura e ativará *GL_TEXTURE_2D* para desenho subsequente.

Parameters

texture: TextureEspecifica a textura a ser vinculada ao índice fornecido.

source

Define/Obtém a origem (nome do arquivo) pra carregar a textura.

class kivy.graphics.context_instructions.PushMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Empurra a matriz para a pilha de matrizes do contexto.

stack

Nome da pilha de matriz a ser usada. Pode ser ‘modelview_mat’ ou ‘projection_mat’.

Novo na versão 1.6.0.

class kivy.graphics.context_instructions.PopMatrix

Bases: *kivy.graphics.instructions.ContextInstruction*

Coloca a matriz da pilha de matrizes do contexto na vista do modelo. (Pop the matrix from the context’s matrix stack onto the model view.)

stack

Nome da pilha de matriz a ser usada. Pode ser ‘modelview_mat’ ou ‘projection_mat’.

Novo na versão 1.6.0.

class kivy.graphics.context_instructions.Rotate

Bases: *kivy.graphics.context_instructions.Transform*

Gire o espaço de coordenadas aplicando uma transformação de rotação na matriz modelview. Você pode definir as propriedades das instruções posteriormente com, por exemplo:

```
rot.angle = 90  
rot.axis = (0, 0, 1)
```

angle

Propriedade para obter/configurar o ângulo da rotação.

axis

Propriedade para obter/configurar o eixo da rotação.

O formato do eixo é (x, y, z).

origin

Origem da rotação.

Novo na versão 1.7.0.

O formato da origem pode ser (x, y) ou (x, y, z).

set()

Defina o ângulo do eixo de rotação.

```
>>> rotationobject.set(90, 0, 0, 1)
```

Obsoleto desde a versão 1.7.0: O método set() não usa a nova propriedade **origin**.

class kivy.graphics.context_instructions.Scale

Bases: kivy.graphics.context_instructions.Transform

Instrução para criar uma transformação de escala não uniforme.

Cria usando um ou três argumentos:

```
Scale(s)      # scale all three axes the same  
Scale(x, y, z)  # scale the axes independently
```

Obsoleto desde a versão 1.6.0: Propriedade de escala simples desprezada em favor de fatores escalonados independentes de eixo x, y, z, xyz.

origin

Origem da escala.

Novo na versão 1.9.0.

O formato da origem pode ser (x, y) ou (x, y, z).

scale

Propriedade paga obter/configurar a escala.

Obsoleto desde a versão 1.6.0: Desprezado em favor de propriedades de escala de eixo x, y, z, xyz, etc.

x

Propriedade paga obter/configurar a escala sobre o eixo X.

Alterado na versão 1.6.0.

xyz

3 vetores de escala de tupla em 3D no eixo x, y e z.

Alterado na versão 1.6.0.

y

Propriedade paga obter/configurar a escala sobre o eixo y.

Alterado na versão 1.6.0.

z

Propriedade paga obter/configurar a escala sobre o eixo Z.

Alterado na versão 1.6.0.

class kivy.graphics.context_instructions.Translate

Bases: [kivy.graphics.context_instructions.Transform](#)

Instrução para criar uma tradução da coordenada espacial da view de modelo

Construa por:

```
Translate(x, y)          # translate in just the two axes  
Translate(x, y, z)       # translate in all three axes
```

x

Propriedade paga obter/configurar a translação sobre o eixo X.

xy

2 tupla com vetor de translação em 2D para o eixo x e y.

xyz

3 vetor de translação da tupla em 3D no eixo x, y, e z.

y

Propriedade paga obter/configurar a translação sobre o eixo Y.

z

Propriedade paga obter/configurar a translação sobre o eixo Z.

class kivy.graphics.context_instructions.MatrixInstruction

Bases: [kivy.graphics.instructions.ContextInstruction](#)

Classe base para instrução Matrix no Canvas.

matrix

Matrix propriedade. Matriz do módulo de transformação. Define a matriz usando esta propriedade quando uma alteração é feita é importante porque ele irá notificar o contexto sobre a atualização.

stack

Nome da pilha de matriz a ser usada. Pode ser ‘modelview_mat’ ou ‘projection_mat’.

Novo na versão 1.6.0.

4.7.6 Gerenciador de Contexto

Novo na versão 1.2.0.

Esta classe gerencia o registro de todas as instruções gráficas criadas. Ele tem a capacidade de liberar ou deletar-los.

Você pode ler mais sobre Contextos Gráficos com Kivy no módulo da documentação [Gráficos](#). Isso foi baseado no [OpenGL graphics contexts](#).

class kivy.graphics.context.Context

Bases: `object`

A classe *Context* gerencia grupos de instruções gráficas. Ela também pode ser usado para gerenciar callbacks do observador. Veja [add_reload_observer\(\)](#) e [remove_reload_observer\(\)](#) para maiores informações.

add_reload_observer()

(Interno) Adiciona um callback a ser chamado depois que todo o contexto gráfico foi recarregado. Este é o lugar onde você pode reupload seus dados personalizados para a GPU.

Parameters

callback: func(context) -> return NoneO primeiro parâmetro será o contexto propriamente dito

before: boolean, o padrão é FalseSe *True*, o callback será executado antes de todos os processos de recarga. Use-o se quiser limpar o cache, por exemplo.

Alterado na versão 1.4.0: *before* parâmetro adicionado.

remove_reload_observer()

(Internal) Remove um callback da lista de observadores previamente adicionado por [add_reload_observer\(\)](#).

4.7.7 FrameBuffer

Fbo é como uma janela fora da tela. Você pode ativar o Fbo para renderizar para uma textura e usá-lo como uma textura para outros desenhos.

O Fbo age como [kivy.graphics.instructions.Canvas](#).

Aqui temos um exemplo de uso do FBO para alguns retângulos coloridos:

```
from kivy.graphics import Fbo, Color, Rectangle

class FboTest(Widget):
```

```

def __init__(self, **kwargs):
    super(FboTest, self).__init__(**kwargs)

        # first step is to create the fbo and use the fbo texture_
        ↵on other
        # rectangle

    with self.canvas:
        # create the fbo
        self.fbo = Fbo(size=(256, 256))

            # show our fbo on the widget in different size
            Color(1, 1, 1)
            Rectangle(size=(32, 32), texture=self.fbo.texture)
            Rectangle(pos=(32, 0), size=(64, 64), texture=self.fbo.
        ↵texture)
            Rectangle(pos=(96, 0), size=(128, 128), texture=self.
        ↵fbo.texture)

        # in the second step, you can draw whatever you want on the_
        ↵fbo
    with self.fbo:
        Color(1, 0, 0, .8)
        Rectangle(size=(256, 64))
        Color(0, 1, 0, .8)
        Rectangle(size=(64, 256))

```

Se alterares alguma coisa no objeto `self.fbo`, ele será atualizado automaticamente. A tela onde o FBO é colocado também será atualizada automaticamente.

Recarregando o conteúdo da FBO

Novo na versão 1.2.0.

Se o contexto OpenGL for perdido, então o FBO também será perdido. Você precisa reupload os dados pra ele. Use o `Fbo.add_reload_observer()` para adicionar uma função de recarga que será automaticamente invocada quando necessário:

```

def __init__(self, **kwargs):
    super(...).__init__(**kwargs)
    self.fbo = Fbo(size=(512, 512))
    self.fbo.add_reload_observer(self.populate_fbo)

        # and load the data now.
        self.populate_fbo(self.fbo)

def populate_fbo(self, fbo):

```

```
with fbo:  
    # .. put your Color / Rectangle / ... here
```

Desta forma, podes usar o mesmo método para inicialização e para recarregar. Mas isso compete a você.

class kivy.graphics.Fbo

Bases: [kivy.graphics.instructions.RenderContext](#)

FBO para envolver a extensão OpenGL Framebuffer. O suporte FBO “com” declaração.

Parameters

clear_color: tuple, o padrão é (0, 0, 0, 0) Define a cor padrão para limpar a framebuffer

size: tuple, o padrão é (1024, 1024) Tamanho padrão do FrameBuffer

push_viewport: bool, o padrão é *True* Se *True*, o Viewport do OpenGL será definido para o tamanho do framebuffer, e será automaticamente restaurado quando o framebuffer for liberado.

with_depthbuffer: bool, o padrão é *False* Se *True*, o framebuffer será alocado com um buffer Z.

with_stencilbuffer: bool, o padrão é *False* Novo na versão 1.9.0.

Se *True*, o framebuffer será alocado com um buffer de Stencil.

texture: Texture, o padrão é *None* Se *None*, a textura padrão será criada.

Nota: Usar o **with_stencilbuffer** e **with_depthbuffer** não é suportado no Kivy 1.9.0

add_reload_observer()

Adiciona um callback pra ser chamado depois que o todo o contexto gráfico for recarregado. Este é o lugar onde podes reupload seus dados personalizados na GPU.

Novo na versão 1.2.0.

Parameters

callback: func(context) -> return None O primeiro parâmetro será o contexto propriamente dito

bind()

Vincula o FBO ao contexto OpenGL atual. *Bind* significa que você habilita o

FrameBuffer, e todas as operações de desenho atuarão dentro do Framebuffer, `release()` seja invocado.

As operações de vinculação/liberação são chamadas automaticamente quando você adiciona objetos gráficos a ele. Se quiseres manipular um FrameBuffer você mesmo, podes usa-lo dessa forma:

```
self.fbo = FBO()  
self.fbo.bind()  
# do any drawing command  
self.fbo.release()  
  
# then, your fbo texture is available at  
print(self.fbo.texture)
```

clear_buffer()

Limpa o FrameBuffer com o `clear_color`.

Precisas vincular o FrameBuffer você mesmo antes de invocar este método:

```
fbo.bind()  
fbo.clear_buffer()  
fbo.release()
```

clear_color

Clear color no formato (red, green, blue, alpha).

get_pixel_color()

Obtém a cor do pixel com as coordenadas de janela especificadas em *wx*, *wy*. O resultado será retornado no formato RGBA.

Novo na versão 1.8.0.

pixels

Obtém a textura dos pixels, apenas no formato RGBA, Unsigned Byte. A origem da imagem está na parte inferior esquerda.

Novo na versão 1.7.0.

release()

Libera o FrameBuffer (unbind).

remove_reload_observer()

Remove o callback da lista de observadores, adicionado anteriormente por `add_reload_observer()`.

Novo na versão 1.2.0.

size

Tamanho do FrameBuffer, no formato (width, height).

Se você alterar o tamanho, o conteúdo do FrameBuffer será perdido.

texture

Retorna a textura do FrameBuffer

4.7.8 Instrução GL

Novo na versão 1.3.0.

Limpia o FBO

Para limpar um FBO, podes usar instruções *ClearColor* e *ClearBuffers* como estas no exemplo:

```
self.fbo = Fbo(size=self.size)
with self.fbo:
    ClearColor(0, 0, 0, 0)
    ClearBuffers()
```

class kivy.graphics.gl_instructions.ClearColor

Bases: *kivy.graphics.instructions.Instruction*

Instruções Gráficas ClearColor

Novo na versão 1.3.0.

Define o ClearColor usado para limpar o buffer com a função glClear ou as instruções gráficas *ClearBuffers*.

a

Componente Alfa, entre 0 e 1.

b

Componente Azul, entre 0 e 1.

g

Componente Verde, entre 0 e 1.

r

Componente Vermelho, entre 0 e 1.

rgb

Cor RGB, uma lista com 3 valores no range entre 0-1 onde o Alfa será 1.

rgba

Cor RGBA usada para o ClearColor, uma lista com 4 valores no intervalo entre 0-1.

class kivy.graphics.gl_instructions.ClearBuffers

Bases: *kivy.graphics.instructions.Instruction*

Instruções Gráficas ClearBuffer

Novo na versão 1.3.0.

Limpe os buffers especificados pela propriedade de máscara de buffer de instruções. Por padrão, apenas o buffer de coloc é limpo/desmarcado.

clear_color

Se *True*, a cor do buffer será limpa.

clear_depth

Se *True*, o buffer de profundidade será apagado/límpio.

clear_stencil

Se *True*, o buffer do Stencil será limpo/removido.

4.7.9 Compilador Gráfico

Antes de renderizar um *InstructionGroup*, nós compilamos o grupo de instrução em ordem para reduzir o número de instrução executadas no momento da renderização.

Reduzindo as instruções de contexto

Imagine que você tem um esquema como este:

```
Color(1, 1, 1)
Rectangle(source='button.png', pos=(0, 0), size=(20, 20))
Color(1, 1, 1)
Rectangle(source='button.png', pos=(10, 10), size=(20, 20))
Color(1, 1, 1)
Rectangle(source='button.png', pos=(10, 20), size=(20, 20))
```

As reais instruções vistas pela tela gráfica seriam:

```
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
Rectangle: push vertices (x1, y1...) to vbo & draw
```

Somente a primeira *Color* e *BindTexture* possuem utilidade e realmente alteram o contexto. Podemos reduzi-las a:

```
Color: change 'color' context to 1, 1, 1
BindTexture: change 'texture0' to `button.png texture`
```

```
Rectangle: push vertices (x1, y1...) to vbo & draw
Rectangle: push vertices (x1, y1...) to vbo & draw
Rectangle: push vertices (x1, y1...) to vbo & draw
```

Isto é o que o compilador faz em inicialmente, sinaliza todas instruções não utilizadas com a flag `GL_IGNORE`. Quando alguma cor for alterada, todo o grupo `InstructionGroup` será recompilado e uma cor anteriormente não utilizada pode ser numa próxima compilação.

Observação para qualquer colaborador Kivy/desenvolvedor interno:

- Todas as instruções do contexto são checadas para averiguar se essas alteram algo no que está em cache.
- Temos que garantir que uma instrução de contexto é necessária para o nosso Canvas atual.
- Temos que assegurar que não dependemos de nenhum outro Canvas.
- Tudo será redefinido caso um dos nossos filhos for de outro grupo de instruções, isso porque não sabemos se ele teve ou não alguma alteração.

4.7.10 OpenGL

Este módulo é um Wrapper em Python para os comandos OpenGL.

Aviso: Nem todos os comandos do OpenGL estão implementados, isso porque, foi feito um bind em C para melhorar o desempenho, utilize sempre a API gráfica do Kivy. Utilizar os comandos do OpenGL diretamente, pode alterar o contexto do OpenGL e causar inconsistência entre o estado em que o Kivy mantém e o estado do OpenGL.

`kivy.graphics.opengl.glActiveTexture()`

Veja: [glActiveTexture\(\) on Kronos website](#)

`kivy.graphics.opengl.glAttachShader()`

Veja: [glAttachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindAttribLocation()`

Veja: [glBindAttribLocation\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindBuffer()`

Veja: [glBindBuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindFramebuffer()`

Veja: [glBindFramebuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindRenderbuffer()`

Veja: [glBindRenderbuffer\(\) on Kronos website](#)

`kivy.graphics.opengl.glBindTexture()`

Veja: [glBindTexture\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendColor()`

Veja: [glBlendColor\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendEquation()`

Veja: [glBlendEquation\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendEquationSeparate()`

Veja: [glBlendEquationSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendFunc()`

Veja: [glBlendFunc\(\)](#) on Kronos website

`kivy.graphics.opengl.glBlendFuncSeparate()`

Veja: [glBlendFuncSeparate\(\)](#) on Kronos website

`kivy.graphics.opengl.glBufferData()`

Veja: [glBufferData\(\)](#) on Kronos website

`kivy.graphics.opengl.glBufferSubData()`

Veja: [glBufferSubData\(\)](#) on Kronos website

`kivy.graphics.opengl.glCheckFramebufferStatus()`

Veja: [glCheckFramebufferStatus\(\)](#) on Kronos website

`kivy.graphics.opengl.glClear()`

Veja: [glClear\(\)](#) on Kronos website

`kivy.graphics.opengl.glClearColor()`

Veja: [glClearColor\(\)](#) on Kronos website

`kivy.graphics.opengl.glClearStencil()`

Veja: [glClearStencil\(\)](#) on Kronos website

`kivy.graphics.opengl.glColorMask()`

Veja: [glColorMask\(\)](#) on Kronos website

`kivy.graphics.opengl.glCompileShader()`

Veja: [glCompileShader\(\)](#) on Kronos website

`kivy.graphics.opengl.glCompressedTexImage2D()`

Veja: [glCompressedTexImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glCompressedTexSubImage2D()`

Veja: [glCompressedTexSubImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glCopyTexImage2D()`

Veja: [glCopyTexImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glCopyTexSubImage2D()`

Veja: [glCopyTexSubImage2D\(\)](#) on Kronos website

`kivy.graphics.opengl.glCreateProgram()`

Veja: [glCreateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glCreateShader()`

Veja: [glCreateShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glCullFace()`

Veja: [glCullFace\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteBuffers()`

Veja: [glDeleteBuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteFramebuffers()`

Veja: [glDeleteFramebuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteProgram()`

Veja: [glDeleteProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteRenderbuffers()`

Veja: [glDeleteRenderbuffers\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteShader()`

Veja: [glDeleteShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDeleteTextures()`

Veja: [glDeleteTextures\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthFunc()`

Veja: [glDepthFunc\(\) on Kronos website](#)

`kivy.graphics.opengl.glDepthMask()`

Veja: [glDepthMask\(\) on Kronos website](#)

`kivy.graphics.opengl.glDetachShader()`

Veja: [glDetachShader\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisable()`

Veja: [glDisable\(\) on Kronos website](#)

`kivy.graphics.opengl.glDisableVertexAttribArray()`

Veja: [glDisableVertexAttribArray\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawArrays()`

Veja: [glDrawArrays\(\) on Kronos website](#)

`kivy.graphics.opengl.glDrawElements()`

Veja: [glDrawElements\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnable()`

Veja: [glEnable\(\) on Kronos website](#)

`kivy.graphics.opengl.glEnableVertexAttribArray()`

Veja: [glEnableVertexAttribArray\(\) on Kronos website](#)

kivy.graphics.opengl.glFinish()
Veja: [glFinish\(\) on Kronos website](#)

kivy.graphics.opengl.glFlush()
Veja: [glFlush\(\) on Kronos website](#)

kivy.graphics.opengl.glFramebufferRenderbuffer()
Veja: [glFramebufferRenderbuffer\(\) on Kronos website](#)

kivy.graphics.opengl.glFramebufferTexture2D()
Veja: [glFramebufferTexture2D\(\) on Kronos website](#)

kivy.graphics.opengl.glFrontFace()
Veja: [glFrontFace\(\) on Kronos website](#)

kivy.graphics.opengl.glGenBuffers()
Veja: [glGenBuffers\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGenFramebuffers()
Veja: [glGenFramebuffers\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGenRenderbuffers()
Veja: [glGenRenderbuffers\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGenTextures()
Veja: [glGenTextures\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGenerateMipmap()
Veja: [glGenerateMipmap\(\) on Kronos website](#)

kivy.graphics.opengl.glGetActiveAttrib()
Veja: [glGetActiveAttrib\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetActiveUniform()
Veja: [glGetActiveUniform\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetAttachedShaders()
Veja: [glGetAttachedShaders\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetAttribLocation()
Veja: [glGetAttribLocation\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetBooleanv()

Veja: [glGetBooleanv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetBufferParameteriv()

Veja: [glGetBufferParameteriv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetError()

Veja: [glGetError\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetFloatv()

Veja: [glGetFloatv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetFramebufferAttachmentParameteriv()

Veja: [glGetFramebufferAttachmentParameteriv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetIntegerv()

Veja: [glGetIntegerv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor (s) será o resultado da chamada.

kivy.graphics.opengl.glGetProgramInfoLog()

Veja: [glGetProgramInfoLog\(\) on Kronos website](#)

Ao contrário da especificação do C, o código fonte será retornado como uma String.

kivy.graphics.opengl.glGetProgramiv()

Veja: [glGetProgramiv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor (s) será o resultado da chamada.

kivy.graphics.opengl.glGetRenderbufferParameteriv()

Veja: [glGetRenderbufferParameteriv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetShaderInfoLog()

Veja: [glGetShaderInfoLog\(\) on Kronos website](#)

Ao contrário da especificação do C, o código fonte será retornado como uma String.

kivy.graphics.opengl.glGetShaderPrecisionFormat()

Veja: [glGetShaderPrecisionFormat\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glGetShaderSource()

Veja: [glGetShaderSource\(\) on Kronos website](#)

Ao contrário da especificação do C, o código fonte será retornado como uma String.

kivy.graphics.opengl.glGetShaderiv()

Veja: [glGetShaderiv\(\) on Kronos website](#)

Ao contrário da especificação C, o valor será o resultado da chamada.

kivy.graphics.opengl.glGetString()

Veja: [glGetString\(\) on Kronos website](#)

Ao contrário da especificação do C, o valor será retornado como uma String.

kivy.graphics.opengl.glGetTexParameterfv()

Veja: [glGetTexParameterfv\(\) on Kronos website](#)

kivy.graphics.opengl.glGetTexParameteriv()

Veja: [glGetTexParameteriv\(\) on Kronos website](#)

kivy.graphics.opengl.glGetUniformLocation()

Veja: [glGetUniformLocation\(\) on Kronos website](#)

kivy.graphics.opengl.glGetUniformfv()

Veja: [glGetUniformfv\(\) on Kronos website](#)

kivy.graphics.opengl.glGetUniformiv()

Veja: [glGetUniformiv\(\) on Kronos website](#)

kivy.graphics.opengl.glGetVertexAttribPointerv()

Veja: [glGetVertexAttribPointerv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glGetVertexAttribfv()

Veja: [glGetVertexAttribfv\(\) on Kronos website](#)

kivy.graphics.opengl.glGetVertexAttribiv()

Veja: [glGetVertexAttribiv\(\) on Kronos website](#)

kivy.graphics.opengl.glHint()

Veja: [glHint\(\) on Kronos website](#)

kivy.graphics.opengl.glIsBuffer()

Veja: [glIsBuffer\(\) on Kronos website](#)

kivy.graphics.opengl.glIsEnabled()
Veja: [glIsEnabled\(\) on Kronos website](#)

kivy.graphics.opengl.glIsFramebuffer()
Veja: [glIsFramebuffer\(\) on Kronos website](#)

kivy.graphics.opengl.glIsProgram()
Veja: [glIsProgram\(\) on Kronos website](#)

kivy.graphics.opengl.glIsRenderbuffer()
Veja: [glIsRenderbuffer\(\) on Kronos website](#)

kivy.graphics.opengl.glIsShader()
Veja: [glIsShader\(\) on Kronos website](#)

kivy.graphics.opengl.glIsTexture()
Veja: [glIsTexture\(\) on Kronos website](#)

kivy.graphics.opengl.glLineWidth()
Veja: [glLineWidth\(\) on Kronos website](#)

kivy.graphics.opengl.glLinkProgram()
Veja: [glLinkProgram\(\) on Kronos website](#)

kivy.graphics.opengl.glPixelStorei()
Veja: [glPixelStorei\(\) on Kronos website](#)

kivy.graphics.opengl.glPolygonOffset()
Veja: [glPolygonOffset\(\) on Kronos website](#)

kivy.graphics.opengl.glReadPixels()
Veja: [glReadPixels\(\) on Kronos website](#)

Nós suportamos somente *GL_RGB/GL_RGBA* como um formato e *GL_UNSIGNED_BYTE* como um tipo.

kivy.graphics.opengl.glReleaseShaderCompiler()
Veja: [glReleaseShaderCompiler\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glRenderbufferStorage()
Veja: [glRenderbufferStorage\(\) on Kronos website](#)

kivy.graphics.opengl.glSampleCoverage()
Veja: [glSampleCoverage\(\) on Kronos website](#)

kivy.graphics.opengl.glScissor()
Veja: [glScissor\(\) on Kronos website](#)

kivy.graphics.opengl.glShaderBinary()
Veja: [glShaderBinary\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glShaderSource()

Veja: [glShaderSource\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilFunc()

Veja: [glStencilFunc\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilFuncSeparate()

Veja: [glStencilFuncSeparate\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilMask()

Veja: [glStencilMask\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilMaskSeparate()

Veja: [glStencilMaskSeparate\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilOp()

Veja: [glStencilOp\(\)](#) on Kronos website

kivy.graphics.opengl.glStencilOpSeparate()

Veja: [glStencilOpSeparate\(\)](#) on Kronos website

kivy.graphics.opengl.glTexImage2D()

Veja: [glTexImage2D\(\)](#) on Kronos website

kivy.graphics.opengl.glTexParameterf()

Veja: [glTexParameterf\(\)](#) on Kronos website

kivy.graphics.opengl.glTexParameterfv()

Veja: [glTexParameterfv\(\)](#) on Kronos website

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glTexParameterI()

Veja: [glTexParameterI\(\)](#) on Kronos website

kivy.graphics.opengl.glTexParameteriv()

Veja: [glTexParameteriv\(\)](#) on Kronos website

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glTexSubImage2D()

Veja: [glTexSubImage2D\(\)](#) on Kronos website

kivy.graphics.opengl.glUniform1f()

Veja: [glUniform1f\(\)](#) on Kronos website

`kivy.graphics.opengl.glUniform1fv()`

Veja: [glUniform1fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glUniform1i()`

Veja: [glUniform1i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform1iv()`

Veja: [glUniform1iv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glUniform2f()`

Veja: [glUniform2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2fv()`

Veja: [glUniform2fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glUniform2i()`

Veja: [glUniform2i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform2iv()`

Veja: [glUniform2iv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glUniform3f()`

Veja: [glUniform3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3fv()`

Veja: [glUniform3fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glUniform3i()`

Veja: [glUniform3i\(\) on Kronos website](#)

`kivy.graphics.opengl.glUniform3iv()`

Veja: [glUniform3iv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniform4f()

Veja: [glUniform4f\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniform4fv()

Veja: [glUniform4fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniform4i()

Veja: [glUniform4i\(\) on Kronos website](#)

kivy.graphics.opengl.glUniform4iv()

Veja: [glUniform4iv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniformMatrix2fv()

Veja: [glUniformMatrix2fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniformMatrix3fv()

Veja: [glUniformMatrix3fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

kivy.graphics.opengl.glUniformMatrix4fv()

Veja: [glUniformMatrix4fv\(\) on Kronos website](#)

kivy.graphics.opengl.glUseProgram()

Veja: [glUseProgram\(\) on Kronos website](#)

kivy.graphics.opengl.glValidateProgram()

Veja: [glValidateProgram\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1f()`

Veja: [glVertexAttrib1f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib1fv()`

Veja: [glVertexAttrib1fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glVertexAttrib2f()`

Veja: [glVertexAttrib2f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib2fv()`

Veja: [glVertexAttrib2fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glVertexAttrib3f()`

Veja: [glVertexAttrib3f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib3fv()`

Veja: [glVertexAttrib3fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glVertexAttrib4f()`

Veja: [glVertexAttrib4f\(\) on Kronos website](#)

`kivy.graphics.opengl.glVertexAttrib4fv()`

Veja: [glVertexAttrib4fv\(\) on Kronos website](#)

Aviso: Ainda não está implementado.

`kivy.graphics.opengl.glVertexAttribPointer()`

Veja: [glVertexAttribPointer\(\) on Kronos website](#)

`kivy.graphics.opengl.glViewport()`

Veja: [glViewport\(\) on Kronos website](#)

4.7.11 Utilitários do OpenGL

Novo na versão 1.0.7.

`kivy.graphics.opengl_utils.gl_get_extensions()`

Retorna uma lista de extensões OpenGL disponíveis. Todos os nomes na lista têm a sintaxe iniciada com '**GL_**'(se existir) e são escritos em minúsculas.

```
>>> print(gl_get_extensions())
['arb_blend_func_extended', 'arb_color_buffer_float', 'arb_
compatibility',
 'arb_copy_buffer'... ]
```

`kivy.graphics.opengl_utils.gl_has_extension()`

Verifique se uma extensão OpenGL está disponível. Se o nome inicia com *GL_*, será retirado para o teste e convertido para minúsculo.

```
>>> gl_has_extension('NV_get_tex_image')
False
>>> gl_has_extension('OES_texture_npot')
True
```

`kivy.graphics.opengl_utils.gl_has_capability()`

Retorna o status de uma Capacidade do OpenGL. Este é um Wrapper que auto-descobre todos os recursos de que o Kivy pode precisar. Os recursos atualmente testados são:

- *GLCAP_BGRA*: Testa o suporte ao formato de textura BGRA
- *GLCAP_NPOT*: Testa o suporte a Non Power de duas texturas
- *GLCAP_S3TC*: Testa o suporte da textura *S3TC* (*DXT1*, *DXT3*, *DXT5*)
- *GLCAP_DXT1*: Testa o suporte da textura *DXT* (subconjunto de *S3TC*)
- *GLCAP_ETC1*: Testa o suporte da textura *ETC1*

`kivy.graphics.opengl_utils.gl_register_get_size()`

Registrar uma associação entre um OpenGL Const usado no *glGet** para um número de elementos.

Por exemplo, o *GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX* é um *pname* especial que retornará o inteiro 1 (apenas NVidia).

```
>>> GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX = 0x9047
>>> gl_register_get_size(GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX,_
    ↪1)
>>> glGetIntegerv(GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX)[0]
524288
```

`kivy.graphics.opengl_utils.gl_has_texture_format()`

Retornar se um formato de textura é suportado pelo seu sistema, nativamente ou por conversão. Por exemplo, se o seu cartão não suporta 'bgra', podemos converter para 'rgba', mas apenas no modo de software.

`kivy.graphics.opengl_utils.gl_has_texture_conversion()`

Retorna 1 se a textura pode ser convertida para um formato nativo.

`kivy.graphics.opengl_utils.gl_has_texture_native_format()`

Retorna 1 se o formato de textura é manipulado nativamente.

```
>>> gl_has_texture_format('azdmok')
0
>>> gl_has_texture_format('rgba')
1
>>> gl_has_texture_format('s3tc_dxt1')
[INFO    ] [GL           ] S3TC texture support is available
[INFO    ] [GL           ] DXT1 texture support is available
1
```

`kivy.graphics.opengl_utils.gl_get_texture_formats()`

Retorna uma lista de formatos de textura reconhecidos pelo Kivy. A lista de texturas é informativa, mas talvez não tenha sido suportada pelo hardware. Se desejas uma lista de texturas com suporte, deverás filtrar essa lista da seguinte maneira:

```
supported_fmts = [gl_has_texture_format(x) for x in gl_get_
                  texture_formats()]
```

`kivy.graphics.opengl_utils.gl_get_version()`

Retorna o (major, minor) versão do OpenGL, analisada a partir de *GL_VERSION*.

Novo na versão 1.2.0.

`kivy.graphics.opengl_utils.gl_get_version_minor()`

Retorna o componente menor da versão do OpenGL.

Novo na versão 1.2.0.

`kivy.graphics.opengl_utils.gl_get_version_major()`

Retorna o componente principal da versão do OpenGL.

Novo na versão 1.2.0.

4.7.12 Instruções Scissor

Novo na versão 1.9.1.

Instruções-tesoura cortam a área de desenho para uma área retangular.

- **ScissorPush**: Begins clipping, sets the bounds of the clip space
- **ScissorPop**: termina o recorte

A área fornecida ao clipe está em pixels do espaço de tela e deve ser fornecida como valores inteiros não flutuantes.

O código a seguir irá desenhar um círculo em cima do nosso Widget enquanto clipping o círculo para que ele não se expande além das bordas do Widget.

```
with self.canvas.after:  
    #If our widget is inside another widget that modified the  
    #coordinates  
    #spacing (such as ScrollView) we will want to convert to Window  
    #coords  
    x,y = self.to_window(*self.pos)  
    width, height = self.size  
    #We must convert from the possible float values provided by kivy  
    #widgets to an integer screenspace, in python3 round returns an  
    #int so  
    #the int cast will be unnecessary.  
    ScissorPush(x=int(round(x)), y=int(round(y)),  
                width=int(round(width)), height=int(round(height)))  
    Color(rgba=(1., 0., 0., .5))  
    Ellipse(size=(width*2., height*2.),  
            pos=self.center)  
    ScissorPop()
```

class kivy.graphics.scissor_instructions.Rect

Bases: object

Classe *Rect* usada internamente por *ScissorStack* e *ScissorPush* para determinar a área de recorte correta.

class kivy.graphics.scissor_instructions.ScissorPop

Bases: *kivy.graphics.instructions.Instruction*

Pop a pilha de *Scissor*. Invoca após *ScissorPush*, uma vez que você terminou o desenho que desejas ser clipped (grampeado).

class kivy.graphics.scissor_instructions.ScissorPush

Bases: *kivy.graphics.instructions.Instruction*

Empurre a pilha de *Scissor*. Fornece o kwargs de ‘x’, ‘y’, ‘width’, ‘height’ para controlar a área e a posição da região do Scissoring. O padrão é 0, 0, 100, 100

Scissor funciona cortando todo o desenho fora de um retângulo começando na posição int x, int y e tendo lados de int width-largura por int height-altura nas coordenadas de espaço da janela.

class kivy.graphics.scissor_instructions.ScissorStack

Bases: object

Classe utilizada internamente para acompanhar o estado atual das regiões *glScissors*. Não instancie, prefira inspecionar a *scissor_stack* do módulo.

4.7.13 Shader

A classe :class:`'Shader'` manipula a compilação do vértice e o sombreado, bem como a criação do programa no OpenGL.

Por fazer

Melhorar a documentação sobre o Shader.

Inclusão do cabeçalho

Novo na versão 1.0.7.

Quando você estiver criando um Shader, o Kivy sempre incluirá parâmetros padrão. Se você não quiser reescrever isso cada vez que quiser personalizar/escrever um novo shader, você pode adicionar o token “\$HEADER\$” e ele será substituído pelo cabeçalho do shader correspondente.

Aqui está o cabeçalho do fragmento Shader:

```
#ifdef GL_ES
    precision highp float;
#endif

/* Outputs from the vertex shader */
varying vec4 frag_color;
varying vec2 tex_coord0;

/* uniform texture samplers */
uniform sampler2D texture0;
```

E o cabeçalho para o vértice Shader:

```
#ifdef GL_ES
    precision highp float;
#endif

/* Outputs to the fragment shader */
varying vec4 frag_color;
varying vec2 tex_coord0;

/* vertex attributes */
attribute vec2      vPosition;
attribute vec2      vTexCoords0;

/* uniform variables */
uniform mat4        modelview_mat;
```

```
uniform mat4      projection_mat;
uniform vec4      color;
uniform float     opacity;
```

Programas de Shaders GLSL de arquivo único

Novo na versão 1.6.0.

Para simplificar o gerenciamento do Shader, os Shaders de vértices e fragmentos podem ser carregados automaticamente a partir de um único arquivo fonte GLSL (texto simples). O arquivo deve conter seções identificadas por uma linha começando com ‘— vértice’ e ‘— fragmento’ respectivamente (sem diferenciação de maiúsculas e minúsculas), por exemplo:

```
// anything before a meaningful section such as this comment are
// ignored

---VERTEX SHADER--- // vertex shader starts here
void main(){
    ...
}

---FRAGMENT SHADER--- // fragment shader starts here
void main(){
    ...
}
```

A propriedade de origem do Shader deve ser definida como sendo o nome do arquivo de um arquivo GLSL Shader (do formato acima), por exemplo, *Phong.gsl*

class kivy.graphics.shader.Shader

Bases: object

Criar um vértice ou fragmento Shader.

Parameters

vs: string, e o padrão é *None*Código Fonte para o Shader Vertex

fs: string, e o padrão é *None*Código-fonte para o Shader de Fragmentos

fs

Fragmento de código-fonte do Shader.

Se definires um novo código fonte de Shader de Fragmentos, ela será automaticamente compilada e substituirá o atual Shader de fragmentos.

source

Código-Fonte GLSL

A fonte deve ser o nome de arquivo de um Shader GLSL que contém tanto o vértice e o código fonte do Shader de fragmento, cada um designado por um cabeçalho de seção consistindo de uma linha começando com “–VERTEX” or “–FRAGMENT” (sem diferenciação de maiúsculas e minúsculas).

Novo na versão 1.6.0.

success

Indica se o Shader foi carregado com êxito e está ou não pronto para uso.

vs

Código fonte do Vertex Shader.

Se definires uma nova fonte de código de Shader de vértice, ela será automaticamente compilada e substituirá o atual Shader de vértices.

4.7.14 Instruções do Stencil

Novo na versão 1.0.4.

Alterado na versão 1.3.0: A operação ‘stêncil’ foi atualizada para solucionar problemas que apareciam quando aninhada. Você DEVE ter uma `StencilInUse` e repetir a mesma operação feita depois de `StencilPush`.

Instruções *Stencil* permitem que você desenhe e use o desenho atual como uma máscara. Eles não fornecem tanto controle como o OpenGL puro, mas você ainda pode fazer coisas interessantes!

O buffer *Stencil* pode ser controlado usando estas 3 instruções:

- ***StencilPush***: empurra uma nova camada de estêncil. Qualquer desenho que aconteça depois disso será usado como máscara.
- ***StencilUse*** : agora desenhe as próximas instruções e use o *Stencil* para mascará-las.
- ***StencilUnUse*** : pare de usar o stencil i.e. remove a máscara e desenha normalmente.
- ***StencilPop*** : pop a atual camada *Stencil*

Você sempre deve respeitar esse esquema:

`StencilPush`

PHASE 1: put any drawing instructions to use as a mask here.

`StencilUse`

*# PHASE 2: all the drawing here will be automatically clipped by the
mask created in PHASE 1.*

StencilUnUse

```
# PHASE 3: drawing instructions will now be drawn without clipping
-but the
# mask will still be on the stack. You can return to PHASE 2 at any
# time by issuing another *StencilUse* command.

StencilPop

# PHASE 4: the stencil is now removed from the stack and unloaded.
```

Limitações

- Desenho em FASE 1 e FASE 3 não deve colidir ou você obterá resultados inesperados
- O Stencil é ativado assim que você executar um *StencilPush*
- O estêncil é desativado assim que você disparou corretamente todas as camadas de estêncil.
- Não deves jogar com stencils você mesmo entre um *StencilPush/StencilPop*
- Você pode empurrar outro estêncil depois de um *StencilUse* / antes do *StencilPop*
- Você pode empurrar até 128 camadas de stencils (8 para kivy <1.3.0)

Exemplo de uso do *Stencil*

Aqui um exemplo, no estilo kv:

StencilPush

```
# create a rectangular mask with a pos of (100, 100) and a (100,
-100) size.
Rectangle:
    pos: 100, 100
    size: 100, 100
```

StencilUse

```
# we want to show a big green rectangle, however, the previous
-stencil
# mask will crop us :)
Color:
    rgb: 0, 1, 0
Rectangle:
```

```
size: 900, 900

StencilUnUse

# you must redraw the stencil mask to remove it
Rectangle:
    pos: 100, 100
    size: 100, 100

StencilPop
```

class kivy.graphics.stencil_instructions.StencilPush
Bases: *kivy.graphics.instructions.Instruction*

Empurre a pilha de Stencil. Consulte a documentação do módulo para obter mais informações.

class kivy.graphics.stencil_instructions.StencilPop
Bases: *kivy.graphics.instructions.Instruction*

Pop a pilha de Stencil. Consulte a documentação do módulo para obter mais informações.

class kivy.graphics.stencil_instructions.StencilUse
Bases: *kivy.graphics.instructions.Instruction*

Use o atual buffer *Stencil* como uma máscara. Verifique a documentação do módulo para obter mais informações.

func_op

Determina a operação de Stencil que sera usada por glStencilFunc(). Pode ser um das seguintes opções ‘never’, ‘less’, ‘equal’, ‘lequal’, ‘greater’, ‘notequal’, ‘gequal’ ou ‘always’.

Por padrão, o operador é definido como ‘equal’.

Novo na versão 1.5.0.

class kivy.graphics.stencil_instructions.StencilUnUse
Bases: *kivy.graphics.instructions.Instruction*

Use buffer *Stencil* atual para desmontar a máscara.

4.7.15 SVG

Novo na versão 1.9.0.

Aviso: Esse é altamente experimental e sujeita a mudança. Não use-o no desenvolvimento.

Carregar um SVG como uma instrução gráfica:

```
from kivy.graphics.svg import Svg
with widget.canvas:
    svg = Svg("image.svg")
```

Não há nenhum widget que possa mostrar Svg diretamente, você tem que fazer o seu próprio jeito por enquanto. Veja o exemplo '[examples/svg](#)' para mais informações.

class kivy.graphics.svg.Svg

Bases: [kivy.graphics.instructions.RenderContext](#)

Classe SVG. Veja o módulo para maiores informações a respeito de como utilizar.

anchor_x

Posição de ancoragem horizontal para escalonamento e rotação. O padrão é igual a 0. Os valores simbólicos como 'left', 'center' e 'right' também são aceitos.

anchor_y

Posição de ancoragem vertical para escalonamento e rotação. O padrão é 0. Os valores simbólicos 'bottom', 'center' e 'top' também são aceitáveis.

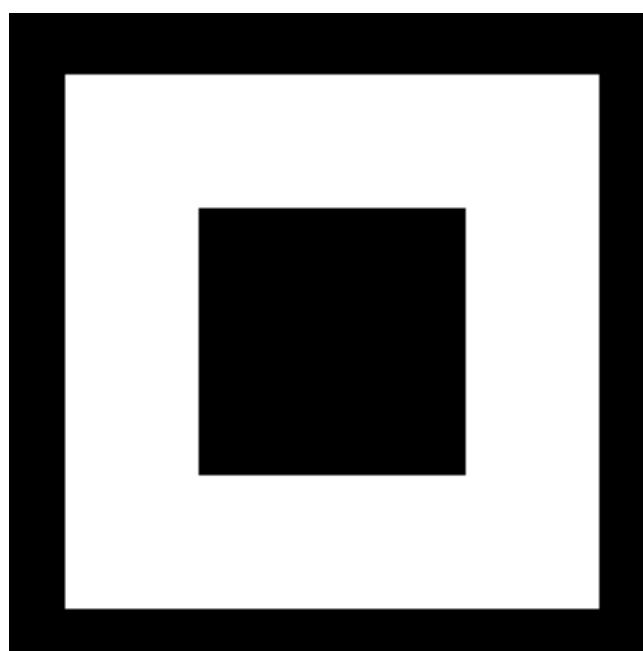
filename

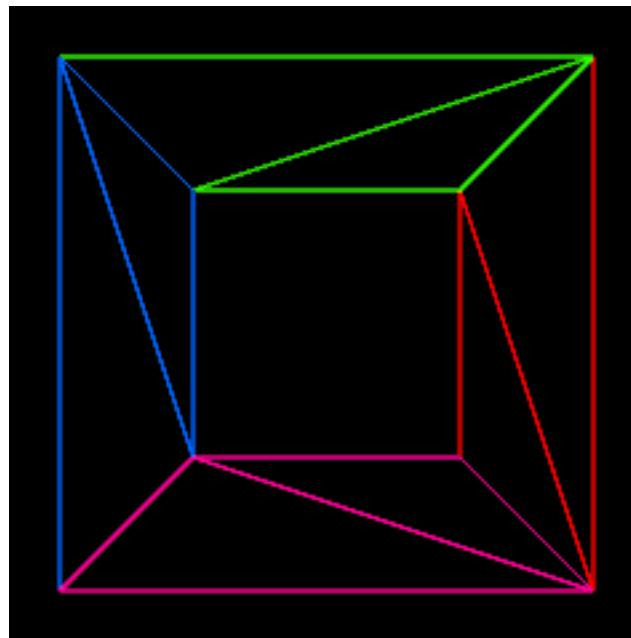
Nome do arquivo para abrir.

A análise e renderização é feita assim que você definir o nome do arquivo.

4.7.16 Tesselator

Novo na versão 1.9.0.





Aviso: Isto é experimental e está sujeito a alterações desde que esta advertência esteja presente. Somente o *TYPE_POLYGONS* suportado atualmente.

O Tesselator é uma biblioteca para construção de polígonos, baseada em *libtess2* <<https://github.com/memononen/libtess2>> . Ele transforma polígonos cheios côncavos primeiro modificando-os, em polígonos convexos. Ele também suporta buracos.

Utilização

Primeiro, você precisa criar um objeto *Tesselator* e adicionar contornos. O primeiro contorno é contorno externo de sua forma e todos os seguintes devem ser buracos:

```
from kivy.graphics.tesselator import Tesselator

tess = Tesselator()
tess.add_contour([0, 0, 200, 0, 200, 200, 0, 200])
tess.add_hole([50, 50, 150, 50, 150, 150, 50, 150])
```

Em segundo lugar, chamar o método *Tesselator.tessellate()* para calcular os pontos. É possível que o tessellator não funcione. Nesse caso, ele pode retornar False:

```
if not tess.tessellate():
    print "Tesselator didn't work :("
    return
```

Após a tessellation, você tem várias maneiras de iterar sobre o resultado. A melhor abordagem é usar *Tesselator.meshes* para obter um formato diretamente utilizável para a *Mesh*:

```

for vertices, indices in tess.meshes:
    self.canvas.add(Mesh(
        vertices=vertices,
        indices=indices,
        mode="triangle_fan"
))

```

Ou, podes obter o resultado “raw”, com apenas polígonos e coordenadas x/y com *Tesselator.vertices()*:

```

for vertices in tess.vertices:
    print "got polygon", vertices

```

class kivy.graphics.tesselator.Tesselator
Bases: object

Classe *Tesselator*. Veja o módulo da documentação para maiores informações.

add_contour()

Adicionar um contorno ao Tesselator. Pode ser feito da seguinte forma:

- uma lista de $[x, y, x2, y2, \dots]$ coodenadas
- um array de float: *array("f", [x, y, x2, y2, ...])*
- qualquer buffer com floats dentro.

element_count

Retorna o número de polígonos convexos.

meshes

Iterar através do resultado do *tessellate()* para dar um resultado que pode ser facilmente empurrado para o objeto Kivy's Mesh.

Ele é uma lista de: $[[vertices, indices], [vertices, indices], \dots]$. Os vértices no formato $[x, y, u, v, x2, y2, u2, v2]$.

Cuidado, as coordenadas de u/v são as mesmas que x/y. Você é responsável por alterá-los para o mapeamento de textura, se for necessário.

Você pode criar objetos Mesh da seguinte forma:

```

tess = Tesselator()
# add contours here
tess.tessellate()
for vertices, indices in self.meshes:
    self.canvas.add(Mesh(
        vertices=vertices,
        indices=indices,
        mode="triangle_fan"))

```

tesselate()

Calcula todos os contornos adicionados com [add_contour\(\)](#), e gere polígonos.

Parameters

*winding_rule: enum*A regra da dobra classifica uma região como interior se o seu número de dobra pertence à categoria escolhida. Pode ser um dos WINDING_ODD, WINDING_NONZERO, WINDING_POSITIVE, WINDING_NEGATIVE, WINDING_ABS_EQTWO. Padrão para WINDING_ODD.

*element_type: enum*O tipo de resultado, você pode gerar os polígonos com TYPE_POLYGONS ou os contornos com TYPE_BOUNDARY_CONTOURS. Padrão para TYPE_POLYGONS.

Retorna1 se o tessellation aconteceu, caso contrário 0.

Tipo de retornoint

vertex_count

Retorna o número de vertex gerados.

Este é o resultado bruto, no entanto, porque o Tesselator formata o resultado para você com [meshes](#) ou [vertices](#) por polígono, teras mais vértices no resultado

vertices

Iterar através do resultado do [tesselate\(\)](#) de forma a gerar uma lista de polígonos $[x, y, x2, y2, \dots]$.

4.7.17 Textura

Alterado na versão 1.6.0: Adicionado suporte para textura paletada em OES: ‘palette4_rgb8’, ‘palette4_rgba8’, ‘palette4_r5_g6_b5’, ‘palette4_rgba4’, ‘palette4_rgb5_a1’, ‘palette8_rgb8’, ‘palette8_rgba8’, ‘palette8_r5_g6_b5’, ‘palette8_rgba4’ e ‘palette8_rgb5_a1’.

Texture é uma classe que lida com texturas OpenGL. Dependendo do hardware, alguns recursos OpenGL podem não estar disponíveis (suporte BGRA, suporte NPOT, etc.)

Você não pode instanciar esta classe nela mesma. Você deve usar a função [Texture.create\(\)](#) para criar uma nova texture:

```
texture = Texture.create(size=(640, 480))
```

Ao criar uma textura, você deve estar ciente da cor padrão e do formato do buffer:

- o formato da cor/pixel (*Texture.colorfmt*) que pode ser uma das opções ‘rgb’, ‘rgba’, ‘luminance’, ‘luminance_alpha’, ‘bgr’ ou ‘bgra’. O padrão é ‘rgb’
- O formato do Buffer determina como um componente de cor é armazenado na memória. Esta pode ser uma das opções ‘ubyte’, ‘ushort’, ‘uint’, ‘byte’, ‘short’, ‘int’ ou ‘float’. O valor padrão e o mais comumente utilizado é ‘ubyte’.

Então, se você quiser criar um RGBA texture:

```
texture = Texture.create(size=(640, 480), colorfmt='rgba')
```

Podes usar sua textura em quase todas as instruções de vértice com o parâmetro `kivy.graphics.VertexInstruction.texture`. Se quiseres usar sua textura com a linguagem kv, podes salvá-la numa *ObjectProperty* dentro do seu Widget.

Blitting dados personalizados

Você pode criar seus próprios dados e enviá-los para a textura usando *Texture.blit_buffer()*.

Por exemplo, para blit dados de bytes imutáveis:

```
# create a 64x64 texture, defaults to rgba / ubyte
texture = Texture.create(size=(64, 64))

# create 64x64 rgb tab, and fill with values from 0 to 255
# we'll have a gradient from black to white
size = 64 * 64 * 3
buf = [int(x * 255 / size) for x in range(size)]

# then, convert the array to a ubyte string
buf = b''.join(map(chr, buf))

# then blit the buffer
texture.blit_buffer(buf, colorfmt='rgb', bufferfmt='ubyte')

# that's all ! you can use it in your graphics now :)
# if self is a widget, you can do this
with self.canvas:
    Rectangle(texture=texture, pos=self.pos, size=(64, 64))
```

Desde 1.9.0, você pode ligar dados armazenados em uma instância que implementa a interface de buffer python, ou um memoryview deles, como numpy arrays, python `array.array`, `bytearray`, ou uma matriz ‘array’ cython. Isso é benéfico se você espera para ligar dados semelhantes, com talvez algumas mudanças nos dados.

Ao usar uma reprodução de bytes dos dados, para cada alteração você tem que regenerar a instância de bytes, de talvez uma lista, o que é muito ineficiente. Ao usar um objeto de buffer, você pode simplesmente editar partes dos dados originais. Da mesma

forma, a menos que comece com um objeto bytes, a conversão para bytes requer uma cópia completa, no entanto, ao usar uma instância de buffer, nenhuma memória é copiada, exceto para carregá-lo para a GPU.

Continuando com o exemplo acima:

```
from array import array

size = 64 * 64 * 3
buf = [int(x * 255 / size) for x in range(size)]
# initialize the array with the buffer values
arr = array('B', buf)
# now blit the array
texture.blit_buffer(arr, colorfmt='rgb', bufferfmt='ubyte')

# now change some elements in the original array
arr[24] = arr[50] = 99
# blit again the buffer
texture.blit_buffer(arr, colorfmt='rgb', bufferfmt='ubyte')
```

Suporta BGR/BGRA

A primeira vez que você tenta criar uma textura BGR ou BGRA, verificamos se seu hardware suporta texturas BGR / BGRA verificando a extensão 'GL_EXT_bgra'.

Se a extensão não for encontrada, a conversão para RGB / RGBA será feita em software.

Textura NPOT

Alterado na versão 1.0.7: Se o seu hardware suporta NPOT, nenhum POT é criado.

Como a documentação do OpenGL diz, uma textura deve ter o poder-de-dois tamanhos. Isso significa que sua largura e altura podem ser uma de 64, 32, 256 ... mas não 3, 68, 42. NPOT significa não-poder-de-dois. OpenGL ES 2 suporta NPOT texturas nativamente, mas com algumas desvantagens. Outro tipo de textura NPOT é chamado de textura de retângulo. POT, NPOT e texturas, todos têm seus próprios prós/contras.

Características	POT	NPOT	Retângulo
Objeto OpenGL	GL_TEXTURE	GL_TEXTURE	GL_TEXTURE_RECTANGLE_(NV ARB EXT)
Coordenadas de textura	0-1 alcance	0-1 alcance	Largura-altura
Mipmapping	Suportado	Em parte	Não
Modo de quebra automática	Suportado	Suportado	Não

Se criares uma textura NPOT, verificamos primeiro se seu hardware o suporta, verificando as extensões GL_ARB_texture_non_power_of_two ou OES_texture_npot. Se

nenhum destes estiver disponível, criamos a textura POT mais próxima que poderá conter a sua textura NPOT. O método `:Texture.create` retornará uma classe `:TextureRegion` em vez disso.

Atlas da textura

Um atlas de textura é uma única textura que contém muitas imagens. Se você quiser separar a textura original em muitos únicos, você não precisa. Você pode obter uma região da textura original. Isso retornará a textura original com as coordenadas de textura personalizadas:

```
# for example, load a 128x128 image that contain 4 64x64 images
from kivy.core.image import Image
texture = Image('mycombinedimage.png').texture

bottomleft = texture.get_region(0, 0, 64, 64)
bottomright = texture.get_region(0, 64, 64, 64)
topleft = texture.get_region(0, 64, 64, 64)
topright = texture.get_region(64, 64, 64, 64)
```

Mipmapping

Novo na versão 1.0.7.

Mipmapping é uma técnica OpenGL para melhorar a renderização de grandes texturas em pequenas superfícies. Sem mipmapping, você pode ver pixelação quando você renderizar em pequenas superfícies. A idéia é pre-calcular a subtextura e aplicar algum filtro de imagem como um filtro linear. Em seguida, quando você renderizar uma pequena superfície, em vez de usar a maior textura, ele usará uma textura filtrada inferior. O resultado pode ficar melhor assim.

Para fazer com que isso aconteça, precisarás especificar `mipmap=True` quando fores criar uma textura. Alguns Widgets fornecem a capacidade de criar texturas `mipmapped`, tais como `Label` e `Image`.

Do OpenGL Wiki: “Então, uma textura 2D 64x16 pode ter 5 mip-mapas: 32x8, 16x4, 8x2, 4x1, 2x1 e 1x1”. Verifique <http://www.opengl.org/wiki/Texture> para obter mais informações.

Nota: Como a tabela na seção anterior disse, se sua textura é NPOT, criamos a textura mais próxima POT e gerar um mipmap a partir dele. Isso pode mudar no futuro.

Recarregando a Textura

Novo na versão 1.2.0.

Se o contexto OpenGL for perdido, a textura deve ser recarregada. As texturas que têm uma origem são automaticamente recarregadas, mas as texturas geradas devem ser recarregadas pelo usuário.

Use o `Texture.add_reload_observer()` para adicionar uma função de recarga que será automaticamente chamada quando necessário:

```
def __init__(self, **kwargs):
    super(...).__init__(**kwargs)
    self.texture = Texture.create(size=(512, 512), colorfmt='RGB',
        bufferfmt='ubyte')
    self.texture.add_reload_observer(self.populate_texture)

    # and load the data now.
    self.cbuffer = '\x00\xf0\xff' * 512 * 512
    self.populate_texture(self.texture)

def populate_texture(self, texture):
    texture.blit_buffer(self.cbuffer)
```

Desta forma, você pode usar o mesmo método para inicialização e recarregamento.

Nota: Para todas as renderizações de texto com nosso processador de texto de núcleo, a textura é gerada, mas já vinculamos um método para refazer a renderização de texto e reenviar o texto para a textura. Você não precisa fazer nada.

class kivy.graphics.texture.Texture

Bases: object

Manipular uma textura OpenGL. Esta classe pode ser usada para criar texturas simples ou texturas complexas baseadas no *ImageData*.

`add_reload_observer()`

Adiciona um callback pra ser chamado depois que todo o contexto gráfico foi recarregado. Este é o lugar onde você pode reupload seus dados personalizados para a GPU.

Novo na versão 1.2.0.

Parameters

`callback: func(context) -> return None`O primeiro parâmetro será o próprio contexto.

`ask_update()`

Indica que o conteúdo da textura deve ser atualizado e que a função de callback precisa ser invocada quando a textura for usada.

`bind()`

Vincule a textura ao estado opengl atual.

blit_buffer()

Blit um buffer na textura.

Nota: A menos que a tela seja atualizada devido a outras alterações, a função `ask_update()` deve ser chamado para atualizar a textura.

Parameters

pbuffer: bytes ou uma classe que implementa a interface do Buffer (incluindo r

Um buffer contendo os dados da imagem. O mesmo poderá ser um objeto em bytes ou uma instância de uma classe que implementa a interface do buffer Python, por exemplo, `array.array`, `bytearray`, `numpy arrays` etc. Se não for um objeto em bytes, o buffer subjacente deverá ser contíguo, ter apenas uma dimensão e não deverá ser *readonly*, mesmo que os dados não sejam modificados, devido a uma limitação do Cython. Consulte a descrição do módulo para obter mais detalhes de como utilizar.

size: tuple, padrão para o tamanho da texturaTamanho da imagem (largura, altura)

colorfmt: str, o parão é 'rgb'Formato da imagem, pode ser um de 'rgb', 'rgba', 'bgr', 'bgra', 'luminance' or 'luminance_alpha'.

pos: tuple, padrão é (0, 0)Posição para voar na textura. (Position to blit in the texture).

bufferfmt: str, o padrão é 'ubyte'O tipo de Buffer de dados, pode ser uma das opções 'ubyte', 'ushort', 'uint', 'byte', 'short', 'int' ou 'float'.

mipmap_level: int, o padrão é 0Indica o nível de mipmap que vamos atualizar.

mipmap_generation: bool, o padrão é TrueIndica se precisamos regenerar o mipmap a partir do nível 0.

Alterado na versão 1.0.7: adicionado `mipmap_level` e `mipmap_generation`

Alterado na versão 1.9.0: 'pbuffer' agora pode ser qualquer instância da classe que implementa a interface de Buffer de Python e/ou respectivos memoryviews.

blit_data()

Substitui uma textura inteira com dados de imagem.

bufferfmt

Retorna um formato de buffer usado nessa textura (somente leitura).

Novo na versão 1.2.0.

colorfmt

Retorna o formato de cor usado nesta textura (somente leitura).

Novo na versão 1.0.7.

static create()

Cria uma textura com base no tamanho.

Parameters

size: tuple, o padrão é (128, 128) Tamanho da textura.

colorfmt: str, o padrão é 'rgba' Formato de cor da textura. Pode ser 'rgba' or 'rgb', 'luminance' ou 'luminance_alpha'. No Desktop, estão disponíveis valores adicionais: 'red', 'rg'.

icolorfmt: str, o padrão é o valor de *colorfmt* Armazenamento em formato interno da textura. Pode ser 'rgba' or 'rgb', 'luminance' ou 'luminance_alpha'. Em Desktop, valores adicionais estão disponíveis: 'r8', 'rg8', 'rgba8'.

bufferfmt: str, o padrão é 'ubyte' Formato do Buffer interno da textura. Pode ser uma das seguintes opções: 'ubyte', 'ushort', 'uint', 'bute', 'short', 'int' ou 'float'.

mipmap: bool, padrão é False Se True, ele gerará automaticamente a textura mipmap.

callback: callable(), e o padrão é False Se for fornecida uma função, ela será chamada quando os dados forem necessários na textura

Alterado na versão 1.7.0: **callback** foi adicionado

static create_from_data()

Cria uma textura de uma classe *ImageData*.

flip_horizontal()

Gira o 'tex_coords' para exibição horizontal.

Novo na versão 1.9.0.

flip_vertical()

Gira o 'tex_coords' para exibição vertical.

get_region()

Retorna uma parte da textura definida pelos argumentos retangulares (x, y, largura, altura). Retorna uma instância *TextureRegion*.

height

Retorna a altura da textura (somente leitura).

id

Retorna o ID OpenGL da textura (somente leitura).

mag_filter

Obtém/define a textura do filtro mag. Valores disponíveis:

- linear
- mais próximo

Veja a documentação OpenGL para obter mais informações sobre o comportamento desses valores: <http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

min_filter

Obtém/defe a textura do filtro min. Valores disponíveis:

- linear
- mais próximo
- linear_mipmap_linear
- linear_mipmap_nearest
- nearest_mipmap_nearest
- nearest_mipmap_linear

Veja a documentação OpenGL para obter mais informações sobre o comportamento desses valores: <http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

mipmap

Retorna *True* se a textura tiver o mipmap ativado (somente leitura).

pixels

Obtém a textura dos pixels, apenas no formato RGBA, Unsigned Byte. A origem da imagem está na parte inferior esquerda.

Novo na versão 1.7.0.

remove_reload_observer()

Remove o callback da lista de observadores, adicionado anteriormente por [*add_reload_observer\(\)*](#).

Novo na versão 1.2.0.

save()

Salve o conteúdo da textura em um arquivo. Verifique [*kivy.core.image.Image.save\(\)*](#) para mais informações.

O parâmetro invertido inverte a imagem salva verticalmente e o padrão é *True*.

Novo na versão 1.7.0.

Alterado na versão 1.8.0: Parâmetro *flipped* adicionado, o padrão é *True*. Toda a textura do OpenGL é lida da parte inferior/esquerda, necessita ser girada antes de conservar. Se não quiseres inverter a imagem, defina *flipped* como sendo *False*.

size

Retorna a (largura, altura) da textura (somente leitura).

target

Retorna o alvo OpenGL da textura (somente leitura).

tex_coords

Retorna a lista de *tex_coords* (OpenGL).

uvpos

Obtém/define a posição UV dentro da textura.

uvsize

Obtém/define o tamanho UV dentro da textura.

Aviso: O tamanho pode ser negativo se a textura for invertida.

width

Retorna a largura da textura (somente leitura).

wrap

Obtém/define a textura de quebra automática. Valores disponíveis:

- repeat
- mirrored_repeat
- clamp_to_edge

Veja a documentação OpenGL para obter mais informações sobre o comportamento desses valores: <http://www.khronos.org/opengles/sdk/docs/man/xhtml/glTexParameter.xml>.

class kivy.graphics.texture.TextureRegion

Bases: [kivy.graphics.texture.Texture](#)

Manipular uma região de uma classe *Texture*. Útil para manipulação de textura sem poder-de-2.

4.7.18 Transformação

Este módulo contém uma classe *Matrix* usada para calcular nossos Graphics. Nós atualmente suportamos:

- rotação, translação e matrizes de rotação

- multiplicação de Matrix
- clip matrix (com ou sem perspectiva)
- Matrix de transformação para toque 3D

Para mais informações sobre transformação de matrizes, por favor, veja [OpenGL Matrices Tutorial](#).

Alterado na versão 1.6.0: Adicionado `Matrix.perspective()`, `Matrix.look_at()` e `Matrix.transpose()`.

`class kivy.graphics.transformation.Matrix`
Bases: `object`

Classe de matriz otimizada para OpenGL:

```
>>> from kivy.graphics.transformation import Matrix
>>> m = Matrix()
>>> print(m)
[[ 1.000000 0.000000 0.000000 0.000000 ]
 [ 0.000000 1.000000 0.000000 0.000000 ]
 [ 0.000000 0.000000 1.000000 0.000000 ]
 [ 0.000000 0.000000 0.000000 1.000000 ]]

[ 0   1   2   3]
[ 4   5   6   7]
[ 8   9   10  11]
[ 12  13  14  15]
```

`get()`

Restabelece o valor atual como uma lista plana. (Retrieve the value of the current as a flat list).

Novo na versão 1.9.1.

`identity()`

Redefine a matriz para a matriz de identidade (inplace). (Reset the matrix to the identity matrix (inplace).)

`inverse()`

Retorna o inverso da Matriz como uma nova Matriz.

`look_at()`

Retorna um novos lookat Matrix (similar ao `gluLookAt`).

Parameters

`eyex: float` Coordenar os olhos X (Eyes X co-ordinate)

`eyey: float` Coordenar os olhos Y (Eyes Y co-ordinate)

`eyez: float` Coordenar os olhos Z (Eyes Z co-ordinate)

`centerx: float` A posição X do ponto de referência

centerx: floatA posição X do ponto de referência
centery: floatA posição Y do ponto de referência
centerz: floatA posição Z do ponto de referência
upx: floatO valor de X até o vetor. (The X value up vector.)
upy: floatO valor de Y até o vetor. (The Y value up vector.)
upz: floatO valor de Z até o vetor. (The Z value up vector.)

Novo na versão 1.6.0.

multiply()

Multiplique a matriz dada com o self (a partir da esquerda), ou seja, pre-multiplicamos a matriz dada pela matriz atual e retornamos o resultado (não inplace):

```
m.multiply(n) -> n * m
```

Parameters

ma: MatrixA matriz para multiplicar por

normal_matrix()

Calcula a matriz normal, que é a transposição inversa da matriz de ModelView 3x3 superior esquerda usada para transformar normais em espaço dentro do olho/câmera.

Novo na versão 1.6.0.

perspective()

Cria uma matriz de perspectiva(inplace).

Parameters

fovy: floatÂngulo do “Campo de Visão”
aspect: floatAspect Ratio (Proporção da Tela)
zNear: floatPerto do plano de recorte (Near clipping plane)
zFar: floatLonge do plano de recorte (Far clippin plane)

Novo na versão 1.6.0.

project()

Projeta um ponto do espaço 3D em uma Viewport 2D.

Parameters

objx: floatCoordenar pontos X (Points X co-ordinate)
objy: floatCoordenar pontos Y (Points Y co-ordinate)
objz: floatCoordenar pontos Z (Points Z co-ordinate)
model: MatrixO modelo da Matriz

proj: MatrixA projeção da Matriz

vx: floatCoordenar as Viewports X (Viewports X co-ordinate)

vy: floatCoordenar as Viewports Y (Viewports Y co-ordinate)

vw: floatLargura do Viewports (janela de visualização)

vh: floatAltura do Viewports (janela de visualização)

Novo na versão 1.7.0.

rotate()

Girar a matriz através do ângulo em torno do eixo (x, y, z) (inplace). (Rotate the matrix through the angle around the axis (x, y, z) (inplace).)

Parameters

angle: floatO ângulo através do qual se deseja girar a matriz
(The angle through which to rotate the matrix)

x: floatPosição X do ponto

y: floatPosição Y do ponto (Y position of the point)

z: floatPosição Z do ponto (Z position of the point)

scale()

Dimensiona a atual matriz pelos fatores especificados em cada dimensão (inplace). (Scale the current matrix by the specified factors over each dimension (inplace).)

Parameters

x: floatO (scale factor) fator de escala ao longo do eixo X

y: floatO (scale factor) fator de escala ao longo do eixo Y

z: floatO (scale factor) fator de escala ao longo do eixo Z

set()

Insert custom values into the matrix in a flat list format or 4x4 array format like below:

```
m.set(array=[  
    [1.0, 0.0, 0.0, 0.0],  
    [0.0, 1.0, 0.0, 0.0],  
    [0.0, 0.0, 1.0, 0.0],  
    [0.0, 0.0, 0.0, 1.0]]  
)
```

Novo na versão 1.9.0.

tolist()

Retrieve the value of the current matrix in numpy format. for example m.tolist() will return:

```
[[1.0, 0.0, 0.0, 0.0],  
 [0.0, 1.0, 0.0, 0.0],  
 [0.0, 0.0, 1.0, 0.0],  
 [0.0, 0.0, 0.0, 1.0]]
```

Você pode usar este formato para conectar o resultado diretamente com NumPy e desta forma que `numpy.array(m.tolist())` (you can use this format to plug the result straight into numpy in this way `numpy.array(m.tolist())`)

Novo na versão 1.9.0.

translate()

Traduz a Matriz.

Parameters

x: floatO fator de conversão ao longo do eixo X

y: floatO fator de conversão ao longo do eixo Y

z: floatO fator de conversão ao longo do eixo Z

transpose()

Retorna a matriz transposta como uma nova matriz

Novo na versão 1.6.0.

view_clip()

Cria uma matriz de clipe(Inplace).

Parameters

left: floatCoordenada

right: floatCoordenada

bottom: floatCoordenada

top: floatCoordenada

near: floatCoordenada

far: floatCoordenada

perspective: intCoordenada

Alterado na versão 1.6.0: Ativa supor para parâmetro de perspectiva.

4.7.19 Instruções Vertex

Este módulo inclui todas as classes para desenhar objetos de vértice simples.

Atualizando Propriedades

Os atributos de lista das classes de instrução gráfica (por exemplo `Triangle.points`, `Mesh.indices` etc) não são propriedades Kivy, mas sim, propriedades Python. Como consequência, os elementos gráficos serão atualizados somente quando o objeto de lista propriamente dito for alterado e não quando os valores da lista forem modificados.

Para um exemplo em Python:

```
class MyWidget(Button):  
  
    triangle = ObjectProperty(None)  
    def __init__(self, **kwargs):  
        super(MyWidget, self).__init__(**kwargs)  
        with self.canvas:  
            self.triangle = Triangle(points=[0,0, 100,100, 200,0])
```

e em kv:

```
<MyWidget>:  
    text: 'Update'  
    on_press:  
        self.triangle.points[3] = 400
```

Embora pressionar o botão irá alterar as coordenadas do triângulo, os gráficos não serão atualizados porque a própria lista não foi alterada. Da mesma forma, nenhuma atualização ocorrerá usando qualquer sintaxe que altere apenas os elementos da lista, e. g. `self.triangle.points[0: 2] = [10,10]` ou `self.triangle.points.insert(10)` etc. Para forçar uma atualização após uma alteração, a variável list deve ser alterada, o que neste caso pode com:

```
<MyWidget>:  
    text: 'Update'  
    on_press:  
        self.triangle.points[3] = 400  
        self.triangle.points = self.triangle.points
```

`class kivy.graphics.vertex_instructions.Triangle`
Bases: `kivy.graphics.instructions.VertexInstruction`

Um triângulo 2D.

Parameters

`points`: listLista de pontos no formato (x1, y1, x2, y2, x3, y3).

points

Propriedade para obtenção/definição de pontos no triângulo.

```
class kivy.graphics.vertex_instructions.Quad  
Bases: kivy.graphics.instructions.VertexInstruction
```

Um quad 2D.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2, x3, y3, x4, y4).

points

Propriedade para obter/configurar ponto no quad.

```
class kivy.graphics.vertex_instructions.Rectangle  
Bases: kivy.graphics.instructions.VertexInstruction
```

Um retângulo 2D.

Parameters

pos: listPosição do retângulo, no formato (x, y).

size: listTamanho do retângulo, no formato (largura, altura).

pos

Propriedade para obter/configurar a posição do retângulo.

size

Propriedade para obter/configurar o tamanho do retângulo.

```
class kivy.graphics.vertex_instructions.RoundedRectangle  
Bases: kivy.graphics.vertex_instructions.Rectangle
```

Um retângulo 2D arredondado.

Novo na versão 1.9.1.

Parameters

segments: int, o padrão é 10Defina quantos segmentos são necessários para desenhar o canto redondo. O desenho será mais suave se você tiver muitos segmentos.

radius: list, o padrão é [(10.0, 10.0), (10.0, 10.0), (10.0, 10.0), (10.0, 10.0)]

Especifica os raios dos cantos redondos no sentido horário: topo-esquerdo, superior-direito, inferior-direito, inferior-esquerdo. Os elementos da lista podem ser números ou tuplas de dois números para especificar diferentes dimensões x, y. Um valor definirá todas as dimensões de canto para esse valor. Quatro valores definem as dimensões para cada canto separadamente. Um maior número de valores será truncado para quatro. O primeiro valor será usado para todos os cantos, se houver menos de quatro valores.

radius

Raios de canto do retângulo arredondado, o padrão é [10,].

segments

Propriedade para obter/configurar o número de segmentos por cada canto.

class kivy.graphics.vertex_instructions.BorderImage

Bases: [kivy.graphics.vertex_instructions.Rectangle](#)

Uma 2D Border Imagem. O comportamento do Border Imagem é semelhante ao conceito de uma imagem de borda CSS3.

Parameters

border: listBorder information in the format (bottom, right, top, left). Each value is in pixels.

auto_scale: boolNovo na versão 1.9.1.

If the BorderImage's size is less than the sum of it's borders, horizontally or vertically, and this property is set to True, the borders will be rescaled to accommodate for the smaller size.

auto_scale

Propriedade para definir se os cantos são automaticamente dimensionados quando o BorderImage é muito pequeno.

border

Propriedade para obter/configurar a borda da classe.

display_border

Propriedade para obter/configurar o tamanho da exibição da borda.

class kivy.graphics.vertex_instructions.Ellipse

Bases: [kivy.graphics.vertex_instructions.Rectangle](#)

Uma elipse 2D.

Alterado na versão 1.0.7: Adicionado *angle_start* e *angle_end*.

Parameters

segments: int, o padrão é 180Denine quantos segmentos são necessários para desenhar a Elipse. O desenho será mais suave se você tiver muitos segmentos.

angle_start: int, o padrão é 0Especifica o ângulo inicial, em graus, da porção do disco.

angle_end: int, o padrão é 360Especifica o ângulo final, em graus, da porção do disco.

angle_end

Ângulo final da elipse em graus, o padrão é 360.

angle_start

Inicia o ângulo da elipse em graus, o padrão é 0.

segments

Propriedade para obter/configurar o número de segmentos de uma Elipse.

class kivy.graphics.vertex_instructions.Line

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma linha 2D.

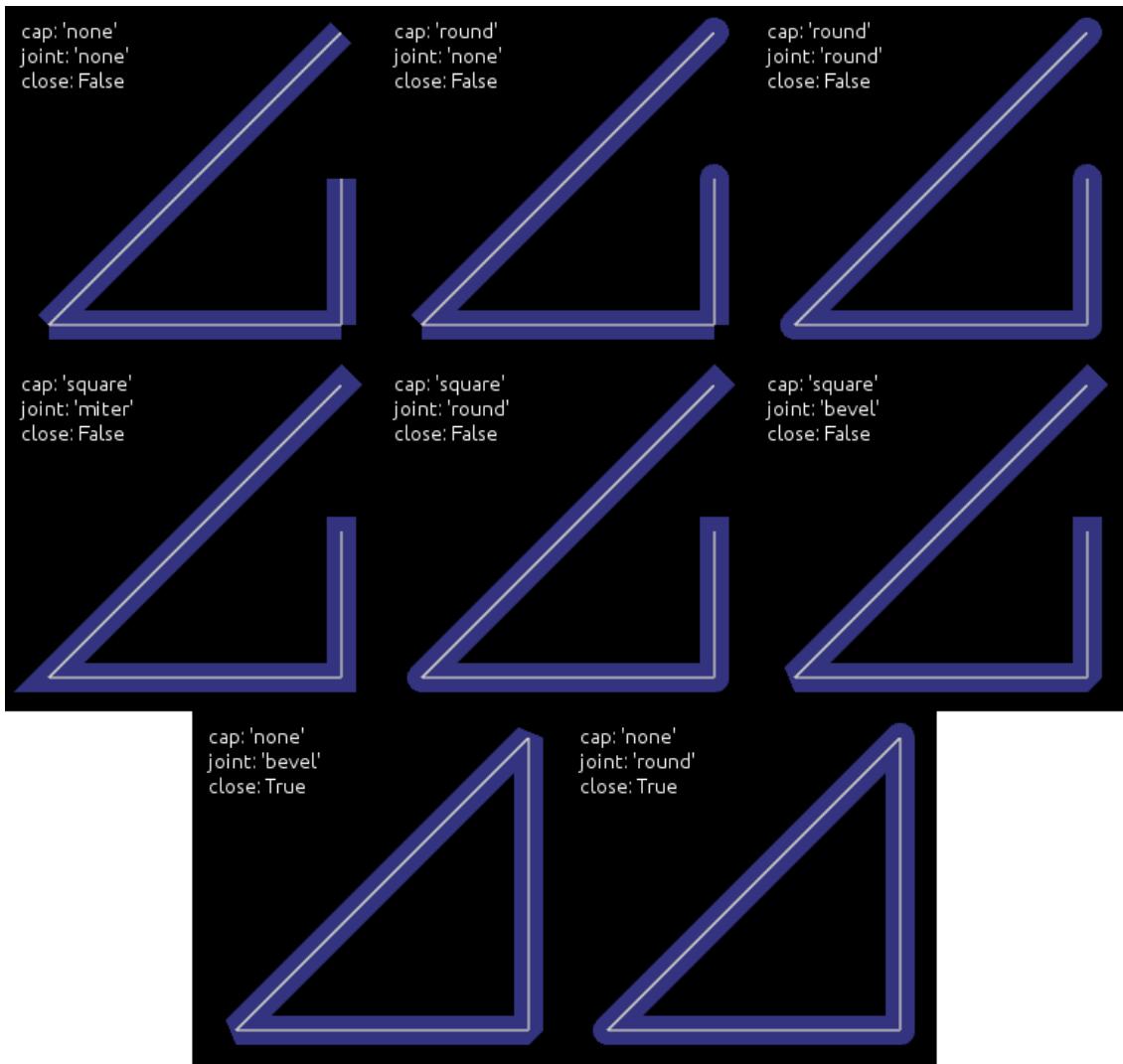
O desenho de uma linha pode ser feito facilmente:

```
with self.canvas:  
    Line(points=[100, 100, 200, 100, 100, 200], width=10)
```

A linha tem 3 modos de desenho interno que você deve estar ciente para obter os melhores resultados:

1.Se o *width* for 1.0, então o padrão de desenho *GL_LINE* do OpenGL será utilizado. *dash_length* e *dash_offset* funcionarão, enquanto as propriedades para o cap e a junção não tiverem nenhum significado aqui.

2.Se o: attr: *width* for maior que 1.0, então um método de desenho personalizado, baseado em triangulação, será usado. :attr: *dash_length* e: attr: *dash_offset* não funcionam neste modo. Além disso, se a cor atual tiver um alpha menor que 1,0, um estêncil será usado internamente para desenhar a linha.



Parameters

points: listLista de pontos no formato (x1, y1, x2, y2...)

dash_length: intTamanho de um segmento (se tracejado), o padrão é 1.

dash_offset: intOffset entre o final de um segmento e o início do próximo, o padrão é 0. Alterando isso torna-o tracejado.

width: floatLargura da linha, o padrão é 1.0.

cap: str, e o padrão é 'round'Veja o [cap](#) para maiores informações.

joint: str, e o padrão é 'round'Veja o [joint](#) para maiores informações.

cap_precision: int, o padrão é 10Veja o [cap_precision](#) para maiores informações

joint_precision: int, o padrão é 10Veja [joint_precision](#) para maiores informações. Veja [cap_precision](#) para maiores informações.

joint_precision: int, o padrão é 10
Veja [joint_precision](#) para maiores informações.

close: bool, defaults to False
Se True, a linha será fechada.

circle: list
Se definido, o [points](#) será definido para construir um círculo. Veja [circle](#) para maiores informações.

ellipse: list
Se definido, o [points](#) será definido para construir uma ellipse. Veja [ellipse](#) para maiores informações.

rectangle: list
Se definido, o [points](#) será definido para construir um retângulo. Veja [rectangle](#) para maiores informações.

bezier: list
Se definido, o [points](#) será definido para construir uma linha bezier. Veja [bezier](#) para maiores informações.

bezier_precision: int, e o padrão é 180
Precisão do desenho Bezier.

Alterado na versão 1.0.8: *dash_offset* e *dash_length* foram adicionados.

Alterado na versão 1.4.1: *width*, *cap*, *joint*, *cap_precision*, *joint_precision*, *close*, *ellipse*, *rectangle* foram adicionados.

Alterado na versão 1.4.1: *bezier*, *bezier_precision* foram adicionados.

bezier

Utilize essa propriedade para construir uma linha Bezier, sem calcular os [points](#). Você pode definir esta propriedade, não obtê-la.

O argumento deve ser uma tupla de $2n$ elementos, n sendo o número de pontos.

Uso:

```
Line(bezier=(x1, y1, x2, y2, x3, y3))
```

Novo na versão 1.4.2.

Nota: Os cálculos das linhas de Bezier usam poucos recursos para um número baixo de pontos, mas a complexidade é quadrática, então as linhas com muitos pontos podem ser muito demoradas para serem construídas, use com cuidado!

bezier_precision

Número de iteração para desenhar o Bezier entre 2 segmentos, o padrão é 180. O *bezier_precision* deve ser pelo menos '1'q.

Novo na versão 1.4.2.

cap

Determine o limite da linha, o padrão é 'round'. Pode ser uma das seguintes opções 'none', 'square' ou 'round'.

Novo na versão 1.4.1.

cap_precision

Número de iteração para desenhar a cápsula “redonda”, o padrão é 10. A *cap_precision* deve ser pelo menos 1.

Novo na versão 1.4.1.

circle

Use this property to build a circle, without calculating the *points*. You can only set this property, not get it.

O argumento deve ser uma tupla de (center_x, center_y, radius, angle_start, angle_end, segments):

- *center_x* e *center_y* representam o centro do círculo
- *radius* representa o raio do círculo
- (optional) *angle_start* and *angle_end* are in degree. The default value is 0 and 360.
- (optional) *segments* is the precision of the ellipse. The default value is calculated from the range between angle.

Observe que cabe a você fechar *close* ou não o círculo.

Por exemplo, para construir uma simples elipse em Python:

```
# simple circle
Line(circle=(150, 150, 50))

# only from 90 to 180 degrees
Line(circle=(150, 150, 50, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(circle=(150, 150, 50, 90, 180, 20))
```

Novo na versão 1.4.1.

close

Se *True*, a linha será fechada.

Novo na versão 1.4.1.

dash_length

Propriedade para obter/configurar o tamanho dos traços na curva

Novo na versão 1.0.8.

dash_offset

Propriedade para obter/configurar o deslocamento entre os traço curva

Novo na versão 1.0.8.

ellipse

Use esta propriedade para construir uma Elipse, sem calcular os *points*. Você somente pode definir essa propriedade, não obtê-la.

O argumento deve ser uma tupla de (*x*, *y*, *width*, *height*, *angle_start*, *angle_end*, *segments*,):

- *x* e *y* representam a parte inferior esquerda da Elipse
- *width* e *height* representam o tamanho da Elipse
- (optional) *angle_start* and *angle_end* are in degree. The default value is 0 and 360.
- (optional) *segments* is the precision of the ellipse. The default value is calculated from the range between angle.

Observe que cabe a você fechar *close* ou não a Elipse.

Por exemplo, para construir uma simples elipse em Python:

```
# simple ellipse
Line(ellipse=(0, 0, 150, 150))

# only from 90 to 180 degrees
Line(ellipse=(0, 0, 150, 150, 90, 180))

# only from 90 to 180 degrees, with few segments
Line(ellipse=(0, 0, 150, 150, 90, 180, 20))
```

Novo na versão 1.4.1.

joint

Determine a junção da linha, o padrão é ‘round’. Pode ser um das opções ‘none’, ‘round’, ‘bevel’, ‘miter’.

Novo na versão 1.4.1.

joint_precision

Número de iteração para desenhar a junção “redonda”, o padrão é 10. A *joint_precision* deve ser pelo menos 1.

Novo na versão 1.4.1.

points

Propriedade para obter/configurar pontos de uma linha

Aviso: Isso sempre reconstruirá os gráficos completamente desde a nova lista de pontos. Pode utilizar muito CPU.

rectangle

Use essa propriedade para construir um retângulo, sem calcular o *points*. Você só pode definir esta propriedade, não obtê-lo.

O argumento deve ser uma tupla de (x, y, width, height):

- *x* e *y* representam a posição bottom-left (inferior esquerda) do retângulo.
- *width* e *height* representam o tamanho

A linha é automaticamente fechada.

Uso:

```
Line(rectangle=(0, 0, 200, 200))
```

Novo na versão 1.4.1.

rounded_rectangle

Use essa propriedade para construir um retângulo, sem calcular o *points*. Você só pode definir esta propriedade, não obtê-lo.

O argumento deve ser uma tupla de uma das seguintes formas:

- (x, y, width, height, corner_radius)
- (x, y, width, height, corner_radius, resolution)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4)
- (x, y, width, height, corner_radius1, corner_radius2, corner_radius3, corner_radius4, resolution)
- *x* e *y* representam a posição bottom-left (inferior esquerda) do retângulo.
- *width* e *height* representam o tamanho
- *corner_radius* é o número de pixels entre as duas bordas e o centro do arco do círculo juntando-os.
- resolução é o número de segmento de linha que será usado para desenhar o arco de círculo em cada canto (o padrão é 30)

A linha é automaticamente fechada.

Uso:

```
Line(rounded_rectangle=(0, 0, 200, 200, 10, 20, 30, 40, ↴100))
```

Novo na versão 1.9.0.

width

Determine a largura da linha, o padrão 1.0.

Novo na versão 1.4.1.

class kivy.graphics.vertex_instructions.Point

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma lista de 2 pontos. Cada ponto é representado como um quadrado com uma largura/altura de 2 vezes o: attr:*pointsize*.

Parameters

points: listLista de pontos no formato (x1, y1, x2, y2 ...), onde cada par de coordenadas especifica o centro de um novo ponto.

pointsize: float, e o padrão é 10 tamanho do ponto, medido a partir do centro para a borda. Um valor de 1,0 significa que o tamanho real será de 2.0 x 2.0.

Aviso: A partir da versão 1.0.7, a instrução de vértice tem um limite de 65535 vértices (índices de vértice para ser exato). 2 entradas na lista (x, y) serão convertidas em 4 vértices. Assim, o limite dentro da classe Point() será de $2^{15}-2$.

add_point()

Adicione um ponto à lista atua *points*.

Se pretender adicionar vários pontos, prefira utilizar este método em vez de reatribuir uma nova lista de *points*. A atribuição de uma nova lista de *points* irá recalcular e recarregar todo o buffer para a GPU. Se usares *add_point*, o mesmo só enviará as alterações.

points

Propriedade para obter/configurar o ponto central na lista de pontos. Cada par de coordenadas especifica o centro de um novo ponto.

pointsize

Propriedade para obter/configurar o tamanho do ponto. O tamanho é medido a partir do centro para a borda, então o valor igual a 1.0 significa que o tamanho real será 2.0 x 2.0.

class kivy.graphics.vertex_instructions.Mesh

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma Malha 2D.

Em OpenGL ES 2.0 e em nossa implementação gráfica, você não pode ter mais do que 65535 índices.

A lista de vértices é descrita como:

```

vertices = [x1, y1, u1, v1, x2, y2, u2, v2, ...]
          |     |   |   |
          +---+ i1 +---+ i2 +---+

```

Se desejas desenhar um triângulo, adicione 3 vértices. Você pode então fazer uma lista de índice da seguinte maneira:

```
indices = [0, 1, 2]
```

Novo na versão 1.1.0.

Parameters

vertices: iterableLista de vértices no formato (x1, y1, u1, v1, x2, y2, u2, v2...).

indices: iterableLista de índices no formato (i1, i2, i3...).

mode: strModo de VBO. Veja *mode* para maiores informações. O padrão é ‘points’.

fmt: listO formato para vértices, por padrão, cada vértice é descrito por coordenadas 2D (x, y) e uma textura 2D coordenada (u, v). Cada elemento da lista deve ser uma tupla ou lista, do formulário

(variable_name, size, type)

que permitirá mapear dados de vértices para as instruções glsl.

```
[('v_pos', 2, 'float'), ('v_tc', 2, 'float')]
```

Permitirá usar

atributo *vec2 v_pos*; atributo *vec2 v_tc*;

No sombreador de vértices do glsl.

Alterado na versão 1.8.1: Antes, *vertices* e ‘índices’ sempre eram convertidos numa lista, agora, eles só são convertidos numa lista se eles não implementarem a interface de buffer. Então, por exemplo, Numpy Arrays, Arrays Python e etc são usados no lugar, sem criar cópias adicionais. No entanto, os buffers não podem ser readonly (mesmo que eles não sejam alterados, devido a uma limitação do Cython) e devem ser contíguos na memória.

Nota: Ao passar um memoryview ou uma instância que implementa a interface do buffer, *vertices* deverá ser um buffer de floats (“f” Código em Python de Array) e *indices* deve ser um buffer de unsigned short (“H” Código em Python de Array). Arrays em outros formatos ainda terão de ser convertidos internamente, negando qualquer ganho potencial.

indices

Índices Vertex usados para especificar a ordem ao desenhar a malha.

mode

Modo VBO usado para desenhar vértices / índices. Pode ser uma das seguintes opções: 'points', 'line_strip', 'line_loop', 'lines', 'triangles', 'triangle_strip' or 'triangle_fan'.

vertices

Lista de coordenadas x, y, u, v usadas para construir a Malha. Agora, a instrução *Mesh* não permite que você altere o formato dos vértices, o que significa que é apenas $x, y +$ uma coordenada de textura.

class kivy.graphics.vertex_instructions.GraphicException

Bases: exceptions.Exception

Exceção gerada quando um erro gráfico é disparado.

class kivy.graphics.vertex_instructions.Bezier

Bases: *kivy.graphics.instructions.VertexInstruction*

Uma curva Bezier 2D.

Novo na versão 1.0.8.

Parameters

points: listLista de pontos no formato ($x_1, y_1, x_2, y_2\dots$)

segments: int, o padrão é 180Define quantos segmentos são necessários para desenhar uma curva. O desenho será mais suave quanto maior for a quantidade de segmentos.

loop: bool, o padrão é FalseDefina a curva Bezier para juntar o último ponto ao primeiro.

dash_length: intTamanho de um segmento (se tracejado), o padrão é 1.

dash_offset: intDistância entre o final do segmento e o início do próximo, o padrão é 0. Alterando isso torna-o tracejado.

dash_length

Propriedade para obter/configurar o tamanho do tracejado na curva.

dash_offset

Propriedade para obter/configurar o deslocamento do tracejado na curva.

points

Propriedade para obter/configurar os pontos do Triângulo.

Aviso: Isso sempre reconstruirá todo o gráfico da nova lista de pontos. Pode ter uso bastante intensivo de CPU.

segments

Propriedade para obter/configurar o número de segmentos numa curva.

class kivy.graphics.vertex_instructions.SmoothLine

Bases: [kivy.graphics.vertex_instructions.Line](#)

Linha experimental usando métodos de over-draw para obter melhores resultados anti-aliasing. Possui algumas desvantagens:

- Desenhar uma linha com alfa provavelmente não terá o resultado desejado da linha cruzando a si mesma.
- A propriedade [cap](#), [joint](#) e [dash](#) não são suportadas.
- Ele usa uma textura personalizada com um alfa premultiplied.
- Linhas com menos de 1px de largura não são suportadas: elas se parecem com as mesmas.

Aviso: Isso é uma trabalho não finalizado, experimental e sujeito a falhas.

Novo na versão 1.9.0.

overdraw_width

Determine a largura de overdraw da linha, o padrão é 1.2.

4.8 Gerenciador de Entrada

Nosso sistema de entrada é vasto e simples ao mesmo tempo. Estamos atualmente aptos a suportar:

- Eventos Multitouch do Windows (caneta e dedo)
- OS X touchpads
- Eventos Multitouch do Linux (Kernel e mtdev)
- Drivers Wacom Linux (caneta e dedo)
- TUIO

Todo os gerenciamento de entrada é configurável no Kivy no mod:[~kivy.config](#). Você pode facilmente usar dispositivos multitouch em uma aplicação Kivy.

Quando o evento tiver sido lido desde o dispositivo, eles são enviados através de um módulo de pré-processamento antes de ser enviado ao seu aplicativo. Temos também vários módulos padrão para:

- Detecção de Duplo Toque
- Diminuição do jittering

- Diminuição da imprecisão do toque em hardware “ruim” DIY
- Regiões Ignoradas

```
class kivy.input.MotionEvent(device, id, args)
    Bases: kivy.input.motionevent.MotionEvent
```

Classe abstrata que representa um evento de entrada (touch ou não).

Parameters

id: strID exclusivo do MotionEvent

args: listlista de parâmetros, passado para a função *depack()*

apply_transform_2d(transform)

Aplica uma transformação sobre x, y, z, px, py, pz, ox, oy, oz, dx, dy, dz

copy_to(to)

Copie algum atributo para outro objeto de toque.

depack(args)

Depack *args* em atributos de classe

distance(other_touch)

Retorna a instância entre o toque atual e outro toque.

dpos

Retorna o Delta entre a última posição e a posição atual, no sistema de coordenada da tela (*self.dx*, *self.dy*)

grab(class_instance, exclusive=False)

Pega este evento de movimento. Pode pegar (grab) um toque se quiseres receber os seguintes eventos:meth:*~kivy.uix.widget.Widget.on_touch_move* e *on_touch_up()* events, mesmo que o toque não seja despachado pelo pai:

```
def on_touch_down(self, touch):
    touch.grab(self)

def on_touch_move(self, touch):
    if touch.grab_current is self:
        # I received my grabbed touch
    else:
        # it's a normal touch

def on_touch_up(self, touch):
    if touch.grab_current is self:
        # I receive my grabbed touch, I must ungrab it!
        touch.ungrab(self)
    else:
        # it's a normal touch
    pass
```

is_mouse_scrolling

Retorna *True* se o toque é um mousewheel scrolling

Novo na versão 1.6.0.

move(args)

Move o toque para outra posição

opos

Retorna a posição inicial do toque no sistema de coordenada da tela (*self.ox*, *self.oy*)

pop()

Pop atributos de valores da pilha

ppos

Retorna a posição anterior do toque no sistema de coordenadas da tela (*self.px*, *self.py*)

push(attrs=None)

Envia valores de atributos em *attrs* para a pilha

scale_for_screen(w, h, p=None, rotation=0, smode='None', kheight=0)

Posição de escala para a tela

spos

Retorna a posição em um sistema de coordenada 0-1 (*self.sx*, *self.sy*)

ungrab(class_instance)

Desgrava um toque previamente detectado

class kivy.input.MotionEventProvider(device, args)

Bases: object

Classe base para um provedor.

start()

Inicia o provedor. Este método é automaticamente chamado quando a aplicação é iniciada e se a configuração usa o provedor atual.

stop()

Para o provedor.

update(dispatch_fn)

Atualiza o provedor e despacha todos os novos eventos de toque através do argumento *dispatch_fn*.

class kivy.input.MotionEventFactory

Fábrica de MotionEvent é uma classe que registra todas as fábricas de entrada disponíveis. Se você criar uma nova fábrica de entrada, precisarás registrá-la aqui:

```
MotionEventFactory.register('myproviderid', MyInputProvider)
```

```
static get(name)
```

Obtém uma classe provedora do ID do provedor

```
static list()
```

Obtém uma lista de todos os provedores disponíveis

```
static register(name, classname)
```

Registra um provedor de entra no database

4.8.1 Entrada de Pós Processamento

Calibration

Novo na versão 1.9.0.

Recalibrate input device to a specific range / offset.

Let's say you have 3 1080p displays, the 2 firsts are multitouch. By default, both will have mixed touch, the range will conflict with each others: the 0-1 range will goes to 0-5760 px (remember, $3 * 1920 = 5760$.)

To fix it, you need to manually reference them. For example:

```
[input]
left = mtdev,/dev/input/event17
middle = mtdev,/dev/input/event15
# the right screen is just a display.
```

Then, you can use the calibration postproc module:

```
[postproc:calibration]
left = xratio=0.3333
middle = xratio=0.3333,xoffset=0.3333
```

Now, the touches from the left screen will be within 0-0.3333 range, and the touches from the middle screen will be within 0.3333-0.6666 range.

```
class kivy.input.postproc.calibration.InputPostprocCalibration
    Bases: object
```

Recalibrate the inputs.

The configuration must go within a section named *postproc:calibration*. Within the section, you must have line like:

```
devicename = param=value,param=value
```

Parameters

*xratio: float*Value to multiply X

yratio: float Value to multiply Y
xoffset: float Value to add to X
yoffset: float Value to add to Y

Dejitter

Prevenir o efeito “bolha tremendo”.

Um problema que muitas vezes é enfrentado (especialmente em configurações de MT óptico) é o de BLOBs jitterish causado por características de câmeras ruins. Com este módulo poderás se livrar desse Jitter. Apenas defina um limiar *jitter_distance* em sua configuração, e todos os movimentos de toque que movem o toque por menos que a distância de jitter serão considerados movimentos ruins causados por Jitter e serão descartados.

class kivy.input.postproc.dejitter.*InputPostprocDejitter*
Bases: object

Livrando-se de BLOBs jitterish. Exemplo:

```
[postproc]
jitter_distance = 0.004
jitter_ignore_devices = mouse,mactouch
```

Configuration

jitter_distance: float Um float no intervalo 0-1.

jitter_ignore_devices: string Uma lista separada por vírgulas de identificadores de dispositivo que não devem ser processados por Dejitter (porque os mesmos não são muito precisos).

Double Tap

Procurar toque para toque duplo

class kivy.input.postproc.doubletap.*InputPostprocDoubleTap*
Bases: object

InputPostProcDoubleTap é um pós-processador para verificar se um toque é um simples toque ou então um toque duplo. O toque duplo pode ser configurado no arquivo de configuração Kivy:

```
[postproc]
double_tap_time = 250
double_tap_distance = 20
```

O parâmetro *Distance* está no intervalo de 0-1000 e o tempo está definido em milissegundos.

find_double_tap(*ref*)

Encontre um duplo toque dentro de *self.touches*. O toque não deve ser um toque duplo anterior e a distância deve estar dentro do limite especificado. Além disso, os perfis de toque devem ser do mesmo tipo de toque.

Lista ignorada

Ignora o Toque em algumas áreas da tela

class kivy.input.postproc.ignorelist.InputPostprocIgnoreList

Bases: object

InputPostprocIgnoreList é um pós-processador que remove toques na Lista a Ignorar. A Lista a Ignorar pode ser configurada no arquivo de configuração Kivy:

```
[postproc]
# Format: [(xmin, ymin, xmax, ymax), ...]
ignore = [(0.1, 0.1, 0.15, 0.15)]
```

As coordenadas da Lista a Ignorar estão no intervalo entre 0-1, não em pixels de tela.

Manter o toque

Reutilizar o toque para conter o comportamento do dedo perdido

class kivy.input.postproc.retaintouch.InputPostprocRetainTouch

Bases: object

InputPostprocRetainTouch é um pós-processador para atrasar o evento ‘up’ de um toque, para reutilizá-lo sob determinadas condições. Este módulo é projetado para evitar toques de dedo perdidos em alguns hardware/configurações.

Retain touch pode ser configurado no arquivo de configuração Kivy:

```
[postproc]
retain_time = 100
retain_distance = 50
```

O parâmetro de distância está no intervalo entre 0-1000 e o tempo está definido em milissegundos.

Triple Tap

Novo na versão 1.7.0.

Procurar toque para toque triplo

```
class kivy.input.postproc.tripletap.InputPostprocTripleTap
    Bases: object
```

InputPostProcTripleTap é um pós-processador para verificar se um toque é um toque triplo ou não. Toque triplo pode ser configurado no arquivo de configuração do Kivy:

```
[postproc]
triple_tap_time = 250
triple_tap_distance = 20
```

O parâmetro de distância está no intervalo entre 0-1000 e o tempo está definido em milissegundos.

find_triple_tap(ref)

Encontre um toque triplo dentro de *self.touches*. O toque não deve ser um toque triplo anterior e a distância deve estar dentro dos limites especificados. Além disso, o perfil de toque deve ser do mesmo tipo de toque.

4.8.2 Provedores

Entrada do Provedor Joystick do Android

Este módulo é baseado no provedor de entrada JoyStick do PyGame. Para maiores informações, veja a documentação <http://www.pygame.org/docs/ref/joystick.html>

Provedor de Entrada do Auto-Criador da entrada de configuração disponibilizada pela MT Hardware (Somente para Linux).

Obrigado ao Marc Tardiff por fornecer o código, retirado de *scan-for-mt-device*.

A descoberta do dispositivo é feita pelo provedor. De qualquer forma, a leitura de entrada pode ser feita por outros provedores, tais como: *hidinput*, *mtdev* e *linuxwacom*. *mtdev* é usado antes para criar outros provedores. Para mais informações sobre *mtdev*, veja *mtdev*.

Aqui temos um exemplo de criação automática:

```
[input]
# using mtdev
device_%(name)s = probesysfs,provider=mtdev
# using hidinput
device_%(name)s = probesysfs,provider=hidinput
# using mtdev with a match on name
device_%(name)s = probesysfs,provider=mtdev,match=acer
```

```

# using hidinput with custom parameters to hidinput (all on one_
↳ line)
%(name)s = probesysfs,
    provider=hidinput,param=min_pressure=1,param=max_pressure=99

# you can also match your wacom touchscreen
touch = probesysfs,match=E3 Finger,provider=linuxwacom,
    select_all=1,param=mode=touch
# and your wacom pen
pen = probesysfs,match=E3 Pen,provider=linuxwacom,
    select_all=1,param=mode=pen

```

Por padrão, o módulo *ProbeSysfs* enumerará o hardware desde o dispositivo */sys/class/input*, e configurará o hardware com capacidade *ABS_MT_POSITION_X*. Por exemplo, a tela *wacom* não suporta esta capacidade. Você pode evitar este comportamento definindo *select_all=1* em suas linhas de configurações.

Definições padrões para Provedores do Windows

Este arquivo fornece a definição comuns às constantes usadas por *WM_Touch*/*WM_Pen*.

Leap Motion - somente dedo

Implementação do Provedor de Mouse

Em sistemas Linux, o provedor de mouse pode ser irritando quando usado com outro provedor MultiTouch (hidinput ou mtdev). O Mouse pode entrar em conflito com: um único toque pode gerar um evento do provedor do mouse e outro do provedor de MultiTouch.

Para evitar este comportamento, você pode ativar o token *disable_on_activity* nas configurações do Mouse. Em seguida, se alguns toques são criados por outro provedor, o evento do mouse será descartado. Adicione isto à sua configuração:

```

[input]
mouse = mouse,disable_on_activity

```

Usando interação com MultiTouch com o mouse

Novo na versão 1.3.0.

Por padrão, os botões do centro e da direita do Mouse, como também a combinação de Ctrl + botão esquerdo do Mouse, são usados para emulação MultiTouch. Se você dese-

jar usa-las para outro propósito, podes desabilitar este comportamento pela ativação do token *disable_multitouch*:

```
[input]
mouse = mouse, disable_multitouch
```

Alterado na versão 1.9.0.

Agora você pode controlar seletivamente se um clique iniciado como descrito acima irá emular MultiTouch. Se o toque tiver sido iniciado da maneira acima (por exemplo, botão direito do mouse), um valor *multitouch_sim* será adicionado ao perfil do toque e uma propriedade *multitouch_sim* será adicionada ao toque. Por padrão, *multitouch_sim* é *True* e MultiTouch será emulado para esse toque. Se, no entanto, *multitouch_on_demand* for adicionado ao config:

```
[input]
mouse = mouse, multitouch_on_demand
```

então o padrão para *multitouch_sim* será *False*. Neste caso, se *multitouch_sim* for definido como *True* antes do mouse ser solto (por exemplo, em *on_touch_down/move*), o toque simulará um evento MultiTouch. Por exemplo:

```
if 'multitouch_sim' in touch.profile:
    touch.multitouch_sim = True
```

Segue uma lista de valores suportados para a lista de propriedades *profile*.

Valor do Perfil (Profile value)	Descrição
botão	Botão do Mouse (um de <i>left</i> , <i>right</i> , <i>middle</i> , <i>scrollup</i> ou <i>scrolldown</i>). Acessados através da propriedade <i>button</i> .
pos	Posição 2D. Também reflete nas propriedades <i>x</i> , <i>y</i> e <i>pos</i> .
<i>multitouch_sim</i>	Especifica se o MultiTouch é ou não simulado. Acessando através das propriedade <i>multitouch_sim</i> .

Suporte nativo para entrada HID do kernel do linux

O suporte começa em 2.6.32-Ubuntu, ou 2.6.34.

Para configurar o HIDInput, adicione isso a sua configuração:

```
[input]
# devicename = hidinput,/dev/input/eventXX
# example with Stantum MTP4.3" screen
stantum = hidinput,/dev/input/event2
```

Nota: Você deve ter acesso de leitura ao evento de entrada.

Você pode usar um range padrão para os valores da pressão de X, Y. Para vários drivers, o range reportado é inválido. Para corrigir isso, você deve adicionar estas opções para a linhas de argumentos:

- *invert_x*: 1 para inverter o eixo X
- *invert_y*: 1 para inverter o eixo y
- *min_position_x*: Mínimo X
- *max_position_x*: máximo X
- *min_position_y*: mínimo Y
- *max_position_y*: máximo Y
- *min_pressure*: pressão mínima
- *max_pressure*: pressão máxima
- *rotation*: giro da coordenada de entrada (0, 90, 180, 270)

Por exemplo, num Asus T101M, o TouchScreen reporta um range entre 0-4095 para os valores de X e o Y, mas os valores reais estão no range entre 0-32768. Para corrigir isso, podes adicionar a seguinte configuração:

```
[input]
t101m = hidinput,/dev/input/event7,max_position_x=32768,max_
        position_y=32768
```

Novo na versão 1.9.1: *rotation*: adiciona o token de configuração.

Suporte nativo a dispositivos com Multitoques em Linux utilizando libmtdev.

O projeto Mtdev é parte da arquitetura do Ubuntu Maverick. Para maiores informações acesse: <http://wiki.ubuntu.com/Multitouch>

Para configurar o MTDev, é preferível usar o provedor *probesysfs*'. Veja a classe a seguir para maiores informações :py:class:'~kivy.input.providers.probesysfs'.

Do contrário, adicione isso a suas configurações:

```
[input]
# devicename = hidinput,/dev/input/eventXX
acert230h = mtdev,/dev/input/event2
```

Nota: Você deve ter acesso de leitura ao evento de entrada.

Você pode usar um range de valor de pressão padrão para X e Y. Em alguns drivers, o range reportado é inválido. Para corrigir isso, você pode adicionar estas opções para a linha de argumento:

- *invert_x*: 1 para inverter o eixo X
- *invert_y*: 1 para inverter o eixo y
- *min_position_x*: Mínimo X
- *max_position_x*: máximo X
- *min_position_y*: mínimo Y
- *max_position_y*: máximo Y
- *min_pressure*: pressão mínima
- *max_pressure*: pressão máxima
- *min_touch_major*: largura mínima do shape (forma)
- *max_touch_major*: largura máxima do shape (forma)
- *min_touch_minor*: largura mínima do shape (forma)
- *max_touch_minor*: altura máxima do shape (forma)
- *totation*: 0,90,180 ou 270 para girar

Suporte nativo do framework MultitouchSupport para MacBook (plataforma Mac OS X)

Suporte nativo do tablet Wacom do driver linuxwacom Drive de suporte nativo com linuxwacom da Wacom

Para configurar o LinuxWacom, adicione isso a suas configurações:

```
[input]
pen = linuxwacom,/dev/input/event2,mode=pen
finger = linuxwacom,/dev/input/event3,mode=touch
```

Nota: Você deve ter acesso de leitura ao evento de entrada.

Você pode usar um range de valor de pressão padrão para X e Y. Em alguns drivers, o range reportado é inválido. Para corrigir isso, você pode adicionar estas opções para a linha de argumento:

- *invert_x*: 1 para inverter o eixo X
- *invert_y*: 1 para inverter o eixo y
- *min_position_x*: Mínimo X
- *max_position_x*: máximo X
- *min_position_y*: mínimo Y

- *max_position_y*: máximo Y
- *min_pressure*: pressão mínima
- *max_pressure*: pressão máxima

Suporte para mensagens WM_PEN (plataforma Windows)

```
class kivy.input.providers.wm_pen.WM_Pen(device, id, args)
Bases: kivy.input.motionevent.MotionEvent
```

MotionEvent representa o evento WM_Pen. Suporta o perfil pos. (Supports the pos profile).

Suporte para mensagem WM_TOUCH (Plataforma Microsoft Windows)

```
class kivy.input.providers.wm_touch.WM_MotionEvent(device, id, args)
Bases: kivy.input.motionevent.MotionEvent
```

MotionEvent representa o evento WM_MotionEvent. Suporta pos, shape e size profiles.

Provedor de Entrada TUIO

TUIO é *de facto* o protocolo padrão de rede para a transmissão de toque e fiducial informação entre servidor e cliente. Para aprender mais sobre TUIO (que também está baseado no protocolo OSC), acesse o link <http://tuio.org> – A especificação deve ser de especial interesse.

Configura um provedor TUIO no arquivo *config.ini*

O provedor TUIO pode ser configurado nas configurações de arquivo na seção de entrada [input]:

```
[input]
# name = tuio,<ip>:<port>
multitouchtable = tuio,192.168.0.1:3333
```

Configura um provedor TUIO na aplicação

Você deve adicionar um provedor antes que a sua aplicação esteja executando, como este:

```

from kivy.app import App
from kivy.config import Config

class TestApp(App):
    def build(self):
        Config.set('input', 'multitouchscreen1', 'tuio,0.0.0.0:3333')
        # You can also add a second TUIO listener
        # Config.set('input', 'source2', 'tuio,0.0.0.0:3334')
        # Then do the usual things
        # ...
        return

```

**class kivy.input.providers.tuio.TuioMotionEventProvider(device,
args)**

Bases: *kivy.input.provider.MotionEventProvider*

O provedor TUIO escuta o socket e manipula algumas mensagens OSC:

- /tuio/2Dcur
- /tuio/2Dobj

Você pode facilmente estender o provedor para lidar com novos PATHs TUIO como:

```

# Create a class to handle the new TUIO type/path
# Replace NEWPATH with the pathname you want to handle
class TuioNEWPATHMotionEvent(MotionEvent):
    def __init__(self, id, args):
        super(TuioNEWPATHMotionEvent, self).__init__(id, args)

    def depack(self, args):
        # In this method, implement 'unpacking' for the received
        # arguments. you basically translate from TUIO args to
        # Kivy
        # MotionEvent variables. If all you receive are x and y
        # values, you can do it like this:
        if len(args) == 2:
            self.sx, self.sy = args
            self.profile = ('pos', )
        self.sy = 1 - self.sy
        super(TuioNEWPATHMotionEvent, self).depack(args)

    # Register it with the TUIO MotionEvent provider.
    # You obviously need to replace the PATH placeholders
    # appropriately.
TuioMotionEventProvider.register('/tuio/PATH',
    TuioNEWPATHMotionEvent)

```

Nota: O nome da classe não é de importância técnica. Sua classe será associada com o path que você passar à função `register()`. Para mentar isso simples, você deve nomear suas classes após o caminho que utiliza.

static create(*oscpath*, ***kargs*)

Cria um evento de toque a partir de um caminho *TUIO*

static register(*oscpath*, *classname*)

Registrar um novo caminho para manipular no provedor *TUIO*

start()

Iniciar o provedor *TUIO*

stop()

Para o provedor *TUIO*

static unregister(*oscpath*, *classname*)

Cancelar o registro do caminho para parar de tratá-lo no provedor *TUIO*

update(*dispatch_fn*)

Atualiza o provedor *TUIO* (pop o evento da fila)

class kivy.input.providers.tuio.Tuio2dCurMotionEvent(*device*, *id*,
 args)

Bases: `kivy.input.providers.tuio.TuioMotionEvent`

Um toque *TUIO* 2dCur.

class kivy.input.providers.tuio.Tuio2dObjMotionEvent(*device*, *id*,
 args)

Bases: `kivy.input.providers.tuio.TuioMotionEvent`

Um objeto *TUIO* 2dObj.

4.8.3 Entrada de Gravação

Novo na versão 1.1.0.

Aviso: Esta parte do Kivy está ainda em fase experimental e este API está sujeito a mudança em versões posteriores.

Esta é uma classe que pode gravar e reproduzir alguns eventos de entrada. Isso pode ser usado para casos de teste, protetores de tela etc.

Uma vez ativado, o gravador irá ouvir qualquer evento de entrada e salvar suas propriedades em um arquivo com a variação do tempo. Mais tarde, você poderá tocar o arquivo de entrada. Ele gerará eventos touch falsos com as propriedades salvas e o mandará para o loop de eventos.

Por padrão, apenas a posição é salva ('pos' profile e 'sx', 'sy', atributos). Altere-os apenas se você entender como funciona a manipulação de entrada.

Gravando Eventos

A melhor forma para usar o módulo “recorder”. Veja a documentação [Módulos](#) para maiores informações.

Uma vez ativado, podes pressionar F8 para iniciar a gravação. Por padrão, os eventos serão gravados em `<currentpath>/recorder.kvi`. Quando quiser interromper a gravação, pressione F8 novamente.

Você pode reproduzir o arquivo pressionando F7.

Veja a API [Módulo de Gravação](#) do módulo para maiores informações.

Reprodução manual

Você pode abrir manualmente um arquivo do gravador e reproduzi-lo da seguinte maneira:

```
from kivy.input.recorder import Recorder

rec = Recorder(filename='myrecorder.kvi')
rec.play = True
```

Se você quiser fazer um loop sobre esse arquivo, podes fazê-lo da seguinte maneira:

```
from kivy.input.recorder import Recorder

def recorder_loop(instance, value):
    if value is False:
        instance.play = True

rec = Recorder(filename='myrecorder.kvi')
rec.bind(play=recorder_loop)
rec.play = True
```

Gravando mais atributos

Podes estender os atributos para salvar em uma condição: valores de atributos devem ser valores simples, não instâncias de classes complexas.

Digamos que você deseja salvar o ângulo e a pressão do toque, se disponível:

```
from kivy.input.recorder import Recorder
```

```

rec = Recorder(filename='myrecorder.kvi',
    record_attrs=['is_touch', 'sx', 'sy', 'angle', 'pressure'],
    record_profile_mask=['pos', 'angle', 'pressure'])
rec.record = True

```

Ou com variáveis de módulos:

```

$ python main.py -m recorder,attrs=is_touch:sx:sy:angle:pressure,
  ↪           profile_mask=pos:angle:pressure

```

Limitações conhecidas

- Não é possível salvar atributos com instâncias de classes complexas.
- Os valores que representam o tempo não serão ajustados.
- Pode reproduzir apenas registros completos. Se um evento de início/atualização/finalização estiver ausente, isso pode levar a “toques de fantasmas” (ghost touches).
- Parar o Replay antes do fim pode levar a toques fantasma.

class kivy.input.recorder.Recorder(kwargs)**
Bases: *kivy.event.EventDispatcher*

Classe Recorder. Veja a documentação do módulo para maiores informações.

Events

on_stop: Disparado quando a reprodução pára.

Alterado na versão 1.10.0: Evento *on_stop* adicionado.

counter

Número de eventos gravados na última sessão.

counter é uma *NumericProperty* e o padrão é 0, somente leitura.

filename

Nome do arquivo para salvar a saída do gravador.

filename é uma *StringProperty* e o padrão é ‘recorder.kvi’.

play

Boolean para iniciar/parar a repetição do arquivo atual (se existir).

play é uma *BooleanProperty* e o padrão é *False*.

record

Boolean para iniciar/parar a gravação de eventos de entrada.

record é uma *BooleanProperty* e o padrão é *False*.

recordAttrs

Atributos para gravar a partir do evento de movimento.

recordAttrs é uma *ListProperty* e o padrão é ['is_touch', 'sx', 'sy'].

recordProfileMask

Perfil para salvar no evento de movimento falso quando reproduzido.

recordProfileMask é uma *ListProperty* e o padrão é ['pos'].

window

Instância da janela para anexar o gravador. Se *None*, será usado a instância padrão.

window é uma *ObjectProperty* e o padrão é *None*.

4.8.4 Eventos de Movimento

A *MotionEvent* é a classe base usada para eventos fornecidos por dispositivos apontadores (toque e não toque). Esta classe define todas as propriedades e métodos necessários para lidar com movimentos 2D e 3D, mas tem muitos mais recursos.

Nota: Você nunca cria a *MotionEvent* você mesmo: este é o papel do *providers*.

Evento de Movimento e Toque

Diferenciamos um Motion Event e um Touch Event. Um evento Touch é uma *MotionEvent* com o profile *pos*. Somente esses eventos são despachados por toda a árvore de Widgets.

1. A *MotionEvent*'s são coletados de provedores de entrada.
2. Todos os *MotionEvent*'s são despachados de *on_motion()*.
3. Se uma *MotionEvent* tem um perfil *pos*, nós despachamos através de *on_touch_down()*, *on_touch_move()* e *on_touch_up()*.

Escuta para um Evento de Movimento

Se você quiser receber todos os *MotionEvents*, seja de Toque ou não, vincule o *MotionEvent* da *Window* para o seu próprio callback:

```
def on_motion(self, etype, motionevent):
    # will receive all motion events.
    pass

Window.bind(on_motion=on_motion)
```

Você pode escutar as modificações da posição do mouse pelo assistindo `mouse_pos`.

Perfis

A `MotionEvent` armazena informações específicas do dispositivo em várias propriedades listadas no `profile`. Por exemplo, podes receber um MotionEvent que tenha um ângulo, um ID fiducial ou mesmo uma Shape (forma). Pode verificar o atributo `profile` para ver o que atualmente é suportado pelo provedor MotionEvent.

Esta é uma pequena lista dos valores de perfil suportados por padrão. Verifique a propriedade `MotionEvent.profile` para ver mais valores de perfil disponíveis.

Valor do Perfil (Profile value)	Descrição
angle	Ângulo 2D. Acessado através da propriedade <code>a</code> .
botão	Mouse button ('left', 'right', 'middle', 'scrollup' or 'scrolldown'). Acessado através da propriedade <code>button</code> .
markerid	Marcador ou ID Fiducial. Acessado através da propriedade <code>fid</code> .
pos	Posição 2D. Acessado através das propriedades <code>x</code> , <code>y</code> ou <code>pos</code> .
pos3d	Posição 3D. Acessado através das propriedades <code>x</code> , <code>y</code> ou <code>z</code> .
pressure	Pressão do contato. Acessado através da propriedade <code>pressure</code> .
shape	Forma do contato. Acessado através da propriedade <code>shape</code> .

Se você desejar conhecer se a atual `MotionEvent` tem um ângulo:

```
def on_touch_move(self, touch):
    if 'angle' in touch.profile:
        print('The touch angle is', touch.a)
```

Se você quiser selecionar apenas os fiduciais:

```
def on_touch_move(self, touch):
    if 'markerid' not in touch.profile:
        return
```

`class kivy.input.motionevent.MotionEvent(device, id, args)`

Bases: `kivy.input.motionevent.MotionEvent`

Classe abstrata que representa um evento de entrada (touch ou não).

Parameters

`id: str`ID exclusivo do MotionEvent

`args: list`lista de parâmetros, passado para a função `depak()`

`apply_transform_2d(transform)`

Aplica uma transformação sobre x, y, z, px, py, pz, ox, oy, oz, dx, dy, dz

`copy_to(to)`

Copie algum atributo para outro objeto de toque.

depack(args)

Depack args em atributos de classe

device = None

Dispositivo usado pra criar este toque

distance(other_touch)

Retorna a instância entre o toque atual e outro toque.

double_tap_time = None

Se o toque é um *is_double_tap*, este é um tempo entre o toque anterior e o toque atual.

dpos

Retorna o Delta entre a última posição e a posição atual, no sistema de coordenada da tela (*self.dx*, *self.dy*)

dsx = None

Delta entre *self.sx* e *self.psx*, no intervalo de 0-1.

dsy = None

Delta entre *self.sy* e *self.psy*, no intervalo de 0-1.

dsz = None

Delta entre *self.sz* e *self.psz*, no intervalo de 0-1.

dx = None

Delta entre *self.x* e *self.px*, no intervalo da janela

dy = None

Delta entre *self.y* e *self.py*, no intervalo da janela

dz = None

Delta entre *self.z* e *self.pz*, no intervalo da janela

grab(class_instance, exclusive=False)

Pega este evento de movimento. Pode pegar (grab) um toque se quiseres receber os seguintes eventos:meth:*~kivy.uix.widget.Widget.on_touch_move* e *on_touch_up()* events, mesmo que o toque não seja despachado pelo pai:

```
def on_touch_down(self, touch):
    touch.grab(self)

def on_touch_move(self, touch):
    if touch.grab_current is self:
        # I received my grabbed touch
    else:
        # it's a normal touch

def on_touch_up(self, touch):
    if touch.grab_current is self:
        # I receive my grabbed touch, I must ungrab it!
```

```

        touch.ungrab(self)
    else:
        # it's a normal touch
        pass

```

grab_current = None

Usado para determinar qual é o Widget que o toque está sendo despachado. Verifique a função [grab\(\)](#) para obter mais informações.

id = None

O *id* do toque, não único. Geralmente, é o ID definido pelo provedor de entrada, como ID no TUIO. Se tiveres várias fontes TUIO, o mesmo id pode ser usado. Em vez disso, prefira usar o atributo [uid](#).

is_double_tap = None

Indica se o toque é ou não um toque duplo

is_mouse_scrolling

Retorna *True* se o toque é um mousewheel scrolling

Novo na versão 1.6.0.

is_touch = None

True se o MotionEvent for um toque. Também pode ser verificado que *pos* é [profile](#).

is_triple_tap = None

Indica se o toque é ou não um toque triplo

Novo na versão 1.7.0.

move(args)

Move o toque para outra posição

opos

Retorna a posição inicial do toque no sistema de coordenada da tela (*self.ox*, *self.oy*)

osx = None

Origem X posição, no intervalo entre 0-1.

osy = None

Origem Y posição, no intervalo entre 0-1.

osz = None

Origem Z posição, no intervalo entre 0-1.

ox = None

Origem X posição, no intervalo da janela

oy = None

Origem Y posição, no intervalo da janela

oz = None

Origem Z posição, no intervalo da janela

pop()

Pop atributos de valores da pilha

pos = None

Posição (X, Y), no intervalo da janela

ppos

Retorna a posição anterior do toque no sistema de coordenadas da tela (*self.px, self.py*)

profile = None

Perfil atual usado no toque

psx = None

Posição X anterior, na faixa 0-1.

psy = None

Posição Y anterior, na faixa 0-1.

psz = None

Posição Z anterior, na faixa 0-1.

push(*attrs=None*)

Envia valores de atributos em *attrs* para a pilha

push_attrs_stack = None

Atributos para empurrar por padrão, quando usamos **push()** : x, y, z, dx, dy, dz, ox, oy, oz, px, py, pz.

px = None

Posição X anterior, no intervalo da janela

py = None

Posição Y anterior, no intervalo da janela

pz = None

Posição Z anterior, no intervalo da janela

scale_for_screen(*w, h, p=None, rotation=0, smode='None', kheight=0*)

Posição de escala para a tela

shape = None

Forma do toque, subclasse de **Shape**. Por padrão a propriedade é definida como sendo *None*.

spos

Retorna a posição em um sistema de coordenada 0-1 (*self.sx, self.sy*)

sx = None

Posição X, no intervalo 0-1.

sy = None

Posição Y, no intervalo 0-1.

sz = None

Posição Z, no intervalo 0-1.

time_end = None

Hora do evento final(uso do último toque)

time_start = None

Tempo inicial da criação do toque

time_update = None

Horário da última atualização

triple_tap_time = None

Se o toque for *is_triple_tap*, este é o tempo entre o primeiro toque e o toque atual.

Novo na versão 1.7.0.

ud = None

Dicionário de dados do usuário. Use este dicionário para salvar seus próprios dados de toque.

uid = None

ID exclusivo do toque. Podes usar essa propriedade com segurança, nunca será o mesmo entre todos os toques existentes.

ungrab(*class_instance*)

Desgrava um toque previamente detectado

x = None

Posição X, no intervalo da janela.

y = None

Posição Y, no intervalo da janela.

z = None

Posição Z, no intervalo da janela.

4.8.5 Fábrica de Eventos de Movimentos

Fábrica de provedores *MotionEvent*.

class kivy.input.factory.MotionEventFactory

Fábrica de MotionEvent é uma classe que registra todas as fábricas de entrada disponíveis. Se você criar uma nova fábrica de entrada, precisarás registrá-la aqui:

```
MotionEventFactory.register('myproviderid', MyInputProvider)
```

static get (name)

Obtém uma classe provedora do ID do provedor

static list()

Obtém uma lista de todos os provedores disponíveis

static register (name, classname)

Registra um provedor de entra no database

4.8.6 Provedor de Eventos de Movimento

Classe abstrata para a implementação de um provedor **MotionEvent**. A implementação deve suportar os métodos **start()**, **stop()** e **update()** methods.

class kivy.input.provider.MotionEventProvider (device, args)

Bases: object

Classe base para um provedor.

start()

Inicia o provedor. Este método é automaticamente chamado quando a aplicação é iniciada e se a configuração usa o provedor atual.

stop()

Para o provedor.

update (dispatch_fn)

Atualiza o provedor e despacha todos os novos eventos de toque através do argumento *dispatch_fn*.

4.8.7 Forma do Evento de Movimento

Representa a forma de **MotionEvent**

class kivy.input.shape.Shape

Bases: object

Classe abstrata para todos as implementações de formas

class kivy.input.shape.ShapeRect

Bases: **kivy.input.shape.Shape**

Classe para a representação de um retângulo.

height

Altura de um rect

width

Largura de um rect

4.9 Linguagem Kivy

A linguagem Kivy é dedicada a descrever interfaces de usuário e suas interações. Kivy pode ser comparada às linguagens Qt e QML (<http://qt.nokia.com>), mas foram incluídos novos conceitos tal como regras e definições (parecidas com as usadas em CSS), modelagens e assim por diante.

Alterado na versão 1.7.0: O Construtor não executa mais as expressões no canvas em tempo real. Ele irá juntar todas as expressões que precisam ser executadas primeiro e executar após enviar as informações das entradas, logo antes de desenhar a tela. Se você quer forçar a execução do desenho no canvas, chame `Builder.sync`.

Uma ferramenta experimental de criação de perfis para o kv lang também está incluso. Podes ativá-la definindo a variável de ambiente `KIVY_PROFILE_LANG = 1`. Em seguida, será gerado um arquivo HTML chamado `builder_stats.html`.

4.9.1 Visão Geral

A linguagem consiste de diversas estruturas que são usadas como:

Regras Uma regra na linguagem KV é semelhante a uma regra CSS. As regras se aplicam a Widgets específicos (ou classes) em sua árvore de Widgets e modifica-os de uma determinada maneira. Podes usar regras para especificar o comportamento interativo ou usá-los para adicionar representações gráficas dos Widgets aos quais eles se aplicam. Pode ter uma classe ou Widgets como alvo (semelhante ao conceito de CSS de uma *classe*) usando o atributo `cls` (por exemplo `cls = MyTestWidget`).

Um Você pode usar a linguagem para criar toda sua interface de usuário. Um arquivo kv deve conter apenas um widget raiz no máximo.

Classes Dinâmicas (*Introduzido na versão 1.7.0*) As classes dinâmicas permitem que você crie novos Widgets e regras on-the-fly, sem qualquer declaração Python.

Templates (em desuso) (*introduzido na versão 1.0.5, tornando-se obsoleto na versão 1.7.0*) Modelos foram usados para preencher partes de um aplicativo, como nomear o conteúdo de uma lista (por exemplo, ícone à esquerda, texto à direita). Eles agora são depreciados por classes dinâmicas.

4.9.2 Sintaxe de um Arquivo kv

Um arquivo de linguagem Kivy deve ter ".kv" como extensão do nome do arquivo

O conteúdo do arquivo deve sempre começar com o cabeçalho Kivy, onde a ‘Versão’ deve ser substituído com a versão da linguagem Kivy que você está usando. Por enquanto, use 1.0:

```
#:kivy `1.0`  
  
# content here
```

O ‘conteúdo’ pode conter definições de regra

```
# Syntax of a rule definition. Note that several Rules can share  
the same  
# definition (as in CSS). Note the braces: they are part of the  
definition.  
<Rule1,Rule2>:  
    # .. definitions ..  
  
<Rule3>:  
    # .. definitions ..  
  
# Syntax for creating a root widget  
RootClassName:  
    # .. definitions ..  
  
# Syntax for creating a dynamic class  
<NewWidget@BaseClass>:  
    # .. definitions ..  
  
# Syntax for create a template  
[TemplateName@BaseClass1,BaseClass2]:  
    # .. definitions ..
```

Independentemente se é uma regra, Widget principal, classe dinâmica ou Template que estás definindo, a definição deve se parecer com o que temos abaixo:

```
# With the braces it's a rule. Without them, it's a root widget.  
<ClassName>:  
    prop1: value1  
    prop2: value2  
  
    canvas:  
        CanvasInstruction1:  
            canvasprop1: value1  
        CanvasInstruction2:  
            canvasprop2: value2  
  
    AnotherClass:  
        prop3: value1
```

Aqui `prop1` e `prop2` são as propriedades de `ClassName` e' `prop3'` é a propriedade de `AnotherClass`. Se o Widget não tiver uma propriedade com o nome definido, uma classe:`~kivy.properties.ObjectProperty` será automaticamente criada e adicionada ao Widget.

`AnotherClass` será criada e adicionada como um filho da instância de `ClassName`.

- A indentação é importante e deve ser utilizada. O espaçamento deve ser múltiplo do número de espaços usados na primeira linha indentada. Espaços são incentivados: não é recomendado mesclar tabulações e espaços.
- O valor de uma propriedade deve ser dada em uma linha única (ao menos por enquanto)
- A propriedade 'canvas' é específica: você pode colocar instruções gráficas nela para criar uma representação gráfica da classe atual

Aqui está um exemplo simples de um arquivo kv que contém um widget raiz

```
#:kivy 1.0

Button:
    text: 'Hello world'
```

Alterado na versão 1.7.0: A indentação não está mais limitado a 4 espaços. O espaçamento deve ser um múltiplo do número de espaços usados na primeira linha indentada.

Ambos `load_file()` e o método `load_string()` Retornam o Widget principal definido no seu arquivo kv/String. Eles também adicionarão quaisquer definições de classe e Template à classe `Factory` para ser usado posteriormente.

4.9.3 Expressões de valor, expressões on_property, ids e palavras-chave reservadas

Quando você especifica o valor de uma propriedade, o valor é avaliado como uma expressão Python. Essa expressão pode ser estática ou dinâmica, o que significa que o valor pode usar os valores de outras propriedades usando palavras-chave reservadas.

`self` A palavra-chave `self` contém a referência à "atual instância do Widget":

```
Button:
    text: 'My state is %s' % self.state
```

`root` Esta palavra-chave está disponível apenas nas definições de regra e representa o Widget principal da regra (a primeira instância definidas nas regras com a linguagem kv):

```
<MyWidget>:  
    custom: 'Hello world'  
    Button:  
        text: root.custom
```

app Esta palavra-chave sempre se refere a instância principal da aplicação. É equivalente a uma chamada a `kivy.app.App.get_running_app()` em Python:

```
Label:  
    text: app.name
```

args Esta palavra-chave está disponível no callbacks `on_<action>`. Refere-se aos argumentos passados para o callback:

```
TextInput:  
    on_focus: self.insert_text("Focus" if args[1] else  
        "No focus")
```

ids

Definição de classe pode conter ids que podem ser usados como keywords:

```
<MyWidget>:  
    Button:  
        id: btn1  
    Button:  
        text: 'The state of the other button is %s' % btn1.state
```

Observe que o *id* não estará disponível na instância do Widget: ele é usado exclusivamente para referências externas. *id* é uma referência fraca (weakref) ao Widget e ao invés do Widget propriamente dito. O Widget propriamente dito pode ser acessado com *id.__self__* (*btn1.__self__* neste caso).

Quando o arquivo *kv* é processado, todas as weakref dos Widgets são adicionados à propriedades *ids* do Widget raiz. Em outras palavras, seguindo o exemplo acima, o estado dos botões também pode ser acessado da seguinte maneira:

```
widget = MyWidget()  
state = widget.ids["btn1"].state  
  
# Or, as an alternative syntax,  
state = widget.ids.btn1.state
```

Observe que o Widget mais externo aplica as regras *kv* a todos os seus Widgets internos antes que outras regras sejam aplicadas. Isto significa que se um Widget interno

contiver *ids*, esses *ids* podem não estar disponíveis durante a função `__init__` do Widget interno.

Expressão Válida

Há dois lugares que aceitam instruções python Num arquivo kv: depois de uma propriedade, que atribui à propriedade o resultado da expressão (como o texto de um botão conforme mostrado acima) e depois de `on_property`, que executa a instrução quando a propriedade é atualizada (como em `on_state`).

No primeiro caso, a **expressão** poderá abranger apenas uma única linha, e não poderá ser estendido para várias linhas usando novas linha de escape e deverá retornar um valor. Um exemplo de uma expressão válida é `text: self.state and ('up' if self.state == 'normal' else 'down')`.

No último caso, várias instruções de single lines são válidas, incluindo instruções de multiple lines que escapam a sua nova linha, desde que não adicione um nível de indentação.

Exemplos de declaração válidas são:

```
on_press: if self.state == 'normal': print('normal')
on_state:
    if self.state == 'normal': print('normal')
    else: print('down')
    if self.state == 'normal':
        print('multiline normal')
        for i in range(10): print(i)
        print([1,2,3,4,
               5,6,7])
```

Um exemplo de uma declaração inválida:

```
on_state:
    if self.state == 'normal':
        print('normal')
```

4.9.4 Relação entre Valores e Propriedades

Quando utilizares a linguagem Kivy, poderás notar que nós fazemos algum trabalho nos bastidores para que as coisas funcionem correta e automaticamente. Deves saber que **Propriedades** implementa o 'Observer Design Pattern <http://en.wikipedia.org/wiki/Observer_pattern>'. Isso significa que poderás vincular as suas própria função pra que sejam invocadas quando o valor de uma propriedade for alterada (isto é, passivamente "observe" a propriedade as eventuais mudanças).

A linguagem Kivy detecta propriedades em sua expressão *value* e criará callbacks para atualizar automaticamente a propriedade através da sua expressão quando houver alguma alteração.

Temos aqui um simples exemplo que demonstra esse comportamento:

```
Button:  
    text: str(self.state)
```

Neste exemplo, o analisador detecta que o *self.state* é um valor dinâmico (uma propriedade). Para fazer isso, usamos a propriedade de estado do Botão e usamos ela numa expressão para a propriedade `text do botão, que controla qual texto é exibido sobre o botão (Estamos convertendo a representação de estado para texto). Agora, se o estado do botão alterar, a propriedade *text* será automaticamente atualizada.

Lembre-se: O valor é um expressão Python! Isso significa que você pode fazer mais algumas coisas interessante, como:

```
Button:  
    text: 'Plop world' if self.state == 'normal' else 'Release me!'
```

O texto do Button muda com o estado do botão. Por padrão, o texto do botão será 'Plop world', mas quando o botão estiver sendo pressionado, o texto mudará para 'Release me!'.

Mais precisamente, o analisador de linguagem Kivy detectará todas as sub-cadeias da forma *X.a.b* onde *X* é *self* ou *root* ou *app* ou um id conhecido, e *a* e *b* são propriedades: então adicione as dependências apropriadas para fazer com que a restrição seja reavaliada sempre que algo for alterado. Por exemplo, isso funciona exatamente como esperado:

```
<IndexedExample>:  
    beta: self.a.b[self.c.d]
```

No entanto, devido a limitações no analisador que, esperamos que possa ser resolvido no futuro, o seguinte trecho de código não funciona:

```
<BadExample>:  
    beta: self.a.b[self.c.d].e.f
```

Na verdade, a parte *.e.f* não é reconhecida porque não segue o padrão esperado e, portanto, não resulta em uma dependência apropriadamente configurada. Ao invés disso, uma propriedade intermediária deverá ser introduzida para permitir a seguinte restrição:

```
<GoodExample>:  
    alpha: self.a.b[self.c.d]  
    beta: self.alpha.e.f
```

4.9.5 Instruções Gráficas

As instruções gráficas são uma parte especial da linguagem Kivy. Elas são manipulados pela definição da propriedade ‘Canvas’:

```
Widget:  
    canvas:  
        Color:  
            rgb: (1, 1, 1)  
        Rectangle:  
            size: self.size  
            pos: self.pos
```

Todas as classes adicionadas dentro da propriedade Canvas devem ser derivadas da classe **Instruction**. Não podes colocar nenhuma classe Widget dentro da propriedade Canvas (e isso não faria sentido porque um Widget não é uma instrução gráfica).

Se quiseres construir um tema, terás a mesma questão como acontece com CSS: quais regras foram executadas primeiro? No nosso caso, as regras são executadas em ordem de processamento (isto é, de cima para baixo).

Se você desejar alterar como os Botões são renderizados, podes criar seus próprio arquivo *kv* e implementar algo do tipo:

```
<Button>:  
    canvas:  
        Color:  
            rgb: (1, 0, 0)  
        Rectangle:  
            pos: self.pos  
            size: self.size  
        Rectangle:  
            pos: self.pos  
            size: self.texture_size  
            texture: self.texture
```

Isso resultará em botões com um fundo vermelho e com a etiqueta na parte inferior esquerda, além de todas as regras anteriores. Pode limpar todas as instruções anteriores usando o comando *Clear*:

```
<Button>:  
    canvas:  
        Clear  
        Color:  
            rgb: (1, 0, 0)  
        Rectangle:  
            pos: self.pos  
            size: self.size  
        Rectangle:
```

```
pos: self.pos  
size: self.texture_size  
texture: self.texture
```

Então, somente suas regras que seguem o comando *Clear* serão levadas em consideração.

4.9.6 Classes dinâmicas

Classes dinâmicas permitem que você crie novos Widgets on-the-fly, sem qualquer declaração Python em primeiro lugar. A sintaxe das classes dinâmicas é semelhante às Regras, mas você precisa especificar a(s) classe(s) base que deseja utilizar como sendo subclasse.

A sintaxe parece algo do tipo:

```
# Simple inheritance  
<NewWidget@Button>:  
    # kv code here ...  
  
# Multiple inheritance  
<NewWidget@ButtonBehavior+Label>:  
    # kv code here ...
```

O carácter '@' é usado para separar o nome de sua classe da classe que você quer para subclasse. O equivalente Python teria sido:

```
# Simple inheritance  
class NewWidget(Button):  
    pass  
  
# Multiple inheritance  
class NewWidget(ButtonBehavior, Label):  
    pass
```

Quaisquer propriedades nova, normalmente adicionadas junto ao código Python, deverá ser declaradas primeiro. Se a propriedade não existir na classe dinâmica, a mesma será criada automaticamente como uma classe: *~kivy.properties.ObjectProperty* (pre 1.8.0) ou como uma propriedade tipificadamente apropriada (a partir da versão 1.8.0).

Alterado na versão 1.8.0: Se o valor da propriedade for uma expressão que poderá ser avaliada imediatamente (sem ligação externa), então o valor será usado como valor padrão da propriedade e o tipo do valor será usado para a especialização da classe *Property*. Em outras palavras: se declarares *hello: "world"*, uma nova *StringProperty* será instanciado, com o valor padrão "world". Listas, tuplas, dicionários e Strings são suportados.

Vamos ilustrar o uso dessas classes dinâmicas com a implementação de um botão de imagem básico. Poderíamos derivar nossas classes a partir do botão e bastaria adicionar uma propriedade para o nome do arquivo da imagem:

```
<ImageButton@Button>:  
    source: None  
  
    Image:  
        source: root.source  
        pos: root.pos  
        size: root.size  
  
# let's use the new classes in another rule:  
<MainUI>:  
    BoxLayout:  
        ImageButton:  
            source: 'hello.png'  
            on_press: root.do_something()  
        ImageButton:  
            source: 'world.png'  
            on_press: root.do_something_else()
```

Em Python, podes criar instâncias da classe dinamicamente da seguinte maneira:

```
from kivy.factory import Factory  
button_inst = Factory.ImageButton()
```

Nota: Usando classes dinâmicas, uma classe filha pode ser declarada antes de ser declarada a classe pai. Isso no entanto, leva à situação não intuitiva, onde as propriedades/métodos do pai substituem as do filho. Tenha cuidado se optares por trabalhar dessa maneira.

4.9.7 Modelos

Alterado na versão 1.7.0: Uso de modelo agora é obsoleto. Por favor use Classes Dinâmicas como alternativa

Sintaxe de modelos

Uso de modelo em Kivy requer 2 coisas :

1. Um contexto para ser passado para o contexto (será ctx dentro do modelo).
2. uma definição kv do modelo.

Sintaxe do modelo:

```

# With only one base class
[ClassName@BaseClass]:
    # .. definitions ..

# With more than one base class
[ClassName@BaseClass1,BaseClass2]:
    # .. definitions ..

```

Por exemplo, para uma lista, você precisará criar uma entrada com uma imagem à esquerda e um label à direita. Você pode criar um modelo para tornar essa definição mais fácil de usar. Então, vamos criar um modelo que usa 2 entradas no contexto: um nome de arquivo de imagem e um título:

```

[IconItem@BoxLayout]:
    Image:
        source: ctx.image
    Label:
        text: ctx.title

```

Em Python, você pode instanciar o modelo usando:

```

from kivy.lang import Builder

# create a template with hello world + an image
# the context values should be passed as kwargs to the Builder.
→template
# function
icon1 = Builder.template('IconItem', title='Hello world',
    image='myimage.png')

# create a second template with other information
ctx = {'title': 'Another hello world',
    'image': 'myimage2.png'}
icon2 = Builder.template('IconItem', **ctx)
# and use icon1 and icon2 as other widget.

```

Exemplo de modelo

Na maioria das vezes, quando estiveres criando uma tela com kv lang, utilizarás um monte de redefinições. No nosso exemplo, criaremos uma Barra de Ferramentas, baseada em um BoxLayout, e colocaremos alguns Widgets **Image** que irão reagir ao evento *on_touch_down*.

```

<MyToolbar>:
    BoxLayout:
        Image:
            source: 'data/text.png'

```

```

        size: self.texture_size
        size_hint: None, None
        on_touch_down: self.collide_point(*args[1].pos) and_
root.create_text()

    Image:
        source: 'data/image.png'
        size: self.texture_size
        size_hint: None, None
        on_touch_down: self.collide_point(*args[1].pos) and_
root.create_image()

    Image:
        source: 'data/video.png'
        size: self.texture_size
        size_hint: None, None
        on_touch_down: self.collide_point(*args[1].pos) and_
root.create_video()

```

Nós podemos ver que o atributo size e size_hint são exatamente iguais. Mais que isso, a retribuição em on_touch_down e a imagem são mudadas. Estas podem ser a parte variável do modelo que podemos colocar em um contexto. Vamos tentar criar um modelo para a Imagem:

```

[ToolbarButton@Image]:
    # This is the same as before
    size: self.texture_size
    size_hint: None, None

    # Now, we are using the ctx for the variable part of the_
    # template
    source: 'data/%s.png' % ctx.image
    on_touch_down: self.collide_point(*args[1].pos) and ctx.
    callback()

```

O modelo pode ser usado diretamente na regra do MyToolbar

```

<MyToolbar>:
    BoxLayout:
        ToolbarButton:
            image: 'text'
            callback: root.create_text
        ToolbarButton:
            image: 'image'
            callback: root.create_image
        ToolbarButton:
            image: 'video'

```

```
callback: root.create_video
```

Isso é tudo :)

Limitações de modelo

Quando você esta criando um contexto

1. Você não pode usar referencias que não sejam “raiz”:

```
<MyRule>:  
    Widget:  
        id: mywidget  
        value: 'bleh'  
    Template:  
        ctxkey: mywidget.value # << fail, this  
        ↵references the id  
        # mywidget
```

2. Nem todas as partes dinâmicas serão entendidas:

```
<MyRule>:  
    Template:  
        ctxkey: 'value 1' if root.prop1 else 'value2' #  
        ↵<< even if  
        # root.prop1 is a property, if it changes value,  
        ↵ ctxkey  
        # will not be updated
```

As definições de modelo também substituem qualquer definição com o mesmo nome na sua totalidade e, portanto, não suportam herança.

4.9.8 Redefinindo um estilo de widget

Às vezes, gostaríamos de herdar de um Widget para usar suas propriedades Python sem ter que também usar estilo definido no arquivo *kv*. Por exemplo, gostaríamos de herdar de um Label, mas também gostaríamos de definir nossas próprias instruções Canvas ao invés de usar automaticamente as instruções herdadas de Canvas do Label. Podemos fazer isso adicionando um traço (-) antes do nome da classe na definição de estilo no arquivo *kv*.

Em *myapp.py*:

```
class MyWidget(Label):  
    pass
```

E em *my.kv*:

```
<-MyWidget>:  
    canvas:  
        Color:  
            rgb: 1, 1, 1  
        Rectangle:  
            size: (32, 32)
```

MyWidget terá agora uma instrução de Cor e de Retângulo em seu Canvas sem qualquer das instruções herdadas do Label

4.9.9 Redefinindo um estilo de propriedade de uma ferramenta

Semelhante ao *redefining style*, algumas vezes desejaremos herdar de um determinado Widget, mantendo todos os seus estilos definidos com a linguagem KV, exceto um estilo aplicado a uma determinada propriedade, como por exemplo, gostaríamos de herdar de a *Button*, mas também gostaríamos de definir nossa própria propriedade *state_image*, em vez de confiar nos valores *background_normal* e *background_down*. Podemos fazer isso adicionando um traço (-) antes do nome da propriedade *state_image* na definição do estilo .kv.

Em myapp.py:

```
class MyWidget(Button):  
  
    new_background = StringProperty('my_background.png')
```

E em my.kv:

```
<MyWidget>:  
    -state_image: self.new_background
```

MyWidget agora tem um plano de fundo *state_image* definido somente pelo *new_background*, e não por quaisquer estilos que possam ter sido anteriormente definido *state_image*.

Nota: Embora as regras anteriores sejam desmarcadas, elas ainda são aplicadas durante a construção do Widget e só são removidas quando a nova regra com o traço for alcançada. Isso significa que inicialmente, regras antigas poderão ser usadas para definir a propriedade.

4.9.10 Ordem dos *kwargs* e aplicação de regra *kv*

Propriedades podem ser inicializada em KV assim como em Python. Por exemplo, em KV:

```
<MyRule@Widget>:  
    text: 'Hello'  
    ramp: 45.  
    order: self.x + 10
```

Então `MyRule()` inicializaria todas as três propriedades Kivy para os valores definidos em KV. Separadamente em Python, se as propriedades já existem como propriedades Kivy será possível fazer, por exemplo `MyRule(line='Bye', side = 55)`.

No entanto, quais serão os valores finais das propriedades quando `MyRule(text='Bye', order = 55)` for executado? A regra rápida é que a inicialização do Python é mais forte do que a inicialização do KV apenas para regras constantes.

Especificamente, os `kwargs` fornecidos ao inicializador Python são sempre aplicados por primeiro. Assim, no exemplo acima, `text` é definido como `'Bye'` e `order` é definido como 55. Em seguida, todas as regras KV são aplicadas, exceto as regras constantes que substituem valores que são fornecidos antes da inicialização do Python.

Isto é, as regras KV que não criam ligações como `text: 'Hello'` e `ramp: 45.`, se um valor para essa propriedade for fornecido com Python, então essa regra não será aplicada.

Então no exemplo `MyRule(text='Bye', order=55)`, `text` será `'Bye'`, `ramp` será `45.`, e `order`, que cria o vínculo, primeiro será definido 55, mas então quando as regras kv forem aplicadas terminará acima de ser o que quer que `self.x + 10` seja.

Alterado na versão 1.9.1: Antigamente, as regras de KV sempre sobrescrevem os valores do Python, agora, os valores do Python não são substituídos pelas regras definidas como constantes.

4.9.11 Diretivas da Linguagem

Podes usar diretivas para adicionar comandos declarativos, como importações ou definições de constantes, aos arquivos lang. As diretivas são adicionadas como comentários no seguinte formato:

```
#:<directivename> <options>
```

`import <package>`

Novo na versão 1.0.5.

Sintaxe

```
#:import <alias> <package>
```

Você pode importar um pacote escrevendo:

```
#:import os os

<Rule>:
    Button:
        text: os.getcwd()
```

Ou mais complexo:

```
#:import ut kivy.utils

<Rule>:
    canvas:
        Color:
            rgba: ut.get_random_color()
```

Novo na versão 1.0.7.

Você pode diretamente importar classes de um módulo:

```
#: import Animation kivy.animation.Animation
<Rule>:
    on_prop: Animation(x=.5).start(self)
```

set <key> <expr>

Novo na versão 1.0.6.

Sintaxe

```
#:set <key> <expr>
```

Define uma chave que será disponível em qualquer parte do código *kv*. Por exemplo:

```
#:set my_color (.4, .3, .4)
#:set my_color_hl (.5, .4, .5)

<Rule>:
    state: 'normal'
    canvas:
        Color:
            rgb: my_color if self.state == 'normal' else my_color_hl
```

include <file>

Novo na versão 1.9.0.

Sintaxe

```
#:include [force] <file>
```

Inclui um arquivo Kivy esterno. Isso permite dividir Widgets complexos em vários arquivos próprios. Se a inclusão for forçada, o primeiro arquivo será descarregado e carregado novamente. Por exemplo:

```
# Test.kv
#:include mycomponent.kv
#:include force mybutton.kv

<Rule>:
    state: 'normal'
    MyButton:
        MyComponent:
```

```
# mycomponent.kv
#:include mybutton.kv

<MyComponent>:
    MyButton:
```

```
# mybutton.kv

<MyButton>:
    canvas:
        Color:
            rgb: (1.0, 0.0, 0.0)
        Rectangle:
            pos: self.pos
            size: (self.size[0]/4, self.size[1]/4)
```

class kivy.lang.Observable

Bases: :class:`kivy.event.ObjectWithUid`

Observable é uma classe de stub e define os métodos necessários para ligar. **EventDispatcher** é (o) um exemplo de classe que implementa o interface de ligação. Veja a classe **EventDispatcher** para maiores informações.

Novo na versão 1.9.0.

fbind()

Veja `EventDispatcher.fbind()`.

Nota: Para manter a compatibilidade com classes derivadas que podem ter herdado de **Observable** antes, o método **fbind()** foi adicionado. A implementação padrão de **fbind()** é criar uma função parcial que passa para ligar enquanto salva o uid e largs/kwargs. No entanto, `meth:funbind` (e `unbind_uid()`) são bastante ineficientes, uma vez que temos de primeiro

pesquisar esta função parcial usando o `largs/kwags` ou `uid` e, em seguida, chamar `unbind()` na função retornada. É recomendável substituir esses métodos em classes derivadas para vincular diretamente para um melhor desempenho.

Similarmente a `EventDispatcher.fbind()`, este método retorna 0 em caso de falha e um `uid` único e positivo no caso de sucesso. Este `uid` pode ser usado com `unbind_uid()`.

funbind()

Veja `fbind()` and `EventDispatcher.funbind()`.

unbind_uid()

Veja `fbind()` e `EventDispatcher.unbind_uid()`.

class kivy.lang.BuilderBase

Bases: `object`

O Builder é responsável por criar uma `Parser` para analisar um arquivo kv, mesclando os resultados em suas regras internas, templates, etc.

Por padrão, `Builder` é uma instância global do Kivy usado pelos Widgets que podes utilizar para abrir outros arquivos KV além de definir os padrões.

apply(widget, ignored_consts=set([]))

Procura todas as regras que correspondam ao Widget e aplique-as.

`Ignored_consts` é um conjunto ou tipo de lista cujos elementos são nomes de propriedade para os quais as regras de KV são constantes (isto é, aquelas que não criam binding (vínculos)) nesses Widget não serão aplicadas. Isto permite, por exemplo, ignorar regras constantes que substituem um valor inicializado em Python.

apply_rules(widget, rule_name, ignored_consts=set([]))

Procura por todas as regras que correspondam ao Widget `rule_name` e aplique-as ao `widget`.

Novo na versão 1.10.0.

`Ignored_consts` é um conjunto ou tipo de lista cujos elementos são nomes de propriedade para os quais as regras de KV são constantes (isto é, aquelas que não criam binding (vínculos)) nesses Widget não serão aplicadas. Isto permite, por exemplo, ignorar regras constantes que substituem um valor inicializado em Python.

load_file(filename, **kwargs)

Insere um arquivo no construtor de linguagem e retorna o Widget raiz (se definido) do arquivo ou `kv`.

Parameters

rulesonly: bool, o padrão é *False*Se Verdadeiro, o Builder vai levantar uma exceção se você tiver uma ferramenta widget raiz dentro da definição.

load_string(string, **kwargs)

Insere uma String no Construtor da Linguagem e retorna o Widget raiz (se definido) da String com código *kv*.

Parameters

rulesonly: bool, o padrão é *False*Se Verdadeiro, o Builder vai levantar uma exceção se você tiver uma ferramenta widget raiz dentro da definição.

match(widget)

Retorna uma lista de objetos correspondentes ao Widget **ParserRule**.

match_rule_name(rule_name)

Retorna uma lista de objetos correspondentes ao Widget **ParserRule**.

sync()

Executa todas as operações em espera, tais como a execução das expressões relacionadas ao *Canvas*.

Novo na versão 1.7.0.

template(*args, **ctx)

Crie um modelo especializado usando um contexto específico.

Novo na versão 1.0.5.

Com templates, você pode construir Widgets personalizados desde a definição da linguagem *kv*, dando-lhes um contexto. Veja [Template usage](#).

unbind_property(widget, name)

Desvincula o manipulador criado por todas as regras dos Widgets que definem o nome.

Isso efetivamente limpa todas as regras de Widget que assumem a forma:

```
name: rule
```

Por exemplo:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.
...     ↪width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
```

```
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_property(w, 'height')
>>> w.width = 222
>>> w.size
[222, 500]
>>> w.y = 500
>>> w.pos
[550, 500]
```

Novo na versão 1.9.1

unbind_widget(*uid*)

Desassociar todos os manipuladores criados pelas regras KV do Widget. O `attr:kivy.uix.widget.Widget.uid` é passado aqui ao invés do próprio Widget, porque o Builder está usando o mesmo no Widget destruidor.

Isso efetivamente limpa todas as regras KV associadas com este Widget. Por exemplo:

```
>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.
... width
...     x: self.y + 50
... ''')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_widget(w.uid)
>>> w.width = 222
>>> w.y = 500
>>> w.size
[222, 500]
>>> w.pos
[50, 500]
```

Novo na versão 172

unload_file(*filename*)

Descarregue todas as regras associadas a um arquivo anteriormente importado.

Novo na versão 1.0.8

Aviso: Isso não removerá as regras ou os templates já aplicados/usados nos Widgets atuais. Isso somente afetará os próximos Widgets criados ou as invocações de template.

class kivy.lang.BuilderException(context, line, message, cause=None)

Bases: [kivy.lang.parser.ParserException](#)

Exceção levantada quando o *Builder* falhar ao aplicar as regras ao Widget.

class kivy.lang.Parser(**kwargs)

Bases: object

Cria um objeto Parse para analisar os arquivos com linguagem Kivy ou outro código Kivy.

parse(content)

Analisar o conteúdo de um arquivo Parser e retornar uma lista de objetos principais.

parse_level(level, lines, spaces=0)

Analisa o recuo (indentação) atual (nível * espaços).

strip_comments(lines)

Remove todos os comentários de todas as linhas no local. Comentários precisam ser em uma única linha e não podem estar no final desta. Ou seja, o primeiro caractere de (menos espaços) em branco) de uma linha de comentário deve ser o caractere #.

class kivy.lang.ParserException(context, line, message, cause=None)

Bases: exceptions.Exception

Exceções levantadas quando alguma coisa errada acontece em um arquivo kivy.

4.9.12 Construtor

Classe usada para registo e aplicação de regras para widgets específicos.

class kivy.lang.builder.Observable

Bases: [kivy.event.ObjectWithUid](#)

Observable é uma classe de stub e define os métodos necessários para ligar. EventDispatcher é (o) um exemplo de classe que implementa o interface de ligação. Veja a classe EventDispatcher para maiores informações.

Novo na versão 1.9.0.

fbind()

Veja EventDispatcher.fbind().

Nota: Para manter a compatibilidade com classes derivadas que podem ter herdado de *Observable* antes, o método *fbind()* foi adicionado. A implementação padrão de *fbind()* é criar uma função parcial que passa para ligar enquanto salva o uid e largs/kwargs. No entanto, meth:*funbind* (e *unbind_uid()*) são bastante ineficientes, uma vez que temos de primeiro pesquisar esta função parcial usando o largs/kwargs ou uid e, em seguida, chamar *unbind()* na função retornada. É recomendável substituir esses métodos em classes derivadas para vincular diretamente para um melhor desempenho.

Similarmente a `EventDispatcher.fbind()`, este método retorna 0 em caso de falha e um *uid* único e positivo no caso de sucesso. Este *uid* pode ser usado com *unbind_uid()*.

funbind()

Veja *fbind()* and `EventDispatcher.funbind()`.

unbind_uid()

Veja *fbind()* e `EventDispatcher.unbind_uid()`.

`kivy.lang.builder.Builder = <kivy.lang.builder.BuilderBase object>`

Instância principal de *BuilderBase*.

`class kivy.lang.builder.BuilderBase`

Bases: `object`

O Builder é responsável por criar uma `Parser` para analisar um arquivo kv, mesclando os resultados em suas regras internas, templates, etc.

Por padrão, *Builder* é uma instância global do Kivy usado pelos Widgets que podes utilizar para abrir outros arquivos KV além de definir os padrões.

apply(widget, ignored_consts=set([]))

Procura todas as regras que correspondam ao Widget e aplique-as.

Ignored_consts é um conjunto ou tipo de lista cujos elementos são nomes de propriedade para os quais as regras de KV são constantes (isto é, aquelas que não criam binding (vínculos)) nesses Widget não serão aplicadas. Isto permite, por exemplo, ignorar regras constantes que substituem um valor inicializado em Python.

apply_rules(widget, rule_name, ignored_consts=set([]))

Procura por todas as regras que correspondam ao Widget *rule_name* e aplique-as ao *widget*.

Novo na versão 1.10.0.

Ignored_consts é um conjunto ou tipo de lista cujos elementos são nomes de propriedade para os quais as regras de KV são constantes (isto é, aquelas que não criam binding (vínculos)) nesses Widget não serão aplicadas. Isto

permite, por exemplo, ignorar regras constantes que substituem um valor inicializado em Python.

load_file(filename, **kwargs)

Insere um arquivo no construtor de linguagem e retorna o Widget raiz (se definido) do arquivo ou do código no formato String *kv*.

Parameters

rulesonly: bool, o padrão é *False*Se Verdadeiro, o Builder vai levantar uma exceção se você tiver uma ferramenta widget raiz dentro da definição.

load_string(string, **kwargs)

Insere uma String dentro do contrutor

Parameters

rulesonly: bool, o padrão é *False*Se Verdadeiro, o Builder vai levantar uma exceção se você tiver uma ferramenta widget raiz dentro da definição.

match(widget)

Retorna uma lista de objetos correspondentes ao Widget ParserRule.

match_rule_name(rule_name)

Retorna uma lista de objetos correspondentes ao Widget ParserRule.

sync()

Executa todas as operações em espera, tais como a execução das expressões relacionadas ao *Canvas*.

Novo na versão 1.7.0.

template(*args, **ctx)

Crie um modelo especializado usando um contexto específico.

Novo na versão 1.0.5.

Com templates, você pode construir Widgets personalizados desde a definição da linguagem *kv*, dando-lhes um contexto. Veja [Template usage](#).

unbind_property(widget, name)

Desvincula o manipulador criado por todas as regras dos Widgets que definem o nome.

Isso efetivamente limpa todas as regras de Widget que assumem a forma:

```
name: rule
```

Por exemplo:

```
>>> w = Builder.load_string('''
... Widget:
```

```

...      height: self.width / 2. if self.disabled else self.
    ↵width
...      x: self.y + 50
...      '')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_property(w, 'height')
>>> w.width = 222
>>> w.size
[222, 500]
>>> w.y = 500
>>> w.pos
[550, 500]

```

Novo na versão 1.9.1.

unbind_widget(*uid*)

Desassociar todos os manipuladores criados pelas regras KV do Widget. O attr:*kivy.uix.widget.Widget.uid* é passado aqui ao invés do próprio Widget, porque o Builder está usando o mesmo no Widget destruidor.

Isso efetivamente limpa todas as regras KV associadas com este Widget. Por exemplo:

```

>>> w = Builder.load_string('''
... Widget:
...     height: self.width / 2. if self.disabled else self.
...     ↵width
...     x: self.y + 50
...     '')
>>> w.size
[100, 100]
>>> w.pos
[50, 0]
>>> w.width = 500
>>> w.size
[500, 500]
>>> Builder.unbind_widget(w.uid)
>>> w.width = 222
>>> w.y = 500
>>> w.size
[222, 500]
>>> w.pos
[50, 500]

```

Novo na versão 1.7.2.

unload_file(filename)

Descarregue todas as regras associadas a um arquivo anteriormente importado.

Novo na versão 1.0.8.

Aviso: Isso não removerá as regras ou os templates já aplicados/usados nos Widgets atuais. Isso somente afetará os próximos Widgets criados ou as invocações de template.

class kivy.lang.builder.BuilderException(context, line, message, cause=None)

Bases: *kivy.lang.parser.ParserException*

Exceção levantada quando o *Builder* falhar ao aplicar as regras ao Widget.

4.9.13 Parser

Classe usada para a análise das regras de arquivos .kv.

class kivy.lang.parser.Parser(kwargs)**

Bases: *object*

Cria um objeto Parse para analisar os arquivos com linguagem Kivy ou outro código Kivy.

parse(content)

Analisar o conteúdo de um arquivo Parser e retornar uma lista de objetos principais.

parse_level(level, lines, spaces=0)

Analisa o recuo (indentação) atual (nível * espaços).

strip_comments(lines)

Remove todos os comentários de todas as linhas no local. Comentários precisam ser em uma única linha e não podem estar no final desta. Ou seja, o primeiro caractere de (menos espaços) em branco) de uma linha de comentário deve ser o caractere #.

class kivy.lang.parser.ParserException(context, line, message, cause=None)

Bases: *exceptions.Exception*

Exceções levantadas quando alguma coisa errada acontece em um arquivo kivy.

4.10 Bibliotecas Externas

Kivy vem com outra biblioteca Python/C:

- Arquivos ***ddsfile*** - used for parsing and saving **DDS**.
- Protocolos ***osc*** - a modified/optimized version of PyOSC for using the **Open Sound Control**.
- ***mtdev*** - provê suporte para o **Kernel multi-touch transformation library**.

Aviso: Mesmo que o Kivy venha com essas bibliotecas externas, nós não fornecemos nenhum apoio as mesmas e essa bibliotecas podem sofrer alterações no futuro. Não confie seus código as mesmas.

4.10.1 GstPlayer

Novo na versão 1.8.0.

GstPlayer é um media player implementado especificamente para o Kivy com Gstreamer 1.0. Isso não utiliza o Gi em tudo e é focado naquilo que nós queremos: a habilidade de ler vídeo e transmitir imagem em um callback, or ler um arquivo de áudio. Não utilize isso diretamente, ao invés disso, utilize nosso provedor principal (Core providers).

Este player é compilado automaticamente se você tiver funcionando *pkg-config --libs* *gstreamer-1.0*.

Aviso: Esta é uma biblioteca esterna e o Kivy não provê nenhum suporte a mesma. Portanto, a mesma pode mudar no futuro e aconselhamos que você não confie seu código a mesma.

4.10.2 OSC

Esta é uma versão fortemente modificada do PyOSC usado para acessar o protocolo **Open Sound Control**. Ele é usado pelo Kivy internamente para provedores TUIO, mas também pode ser usado por aplicativos para interagir com dispositivos ou processos usando a API OSC.

Aviso: Esta é uma biblioteca esterna e o Kivy não provê nenhum suporte a mesma. Portanto, a mesma pode mudar no futuro e aconselhamos que você não confie seu código a mesma.

SEM DOCUMENTAÇÃO (módulo kivy.lib.osc)

class kivy.lib.osc.CallbackManager

Este utilitário de classe mapeia endereços OSC para ‘invocáveis’ (callables).

O CallbackManager chama seus callbacks com uma lista de argumentos OSC descodificados, incluindo o endereço e os *typetags* como os dois primeiros argumentos.

add(callback, name)

Adiciona um callback ao nosso conjunto de callbacks ou remove o callback com nome se o retorno da chamada for igual a *None*.

dispatch(message, source=None)

Envia dados OSC descodificados para um callback apropriado

handle(data, source=None)

Tendo dados OSC, tenta invocar o callback com o endereço correto.

unbundler(messages)

Enviar as mensagens em um pacote descodificado.

kivy.lib.osc.OSCArgument(data)

Converte alguns tipos do Python para suas representações binárias OSC, retornando uma tupla (*typetag*, *data*).

kivy.lib.osc.OSCBlob(next)

Converte uma string num blob OSC, retornando uma tupla (*typetag*, *data*).

class kivy.lib.osc.OSCMessage

Constrói mensagens OSC typetagged.

append(argument, typehint=None)

Adiciona dados à mensagem, atualizando as *typetags* baseadas no tipo do argumento. Se o argumento for um blob (string contada) envie um ‘b’ como *typehint*.

getBinary()

Retorna a mensagem binária (até agora) com *typetags*.

rawAppend(data)

Anexa dados brutos a mensagem. Use *append()*.

kivy.lib.osc.hexDump(data)

Utilitário útil; Imprime a seqüência em hexadecimal

kivy.lib.osc.parseArgs(args)

Dada uma lista de Strings, produz uma lista onde essas Strings foram analisadas (quando possível) como sendo Floats ou Integers.

kivy.lib.osc.readDouble(data)

Tenta interpretar os próximos 8 bytes dos dados como um double float de 64 bits.

kivy.lib.osc.OSC.readLong(*data*)

Tenta interpretar os próximos 8 bytes dos dados como um *signed integer* de 64 bits.

simpleOSC 0.2

ixi software - July, 2006 www.ixi-software.net

Simples API para o Open SoundControl para o Python (por Daniel Holth, Clinton McChesney -> arquivo pyKit.tar.gz em <http://wiretap.stetson.edu>) Documentação em <http://wiretap.stetson.edu/docs/pyKit/>

O principal alvo desta implementação é prover uma simples maneira de lidar com a implementação OSC que facilita a vida daqueles que não possuem conhecimento sobre sockets ou programação. Isto não estaria na sua tela sem a ajuda de Daniel Holth.

Esta biblioteca é software livre; você pode redistribuí-la e/ou modificá-la sob os termos da GNU Lesser General Public License, tal como publicado pela Free Software Foundation; quer seja a versão 2.1 da licença, ou (a sua opção) qualquer versão posterior.

Esta biblioteca é distribuída na esperança de que seja útil, mas SEM NENHUMA GARANTIA; sem mesmo a garantia implícita de COMERCIALIZAÇÃO ou ADEQUAÇÃO A UM DETERMINADO PROPÓSITO. Consulte a GNU Lesser General Public License para obter mais detalhes.

Você deve ter recebido uma cópia da GNU Lesser General Public License juntamente com esta biblioteca; se não, escreva para a Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 EUA

Obrigado pelo apoio a Buchsenhausen, Innsbruck, Áustria.

kivy.lib.osc.oscAPI.appendToBundle(*bundle*, *oscAddress*, *dataArray*)

criar uma mensagem OSC e anexá-a a um determinado pacote

kivy.lib.osc.oscAPI.bind(*oscid*, *func*, *oscaddress*)

Vincula dado oscaddresses com funções dadas no gerenciador de endereços

kivy.lib.osc.oscAPI.createBinaryMsg(*oscAddress*, *dataArray*, *typehint=None*)

criar e retornar o tipo geral de mensagem binária OSC

kivy.lib.osc.oscAPI.createBundle()

cria o tipo de pacote de mensagens OSC

kivy.lib.osc.oscAPI.dontListen(*thread_id=None*)

Fechá o socket e mata a thread.

kivy.lib.osc.oscAPI.init()

Instancia o gerenciador de endereços e o soquete externo como globais

kivy.lib.osc.oscAPI.listen(ipAddr='127.0.0.1', port=9001)

Cria um novo Thread que escuta a porta padrão do ipAddr = '127.0.0.1', porta 9001

kivy.lib.osc.oscAPI.readQueue(thread_id=None)

Lê filas de todos os Threads e envia a mensagem. Isso deve ser chamada no Thread principal.

Você pode passar o *id* de Thread para a mensagem deque de um Thread específico. Esse id é retornado pela função *listen()*

kivy.lib.osc.oscAPI.sendBundle(bundle, ipAddr='127.0.0.1', port=9000)

Converter um pacote para binário e enviá-o

**kivy.lib.osc.oscAPI.sendMsg(oscAddress, dataArray=[],
ipAddr='127.0.0.1', port=9000, typehint=None)**

cria e envia mensagens padrão OSC para '127.0.0.1', porta 9000

4.10.3 Arquivo de Biblioteca DDS

A biblioteca pode ser usada para analisar e salvar arquivos DDS (*DirectDraw Surface* <https://en.wikipedia.org/wiki/DirectDraw_Surface>).

A versão inicial foi escrita por:

Alexey Borzenkov (snaury@gmail.com)

Todos os créditos de trabalho iniciais vão para ele! Obrigado :)

Esta versão utiliza Struct ao invés de ctypes.

Formato DDS

```
[DDS ][SurfaceDesc][Data]  
  
[SurfaceDesc]:: (everything is uint32)  
    Size  
    Flags  
    Height  
    Width  
    PitchOrLinearSize  
    Depth  
    MipmapCount  
    Reserved1 * 11  
    [PixelFormat]::  
        Size  
        Flags
```

```
FourCC
RGBBitCount
RBitMask
GBitMask
BBitMask
ABitMask
[Caps]::
    Caps1
    Caps2
    Reserved1 * 2
Reserverd2
```

Aviso: Esta é uma biblioteca esterna e o Kivy não provê nenhum suporte a mesma. Portanto, a mesma pode mudar no futuro e aconselhamos que você não confie seu código a mesma.

4.10.4 Python mtdev

O módulo mtdev provê a ligação com o Python e o [Kernel multi-touch transformation library](#), também conhecida como mtdev (licença MIT).

A biblioteca mtdev transforma todas as variantes de eventos de MT do kernel para o protocolo de tipo B com slotted. Os eventos colocados em mtdev podem ser de qualquer dispositivo MT, especificamente tipo A sem rastreamento de contatos, tipo A com rastreamento de contatos ou tipo B com rastreamento de contatos. Consulte a documentação do kernel para obter mais detalhes.

Aviso: Esta é uma biblioteca esterna e o Kivy não provê nenhum suporte a mesma. Portanto, a mesma pode mudar no futuro e aconselhamos que você não confie seu código a mesma.

4.11 Módulos

Módulos são classes que podem ser carregadas com a aplicação Kivy inicia. A carga dos módulos é gerenciada pelo arquivo config. Atualmente, inclui:

- [**touchring**](#): Desenha círculo no entorno de cada toque.
- [**monitor**](#): Adiciona barra superior vermelha que indica que FPS e indicação de atividade entrada gráfica.
- [**keybinding**](#): Vincula algumas teclas para ações como captura de tela.

- *recorder*: Grava e reproduz sequência de eventos.
- *screen*: Emular características de diferentes monitores (dpi/densidade/ resolução).
- *inspector*: Examinar hierarquia do widget bem como suas propriedades.
- *webdebugger*: Examinar em tempo real os internals de sua app em um navegador.
- *joycursor*: Navigate in your app with a joystick.

Módulos são automaticamente carregados a partir do path Kivy e do path do Usuário:

- *PATH_TO_KIVY/kivy/modules*
- *HOME/.kivy/mods*

4.11.1 Ativando um módulo

Existem várias maneiras para ativar um módulo kivy.

Ativar módulo no config

Para ativar o módulo dessa maneira, editar seu arquivo de configuração (*HOME/.kivy/config.ini*):

```
[modules]
# uncomment to activate
touchring =
# monitor =
# keybinding =
```

Apenas o nome do módulo seguido por um “=” é suficiente para ativar o módulo.

Ativar um módulo Python

Antes de ativar sua aplicação, preferencialmente no início do seu import, faça algo como isso:

```
import kivy
kivy.require('1.0.8')

# Activate the touchring module
from kivy.config import Config
Config.set('modules', 'touchring', '')
```

Ativar o módulo via linha de comando

Quando iniciando sua aplicação a partir da linha de comando, pode adicionar um `-m <nome-módulo>` aos argumentos. Exemplo:

```
python main.py -m webdebugger
```

Nota: Alguns módulos, como tela, podem requerer parâmetros adicionais. Quando ativados sem esses parâmetros, haverá exibição na console.

4.11.2 Criar seu próprio módulo

Criar um arquivo em `HOME/.kivy/mods` e criar 2 funções:

```
def start(win, ctx):
    pass

def stop(win, ctx):
    pass
```

Iniciar / Parar são funções que serão chamadas para toda janela aberta no Kivy. Quanto iniciar um módulo, pode usar isso para armazenar e gerenciar o status do módulo. Usar variável `ctx` como dicionário. Esse contexto é único para cada execução de instância `start()` na chamada de cada módulo bem como no `stop()`.

4.11.3 Console

Novo na versão 1.9.1.

Reinicialização do inspetor antigo, projetado para trabalhar de forma modular e as referências separadas. Também possui uma arquitetura com ‘add-ons’ que permitem ser acrescentados: botões, painéis ou outros widgets no próprio console.

Aviso: Este módulo funciona, mas pode falhar em alguns casos. Por favor, contribua!

Utilização

Para uso normal do módulo, consulte a documentação [modules](#):

```
python main.py -m console
```

Navegação do Mouse

Quando o botão “Select” está ativado, você pode:

- Toque uma vez num Widget para selecioná-lo sem sair do modo de inspeção
- Toque duas vezes num Widget para seleciona-lo e deixa-lo ser inspecionado no modo de inspeção (então você pode manipular o Widget novamente)

Navegação do Teclado

- “Ctrl + e”: Alterna entre o Console
- “Escape”: cancelar a pesquisa do Widget e, em seguida, oculta a visualização do inspetor
- “Top”: selecione o widget Pai.
- “Down”: Selecione os primeiros filhos do Widget atualmente selecionado
- “Left”: Seleciona o irmão anterior
- “Right”: Selecione o próximo irmão seguinte

Informações adicionais

Algumas propriedades podem ser editadas ao vivo. No entanto, devido ao uso atrasado de algumas propriedades, pode ocorrer problemas se você não lidar com todos os casos.

Addons

Os Addons devem ser adicionados ao `Console.addons` antes do primeiro tick do Clock do aplicativo, ou antes que o `create_console` seja invocado. Não é possível adicionar Addons durante o tempo de execução. Os Addons são bem leves até que o Console seja ativado. O Painel é ainda mais leve, nada é feito até que o usuário selecione-o.

Por padrão, fornecemos vários AddOns ativados:

- `ConsoleAddonFps`: exibe o FPS no canto superior direito
- `ConsoleAddonSelect`: Ativa o modo de seleção
- `ConsoleAddonBreadcrumb`: exibe a hierarquia do atual Widget na parte inferior
- `ConsoleAddonWidgetTree`: painel que exibe a árvore de Widgets do aplicativo
- `ConsoleAddonWidgetPanel`: painel que exibe as propriedades do Widget selecionado

Se precisares adicionar um Widget personalizado no Console, utilize uma *ConsoleButton*, *ConsoleToggleButton* ou *ConsoleLabel*

Um Addon deve herdade da classe *ConsoleAddon*.

Por exemplo, aqui está um simples Addon para exibir o FPS no topo/direita do Console:

```
from kivy.modules.console import Console, ConsoleAddon

class ConsoleAddonFps(ConsoleAddon):
    def __init__(self):
        self.lbl = ConsoleLabel(text="0 Fps")
        self.console.add_toolbar_widget(self.lbl, right=True)

    def activate(self):
        self.event = Clock.schedule_interval(self.update_fps, 1 / 2.)
        ↴

    def deactivated(self):
        self.event.cancel()

    def update_fps(self, *args):
        fps = Clock.get_fps()
        self.lbl.text = "{} Fps".format(int(fps))

Console.register_addon(ConsoleAddonFps)
```

Podes criar Addon que adiciona outros painéis. A ativação/desativação do painel não está vinculada à ativação/desativação do complemento, mas em alguns casos, podes usar o mesmo retorno de chamada para desativar o complemento e o painel. Aqui temos um simples painel Addons como exemplo:

```
from kivy.modules.console import Console, ConsoleAddon, ConsoleLabel

class ConsoleAddonAbout(ConsoleAddon):
    def __init__(self):
        self.console.add_panel("About", self.panel_activate,
                              self.panel_deactivate)

    def panel_activate(self):
        self.console.bind(widget=self.update_content)
        self.update_content()

    def panel_deactivate(self):
        self.console.unbind(widget=self.update_content)

    def deactivate(self):
        self.panel_deactivate()
```

```

def update_content(self, *args):
    widget = self.console.widget
    if not widget:
        return
    text = "Selected widget is: {!r}".format(widget)
    lbl = ConsoleLabel(text=text)
    self.console.set_content(lbl)

Console.register_addon(ConsoleAddonAbout)

```

`kivy.modules.console.start(win, ctx)`

Cria uma instância de Console anexada ao `ctx` e vinculada ao evento da janela :meth: `~kivy.core.window.WindowBase.on_keyboard` para capturar o atalho de teclado.

Parameters

`win`: A **Window**A janela do aplicativo para vincular a.

`ctx`: Uma **Widget** ou subclasseO Widget a ser inspecionado.

`kivy.modules.console.stop(win, ctx)`

Para e descarrega todos os Inspetores ativos para um dado `ctx`.

class kivy.modules.console.Console(kwargs)**
Bases: `kivy.uix.relativelayout.RelativeLayout`

Interface do Console

Este Widget é criado por `create_console()`, quando o módulo é carregado. Durante esse tempo, poderás adicionar Addons no Console para estender as funcionalidades, ou adicionar seu próprio stats de aplicativo/módulo de depuração.

activated

`True` se o Console estiver ativado (mostrado)

add_panel(name, cb_activate, cb_deactivate, cb_refresh=None)

Adiciona um novo painel no Console.

- `cb_activate` é um callable que será invocado quando o painel for ativado pelo usuário.
- `cb_deactivate` é um callable que será invocado quando o painel for desativado ou quando o Console for ocultado.
- `cb_refresh` é um callable opcional chamado se o usuário clicar novamente no botão para exibir o painel

Quando ativado, cabe ao painel exibir um conteúdo no Console pelo uso do método `set_content()`.

add_toolbar_widget(*widget*, *right=False*)

Adiciona um Widget na barra de ferramentas superior esquerda do Console. Use *right = True* se quiser adicionar o Widget à direita.

addons = [*<class 'kivy.modules.console.ConsoleAddonSelect'>*, *<class 'kivy.modules.console.ConsoleAddonSelect'>*]

Array de Addons que serão criados na criação do Console

highlight_at(*x, y*)

Seleciona um Widget a partir de uma coordenada de janela x/y. Isso é usado principalmente internamente quando o modo Select está ativado

inspect_enabled

Indica se a inspeção do inspetor está habilitada. Caso esteja, o próximo toque para baixo irá selecionar um Widget abaixo do item selecionado

mode

Modo de exibição do Console, fixado na parte inferior, ou como uma janela flutuante.

pick(*widget*, *x, y*)

Escolha um Widget em x/y, dado uma raiz *widget*

remove_toolbar_widget(*widget*)

Remove um widget da barra de ferramentas

set_content(*content*)

Substitui o conteúdo do Console por outro.

widget

O Widget atualmente selecionado

class kivy.modules.console.ConsoleAddon(*console*)

Bases: *object*

Classe base para implementação de AddOns

activate()

Método invocado quando o Addon é ativado pelo Console (quando o Console for exibido)

console=None

Instância do Console

deactivate()

Método invocado quando o AddOn é desativado pelo Console (quando o Console está oculto)

init()

Método disparado quando o AddOn é instanciado pelo Console

class kivy.modules.console.ConsoleButton(***kwargs*)

Bases: *kivy.uix.button.Button*

Button especializado para o Console

```
class kivy.modules.console.ConsoleToggleButton(**kwargs)
```

Bases: [kivy.uix.togglebutton.ToggleButton](#)

ToggleButton especializado para o Console

```
class kivy.modules.console.ConsoleLabel(**kwargs)
```

Bases: [kivy.uix.label.Label](#)

LabelButton especializado para o Console

4.11.4 Inspetor

Novo na versão 1.0.9.

Aviso: Este módulo é altamente experimental, use-o com cuidado.

O Inspector é uma ferramenta para encontrar um Widget na árvore de Widgets clicando ou tocando nele. Alguns atalhos de teclado são ativados:

- “Ctrl + e”: ativa/desativa a View Inspector
- “Escape”: cancela a primeira pesquisa do Widget, depois oculta a visualização do inspetor

Interações com o Inspetor Disponíveis:

- Toque uma vez num Widget para selecioná-lo sem sair do modo de inspeção
- Toque duas vezes num Widget para seleciona-lo e deixa-lo ser inspecionado no modo de inspeção (então você pode manipular o Widget novamente)

Algumas propriedades podem ser editadas ao vivo. No entanto, devido ao uso atrasado de algumas propriedades, pode ocorrer problemas se você não lidar com todos os casos.

Utilização

Para uso normal do módulo, por favor, veja a documentação [modules](#).

O Inspector, no entanto, também pode ser importado e usado como um módulo normal do Python. Isto tem a vantagem adicional de ser capaz de Ativar e Desativar o módulo de forma programática:

```
from kivy.core.window import Window
from kivy.app import App
from kivy.uix.button import Button
from kivy.modules import inspector

class Demo(App):
```

```
def build(self):
    button = Button(text="Test")
    inspector.create_inspector(Window, button)
    return button
```

```
Demo().run()
```

Para remover o inspector, você pode fazer o seguinte:

```
inspector.stop(Window, button)
```

kivy.modules.inspector.stop(*win, ctx*)

Para e descarrega todos os Inspetores ativos para um dado *ctx*.

kivy.modules.inspector.create_inspector(*win, ctx, *l*)

Crie uma instância do Inspector conectada ao *ctx* e vinculada ao evento da janela: *on_keyboard()* para capturar o atalho do teclado.

Parameters

win: A **Window**A janela do aplicativo para vincular a.

ctx: Uma **Widget** ou subclasseO Widget a ser inspecionado.

4.11.5 Keybinding

Este módulo força o mapeamento de algumas funções chaves (específicas):

- F11: Gira a janela através dos graus 0, 90, 180 and 270 graus
- Shift + F11: Alterna entre portrait e landscape num computador Desktops
- F12: Pega um ScreenShot (foto)

Nota: Isto não funciona se o aplicativo solicitar o teclado de antemão.

Utilização

Para uso normal do módulo, por favor, veja a documentação **modules**.

O módulo KeyBinding, de qualquer forma, pode ser importado e utilizado como um módulo Python normal. Isso possui a vantagem adicional de estar apto a ativar e desativar módulos programaticamente:

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.modules import keybinding
from kivy.core.window import Window
```

```
class Demo(App):  
  
    def build(self):  
        button = Button(text="Hello")  
        keybinding.start(Window, button)  
        return button  
  
Demo().run()
```

Para remover todos os KeyBindings, você deve fazer da seguinte forma:

```
Keybinding.stop(Window, button)
```

4.11.6 Módulo de Monitoria

O módulo do Monitor é uma barra de ferramenta que exibe a atividade atual da sua aplicação:

- FPS
- Gráfico da Entrada de Eventos

Utilização

Para uso normal do módulo, por favor, veja a documentação [modules](#).

4.11.7 Módulo de Gravação

Novo na versão 1.1.0.

Cria uma instância da classe: [*Recorder*](#), que se liga a classe e associa teclas com a função de gravar/reproduzir sequências.

- F6: reproduzir o último registro em um loop
- F7: leia a última gravação
- F8: grava os eventos de entrada

Configuração

Parameters

attrs: str, o padrão é o valor *recordAttrs* Atributos para gravar a partir do evento de movimento

profile_mask: str, o valor padrão é **record_profile_mask**.

Máscara para o perfil de eventos de movimento. Usado para filtrar qual perfil aparecerá no evento de movimento falso quando reproduzido.

filename: str, o padrão é ‘recorder.kvi’ Nome do arquivo para gravar / reproduzir com

Utilização

Para uso normal do módulo, por favor, veja a documentação **modules**.

4.11.8 Tela

Este módulo modifica algumas variáveis de ambiente e de configuração para que correspondam a densidade/dpi/ScreenSize de um dispositivo em específico.

Para ver lista de ScreenIds disponíveis, basta executar:

```
python main.py -m screen
```

Para simular uma densidade média de tela como o Motorola Droid 2:

```
python main.py -m screen:droid2
```

Para simular um tela de alta-densidade como um HTC One X, em portrait:

```
python main.py -m screen:onex,portrait
```

Para simular uma tela de iPad 2:

```
python main.py -m screen:ipad
```

Se a janela gerada for muito grande, é possível determinar uma escala:

```
python main.py -m screen:note2,portrait,scale=.75
```

Note que para exibir o conteúdo corretamente numa janela escalável você deve consistentemente utilizar as unidade ‘dp’ e ‘sp’ na construção da sua aplicação. Veja **metrics** para maiores informações.

4.11.9 Toque

Mostra anéis em volta de todos os toques feitos na superfície / tela. Este módulo pode ser usado para checar se não há problemas de calibração com toques na tela.

Configuração

Parameters

image: str, padrão para '`<kivy>/data/images/rings.png`' Nome do arquivo para a imagem a ser usada
scale: float, padrão para 1 Escala da imagem.
alpha: float, padrão para 1 Opacidade da imagem

Exemplo

Na sua configuração (`~/kivy/config.ini`), você pode adicionar algo como isto:

```
[modules]
touchring = image=mypointer.png, scale=.3, alpha=.7
```

4.11.10 Web Debugger

Novo na versão 1.2.0.

Aviso: Este módulo é altamente experimental, use-o com cuidado.

Este módulo inicia um WebServer e roda em background. Pode ver como seu aplicativo evolui durante o tempo de execução, examinar o cache interno etc.

Execute com:

```
python main.py -m webdebugger
```

Feito isso, abra seu navegador e acesse: `http://localhost:5000/`

4.12 Suporte de Rede

Atualmente, o Kivy suporta solicitações básicas de rede assíncrona. Por favor, consulte [`kivy.network.urlrequest.UrlRequest`](#).

4.12.1 UrlRequest

Novo na versão 1.0.8.

Você pode usar a `URLRequest` para fazer requisições assíncronas à Web e pegar o resultado quando o solicitação estiver concluída. O espírito é o mesmo do que um objeto JavaScript XHR.

O conteúdo é também decodificado se o Content-Type é `application/json` e o resultado é automaticamente enviado através de `json.loads`.

A sintaxe para criar um requeste é:

```
from kivy.network.urlrequest import URLRequest
req = URLRequest(url, on_success, on_redirect, on_failure, on_error,
                 on_progress, req_body, req_headers, chunk_size,
                 timeout, method, decode, debug, file_path, ca_file,
                 verify)
```

Somente o primeiro argumento é obrigatório: o restante é opcional. Por padrão, uma requisição `GET` é enviada. Se o `URLRequest.req_body` não for `None`, uma requisição `POST` será enviada. Cabe a você ajustar o `URLRequest.req_headers` para atender às suas necessidades e a resposta à solicitação estará acessível como o parâmetro chamado “`result`” na função callback do evento `on_success`.

Example of fetching JSON:

```
def got_json(req, result):
    for key, value in result['headers'].items():
        print('{}: {}'.format(key, value))

req = URLRequest('https://httpbin.org/headers', got_json)
```

Exemplo de envio de dados (adaptado de um exemplo `httplib`):

```
import urllib

def bug_posted(req, result):
    print('Our bug is posted!')
    print(result)

params = urllib.urlencode({'@number': 12524, '@type': 'issue',
                           '@action': 'show'})
headers = {'Content-type': 'application/x-www-form-urlencoded',
           'Accept': 'text/plain'}
req = URLRequest('bugs.python.org', on_success=bug_posted, req_
                body=params,
                req_headers=headers)
```

Se você deseja uma requisição assíncrona, você pode invocar o método `wait()`.

```
class kivy.network.urlrequest.UrlRequest(url,      on_success=None,
                                           on_redirect=None,
                                           on_failure=None,
                                           on_error=None,
                                           on_progress=None,
                                           req_body=None,
                                           req_headers=None,
                                           chunk_size=8192,    time-
                                           out=None,   method=None,
                                           decode=True,           de-
                                           bug=False,   file_path=None,
                                           ca_file=None, verify=True,
                                           proxy_host=None,
                                           proxy_port=None,
                                           proxy_headers=None)
```

Bases: `threading.Thread`

Um UrlRequest. Veja o módulo da documentação para utiliza-lo.

Alterado na versão 1.5.1: Adiciona parâmetro `debug`

Alterado na versão 1.0.10: Adiciona parâmetro `method`

Alterado na versão 1.8.0: Parâmetro `decode` adicionado. Parâmetro `file_path` adicionado. Parâmetro `on_redirect` adicionado. Parâmetro `on_failure` adicionado.

Alterado na versão 1.9.1: Parâmetro `ca_file` adicionado. Parâmetro `verify` adicionado.

Alterado na versão 1.10.0: Parâmetro `proxy_host`, `proxy_port` e `proxy_headers` adicionados.

Parameters

`url: str` String de URL completa para chamada.

`on_success: callback(request, result)` Função de callback para chamada quando o resultado foi buscado.

`on_redirect: callback(request, result)` Função de callback invocado se o servidor retornar um Redirect.

`on_failure: callback(request, result)` Função de callback retornado para invocar se o servidor retornar um Cliente ou um Erro de Servidor.

`on_error: callback(request, error)` Função de callback invocada se um erro acontecer.

`on_progress: callback(request, current_size, total_size)` Função de callback que será enviada para informar o progresso do download. `total_size` pode ser igual a -1 se nenhum Content-Length

tiver sido relatado na resposta http. Esse retorno de chamada será chamado após cada *chunk_size* ser lido.

***req_body*: str, o padrão é None**Dados enviados numa requisição. Caso seja enviado *None*, uma chamada POST será enviado ao invés de um GET.

***req_headers*: dict, e o padrão é None**Cabeçalho para adicionar a solicitação.

***chunk_size*: int, o padrão é 8192**O tamanho de cada pedaço a ser lido, usado somente quando o callback de *on_progress* tiver sido enviado. Se você decrementar isso muito, um monte de callbacks de *on_progress* será disparado e demorará até você baixar tudo. Se você desejar usar a velocidade máxima de download, aumente o *chunk_size* ou não utilize *on_progress*.

***timeout*: int, o padrão é None**Se definido, as operações de bloqueio terão seu tempo esgotado após muitos segundos.

method*: str, o padrão é 'GET' (ou 'POST' se ***body for especificado)**O método HTTP a ser usado.

***decode*: bool, e o padrão é True**Se *False*, ignora a decodificação da resposta.

***debug*: bool, o padrão é False**Se *True*, será usado o *Logger.debug* para imprimir informações sobre a URL acessada/progresso/erros.

***file_path*: str, o padrão é None**Se definido, o resultado do *UrlRequest* será gravado neste caminho ao invés da memória.

***ca_file*: str, o padrão é None**Indica o caminho do arquivo de certificado da CA SSL para validar certificados HTTPS

***verify*: bool, o padrão é True**Se *False*, desabilita a verificação de certificados SSL CA

***proxy_host*: str, o padrão é None**Se definido, o host proxy usado para esta conexão.

***proxy_port*: int, o padrão é None**Se definido, e o *proxy_host* também estiver definido, a porta a ser usada para conexão com o servidor proxy.

***proxy_headers*: dict, o padrão é None**Se definido, e *proxy_host* também estiver definido, os cabeçalhos a serem enviados para o servidor de proxy na solicitação CONNECT

chunk_size

Retorna o tamanho do pedaço, usado somente no modo *progress* (quando o callback *on_progress* estiver definido)

decode_result(result, resp)

Decodifica o resultado retornado pela URL de acordo com seu Content-Type. Atualmente, suporta somente o *application/json*.

error

Retorna o erro da solicitação. Este valor não é determinado até a solicitação ser completada.

get_connection_for_scheme(scheme)

Retorna a classe de Connection para um Scheme particular. Esta é uma função interna que pode ser expandida para suportar esquemas padrões.

Esquemas atualmente suportados: http, https.

is_finished

Retorna *True* se a requisição tiver finalizado, se ela teve sucesso ou se falhou.

req_body = None

Corpo da solicitação enviado em `__init__`

req_headers = None

Cabeçalho da solicitação enviado em `__init__`

resp_headers

Se a solicitação tiver sido completada, retorna um dicionário contendo o cabeçalho da resposta. Senão, isso retornará *None*.

resp_status

Retorna o código do status da resposta se o requisição estiver completa, senão, retorna *None*.

result

Retorna o resultado da requisição. Este valor não é determinado até que a solicitação seja finalizada.

url = None

URL da solicitação

wait(delay=0.5)

Espera até a solicitação finalizar (até `resp_status` não é *None*)

Nota: Este método destina-se a ser usado no thread principal, e o callback será despachado desde o mesmo thread do qual você está chamando.

Novo na versão 1.1.0.

4.13 Storage (Armazenamento)

Novo na versão 1.7.0.

Aviso: Este módulo é experimental! Portanto a API está sujeita a alterações em versões futuras.

4.13.1 Utilização

A ideia por detrás do módulo Storage é ser capaz de abrir e armazenar qualquer quantidade de informações chave-valor através de uma chave indexada. O modelo padrão é abstrato, portanto, você não pode utilizá-lo diretamente. Disponibilizamos implementações tais como:

- **kivy.storage.dictstore.DictStore**: utilize um dicionário Python (dict) para armazenar
- **kivy.storage.jsonstore.JsonStore**: utilize um JSON arquivo de armazenamento
- **kivy.storage.redisstore.RedisStore**: utilize o Redis database with redis-py

4.13.2 Exemplos

Por exemplo, vamos utilizar um JsonStore:

```
from kivy.storage.jsonstore import JsonStore

store = JsonStore('hello.json')

# put some values
store.put('tito', name='Mathieu', org='kivy')
store.put('tshirtman', name='Gabriel', age=27)

# using the same index key erases all previously added key-value
→pairs
store.put('tito', name='Mathieu', age=30)

# get a value using a index key and key
print('tito is', store.get('tito')['age'])

# or guess the key/entry for a part of the key
for item in store.find(name='Gabriel'):
    print('tshirtmans index key is', item[0])
    print('his key value pairs are', str(item[1]))
```

Como os dados são armazenados, você pode verificar depois pra ver se a chave existe:

```
from kivy.storage.jsonstore import JsonStore

store = JsonStore('hello.json')
if store.exists('tito'):
    print('tite exists:', store.get('tito'))
    store.delete('tito')
```

4.13.3 API síncrona/assíncrona

Todos os métodos padrões (`get()`, `put()`, `exists()`, `delete()`, `find()`) possuem uma versão assíncrona.

Por exemplo, o método `get` possui um parâmetro de ‘callback’. Se for definido, o `callback` será utilizado para retornar o resultado para o usuário quando disponível: a requisição será assíncrona. Caso o `callback` seja igual a `None`, então, a requisição será síncrona e o resultado será retornado diretamente.

Sem callback (API síncrona):

```
entry = mystore.get('tito')
print('tito =', entry)
```

Com callback (API assíncrona):

```
def my_callback(store, key, result):
    print('the key', key, 'has a value of', result)
mystore.get('plop', callback=my_callback)
```

A assinatura do callback (na maior parte das vezes) será:

```
def callback(store, key, result):
    """
    store: the `Store` instance currently used.
    key: the key sought for.
    result: the result of the lookup for the key.
    """
```

4.13.4 Tipo de container Síncrono

A API de armazenamento emula o tipo de container para a API síncrona:

```
store = JsonStore('hello.json')

# original: store.get('tito')
store['tito']
```

```

# original: store.put('tito', name='Mathieu')
store['tito'] = {'name': 'Mathieu'}

# original: store.delete('tito')
del store['tito']

# original: store.count()
len(store)

# original: store.exists('tito')
'tito' in store

# original: for key in store.keys()
for key in store:
    pass

```

class kivy.storage.AbstractStore(kwargs)**

Bases: *kivy.event.EventDispatcher*

Classe abstrata utilizada para implementar o Store

async_clear(callback)

Versão assíncrona de *clear()*.

async_count(callback)

Retorna de forma assíncrona o número de entradas no armazenamento.

async_delete(callback, key)

Versão assíncrona de *delete()*.

Callback arguments

store: AbstractStore instânciaInstância de Armazena-
mento

*key: string*Nome da chave para pesquisar

*result: bool*Indicará True se o armazenamento tiver sido atualizado, or False caso não tenha sido (sem modificações). None caso ocorra algum erro.

async_exists(callback, key)

Versão assíncrona de *exists()*.

Callback arguments

store: AbstractStore instânciaInstância de Armazena-
mento

*key: string*Nome da chave para pesquisar

*result: bool*Resultado da pesquisa, None caso ocorra algum erro

async_find(callback, **filters)

Versão assíncrona de [find\(\)](#).

O callback será invocado para cada entrada no resultado.

Callback arguments

store: AbstractStore instânciaInstância de Armazenamento

key: stringNome da chave a ser pesquisada, ou None se chegarmos ao final dos resultados

result: boolIndicará True se o armazenamento tiver sido atualizado, or False caso não tenha sido (sem modificações). None caso ocorra algum erro.

async_get(callback, key)

Versão assíncrona de [get\(\)](#).

Callback arguments

store: AbstractStore instânciaInstância de Armazenamento

key: stringNome da chave para pesquisar

result: dictResultado da pesquisa, None caso ocorra algum erro

async_keys(callback)

Retorna todas as chaves do armazenamento de forma assíncrona.

async_put(callback, key, **values)

Versão assíncrona de [put\(\)](#).

Callback arguments

store: AbstractStore instânciaInstância de Armazenamento

key: stringNome da chave para pesquisar

result: boolIndicará True se o armazenamento tiver sido atualizado, or False caso não tenha sido (sem modificações). None caso ocorra algum erro.

clear()

Limpe todo o armazenamento.

count()

Retorna o número de entradas armazenadas.

delete(key)

Deleta a chave de armazenamento. Caso a chave não seja encontrada, uma exceção *KeyError* será levantada.

exists(key)

Verifica se a chave está armazenada.

find(filters)**

Retorna todas as entradas correspondentes aos filtros. As entradas são retornadas através de um gerador como uma lista de pares (key, entry) onde *entry* é um dict chave-valor

```
for key, entry in store.find(name='Mathieu'):
    print('key:', key, ', entry:', entry)
```

Devido ao fato de ser uma generator, você não pode usá-lo diretamente como sendo uma lista. Você precisa fazer:

```
# get all the (key, entry) availables
entries = list(store.find(name='Mathieu'))
# get only the entry from (key, entry)
entries = list((x[1] for x in store.find(name='Mathieu')))
```

get(key)

Pegue o par key-value armazenado em *key*. Caso a chave não exista, uma exceção *KeyError* será levantada.

keys()

Retorna uma lista com todas as chaves armazenadas.

put(key, **values)

Coloca um novo para key-value (fornecido em *values*) dentro do armazenamento. Quaisquer pares key-value existentes serão removidos.

4.13.5 Dicionários de Armazenamento

Utiliza um dicionário Python para o armazenamento.

class kivy.storage.dictstore.DictStore(filename, data=None, **kwargs)

Bases: *kivy.storage.AbstractStore*

Veja a implementação utilizando Pickled *dict*. Veja o módulo *kivy.storage* para maiores informações.

4.13.6 Armazenamento JSON

Um módulo *Storage* utilizado para salvar e abrir pares key-value de arquivos JSON.

class kivy.storage.jsonstore.JsonStore(filename, indent=None, sort_keys=False, **kwargs)

Bases: *kivy.storage.AbstractStore*

Implementação de armazenamento utilizando arquivo JSON para armazenar pares key-value. Veja a documentação do módulo [kivy.storage](#) para maiores informações.

4.13.7 Armazenamento Redis

Implementação para armazenamento Redis. Você precisa ter instalado redis-py.

Exemplo de uso:

```
from kivy.storage.redisstore import RedisStore

params = dict(host='localhost', port=6379, db=14)
store = RedisStore(params)
```

Toda par key-value será armazenado por padrão com o prefixo ‘store’. Você pode instanciar o armazenamento com outro prefixo, como este:

```
from kivy.storage.redisstore import RedisStore

params = dict(host='localhost', port=6379, db=14)
store = RedisStore(params, prefix='mystore2')
```

O dicionário de parâmetro será passado para a classe redis.StrictRedis.

Veja [redis-py](#).

```
class kivy.storage.redisstore.RedisStore(redis_params, **kwargs)
    Bases: kivy.storage.AbstractStore
```

Implementação de armazenamento utilizando base de dados Redis. Veja a documentação do módulo [kivy.storage](#) para maiores informações.

4.14 Ferramentas

O módulo Tools fornece vários Scripts úteis, módulos e exemplos.

4.14.1 Scripts

Alguns Scripts úteis são:

- `kvviewer.py`: para visualizar arquivos kv com atualização automática
- `benchmark.py`: fornece informações detalhadas de hardware OpenGL, bem como alguns benchmarks que medem o desempenho específico de kivy
- `reports.py`: fornece um relatório abrangente que cobre os provedores do seu sistema, bibliotecas, configuração, ambiente, dispositivos de entrada e opções.

- `texturecompress.py`: utilitário de linha de comando para compactar imagens nos formatos PVRTC ou ETC1
- `generate-icons.py`: gera um conjunto de ícones apropriados para os vários formatos de armazenamento e pacote
- `gles_compat/subset_gles.py`: examina a compatibilidade entre cabeçalhos GLEXT e GLES2 para encontrar subconjuntos compatíveis

4.14.2 Módulos

Os módulos de Ferramentas (Tool Modules) fornecem vários recursos para:

- `packaging`
- `text editor highlighting`

4.14.3 Outro

Outros recursos diversos incluem

- `pep8checker`: scripts de verificação PEP8 e hook git
- `theming`: demonstração de um tema alternativo para kivy
- `travis`: integração contínua com Travis

Este documento de ajuda está em constante desenvolvimento e está em desenvolvimento neste momento.

4.14.4 Empacotamento

Este módulo contém hooks do [PyInstaller](#) para ajudar no processo de construção de pacotes binários. O PyInstaller permite a produção de executáveis stand-alone (independentes), autônomos do seu aplicativo construído com Kivy para Windows, Linux e Mac.

Para maiores informações, por favor, veja [PyInstaller website](#)

Pyinstaller hooks

Módulos que exportam os métodos e parâmetros relacionados ao `pyinstaller`.

Hooks

PyInstaller vem como um hook padrão para o kivy que lista os módulos indiretamente importados que o pyinstaller não encontraria por conta própria :

`func:get_deps_all`. :func: `hookspath` retorna um caminho kivy hook alternativo, `kivy/tools/packaging/pyinstaller_hooks/kivy-hook.py` que não adiciona essas dependências à sua lista de importações ocultas e eles tem que ser explicitamente incluídos em vez disso.

É possível substituir um hook padrão fornecendo na linha de comando a opção `--additional-hooks-dir=HOOKSPATH`. Porque, embora o hook padrão ainda será executado, as variáveis globais importantes <<https://pythonhosted.org/PyInstaller/#hook-global-variables>>_, e.g. `excludedimports` and “`hiddenimports`”, serão substituídos pelo novo hook, se definido lá.

Além disso, pode-se adicionar um hook para ser executado após o hook padrão, passando e.g. `hookspath=[HOOKSPATH]` to the `Analysis` class. In both cases, `HOOKSPATH` é o caminho para um diretório que contém um arquivo chamado `hook-kivy.py` que é o pyinstaller hook para kivy poder processar após o hook padrão.

hiddenimports

Quando um módulo é importado indiretamente, e.g. com “`__import__`”, o pyinstaller não saberá sobre ele e o módulo tem que ser adicionado através de “`hiddenimports`”.

`hiddenimports` e outras variáveis hook, podem ser especificadas dentro de um hook como descrito acima. Além disso, essas variáveis podem ser passadas para `Analysis` e seus valores são então anexados aos valores do hook para essas variáveis.

A maioria dos módulos principais de kivy, e.g. Vídeo são importados indiretamente e, portanto, precisam ser adicionados em `hiddenimports`. O PyInstaller hook padrão adiciona todos os provedores. Para substituir, um kivy-hook modificado semelhante ao hook padrão, como: `func: hookspath` que só importa os módulos desejados pode ser adicionado. Um então usa: `func: get_deps_minimal` ou: `func: 'get_deps_all'` para obter a lista de módulos e os adiciona manualmente em um hook modificado ou os passa para “`Analysis`” no arquivo `spec`.

Gerador de Hook

`pyinstaller_hooks` inclui uma ferramenta para gerar um Hook que lista todos os módulos do provedor numa lista de modo que um possa comentar manualmente os provedores que não devem ser incluso. Para usar, faça:

```
python -m kivy.tools.packaging.pyinstaller_hooks hook filename
```

“filename” é o nome e caminho do arquivo hook a ser criado. Se o “filename” não é fornecido o hook é impresso no terminal.

`kivy.tools.packaging.pyinstaller_hooks.add_dep_paths()`

Deve ser chamado pelo hook. Adiciona os caminhos com as dependências bi-

nárias ao caminho do sistema para que o pyinstaller possa localizar os binários durante sua fase de rastreamento.

`kivy.tools.packaging.pyinstaller_hooks.get_deps_all()`

Igual a: `func:get_deps_minimal`, mas isso retorna todos os módulos kivy, quem podem indiretamente ser importados. Que inclui todos os possíveis provedores kivy.

Isso pode ser usado para obter uma lista de todos os possíveis provedores que podem então ser incluídos/excluídos manualmente, comentando os elementos na lista em vez de passar todos os itens. Veja a descrição do módulo.

RetornaUm dicionário com duas chaves `hiddenimports` e `excludes`. Seus valores são uma lista dos módulos correspondentes para incluir/excluir. Isso pode ser passado diretamente para `Analysis` com e.g. `:: a = Analysis(['..kivyexamplesdemotouchtracermain.py'], ... **get_deps_all())`

`kivy.tools.packaging.pyinstaller_hooks.get_deps_minimal(exclude_ignored=True, **kwargs)`

Retorna os módulos ocultos do Kivy bem como os módulos excluídos pra serem usados com o `Analysis`.

A função leva os módulos principais como argumento de palavra-chave e o seu valor indica qual dos fornecedores a incluir/excluir a partir da aplicação compilada.

Os possíveis nomes de palavra-chave são “audio, camera, clipboard, image, spelling, text, video, e window”. Seus valores podem ser:

True: Inclui o provedor atualOs provedores importados quando o módulo central é carregado nesse sistema são adicionados às importações ocultas. Isso é o padrão se o nome da palavra-chave não for especificado.

None: ExcluirNão devolva este módulo principal.

Uma String ou uma lista de Strings: Fornecedores para incluir
Cada String é o nome de um fornecedor para este módulo a ser incluído.

Por exemplo, `get_deps_minimal(video=None, window=True, audio=['gstplayer', 'ffpyplayer'], spelling='enchant')`

Irá excluir todos os provedores de vídeo, irá incluir os provedores gstreamer e ffpyplayer para áudio, irá incluir o provedor de encantamento para a ortografia e usará o provedor padrão atual para “window”.

`exclude_ignored`` se ``True (o padrão), se o valor para a biblioteca central é `None`, depois se `exclude_ignored` é True, não só a biblioteca não será incluída no `hiddenimports`, mas também será adicionado às importações excluídas para impedir que ele seja incluído acidentalmente por pyinstaller.

RetornaUm dicionario com duas chaves, `hiddenimports` e `excludes`. Seus valores são uma lista dos módulos correspondentes para incluir/excluir. Isso pode ser passado diretamente para `Analysis'` com e.g. :: a

```
= Analysis(['..kivyexamplesdemotouchtracermain.py'],
...           hookspath=hookspath(), runtime_hooks=[],
win_no_prefer_redirects=False, win_private_assemblies=False,
cipher=block_cipher, **get_deps_minimal(video=None, audio=None))
```

`kivy.tools.packaging.pyinstaller_hooks.get_factory_modules()`

Retorna uma lista de todos os módulos registrado na fábrica Kivy (Kivy Factory).

`kivy.tools.packaging.pyinstaller_hooks.get_hooks()`

Retorna o dict para os valores spec `hookspath` e `runtime_hooks`.

`kivy.tools.packaging.pyinstaller_hooks.hookspath()`

Retorna uma lista com o diretório que contém o alternativo (não o padrão incluído com pyinstaller) pyinstaller hook para kivy, `kivy/tools/packaging/pyinstaller_hooks/kivy-hook.py`. Geralmente é usado com o `"hookspath=hookspath()'` no arquivo spec.

O padrão do pyinstaller hook retorna todos os provedores principais usados, usando `get_deps_minimal()` para adicionar à sua lista de importações ocultas. Esse hook alternativo inclui apenas os módulos essenciais e deixa os provedores principais a serem incluídos adicionalmente com `get_deps_minimal()` or `get_deps_all()`.

`kivy.tools.packaging.pyinstaller_hooks.runtime_hooks()`

Retorna uma lista com os hooks de tempo de execução para kivy. Ele pode ser usado com `runtime_hooks=runtime_hooks()` no arquivo spec. Pyinstaller vem pré-instalado com esse hook.

4.15 Widgets

Widgets são elementos de usuários gráficos que fazem parte da **User Experience** (Experiência de Usuário). O módulo `kivy.uix` contém as classes para a criação, gerenciamento e manipulação dos Widgets. Por favor, para mais informações acesse: [Classe Widget](#).

Os Widgets do Kivy podem ser categorizados como:

- **UX widgets:** Widgets clássicos que compõem a interface de usuário e estão prontos para a construção de telas e outros Widgets mais complexos.

Rótulo, Botão, CheckBox, Imagem, Slider, Progress Bar, Entrada de Texto, Botão de alternar, Switch, Vídeo

- **Layouts:** Um Widget de leiaute não possui visualização, mas age como um gatilho e organiza os componentes contidos numa determinada forma ou num determinado sentido. Para maiores informações, acesse: [Layouts here](#).

Anchor Layout, Box Layout, Float Layout, GridLayout, PageLayout, Relative Layout, Scatter Layout, Stack Layout

- **Complex UX widgets:** Widgets não atômicos que são resultado de combinações de múltiplos widgets clássicos. Estes são chamados de complexos por serem formados pela junção de outros Widgets clássicos e por não possuírem um uso tão comum quanto os Widgets clássicos.

Bubble, Lista Drop-Down, FileChooser, Popup, Spinner, ListView, TabbedPanel, Player de Vídeo, VKeyboard,

- **Behaviors widgets:** Esses widgets não possuem renderização mas agem sobre as instruções gráficas ou interações do usuário como o toque, ou então, pelo comportamento de seus filhos.

Scatter, Stencil View

- **Screen manager:** Gerenciador de transições de telas, ou seja, quando é necessário alternar de uma tela para outra.

Gerenciador de Vídeo

4.15.1 Comportamentos

Novo na versão 1.8.0.

Mixin de classes de Comportamento

Este módulo implementa os comportamento que podem ser mixados [mixed in](#) com Widgets de base existentes. A ideia por detrás destas classes é encapsular propriedades e eventos associados com certos tipos de Widgets.

Isolando estas propriedades e eventos em uma classe mixin lhe permite definir sua própria implementação para widgets kivy padrões que podem agir como substitutos drop-in. Significa que você pode re-estilizar e redefinir widgets como desejado sem quebrar a compatibilidade: contanto que eles implementem os comportamentos corretamente, eles podem simplesmente substituir os widgets padrões.

Adicionando comportamentos

Digamos que você queira adicionar as capacidades da [Button](#) à uma [Image](#), você poderia fazer:

```
class IconButton(ButtonBehavior, Image):  
    pass
```

Isso lhe daria uma *Image* com eventos e propriedades herdados de *ButtonBehavior*. Por exemplo, os eventos *on_press* e *on_release* seriam disparados quando apropriados:

```
class IconButton(ButtonBehavior, Image):  
    def on_press(self):  
        print("on_press")
```

Ou em kv:

```
IconButton:  
    on_press: print('on_press')
```

Naturalmente, você também poderia ligar à qualquer mudança de propriedades que a classe comportamento oferece:

```
def state_changed(*args):  
    print('state changed')  
  
button = IconButton()  
button.bind(state=state_changed)
```

Nota: A classe comportamento deve sempre estar *antes* da classe widget. Se você não especificar a herança nessa ordem, o comportamento não funcionará, pois os métodos do comportamento são sobreescritos pelo método da classe listada primeiro.

Similarmente, se você combinar uma classe comportamento com uma classe que requere o uso de métodos também definidos pela classe comportamento, a classe resultante poderá não funcionar devidamente. Por exemplo, quando combinando a *ButtonBehavior* com uma *Slider*, duas das quais usam o método *on_touch_up()*, a classe resultante pode não funcionar devidamente.

Alterado na versão 1.9.1: As classes comportamento individuais, previamente em um grande arquivo *behaviors.py*, foi dividida em um único arquivo sob o módulo *behaviors*. Todos os comportamentos ainda são importados no módulo *behaviors* para serem acessíveis como antes (ex. ambos *from kivy.uix.behaviors import ButtonBehavior* e *from kivy.uix.behaviors.button import ButtonBehavior* funcionam).

```
class kivy.uix.behaviors.ButtonBehavior(**kwargs)  
    Bases: object
```

Esta classe *mixin* fornece um comportamento *Button*. Por favor veja a documentação do módulo *button behaviors* para maiores informações.

Events

on_press Disparado quando o botão é pressionado.

on_release Disparado quando o botão é solto (Ou seja, o toque/clique que pressionou o botão desaparece).

trigger_action(*duration=0.1*)

Dispara qualquer ação atrelada ao botão chamando ambas as chamadas *on_press* e *on_release*.

Isso simula um rápido pressionar de botão sem usar qualquer evento de toque.

duration é o comprimento da impressão em segundos. Passe 0 se quiseres que a ação aconteça instantaneamente.

Novo na versão 1.8.0.

class kivy.uix.behaviors.ToggleButtonBehavior(kwargs)**

Bases: *kivy.uix.behaviors.button.ButtonBehavior*

A classe *mixin* provê comportamento de *togglebutton*. Por favor, veja o módulo *togglebutton behaviors module* da documentação para maiores informações.

Novo na versão 1.8.0.

static get_widgets(*groupname*)

Retorna uma lista de Widgets contidas num determinado grupo. Caso o grupo não exista, uma lista vazia será retornada.

Nota: Sempre libere o resultado deste método! Manter a referência de um desses Widgets pode impedir que os mesmos seja destruídos pelo Garbage Collector. Em caso de dúvida, veja:

```
l = ToggleButtonBehavior.get_widgets('mygroup')
# do your job
del l
```

Aviso: É possível que algum Widgets que você tenha anteriormente deletado ainda esteja na lista. O garbage collector pode precisar liberar outros objetos antes de limpá-los.

class kivy.uix.behaviors.DragBehavior(kwargs)**

Bases: *object*

O DragBehavior *mixin* provê o comportamento de Arrastar (Drag). Quando combinando com um Widget, um retângulo de arrasto será definido

`drag_rectangle`. Por favor, veja o módulo [drag behaviors module](#) da documentação para maiores informações.

Novo na versão 1.8.0.

```
class kivy.uix.behaviors.FocusBehavior(**kwargs)
Bases: object
```

Comportamento dos provedores de teclado. Quando combinado com outros Widgets FocusBehavior eles permitirão o ciclo de passagem de foco pelo pressionar da tecla Tab. Por favor, veja [focus behavior module documentation](#) para maiores informações.

Novo na versão 1.9.0.

get_focus_next()

Returns the next focusable widget using either `focus_next` or the `children` similar to the order when tabbing forwards with the `tab` key.

get_focus_previous()

Returns the previous focusable widget using either `focus_previous` or the `children` similar to the order when `tab + shift` key are triggered together.

hide_keyboard()

Função de conveniência para esconder o teclado no modo de gerenciamento.

keyboard_on_key_down(window, keycode, text, modifiers)

O método vinculado ao teclado quando a instância estiver focada.

Quando a instancia se torna focada, esse método é vinculado ao teclado a será chamado por cada entrada pressionada. Os parâmetros são os mesmos ao [kivy.core.window.WindowBase.on_key_down\(\)](#).

Quando sobrescrevendo o método no widget derivado, super deve ser chamado para permitir ciclo de abas. Se o widget derivado deseja utilizar aba para seus próprios fins, pode chamar super após ter processado o personagem (se não deseja consumir a aba).

Semelhante a outras funções de teclado, deve retornar `True` se a tecla foi consumida.

keyboard_on_key_up(window, keycode)

O método vinculado ao teclado quando a instância estiver focada.

Quando a instancia se torna focada, esse método é vinculado ao teclado a será chamado por cada entrada pressionada. Os parâmetros são os mesmos aos do [kivy.core.window.WindowBase.on_key_down\(\)](#).

Quando sobrescrevendo o método no widget derivado, super deve ser chamado para permitir desativação no escape. Se o widget derivado deseja

utilizar o escape para seus próprios fins, pode chamar super após ter processado o personagem (se não deseja consumir o escape).

Veja [keyboard_on_key_down\(\)](#)

show_keyboard()

Função de conveniência para mostrar o teclado no modo gerenciado.

```
class kivy.uix.behaviors.CompoundSelectionBehavior(**kwargs)
Bases: object
```

O comportamento de seleção *mixin* <<https://en.wikipedia.org/wiki/Mixin>> _ implementa a lógica por trás do teclado e do toque na seleção de Widgets selecionáveis gerenciados pelo Widget derivado. Consulte a documentação [Módulo de Comportamentos de Seleção Composto](#) para obter mais informações.

Novo na versão 1.9.0.

clear_selection()

Desmarca todos os nós atualmente selecionados.

deselect_node(node)

Desmarca um nó possivelmente selecionado.

Ele é chamado pelo controlador quando ele desmarca um nó e também pode ser chamado de fora para desmarcar um nó diretamente. O widget derivado deve substituir esse método e alterar o nó para seu estado não selecionado quando isso é chamado

Parameters

node O nó a ser desselecionado

Aviso: Este método deve ser invocado por um Widget derivado utilizando super caso o mesmo tenha sido sobreescrito.

get_index_of_node(node, selectable_nodes)

(interno) retorna o índice do *node* com o 'selectable_nodes' retornado pelo [get_selectable_nodes\(\)](#).

get_selectable_nodes()

(Internal) Retorna uma lista dos nós que podem ser selecionados. Ele pode ser substituído pelo Widget derivado para retornar a lista correta.

Esta lista é usada para determinar quais nós deve ser selecionados com a seleção de grupo. Por exemplo, o último elemento na lista será selecionado quando a tecla Home for pressionada, PageDown moverá (ou adicione a, se Shift for mantido) a seleção da posição atual por nós negativos `page_count` a partir da posição selecionada atualmente no nessa lista e assim por diante. Ainda assim, os nós podem ser selecionados mesmo se não estiverem nessa lista.

Nota: É seguro alterar dinamicamente esta lista, incluindo remoção, adição ou reorganização dos elementos. Os nós podem ser selecionados mesmo se não estiverem nessa lista. E os nós selecionados removidos da lista permanecerão selecionados até que `deselect_node()` seja invocado.

Aviso: Os Layouts exibem seus filhos na ordem inversa. Ou seja, o conteúdo de `children` é exibido da direita para a esquerda, de baixo para cima. Portanto, internamente, os índices dos elementos retornados por esta função são invertidos para que o mesmo funcione por padrão na maioria dos Layouts, de modo que o resultado final seja consistente, por exemplo, Home, embora o mesmo selecionará o último elemento visualmente na lista, irá selecionar o primeiro elemento quando a contagem de cima para baixo e da esquerda para a direita. Se esse comportamento não for desejado, uma lista invertida deve ser retornada em vez disso.

O padrão é retornar `children`.

goto_node(key, last_node, last_node_idx)

(Interno) Utilizado pelo controlador para obter o nó na posição indicada pela tecla. A Key pode ser uma entradas de teclado, por exemplo, ou as entradas da rolagem da roda do mouse, por exemplo, scrollup 'last_node' é o último nó selecionado e será usado para encontrar o nó resultante. Por exemplo, se a chave está para cima, o nó retornado será um nó acima do último nó.

Pode ser substituído por um Widget derivado.

Parameters

`keystr`, a string usada para encontrar o nó desejado. Pode ser qualquer uma das teclas do teclado, bem como as teclas de rolagem do mouse, ScrollDown, ScrollRight e ScrollLeft. Se as letras forem rapidamente digitadas uma após a outra, as letras serão combinadas antes de serem passadas como Key e podem ser usadas para encontrar nós que tenham uma string associada que comece com essas letras.

`last_node` O último nós que foi selecionado.

`last_node_idx` O índice em cache do último nó selecionado na lista `get_selectable_nodes()`. Se a lista não tiver sido alterada, será economizado ter que procurar no índice de `last_node` dessa lista.

Returnstuple, o nó alvo de chave e seu índice na lista `get_selectable_nodes()`. Retornando (`last_node`,

last_node_idx) indica que um nó não foi encontrado.

select_node(node)

Seleciona um nó.

Ele é chamado pelo controlador quando ele seleciona um nó e pode ser chamado desde o lado de fora para selecionar um nó diretamente. Um Widget derivado deve substituir este método e alterar o estado do nó selecionado quando chamado.

Parameters

*node*O nó que será selecionado.

Returnsbool, *True* se o nó foi selecionado, *False* caso não seja.

Aviso: Este método deve ser invocado por um Widget derivado utilizando super caso o mesmo tenha sido sobreescrito.

select_with_key_down(keyboard, scancode, codepoint, modifiers, **kwargs)

Processa uma tecla selecionada. Isso é chamado quando uma tecla é usada para a seleção. Dependendo das teclas do teclado pressionadas e da configuração, isso por selecionar ou desmarcar nós ou um conjunto de nós de uma lista que estão selecionados, [get_selectable_nodes\(\)](#).

O parâmetro é tal que ele pode ser vinculado diretamente ao evento *on_key_down* do teclado. Portanto, é seguro chamar repetidamente quando a tecla é mantida pressionada, como é feito pelo teclado.

Returnsbool, *True* se o evento keypress foi utilizado, *False* caso não tenha sido.

select_with_key_up(keyboard, scancode, **kwargs)

(Interno) Processa a liberação de tecla. Isso deve ser chamado pelo Widget derivado quando uma tecla que [select_with_key_down\(\)](#) retornou *True* for liberado.

Os parâmetros são de tal forma que podem ser ligados diretamente ao evento *on_key_up* do teclado.

Returnsbool, *True* se a tecla solta foi usada, *False* caso não tenha sido.

select_with_touch(node, touch=None)

(Interno) Processa um toque sobre o nó. Isso deve ser chamado pelo Widget derivado quando um nó for tocado e deve ser usado pela seleção. Dependendo das teclas que forem pressionadas e da configuração, ela poderia selecionar ou desmarcar este e outros nós na lista de nós selecionáveis, [get_selectable_nodes\(\)](#).

Parameters

*node*O nó que recebeu o toque. Pode ser *None* para um tipo de toque de rolagem.

*touch*Opcionalmente, o toque. O padrão é *None*.

Returnsbool, *True* se o toque foi usado, *False* caso não tenha sido.

class kivy.uix.behaviors.CodeNavigationBehavior

Bases: *kivy.event.EventDispatcher*

Comportamento da Navegação de Código. Modifica o comportamento de navegação do TextInput fazendo-o como uma IDE ao invés de um processador de texto. Por favor, veja a documentação <kivy.uix.behaviors.codenavigation> para maiores informações.

Novo na versão 1.9.1.

class kivy.uix.behaviors.EmacsBehavior(kwargs)**

Bases: *object*

Um *mixin* que ativa o estilo de teclado Emacs e as teclas de atalho para o Widget *TextInput*. Por favor, veja a documentação *Emacs behaviors module* para obteres maiores informações.

Novo na versão 1.9.1.

delete_word_left()

Excluir texto à esquerda do cursor para o início da palavra

delete_word_right()

Remove o texto a direita do cursor para o fim da palavra

class kivy.uix.behaviors.CoverBehavior(kwargs)**

Bases: *object*

The CoverBehavior *mixin* provides rendering a texture covering full widget size keeping aspect ratio of the original texture.

Novo na versão 1.10.0.

Button Behavior

A classe *ButtonBehavior* *mixin* fornece o comportamento *Button*. Você pode combinar esta classe com outros Widgets, como uma *Image*, para fornecer botões alternativos que preservam o comportamento do botão do Kivy.

Para uma visão geral sobre o comportamento, por favor, veja a documentação a seguir: *behaviors*.

Exemplo

O exemplo seguinte adiciona comportamento de botão para uma imagem para fazer um CheckBox que comporta como um botão:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.uix.behaviors import ButtonBehavior

class MyButton(ButtonBehavior, Image):
    def __init__(self, **kwargs):
        super(MyButton, self).__init__(**kwargs)
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'
    ↴

    def on_press(self):
        self.source = 'atlas://data/images/defaulttheme/checkbox_on'

    def on_release(self):
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'
    ↴

class SampleApp(App):
    def build(self):
        return MyButton()

SampleApp().run()
```

Para maiores informações veja [ButtonBehavior](#).

`class kivy.uix.behaviors.button.ButtonBehavior(**kwargs)`
Bases: `object`

A classe `mixin` fornece um comportamento `Button`. Veja o módulo [button behaviors](#) da documentação para maiores informações.

Events

`on_press` Disparado quando o botão é pressionado.

`on_release` Disparado quando o botão é solto (Ou seja, o toque/clique que pressionou o botão desaparece).

always_release

Isso determina se o Widget ativa ou não um evento `on_release` se o `touch_up` estiver fora do Widget.

Novo na versão 1.9.0.

Alterado na versão 1.10.0: O valor padrão agora é *False*.

always_release é uma *BooleanProperty* e o padrão é *False*.

last_touch

Contém o último toque relevante recebido pelo Button. Isso pode ser usado em *on_press* ou *on_release* para saber qual toque despachou o evento.

Novo na versão 1.8.0.

last_touch é uma *ObjectProperty* e o padrão é *None*.

min_state_time

O período mínimo de tempo que o Widget deve permanecer no estado ‘down’.

Novo na versão 1.9.1.

min_state_time é um float e o padrão é 0.035. Esse valor é extraído desde *Config*.

state

O estado do botão, deve ser uma das opções ‘normal’ ou ‘down’. O estado será ‘down’ somente quando o botão estiver atualmente tocado/clicado, senão será ‘normal’.

state é uma *OptionProperty* e o padrão é ‘normal’.

trigger_action(duration=0.1)

Dispara qualquer ação atrelada ao botão chamando ambas as chamadas *on_press* e *on_release*.

Isso simula um rápido pressionar de botão sem usar qualquer evento de toque.

duration é o comprimento da impressão em segundos. Passe 0 se quiseres que a ação aconteça instantaneamente.

Novo na versão 1.8.0.

Comportamento de Navegação de Código

A *CodeNavigationBehavior* modifica o comportamento de navegação do *TextInput*, fazendo-o trabalhar como uma IDE ao invés de um processador de texto.

Usando este mixin dando ao *TextInput* a habilidade de reconhecer espaços em branco, pontuação e variância de caso (por exemplo, CamelCase) quando movido sobre o texto. Isso é atualmente usado pelo Widget *CodeInput*.

class kivy.uix.behaviors.codenavigation.CodeNavigationBehavior

Bases: *kivy.event.EventDispatcher*

Comportamento da Navegação de Código. Modifica o comportamento de navegação do *TextInput* fazendo-o como uma IDE ao invés de um processador

de texto. Por favor, veja a documentação <kivy.uix.behaviors.codenavigation> para maiores informações.

Novo na versão 1.9.1.

Comportamento de Seleção Composto

As classes class:~*kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior* mixin class Implementa a lógica de seleção sobre o teclado e o toque de widgets disponíveis para serem selecionados, gerenciados pelo widget derivado. Por exemplo, pode ser combinado com uma classe *GridLayout* para adicionar uma seleção para o layout.

Conceitos de seleção compostos

No seu núcleo, ele mantém uma lista dinâmica de widgets que podem ser selecionados. Então, como os toques e entrada de teclado são passados, ele seleciona um ou mais dos widgets com base nessas entradas. Por exemplo, ele usa os botões de rolagem do mouse e teclado para cima / para baixo para percorrer a lista de widgets. Multiselection também pode ser alcançado usando as teclas shift e ctrl do teclado.

Finalmente, além das entradas de teclado tipo up / down, a seleção composta também pode aceitar letras do teclado para ser usado para selecionar nós com sequências associadas que começam com essas letras, semelhante a como os arquivos são selecionados por um navegador de arquivos.

Mecanismo de Seleção

Quando o controlador precisa selecionar um nó, ele chama `select_node()` e :meth:`deselect_node`. Portanto, eles devem ser sobrescritos para alterar a seleção de nó. Por padrão, a classe não escuta os eventos de teclado ou de toque, então o Widget derivado deverá invocar `select_with_touch()`, `select_with_key_down()` e `select_with_key_up()` nos eventos em que desejar passar para fins de seleção.

Exemplo

Para adicionar a seleção a um GridLayout que contenha Widgetclass:~*kivy.uix.Button*. Para cada botão adicionado ao layout, precisarás vincular o `on_touch_down` do botão para passar `select_with_touch()` os eventos de toque:

```
from kivy.uix.behaviors.compoundselection import_
    CompoundSelectionBehavior
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
```

```

from kivy.uix.behaviors import FocusBehavior
from kivy.core.window import Window
from kivy.app import App


class SelectableGrid(FocusBehavior, CompoundSelectionBehavior, _  

    GridLayout):
    def keyboard_on_key_down(self, window, keycode, text, _  

        modifiers):
        """Based on FocusBehavior that provides automatic keyboard  

        access, key presses will be used to select children.  

        """
        if super>SelectableGrid, self).keyboard_on_key_down(  

            window, keycode, text, modifiers):
            return True
        if self.select_with_key_down(window, keycode, text, _  

            modifiers):
            return True
        return False

    def keyboard_on_key_up(self, window, keycode):
        """Based on FocusBehavior that provides automatic keyboard  

        access, key release will be used to select children.  

        """
        if super>SelectableGrid, self).keyboard_on_key_up(window, _  

            keycode):
            return True
        if self.select_with_key_up(window, keycode):
            return True
        return False

    def add_widget(self, widget):
        """Override the adding of widgets so we can bind and catch  

        their
        *on_touch_down* events. """
        widget.bind(on_touch_down=self.button_touch_down,
                   on_touch_up=self.button_touch_up)
        return super>SelectableGrid, self).add_widget(widget)

    def button_touch_down(self, button, touch):
        """Use collision detection to select buttons when the  

        touch occurs
        within their area. """
        if button.collide_point(*touch.pos):
            self.select_with_touch(button, touch)

    def button_touch_up(self, button, touch):

```

```

    """ Use collision detection to de-select buttons when the_
→touch
    occurs outside their area and *touch_multiselect* is not_
→True. """
    if not (button.collide_point(*touch.pos) or
            self.touch_multiselect):
        self.deselect_node(button)

    def select_node(self, node):
        node.background_color = (1, 0, 0, 1)
        return super(SelectableGrid, self).select_node(node)

    def deselect_node(self, node):
        node.background_color = (1, 1, 1, 1)
        super(SelectableGrid, self).deselect_node(node)

    def on_selected_nodes(self, grid, nodes):
        print("Selected nodes = {}".format(nodes))

class TestApp(App):
    def build(self):
        grid = SelectableGrid(cols=3, rows=2, touch_
→multiselect=True,
                           multiselect=True)
        for i in range(0, 6):
            grid.add_widget(Button(text="Button {}".format(i)))
        return grid

TestApp().run()

```

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

`class kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior(**kwargs)`
Bases: `object`

O comportamento de seleção *mixin* <<https://en.wikipedia.org/wiki/Mixin>> _ implementa a lógica por trás do teclado e do toque na seleção de Widgets selecionáveis gerenciados pelo Widget derivado. Consulte a documentação *Módulo de Comportamentos de Seleção Composto* para obter mais informações.

Novo na versão 1.9.0.

`clear_selection()`

Remove a seleção de todos os nós selecionados.

deselect_node(node)

Desmarca um nó possivelmente selecionado.

Ele é chamado pelo controlador quando ele desmarca um nó e também pode ser chamado de fora para desmarcar um nó diretamente. O widget derivado deve substituir esse método e alterar o nó para seu estado não selecionado quando isso é chamado

Parameters

node O nó a ser desselecionado

Aviso: Este método deve ser invocado por um Widget derivado utilizando super caso o mesmo tenha sido sobreescrito.

get_index_of_node(node, selectable_nodes)

(interno) retorna o índice do *node* com o 'selectable_nodes' retornado pelo **get_selectable_nodes()**.

get_selectable_nodes()

(Internal) Retorna uma lista dos nós que podem ser selecionados. Ele pode ser substituído pelo Widget derivado para retornar a lista correta.

Esta lista é usada para determinar quais nós devem ser selecionados com a seleção de grupo. Por exemplo, o último elemento na lista será selecionado quando a tecla Home for pressionada, PageDown moverá (ou adicione a, se Shift estiver pressionado) a seleção da posição atual por nós negativos **page_count** começa a partir da posição do nó atualmente selecionada nessa lista e assim por diante. Ainda assim, os Nós podem ser selecionados mesmo se não estiverem nessa lista.

Nota: É seguro alterar dinamicamente esta lista, incluindo remoção, adição ou reorganização dos elementos. Os nós podem ser selecionados mesmo se não estiverem nessa lista. E os nós selecionados removidos da lista permanecerão selecionados até que **deselect_node()** seja invocado.

Aviso: Os Layouts exibem seus filhos na ordem inversa. Ou seja, o conteúdo de **children** é exibido da direita para a esquerda, de baixo para cima. Portanto, internamente, os índices dos elementos retornados por esta função são invertidos para que o mesmo funcione por padrão na maioria dos Layouts, de modo que o resultado final seja consistente, por exemplo, Home, embora o mesmo selecionará o último elemento visualmente na lista, irá selecionar o primeiro elemento quando a contagem de cima para baixo e da esquerda para a direita. Se esse comportamento

não for desejado, uma lista invertida deve ser retornada em vez disso.

O padrão é retornar `children`.

goto_node(key, last_node, last_node_idx)

(Interno) Utilizado pelo controlador para obter o nó na posição indicada pela tecla. A Key pode ser uma entradas de teclado, por exemplo, ou as entradas da rolagem da roda de do mouse, por exemplo, scrollup 'last_node' é o último nó selecionado e será usado para encontrar o nó resultante. Por exemplo, se a chave está para cima, o nó retornado será um nó acima do último nó.

Pode ser substituído por um Widget derivado.

Parameters

`keystr`, a string usada para encontrar o nó desejado. Pode ser qualquer uma das teclas do teclado, bem como as teclas de rolagem do mouse, ScrollDown, ScrollRight e ScrollLeft. Se as letras forem rapidamente digitadas uma após a outra, as letras serão combinadas antes de serem passadas como Key e podem ser usadas para encontrar nós que tenham uma string associada que comece com essas letras.

`last_node`O último nós que foi selecionado.

`last_node_idx`O índice em cache do último nó selecionado na lista `get_selectable_nodes()`. Se a lista não tiver sido alterada, será economizado ter que procurar no índice de `last_node` dessa lista.

`Returnstuple`, o nó alvo de chave e seu índice na lista `get_selectable_nodes()`. Retornando (`last_node`, `last_node_idx`) indica que um nó não foi encontrado.

keyboard_select

Determina se o teclado pode ser usado para seleção. Se `False`, a entrada de teclado será ignorada.

`keyboard_select` é uma `BooleanProperty` e o padrão é `True`.

multiselect

Determina se vários nós podem ser selecionados. Se ativado, o Shift e o CTRL do teclado, opcionalmente combinados com o toque, por exemplo, poderão selecionar vários Widgets da maneira normalmente esperada. Isso comanda `touch_multiselect` quando `False`.

`multiselect` é uma `BooleanProperty` e o padrão é `False`.

nodes_order_reversed

(Interno) Indica se a ordem dos nós mostrada de cima para baixo é invertida

em comparação com a sua ordem em `get_selectable_nodes()` (por exemplo, como a propriedade de filhos é invertida em comparação com a forma como ela é exibida).

page_count

Determina o quanto o nó selecionado é movido para cima ou para baixo, em relação à posição do último nó selecionado, quando o PageUp (ou PageDown) for pressionado.

`page_count` é uma *NumericProperty* e o padrão é 10.

right_count

Determina o quanto o nó selecionado é movido para cima ou para baixo, em relação à posição do último nó selecionado, quando a seta direita (ou esquerda) no teclado for pressionada.

`right_count` é uma *NumericProperty* e o padrão é 1.

scroll_count

Determina o quanto o nó selecionado é movido para cima ou para baixo, em relação à posição do último nó selecionado, quando a (roda do meio) rolagem do mouse for girada.

`right_count` é uma *NumericProperty* e o padrão é 0.

select_node(node)

Seleciona um nó.

Ele é chamado pelo controlador quando ele seleciona um nó e pode ser chamado desde o lado de fora para selecionar um nó diretamente. Um Widget derivado deve substituir este método e alterar o estado do nó selecionado quando chamado.

Parameters

`node` O nó que será selecionado.

Returns bool, *True* se o nó foi selecionado, *False* caso não seja.

Aviso: Este método deve ser invocado por um Widget derivado utilizando super caso o mesmo tenha sido sobreescrito.

select_with_key_down(keyboard, scancode, codepoint, modifiers, **kwargs)

Processa uma tecla selecionada. Isso é chamado quando uma tecla é usada para a seleção. Dependendo das teclas do teclado pressionadas e da configuração, isso por selecionar ou desmarcar nós ou um conjunto de nós de uma lista que estão selecionados, `get_selectable_nodes()`.

O parâmetro é tal que ele pode ser vinculado diretamente ao evento `on_key_down` do teclado. Portanto, é seguro chamar repetidamente quando

a tecla é mantida pressionada, como é feito pelo teclado.

Returnsbool, *True* se o evento keypress foi utilizado, *False* caso não tenha sido.

select_with_key_up(*keyboard*, *scancode*, ***kwags*)

(Interno) Processa a liberação de tecla. Isso deve ser chamado pelo Widget derivado quando uma tecla que [select_with_key_down\(\)](#) retornou *True* for liberado.

Os parâmetros são de tal forma que podem ser ligados diretamente ao evento *on_key_up* do teclado.

Returnsbool, *True* se a tecla solta foi usada, *False* caso não tenha sido.

select_with_touch(*node*, *touch=None*)

(Interno) Processa um toque sobre o nó. Isso deve ser chamado pelo Widget derivado quando um nó for tocado e deve ser usado pela seleção. Dependendo das teclas que forem pressionadas e da configuração, ela poderia selecionar ou desmarcar este e outros nós na lista de nós selecionáveis, [get_selectable_nodes\(\)](#).

Parameters

*node*O nó que recebeu o toque. Pode ser *None* para um tipo de toque de rolagem.

*touch*Opcionalmente, o toque. O padrão é *None*.

Returnsbool, *True* se o toque foi usado, *False* caso não tenha sido.

selected_nodes

A lista de nós selecionados.

Nota: Vários nós podem ser selecionados imediatamente após outro, por ex. usando o teclado. Ao ouvir [selected_nodes](#), deve-se estar ciente disso.

[selected_nodes](#) é uma *ListProperty* e o padrão é uma lista vazia `[]`. Ele é somente leitura e não deve ser modificado.

text_entry_timeout

Ao digitar caracteres rapidamente (ou seja, a diferença de tempo desde o último caractere é menor que [text_entry_timeout](#)), as teclas são concatenadas e o texto combinado é passado como o argumento-chave de [goto_node\(\)](#).

Novo na versão 1.10.0.

touch_deselect_last

Determines whether the last selected node can be deselected when

`multiselect` or `touch_multiselect` is False.

Novo na versão 1.10.0.

`touch_deselect_last` is a *BooleanProperty* and defaults to True on mobile, False on desktop platforms.

touch_multiselect

Um modo de toque especial que determina se os eventos de toque, conforme processados por `select_with_touch()`, adicionarão o nó atualmente tocado à seleção ou se limpará a seleção antes de adicionar o nó. Isso permite a seleção de vários nós simplesmente tocando-os.

Isso é diferente de `multiselect` porque quando é *True*, simplesmente tocando em um nó não selecionado irá selecioná-lo, mesmo se o CTRL não estiver pressionado. Se for *Falso*, no entanto, CTRL deve ser pressionado para adicionar a seleção quando `multiselect` for *True*.

Nota: `multiselect`, quando *Falso*, desabilitará `touch_multiselect`.

`touch_multiselect` é uma *BooleanProperty* e o padrão é *False*.

up_count

Determina o quanto o nó selecionado é movido para cima ou para baixo, em relação à posição do último nó selecionado, quando a seta para cima (ou para baixo) no teclado é pressionada.

`up_count` é uma *NumericProperty* e o padrão é 1.

Cover Behavior

The `CoverBehavior` mixin is intended for rendering textures to full widget size keeping the aspect ratio of the original texture.

Use cases are i.e. rendering full size background images or video content in a dynamic layout.

For an overview of behaviors, please refer to the `behaviors` documentation.

Example

The following examples add cover behavior to an image:

In python:

```
from kivy.app import App
from kivy.uix.behaviors import CoverBehavior
from kivy.uix.image import Image
```

```

class CoverImage(CoverBehavior, Image):

    def __init__(self, **kwargs):
        super(CoverImage, self).__init__(**kwargs)
        texture = self._coreimage.texture
        self.reference_size = texture.size
        self.texture = texture

class MainApp(App):

    def build(self):
        return CoverImage(source='image.jpg')

MainApp().run()

```

In Kivy Language:

```

CoverImage:
    source: 'image.png'

<CoverImage@CoverBehavior+Image>:
    reference_size: self.texture_size

```

See [CoverBehavior](#) for details.

`class kivy.uix.behaviors.cover.CoverBehavior(**kwargs)`
Bases: object

The CoverBehavior [mixin](#) provides rendering a texture covering full widget size keeping aspect ratio of the original texture.

Novo na versão 1.10.0.

cover_pos

Position of the aspect ratio aware texture. Gets calculated in `CoverBehavior.calculate_cover`.

`cover_pos` is a [ListProperty](#) and defaults to [0, 0].

cover_size

Size of the aspect ratio aware texture. Gets calculated in `CoverBehavior.calculate_cover`.

`cover_size` is a [ListProperty](#) and defaults to [0, 0].

reference_size

Reference size used for aspect ratio approximation calculation.

`reference_size` is a [ListProperty](#) and defaults to [].

Comportamento de Arrastar

A classe **DragBehavior mixin** provê o comportamento de arrastar. Quando combinado com um Widget, um retângulo de arrastamento será definido pelo Widget de arrasto **drag_rectangle**.

Exemplo

O exemplo seguinte cria um label arrastável:

```
from kivy.uix.label import Label
from kivy.app import App
from kivy.uix.behaviors import DragBehavior
from kivy.lang import Builder

# You could also put the following in your kv file...
kv = '''
<DragLabel>:
    # Define the properties for the DragLabel
    drag_rectangle: self.x, self.y, self.width, self.height
    drag_timeout: 10000000
    drag_distance: 0

FloatLayout:
    # Define the root widget
    DragLabel:
        size_hint: 0.25, 0.2
        text: 'Drag me'
'''


class DragLabel(DragBehavior, Label):
    pass


class TestApp(App):
    def build(self):
        return Builder.load_string(kv)

TestApp().run()
```

```
class kivy.uix.behaviors.drag.DragBehavior(**kwargs)
Bases: object
```

O DragBehavior **mixin** provê o comportamento de Arrastar (Drag). Quando combinando com um Widget, um retângulo de arrasto será definido **drag_rectangle**. Por favor, veja a o módulo **drag behaviors module** da

documentação para maiores informações.

Novo na versão 1.8.0.

drag_distance

Distância a ser movida antes de arrastar o *DragBehavior*, em pixels. Assim que a distância for percorrida, a classe *DragBehavior* começará a arrastar e nenhum evento de toque será enviado para os Widgets filhos. É aconselhável basear esse valor em dpi da tela do seu dispositivo de destino.

drag_distance é um *NumericProperty* e o padrão para *scroll_distance* é definido pelo usuário *Config* (20 pixels por padrão).

drag_rect_height

Altura do eixo retangular delimitador alinhado onde é permitido arrastar.

drag_rect_height é um *NumericProperty* e o padrão é 100.

drag_rect_width

Largura do eixo retângulo delimitador onde o arrastar é permitido.

drag_rect_width é um *NumericProperty* e o padrão é 100.

drag_rect_x

Posição X do eixo retangular delimitador alinhado onde é permitido arrastar (em coordenadas de janela).

drag_rect_x é um *NumericProperty* e o padrão é 0.

drag_rect_y

Posição Y do eixo retangular delimitador alinhado onde é permitido arrastar (em coordenadas de janela).

drag_rect_Y é um *NumericProperty* e o padrão é 0.

drag_rectangle

Posição e tamanho do eixo de alinhamento retângulo onde é permitido arrastar.

drag_rectangle é uma propriedade *ReferenceListProperty* de (*drag_rect_x*, *drag_rect_y*, *drag_rect_width*, *drag_rect_height*).

drag_timeout

Tempo máximo permitido para o acionamento do *drag_distance*, em milissegundos. Se o usuário não tiver movido *drag_distance* dentro do tempo máximo, a ação de arrastar será desativada, e o evento de toque será despachado para os Widgets filhos.

drag_timeout é um *NumericProperty* e o padrão para *scroll_timeout* como definido pelo usuário em *Config* (55 milliseconds por padrão).

Comportamento Emacs

A *EmacsBehavior* mixin permite que você adicione teclas de atalho *Emacs* para movimentos básicos como o de edição de Widget *TextInput*. Os atalhos atualmente disponíveis estão listados abaixo:

Teclas de Atalho Emacs

Teclas de Atalho	Descrição
Controle + A	Move o cursor para o início da linha
Control + E	Move o cursor para o final da linha
Control + F	Move o cursor para o caractere da direita
Control + B	Move o cursor para o caractere da direita
Alt + F	Move o cursor para o final da para a direita
Alt + B	Move o Cursor para o início da palavra a esquerda
Alt + Backspace	Excluir texto à esquerda do cursor para o início da palavra
Alt + D	Remove o texto a direita do cursor para o fim da palavra
Alt + W	Copia a seleção
Control + W	Recorta a seleção
Control + Y	Cola a seleção

Aviso: Se você tiver o módulo *inspector* ativado, a tecla de atalho para abrir o inspetor é (Control + E) entrará em conflito com a tecla de atalho para ir ao final da linha (o cursor ainda se moverá ao final da linha, e o inspetor também se abrirá).

`class kivy.uix.behaviors.emacs.EmacsBehavior(**kwargs)`

Bases: `object`

Um *mixin* que ativa o estilo de teclado Emacs e as teclas de atalho para o Widget *TextInput*. Por favor, veja a documentação *Emacs behaviors module* para obteres maiores informações.

Novo na versão 1.9.1.

`delete_word_left()`

Excluir texto à esquerda do cursor para o início da palavra

`delete_word_right()`

Remove o texto a direita do cursor para o fim da palavra

`key_bindings`

Nome da String que determina o tipo de ligações de chave para usar com o *TextInput*. Isso permite vincular teclas de atalho Emacs para ativar/desativar programaticamente widgets que herdam de *EmacsBehavior*. Se o valor não for 'emacs', as ligações do Emacs serão

desabilitadas. Use 'default' para trocar o padrão de ligação das teclas de atalho do TextInput.

`key_bindings` é um *StringProperty* e o padrão é 'emacs'.

Novo na versão 1.10.0.

Comportamento do Foco

A classe *FocusBehavior mixin* fornece o controle do foco do teclado. Quando combinado com outros widgets o FocusBehavior permite o controle do foco entre eles pressionando a tecla Tab. Além disso, ao obter o foco, a instância receberá automaticamente a entrada do teclado.

O foco é bastante diferente da seleção e está intimamente ligado com o teclado; cada teclado pode ter o foco em zero ou um Widget e cada Widget pode ter somente um foco de teclado. Mesmo assim, vários teclados podem ter o foco ao mesmo tempo em diferentes Widgets. Quando for preciso remover o foco, o Widget que tiver com o focado e com teclado ativo será desfocado.

Gerenciamento do Foco

Em essência, o foco é implementado como uma lista duplamente encadeada, onde cada nó contém uma referência (fraca) à instância anterior e posterior a ela, como pode ser visto quando se vai pressionando tab (frente) ou shift + tab (para trás) para mover o foco. Se o widget anterior ou posterior não for definido, `focus_next` e `focus_previous` o padrão será *None*.

Por exemplo, o ciclo do foco entre os elementos de um *Button* e um *GridLayout*:

```
class FocusButton(FocusBehavior, Button):
    pass

grid = GridLayout(cols=4)
for i in range(40):
    grid.add_widget(FocusButton(text=str(i)))
# clicking on a widget will activate focus, and tab can now be used
# to cycle through
```

Quando utilizado um software de teclado, geralmente em dispositivos mobile e dispositivos de toque, o comportamento do teclado que aparecerá será determinado pela propriedade *softinput_mode*. Você pode utilizar essa propriedade para assegurar que o Widget focado não será sobreposto pelo teclado virtual.

Inicialização do Foco

Os Widgets precisam estar visíveis para receberem o foco. Isso significa que definir a propriedade *focus* deles como sendo True antes de estarem visíveis não terá qualquer efeito. Para inicializar o foco, você pode utilizar o evento ‘on_parent’:

```
from kivy.app import App
from kivy.uix.textinput import TextInput

class MyTextInput(TextInput):
    def on_parent(self, widget, parent):
        self.focus = True

class SampleApp(App):
    def build(self):
        return MyTextInput()

SampleApp().run()
```

Se você estiver utilizando a classe *popup*, você pode utilizar o evento ‘on_open’.

Para uma visão geral sobre o comportamento, por favor, veja a documentação a seguir: *behaviors*.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

class kivy.uix.behaviors.focus.FocusBehavior(kwargs)**
Bases: object

Comportamento dos provedores de teclado. Quando combinado com outros Widgets FocusBehavior eles permitirão o ciclo de passagem de foco pelo pressionar da tecla Tab. Por favor, veja *focus behavior module documentation* para maiores informações.

Novo na versão 1.9.0.

focus

Se a instância atual não tiver o foco.

Configurá-lo como True ligará e/ou solicitará o teclado, e a entrada será encaminhada para a instância. Configurá-lo como sendo False desvinculará e/ou libertará o teclado. Para um determinado teclado, apenas um único Widget poderá ter o foco, portanto, quando um receber o foco o outro o perderá automaticamente, isto é, não terá mais o foco.

Ao utilizar um teclado de software, consulte a propriedade *softinput_mode* para determinar como a exibição do teclado será

manipulada.

`focus` é um *BooleanProperty* e o padrão é False.

focus_next

A instância do foco é adquirida *FocusBehavior* quando a tecla Tab é pressionada e a instância tiver o foco, se não tiver *None* ou *StopIteration*.

Quando a aba é pressionada, o ciclo do foco atravessa todos os Widgets *FocusBehavior* que estão ligados através de `focus_next` e podem receber o foco. Se `focus_next` for *None*, ao invés de percorrer os filhos da lista para encontrar o próximo Widget que pode ser focado. Finalmente, se `focus_next` é a classe *StopIteration*, o foco não avançará, mas terminará aqui.

`focus_next` é um *ObjectProperty* e o padrão é *None*.

focus_previous

A instância de *FocusBehavior* recebe o foco quando Shift+Tab é pressionado nesta instância, senão None ou *StopIteration*.

Quando Shift + Tab é pressionado, o foco faz o ciclo através de todos os Widgets *FocusBehavior* que estão ligados através de `focus_previous` e podem receber o foco. Se `focus_previous` for *None*, ao invés de percorrer a árvore de filhos para encontrar o Widget focusable anterior. Finalmente, se `focus_previous` for a classe *StopIteration*, o foco não se moverá para trás, mas terminará aqui.

`focus_previous` é um *ObjectProperty* e o padrão é *None*.

focused

Um apelido de `focus`.

`focused` é uma *BooleanProperty* e o padrão é False.

Aviso: `focused` é um apelido de `focus` e será removido na versão 2.0.0.

get_focus_next()

Returns the next focusable widget using either `focus_next` or the `children` similar to the order when tabbing forwards with the tab key.

get_focus_previous()

Returns the previous focusable widget using either `focus_previous` or the `children` similar to the order when tab + shift key are triggered together.

hide_keyboard()

Função de conveniência para esconder o teclado no modo de gerenciamento.

ignored_touch = []

Uma lista de toques que não devem ser usados para desfocar. Depois de on_touch_up, cada toque que não esteja em: attr: *ignored_touch* irá desfocar todos os widgets focados se o modo de teclado config não for multi. Os toques em widgets que podem ser focados e que foram usados para focar são automaticamente adicionados aqui.

Exemplo de uso:

```
class Unfocusable(Widget):  
  
    def on_touch_down(self, touch):  
        if self.collide_point(*touch.pos):  
            FocusBehavior.ignored_touch.append(touch)
```

Observe que você precisa acessar isso como um classe, não como uma variável de instância.

input_type

Tipo de entrada de teclado para solicitação.

Novo na versão 1.8.0.

input_type é uma OptionsProperty e o padrão é ‘text’. Pode ser um de ‘text’, ‘number’, ‘url’, ‘mail’, ‘datetime’, ‘tel’ ou ‘address’.

is_focusable

Se a instância puder ser focada. Se focada, esta perderá o foco quando for definido como *False*.

is_focusable é um BooleanProperty e o padrão é *True* nos Desktop (por exemplo *desktop* é *True* em *config*), do contrário será *False*.

keyboard

O teclado será vinculado com (ou vinculado com o Widget) quando focado.

Quando *None*, um teclado é solicitado e liberado sempre que o Widget vier de dentro pra fora do foco. Se não for *None*, ele deve ser um teclado, que ficará vinculado e não vinculado a partir do Widget sempre que estiver dentro ou fora do foco. Será útil somente quando mais de um teclado estiver disponível, portanto é recomendável que seja definido como *None* quando somente um teclado estiver disponível.

Se mais de um teclado estiver disponível, sempre que uma instância for focada, um novo teclado será solicitado se *None*. A menos que as outras instâncias percam o foco (por exemplo, se a guia foi usada), um novo teclado aparecerá. Quando isso não for desejado, a propriedade do teclado poderá ser usada. Por exemplo, se houver dois usuários com dois teclados, cada teclado poderá ser atribuído a diferentes grupos de instâncias de *FocusBehavior*, garantindo que dentro de cada grupo, apenas um *FocusBehavior*

tenha o foco e receba entrada do teclado correto. Consulte `keyboard_mode` em [config](#) para obter mais informações sobre os modos de teclado.

Teclado e Comportamento do Foco

Quando utilizar o teclado, existem alguns comportamentos padrão importantes que você deve ter em mente.

- Quando o `keyboard_mode` do Config for multi, cada novo toque será considerado um toque por um usuário diferente e irá definir o foco (se clicado em um focusable) com um novo teclado. Os elementos já focados não perderão o foco (mesmo que um Widget não-focado seja tocado).
- Se a propriedade do teclado estiver definida, esse teclado será usado quando a instância for focalizada. Se os widgets com teclados diferentes estiverem ligados através de: attr: `focus_next` e: attr: `focus_previous`, então, à medida que forem tabulados, teclados diferentes ficarão ativos. Portanto, normalmente é indesejável ligar instâncias às quais são atribuídos teclados diferentes.
- Quando um Widget tiver o foco, definir o seu teclado como `None` irá remover seu teclado, mas o Widget, em seguida, tentará obter imediatamente outro teclado. Para remover o teclado, defina a propriedade [focus](#) como sendo `False`.
- Ao usar um teclado de software, típico em dispositivos móveis e de toque, o comportamento de exibição do teclado será determinado pela propriedade `softinput_mode`. Poderás usar esta propriedade para garantir que o Widget focado não seja coberto ou obscurecido.

`keyboard` é um [AliasProperty](#) e o padrão é `None`.

`keyboard_mode`

Determina como a visibilidade do teclado deve ser gerenciada. ‘auto’ resultará no comportamento padrão de exibir/ocultar o foco. ‘managed’ requererá a definição da visibilidade do teclado manualmente, ou invocará as funções auxiliares `show_keyboard()` e `hide_keyboard()`.

`keyboard_mode` é um [OptionsProperty](#) e o padrão é ‘auto’. Pode ser ‘auto’ ou então ‘managed’.

`keyboard_on_key_down`(*window, keycode, text, modifiers*)

O método vinculado ao teclado quando a instância estiver focada.

Quando a instancia se torna focada, esse método é vinculado ao teclado a ser chamado por cada entrada pressionada. Os parâmetros são os mesmo ao [kivy.core.window.WindowBase.on_key_down\(\)](#).

Quando sobrescrevendo o método no widget derivado, super deve ser chamado para permitir ciclo de abas. Se o widget derivado deseja utilizar aba para seus próprios fins, pode chamar super após ter processado o personagem (se não deseja consumir a aba).

Semelhante a outras funções de teclado, deve retornar *True* se a tecla foi consumida.

keyboard_on_key_up(*window, keycode*)

O método vinculado ao teclado quando a instância estiver focada.

Quando a instância se torna focada, esse método é vinculado ao teclado e será chamado por cada entrada pressionada. Os parâmetros são os mesmos aos do [*kivy.core.window.WindowBase.on_key_down\(\)*](#).

Quando sobrescrevendo o método no widget derivado, super deve ser chamado para permitir ciclo de abas. Se o widget derivado deseja utilizar aba para seus próprios fins, pode chamar super após ter processado o personagem (se não deseja consumir o escape).

Veja [*keyboard_on_key_down\(\)*](#)

show_keyboard()

Função de conveniência para mostrar o teclado no modo gerenciado.

unfocus_on_touch

Se a instância deve perder o foco quando houver um clique em alguma região fora da sua área;

Quando um usuário clica num Widget focado e partilha o mesmo teclado que este Widget (no caso de haver apenas um teclado, serão todos os Widgets conscientes do foco), então, à medida que os outros Widgets ganham o foco, este Widget perde o foco. Além disso, se esta propriedade for *True*, clicando em qualquer Widget que não seja este Widget, removerá o foco deste Widget.

unfocus_on_touch é uma *BooleanProperty* e o padrão é *False* se o *keyboard_mode* em [*Config*](#) é ‘multi’ ou ‘systemandmulti’, caso contrário o padrão é *True*.

Kivy Namespaces

Novo na versão 1.9.1.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

A classe [*KNSpaceBehavior mixin*](#) provê funcionalidades de namespaces para objetos Kivy. Isso permite que objetos Kivy sejam nomeados e então acessados utilizando um namespace.

[*KNSpace*](#) instâncias são os Namespaces que armazenam os objetos nomeados em instâncias no Kivy [*ObjectProperty*](#). Além disso, ao herdar de [*KNSpaceBehavior*](#), se

o objeto derivado for nomeado, o nome será automaticamente adicionado ao namespace associado e apontará para o objeto derivado `proxy_ref`.

Simples exemplo

Por padrão, há somente um namespace: o namespace `knspace`. O exemplo mais simples é adicionar um Widget para o namespace:

```
from kivy.uix.behaviors.knspace import knspace
widget = Widget()
knspace.my_widget = widget
```

Isso adiciona uma classe Kivy `ObjectProperty` com `rebind=True` e `allownone=True` para o namespace `knspace` com uma propriedade de nome `my_widget`. E a propriedade agora também aponta para este Widget.

Isso pode ser feito automaticamente com:

```
class MyWidget(KNSpaceBehavior, Widget):
    pass

widget = MyWidget(knsname='my_widget')
```

Ou em kv:

```
<MyWidget@KNSpaceBehavior+Widget>

MyWidget:
    knsname: 'my_widget'
```

Agora, `knspace.my_widget` apontará para o Widget.

Quando se cria um segundo Widget com o mesmo nome, o namespace também será alterado para apontar para o novo Widget. Por exemplo.:

```
widget = MyWidget(knsname='my_widget')
# knspace.my_widget now points to widget
widget2 = MyWidget(knsname='my_widget')
# knspace.my_widget now points to widget2
```

Definindo o namespace

Também é possível criar o próprio namespace em vez de usar o padrão `knspace` definindo directamente `KNSpaceBehavior.knspace`:

```

class MyWidget(KNSpaceBehavior, Widget):
    pass

widget = MyWidget(knsname='my_widget')
my_new_namespace = KNSpace()
widget.knspace = my_new_namespace

```

Inicialmente, *my_widget* é adicionado ao namespace padrão, mas quando o namespace do Widget é alterado para 'my_new_namespace', a referência a *my_widget* é movida pra esse namespace. Poderíamos ter também, naturalmente, primeiro definido o namespace para *my_new_namespace* e, em seguida, ter chamado o Widget *my_widget*, evitando assim a atribuição inicial para o namespace padrão.

Da mesma forma temos em kv:

```

<MyWidget@KNSpaceBehavior+Widget>

MyWidget:
    knspace: KNSpace()
    knsname: 'my_widget'

```

Namespace herdado

No exemplo anterior, definimos diretamente o namespace que desejamos usar. No exemplo a seguir, herdaremos do pai, portanto, só precisaremos configurá-lo uma vez:

```

<MyWidget@KNSpaceBehavior+Widget>
<MyLabel@KNSpaceBehavior+Label>

<MyComplexWidget@MyWidget>:
    knsname: 'my_complex'
    MyLabel:
        knsname: 'label1'
    MyLabel:
        knsname: 'label2'

```

Então, nós fazemos:

```

widget = MyComplexWidget()
new_knspace = KNSpace()
widget.knspace = new_knspace

```

A regra é que no caso de nenhum knspace ter sido atribuído a um Widget, o mesmo procurará um namespace em seu pai e pai a pai e assim por diante até encontrar um para utilizar. Se nenhum for encontrado, será usado padrão *knspace*.

Quando *MyComplexWidget* é criado, ele ainda usa o padrão de namespace. No entanto,

quando nós atribuímos o Widget principal ele cria um novo namespace, e temos que todos os seus filhos também mudaram pra esse novo namespace. Assim, `new_knspace` agora contém `label1` e' `label2`, bem como `my_complex`.

Se tivéssemos feito primeiro:

```
widget = MyComplexWidget()
new_knspace = KNSpace()
knspace.label1.knspace = knspace
widget.knspace = new_knspace
```

Em seguida, `label1` permaneceria armazenado no padrão: attr:' knspace' uma vez que foi definido diretamente, mas `label2` e `my_complex` ainda seriam adicionados ao novo namespace.

Pode-se personalizar o atributo usado para pesquisar na árvore pai, alterando `KNSpaceBehavior.knspace_key`. Se o knspace desejado não estiver acessível através da árvore pai dos Widgets, por exemplo, em um Popup que não é filho de um Widget `KNSpaceBehavior.knspace_key` pode ser usado para estabelecer uma ordem de busca diferente.

Acessando o namespace

Como visto no exemplo anterior, se não for atribuído diretamente, o namespace é encontrado pesquisando a árvore pai. Consequentemente, se um namespace foi atribuído mais acima na árvore pai, todos os seus filhos e aqueles que estão abaixo poderiam acessar esse namespace por meio de sua propriedade `KNSpaceBehavior.knspace`.

Isso permite a criação de vários widgets com nomes idênticos se cada instância do widget raiz 'root' é atribuída a um novo namespace. Por exemplo:

```
<MyComplexWidget@KNSpaceBehavior+Widget>:
    Label:
        text: root.knspace.pretty.text if root.knspace.pretty else ''
    ''

<MyPrettyWidget@KNSpaceBehavior+TextInput>:
    knsname: 'pretty'
    text: 'Hello'

<MyCompositeWidget@KNSpaceBehavior+BoxLayout>:
    MyComplexWidget
    MyPrettyWidget
```

Agora, quando fazemos:

```

knspace1, knspace2 = KNSpace(), KNSpace()
composite1 = MyCompositeWidget()
composite1.knspace = knspace1

composite2 = MyCompositeWidget()
composite2.knspace = knspace2

knspace1.pretty = "Here's the ladder, now fix the roof!"
knspace2.pretty = "Get that raccoon off me!"

```

Como cada uma das instâncias *MyCompositeWidget* tem um namespace diferente, seus filhos também usam namespaces diferentes. Consequentemente, os Widgets bonitos e complexos de cada instância terão texto diferente.

Além disso, porque tanto a referência do namespace *ObjectProperty*, e *KNSpaceBehavior.knspace* possuem *rebind=True*, o texto do Label *MyComplexWidget* é rebound para corresponder ao texto do *MyPrettyWidget* quando o espaço de nomes da raiz é alterado ou quando a propriedade *root.knspace.pretty* for alterada, conforme o esperado.

Forking um namespace

o Forking de um namespace fornece a oportunidade de criar um novo namespace a partir de um namespace pai para que o namespace bifurcado (forking) conterá tudo no namespace de origem, mas o namespace de origem não terá acesso a nada adicionado ao namespace bifurcado.

Por exemplo:

```

child = knspace.fork()
grandchild = child.fork()

child.label = Label()
grandchild.button = Button()

```

Agora o Label é acessível por ambos os filhos e netos, mas não por *knspace*. E o Button só é acessível pelo neto, mas não pelos filhos ou pelo *knspace*. Finalmente, fazer *grandchild.label = Label()* deixará *grandchild.label* e *child.label* apontando para Labels diferentes.

Um exemplo motivador é o exemplo acima:

```

<MyComplexWidget@KNSpaceBehavior+Widget>:
    Label:
        text: root.knspace.pretty.text if root.knspace.pretty else '

```

```

<MyPrettyWidget@KNSpaceBehavior+TextInput>:
    knsname: 'pretty'
    text: 'Hello'

<MyCompositeWidget@KNSpaceBehavior+BoxLayout>:
    knspace: 'fork'
    MyComplexWidget
    MyPrettyWidget

```

Observe a adição de `knspace: 'fork'`. Isso é idêntico ao fazer `knspace:self.knspace.fork()`. No entanto, fazer isso levaria a recursão infinita, pois essa regra kv seria executada recursivamente porque `self.knspace` continuará mudando. No entanto, permitindo `knspace:'fork'` cirumvents que. Veja [KNSpaceBehavior.knspace](#).

Agora, tendo forked (bifurcado), só precisamos fazer:

```

composite1 = MyCompositeWidget()
composite2 = MyCompositeWidget()

composite1.knspace.pretty = "Here's the ladder, now fix the roof!"
composite2.knspace.pretty = "Get that raccoon off me!"

```

Como, por meio do forking, criamos automaticamente um namespace exclusivo para cada instância do `MyCompositeWidget`.

```

class kivy.uix.behaviors.knspace.KNSpace(parent=None,
                                         keep_ref=False, **kwargs)
Bases: kivy.event.EventDispatcher

```

Cada instância de class:`KNSpace` é um namespace que armazena os objetos Kivy nomeados associados a este namespace. Cada objeto nomeado é armazenado como o valor de uma classe Kivy [ObjectProperty](#) dessa instância cujo nome de propriedade é o nome dado do objeto. Ambos propriedades `rebind` e `allownone` são por padrão iguais a `True`.

Veja [KNSpaceBehavior.knspace](#) para detalhes sobre como um namespace está associado a um objeto nomeado.

Ao armazenar um objeto no namespace, o `proxy_ref` do objeto é armazenado se o objeto tiver tal atributo.

Parameters

`parent: (interno) Uma instância de KNSpace ou None.` Se especificado, é um namespace pai, nesse caso, o espaço de nomes atual terá em seu namespace todos os seus objetos nomeados, bem como os objetos nomeados de seu pai e pai pai etc. Veja [fork\(\)](#) para mais detalhes.

`fork()`

Retorna uma nova instância [KNSpace](#) que terá acesso a todos os objetos

nomeados no namespace atual, mas também terá um namespace próprio que é exclusivo para ele.

Por exemplo:

```
forked_knspace1 = knspace.fork()  
forked_knspace2 = knspace.fork()
```

Agora, qualquer nome adicionado ao *knspace* será acessível pelos namespaces *forked_knspace1* e *forked_knspace2* pelos meios normais. No entanto, quaisquer nomes adicionados a *forked_knspace1* não estarão acessíveis a partir de *knspace* ou *forked_knspace2*. Similar ao que temos com *forked_knspace2*.

keep_ref = False

Se uma referência direta deve ser mantida para os objetos armazenados. Se `True`, nós usamos o objeto direto, caso contrário nós usamos: attr: `~kivy.uix.widget.proxy_ref` quando presente.

Padrão para `False`.

parent = None

(Interno) The parent namespace instance, *KNSpace*, ou 'None'. Veja *fork()*.

```
class kivy.uix.behaviors.knspace.KNSpaceBehavior(knspace=None,  
                                                 **kwargs)
```

Bases: *object*

Herdar dessa classe permite nomear os objetos herdados, que são então adicionados ao namespace associado: attr: *knspace* e acessível através dele.

Consulte a documentação: mod: *knspace behaviors module <kivy.uix.behaviors.knspace>* para obter mais informações.

knsname

O nome dado a esta instância. Se nomeado, o nome será adicionado ao namespace associado: attr: *knspace*, que irá, em seguida, apontar para o *proxy_ref* desta instância.

Quando nomeado, um pode acessar este objeto, por exemplo `self.knspace.name`, onde *name* é o nome fornecido desta instância. Veja *knspace* e a descrição do módulo para mais detalhes.

knspace

O namespace da instância associada , *KNSpace*, com este Widget. O namespace *knspace* armazena esse Widget Ao nomear este Widget com *knsname*.

Se o namespace tiver sido definido com uma instância *KNSpace*, por exemplo, com `self.knspace = KNSpace()`, então essa instância é retornada (configurações iguais a *None* não são contabilizadas). Caso contrário, se

`knspace_key` não for `None`, procuramos um namespace para usar no objeto que está armazenado na propriedade chamada `knspace_key`, desta instância. I.e. `Object = getattr(self, self.knspace_key)`.

Se esse objeto tiver uma propriedade `knspace`, então retornamos seu valor. Caso contrário, vamos mais adiante, por exemplo, com `getattr(object, self.knspace_key)` e procure a propriedade `knspace`.

Finalmente, se atingimos um valor de `None`, ou `knspace_key` for igual a `None`, o namespace padrão `knspace` será retornado.

Se `knspace` estiver definido para a string '`fork`', o namespace atual em `knspace` será bifurcado com `KNSpace.fork()` e o namespace resultante será atribuído a esta instância `knspace`. Veja os exemplos do módulo para um exemplo de motivação.

Ambos `rebind` e `allownone` são `True`.

`knspace_key`

O nome da propriedade dessa instância, para usar para procurar para cima um namespace para usar por esta instância. O padrão é “parent” para que possamos pesquisar a árvore pai. Veja: attr: `knspace`.

Quando `None`, não procuraremos a árvore pai para o namespace.`'allownone'` é `True`.

`kivy.uix.behaviors.knspace.knspace = <kivy.uix.behaviors.knspace.KNSpace object>`

O namespaces padrão é `KNSpace`. Veja `KNSpaceBehavior.knspace` para maiores informações.

Comportamento ToggleButton

A classe `ToggleButtonBehavior` mixin provê o comportamento do `ToggleButton`. Você pode combinar esta classe com outros Widgets, tais como um `Image`, para disponibilizar uma alternativa de ToggleButtons, mas que preserve o comportamento padrão do ToggleButton da biblioteca Kivy.

Para uma visão geral sobre o comportamento, por favor, veja a documentação a seguir: `behaviors`.

Exemplo

Os exemplos a seguir adicionam comportamentos do ToggleButton a uma imagem pra fazer com que um CheckBox funcione como um ToggleButton:

```
from kivy.app import App
from kivy.uix.image import Image
from kivy.uix.behaviors import ToggleButtonBehavior
```

```

class MyButton(ToggleButtonBehavior, Image):
    def __init__(self, **kwargs):
        super(MyButton, self).__init__(**kwargs)
        self.source = 'atlas://data/images/defaulttheme/checkbox_off'
    def on_state(self, widget, value):
        if value == 'down':
            self.source = 'atlas://data/images/defaulttheme/
checkbox_on'
        else:
            self.source = 'atlas://data/images/defaulttheme/
checkbox_off'

class SampleApp(App):
    def build(self):
        return MyButton()

SampleApp().run()

```

class kivy.uix.behaviors.togglebutton.ToggleButtonBehavior(kwargs)**
Bases: *kivy.uix.behaviors.button.ButtonBehavior*

A classe **mixin** provê comportamento de **togglebutton**. Por favor, veja o módulo **togglebutton behaviors module** da documentação para maiores informações.

Novo na versão 1.8.0.

allow_no_selection

Especifica se os Widgets de um grupo não permitem seleção, isto é, tudo será desmarcado.

Novo na versão 1.9.0.

allow_no_selection é um BooleanProperty e o padrão é *True*

static get_widgets(groupname)

Retorna uma lista de Widgets contidas num determinado grupo. Caso o grupo não exista, uma lista vazia será retornada.

Nota: Sempre libere o resultado deste método! Manter a referência de um desses Widgets pode impedir que os mesmos seja destruídos pelo Garbage Collector. Em caso de dúvida, veja:

```
l = ToggleButtonBehavior.get_widgets('mygroup')
# do your job
del l
```

Aviso: É possível que algum Widgets que você tenha anteriormente deletado ainda esteja na lista. O garbage collector pode precisar liberar outros objetos antes de limpá-los.

group

Grupo de botões. Se *None*, nenhum grupo será utilizado (o botão será independente). Se especificado, :attr:`group` deve ser um objeto hashable, como uma String. Somente um botão no grupo pode ter o estado 'down'.

group é um *ObjectProperty* e o padrão é *None*.

4.15.2 RecycleView

Novo na versão 1.10.0.

A opção 'RecycleView' é um modelo flexível para visualizar grandes coleções de dados selecionados. Esta opção previne perda de performance quando se usa um grande número de widgets e, consequentemente, é necessário processar muitos dados.

A View é gerada processando o *data*, essencialmente uma lista de dicts, e usa esses dicionários para gerar instâncias de *viewclass* conforme necessário. Seu design é baseado no padrão MVC (*Model-view-controller* <<https://en.wikipedia.org/wiki/Model%20view%20controller>>).

- Model: O modelo é formado por *data* que você passa através de uma lista de dicts.
- View: A View é dividida entre layout e visualizações e implementada por...
- Controller: O controlador é implementado por *RecycleViewBehavior*.

Estas são classes abstratas e não podem ser usadas diretamente. A implementação concreta do padrão é *RecycleDataModel* para o modelo, o *RecycleLayout* e ... para View, e a *RecycleView* para o Controlador.

Quando um *RecycleView* é instanciado, ele cria automaticamente as exibições e classes de dados. No entanto, é necessário criar manualmente as classes de layout e adicioná-las ao *RecycleView*.

Um gerenciador de layout é automaticamente criado como *layout_manager* quando adicionado como filho do *RecycleView*. Similarmente quando removido. Um requisito é que o gerenciador de layout deve estar contido como um filho em algum lugar

dentro da árvore de Widgets do *RecyclerView* para que a porta de exibição possa ser encontrada.

Um exemplo mínimo pode ser algo como isto:

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.recycleview import RecycleView

Builder.load_string('''
<RV>:
    viewclass: 'Label'
    RecycleBoxLayout:
        default_size: None, dp(56)
        default_size_hint: 1, None
        size_hint_y: None
        height: self.minimum_height
        orientation: 'vertical'
''')

class RV(RecycleView):
    def __init__(self, **kwargs):
        super(RV, self).__init__(**kwargs)
        self.data = [{'text': str(x)} for x in range(100)]

class TestApp(App):
    def build(self):
        return RV()

if __name__ == '__main__':
    TestApp().run()
```

Para suportar a seleção na visualização, podes adicionar os comportamentos necessários da seguinte forma:

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.recycleview import RecycleView
from kivy.uix.recycleview.views import RecycleDataViewBehavior
from kivy.uix.label import Label
from kivy.properties import BooleanProperty
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.uix.behaviors import FocusBehavior
from kivy.uix.recycleview.layout import LayoutSelectionBehavior

Builder.load_string('''
<SelectableLabel>:
```

```

# Draw a background to indicate selection
canvas.before:
    Color:
        rgba: (.0, .9, .1, .3) if self.selected else (0, 0, 0,_
        ↵1)
    Rectangle:
        pos: self.pos
        size: self.size

<RV>:
    viewclass: 'SelectableLabel'
    SelectableRecycleBoxLayout:
        default_size: None, dp(56)
        default_size_hint: 1, None
        size_hint_y: None
        height: self.minimum_height
        orientation: 'vertical'
        multiselect: True
        touch_multiselect: True
    ''')

class SelectableRecycleBoxLayout(FocusBehavior,_
    ↵LayoutSelectionBehavior,
    RecycleBoxLayout):
    ''' Adds selection and focus behaviour to the view. '''

class SelectableLabel(RecycleDataViewBehavior, Label):
    ''' Add selection support to the Label '''
    index = None
    selected = BooleanProperty(False)
    selectable = BooleanProperty(True)

    def refresh_view_attrs(self, rv, index, data):
        ''' Catch and handle the view changes '''
        self.index = index
        return super(SelectableLabel, self).refresh_view_attrs(
            rv, index, data)

    def on_touch_down(self, touch):
        ''' Add selection on touch down '''
        if super(SelectableLabel, self).on_touch_down(touch):
            return True
        if self.collide_point(*touch.pos) and self.selectable:
            return self.parent.select_with_touch(self.index, touch)

    def apply_selection(self, rv, index, is_selected):
        ''' Respond to the selection of items in the view. '''

```

```

        self.selected = is_selected
        if is_selected:
            print("selection changed to {}".format(rv.data[index]))
        else:
            print("selection removed for {}".format(rv.
data[index]))
    
```



```

class RV(RecyclerView):
    def __init__(self, **kwargs):
        super(RV, self).__init__(**kwargs)
        self.data = [{text: str(x)} for x in range(100)]
    
```



```

class TestApp(App):
    def build(self):
        return RV()

if __name__ == '__main__':
    TestApp().run()
    
```

Consulte o arquivo `examples/widgets/recycleview/basic_data.py` para obter um exemplo mais completo.

TODO:

- Método para limpar instâncias de classe em cache.
- Teste quando as Views não podem ser encontradas (por exemplo, quando ViewClass é `None`).
- Corrigir seleção goto.

Aviso: Quando `Views` são reutilizadas, elas não podem conseguem disparar se os dados permanecem os mesmos.

`class kivy.uix.recycleview.RecycleViewBehavior(**kwargs)`
Bases: `object`

`RecycleViewBehavior` fornece um modelo comportamental em que o `RecycleView` é construído. Juntos, eles oferecem uma maneira extensível e flexível para produzir exibições com janelas limitadas em grandes conjuntos de dados.

Veja o módulo da documentação para maiores informações.

`data_model`

O Modelo de Dados responsável pela manutenção do conjunto de dados.

data_model é uma **AliasProperty** que obtém e define o modelo de dados atual.

layout_manager

O Gerenciador de Layout responsável por posicionar as vistas dentro da classe *:RecycleView*.

layout_manager é uma **AliasProperty** que obtém e define o *layout_manger*.

refresh_from_data(*largs, **kwargs)

This should be called when data changes. Data changes typically indicate that everything should be recomputed since the source data changed.

Este método é automaticamente vinculado ao método *on_data_changed* da classe *RecycleDataModelBehavior* e, portanto, responde e aceita os argumentos de palavra-chave desse evento.

Ele pode ser chamado manualmente para acionar uma atualização.

refresh_from_layout(*largs, **kwargs)

Isso deve ser invocado quando o layout for alterado ou precisar ser alterado. Geralmente é chamado quando um parâmetro de layout foi alterado e, portanto, o layout precisa ser recalculado.

refresh_from_viewport(*largs)

Isso deve ser chamado quando o ViewPort for alterada e os dados exibidos precisarem ser atualizados. Nem os dados nem o layout serão recalculados.

view_adapter

O adaptador responsável por fornecer Views que representam itens em um conjunto de dados.

view_adapter é uma **AliasProperty** que obtém e define o adaptador de exibição atual.

class kivy.uix.recycleview.RecycleView(kwargs)**

Bases: ***kivy.uix.recycleview.RecycleViewBehavior, kivy.uix.scrollview.ScrollView***

RecycleView é uma visão flexível para fornecer uma janela limitada em um grande conjunto de dados.

Veja o módulo da documentação para maiores informações.

data

Os dados usados pelo adaptador de exibição atual. Esta é uma lista de dicts cujas Keys mapeiam para os nomes de propriedade correspondentes ao *viewclass*.

Data é uma **AliasProperty** que obtém e define os dados usados para gerar as visualizações.

key_viewclass

key_viewclass é uma **AliasProperty** que obtém e define a chave *viewclass*

para o atual `layout_manager`.

viewclass

O `viewclass` é usado pelo atual `layout_manager`.

`viewclass` é uma **AliasProperty** que obtém e define a classe usada para gerar os itens individuais apresentados na View.

RecycleView Data Model

Novo na versão 1.10.0.

Parte do Modelo de Dados do padrão MVC (model-view-controller) RecycleView.

define os modelos (classes) que armazenam os dados associados a `RecycleViewBehavior`. Cada modelo (classe) determina como os dados são armazenados e emite solicitações ao controlador (`:class:~kivy.uix.recycleview.RecycleViewBehavior`) quando os dados são modificados.

```
class kivy.uix.recycleview.datamodel.RecycleDataModelBehavior  
Bases: object
```

`RecycleDataModelBehavior` é a classe base para os modelos que descreve e fornece os dados para `RecycleViewBehavior`.

Events

`on_data_changed`: Disparado quando os dados mudam. O evento pode enviar argumentos de palavra-chave específicos para cada implementação do modelo de dados. Quando enviados, o evento e os keyword argumentos são encaminhados para `refresh_from_data()`.

attach_recycleview(rv)

Associa uma `RecycleViewBehavior` como este “Data Model”.

detach_recycleview()

Remove a `RecycleViewBehavior` associada com este “Data Model”.

recycleview

A instância de `RecycleViewBehavior` associada com este “Data Model”.

```
class kivy.uix.recycleview.datamodel.RecycleDataModel(**kwargs)  
Bases: kivy.uix.recycleview.datamodel.  
RecycleDataModelBehavior, kivy.event.EventDispatcher
```

Uma implementação de `RecycleDataModelBehavior` que mantém os dados em uma lista indexável. Veja `data`.

Quando os dados mudam, esta classe atualmente envia `on_data_changed` com um dos seguintes keyword argumentos adicionais.

none: nenhum argumento keywordSem argumento adicional, isso significa uma mudança genérica de dados.

removed: uma fatia ou inteiroO valor é uma fatia ou número inteiro que indica os índices removidos.

appended: uma fatiaA fatia em *data* indicando o primeiro e último item novos (ou seja, a fatia apontando para os novos itens adicionados no final).

inserted: um inteiroO índice em *data* onde um novo item de dados foi inserido.

modified: uma fatiaA fatia com os índices onde os dados foram modificados. Isso atualmente não permite a mudança de tamanho etc.

data

Armazena os dados do modelo usando uma lista.

Os dados de um item no índice *i* também podem ser acessados com *RecycleDataModel* [*i*].

observable_dict

Uma instância de dicionário que, quando modificada, acionará um *data* e, consequentemente, um despacho em *on_data_changed*.

RecycleView Layouts

Novo na versão 1.10.0.

A opção “Layouts” manipula a apresentação das visualizações da classe *RecycleView*.

Aviso: Este módulo é altamente experimental, sua API pode mudar no futuro e a documentação ainda não está completa.

```
class kivy.uix.recycleview.layout.LayoutSelectionBehavior(**kwargs)
    Bases: kivy.uix.behaviors.compoundselection.CompoundSelectionBehavior
```

A *LayoutSelectionBehavior* pode ser combinada com *RecycleLayoutManagerBehavior* para permitir que seus comportamentos de seleção de classes derivadas sejam *CompoundSelectionBehavior* pode ser usado para adicionar comportamentos de seleção ao layout normal.

RecycleLayoutManagerBehavior cerencia seus filhos de forma diferente dos layouts ou Widgets normais para que esta classe se adapte *CompoundSelectionBehavior* para também trabalhar com *RecycleLayoutManagerBehavior*.

Da mesma forma que *CompoundSelectionBehavior*, pode-se selecionar usando o teclado ou toque, que invocará *select_node()* ou

`deselect_node()`, ou pode chamar esses métodos diretamente . Quando um item for selecionado ou desmarcado:meth `:apply_selection` será chamado. Veja [apply_selection\(\)](#).

apply_selection(index, view, is_selected)

Aplica a seleção à View. Isso é chamado internamente quando uma View é exibida e precisa ser mostrada como selecionada ou não selecionada.

É chamado quando `select_node()` ou `deselect_node()` é chamado ou quando uma View precisar ser atualizada. Sua função é puramente para atualizar a View e refletir o estado de seleção. Assim, a função pode ser chamada várias vezes mesmo se o estado da seleção não tiver sido alterado.

Se a View for uma instância de [RecycleDataViewBehavior](#), seu método `apply_selection()` será invocado todas as vezes que a View precisar atualizar o estado de seleção. Caso contrário, este método será responsável pela aplicação da seleção.

Parameters

`index: int`O índice do item de dados associado à exibição.

`view: widget`O Widget que é a View desse item de dado.

`is_selected: bool`Se o item está selecionado.

key_selection

A chave usada para verificar se uma View de um item de dados pode ser selecionada com o toque ou com o teclado.

`key_selection` é a chave em dados, que se presente e `True` habilitarão a seleção para este item do teclado ou através de um toque. Quando `None`, o item padrão, não será selecionável.

`key_selection` é uma `StringProperty` e o padrão é `None`.

Nota: Todos os itens de dados podem ser usados `select_node()` ou `deselect_node()`, even if `key_selection` é `False`.

class kivy.uix.recycleview.layout.RecycleLayoutManagerBehavior
Bases: `object`

Um RecycleLayoutManagerBehavior é responsável por posicionar as visualizações no `RecycleView.data` dentro de uma `RecycleView`. Isso adiciona novas Views aos dados quando ficar visível para o usuário e os remove quando deixarem a área de visualização.

compute_visible_views(data, viewport)

`Viewport` está definido nas coordenadas do gerenciador de layout.

get_view_index_at(pos)

Retornar a View índice em que a posição,'pos', cai.

Pos está em coordenadas do gerenciador de layout.

goto_view(index)

Move as Views para que a View correspondente a *index* seja visível.

key_viewclass

Veja `RecyclerView.key_viewclass`.

refresh_view_layout(index, layout, view, viewport)

Veja :meth:`~kivy.uix.recycleview.views.RecycleDataAdapter.refresh_view_layout`.

set_visible_views(indices, data, viewport)

Viewport está definido nas coordenadas do gerenciador de layout.

viewclass

Veja `RecyclerView.viewclass`.

RecycleView Views

Novo na versão 1.10.0.

O adapter do RecycleView que junto com o layout é a camada view do padrão model-view-controller.

O módulo de visualização manipula a conversão dos dados para a exibição usando a classe Adapter que é exibida pelo layout. Uma View é qualquer classe derivada de Widget. No entanto, herdando de `RecycleDataViewBehavior` adicionará métodos para converter os dados para uma View.

TODO:

- Crie caches de Views específicos para cada tipo de classe de exibição.

class kivy.uix.recycleview.views.RecycleDataAdapter

Bases: `kivy.event.EventDispatcher`

A classe que converte dados para uma View.

— Detalhes internos — Uma View pode ter 3 estados.

- Ele pode estar completamente sincronizado com os dados, o que ocorre quando a View é exibida. Os mesmos são armazenados em `views`.
- Ele pode estar sujo, o que ocorre quando a View está sincronizada com os dados, exceto para os parâmetros `size/pos`, que são controlados pelo layout. Isso ocorre quando a View não é exibida no momento, mas os dados não foram alterados. Essas Views são armazenadas em `dirty_views`.
- Finalmente, a View pode estar inoperante, o que ocorre quando os dados mudam e a View não é atualizada ou quando uma View é criada. Tais Views são tipicamente adicionadas ao cache interno.

Normalmente, o que acontece é que o gerenciador de layout coloca os dados e, em seguida, solicita que as Views, usando:method:`set_visible_views`, para alguns itens de dados específicos que ele exibe.

Essas Views são obtidas a partir das Views atuais, cache sujo ou global. Em seguida, dependendo do estado da View `refresh_view_attrs()` será chamada para trazer a View atualizada com os dados (exceto para parâmetros de dimensionamento). Finalmente, o gerenciador de layout obtém essas Views, atualiza seu tamanho e exibe-as.

attach_recycleview(*rv*)

Associa uma `RecycleViewBehavior` com essa instância. Ele é armazenado em `recycleview`.

create_view(*index*, *data_item*, *viewclass*)

(Internal) Cria e inicializa a View para os dados em *index*.

A View retornada é sincronizada com os dados, exceto para as informações de *pos/size*.

detach_recycleview()

Remove a `RecycleViewBehavior` associada com essa instância e limpa o `recycleview`.

get_view(*index*, *data_item*, *viewclass*)

(Internal) Retorna uma instância de View para os dados em *index*

Isso vê através dos vários caches e finalmente cria uma View caso não exista. A View retornada é sincronizada com os dados, exceto para as informações *pos/size*.

Se encontrado no cache ele é removido da fonte antes de retornar. O mesmo não verifica as Views atuais.

get_visible_view(*index*)

Retorna a View atualmente visível associada ao *index*.

Se nenhuma View for atualmente exibida para *index*, será retornado `None`.

invalidate()

Move todas as atuais Views para o cache global.

Ao contrário de tornar uma View suja onde a View está em sincronia com os dados, exceto para informações de dimensionamento, isso irá desconectar completamente a View dos dados, pois é assumido que os dados não estão sincronizados com a View.

Normalmente, isso é chamado quando os dados são alterados.

make_view_dirty(*view*, *index*)

(Internal) Usado para sinalizar esta View como “suja”, pronta para ser usada reutilizada por outro. Veja: `make_views_dirty()`.

`make_views_dirty()`

Torna todas as Views atuais sujas.

As Views sujas ainda são sincronizadas com os dados correspondentes. No entanto, as informações de tamanho podem ficar fora de sincronia. Portanto, uma View suja pode ser reutilizada pelo mesmo índice apenas atualizando as informações de dimensionamento.

Uma vez que os dados subjacentes a este índice são alterados, a View deve ser removida das Views sujas e movida para o cache global com `invalidate()`.

Isso normalmente é chamado quando o gerenciador de layout precisa recalcular todos os dados.

`recycleview`

A `RecyclerViewBehavior` associado a esta instância.

`refresh_view_attrs(index, data_item, view)`

(Interno) Sincroniza a View e atualiza os com os novos dados.

Este método invoca `RecycleDataViewBehavior.refresh_view_attrs()` caso a View herde de `RecycleDataViewBehavior`. Consulte esse método para obter mais detalhes.

Nota: Qualquer informação de dimensionamento e posição é ignorada ao sincronizar com os dados.

`refresh_view_layout(index, layout, view, viewport)`

Atualiza as informações de dimensionamento da visualização.

Viewport está em coordenadas do gerenciador de layout.

Este método invoca `RecycleDataViewBehavior.refresh_view_attrs()` caso a View herde de `RecycleDataViewBehavior`. Consulte esse método para obter mais detalhes.

Nota: Qualquer informação de dimensionamento e posição é ignorada ao sincronizar com os dados.

`set_visible_views(indices, data, viewclasses)`

Obtém uma 3-tupla das exibições novas, restantes e antigas para o Viewport atual.

As novas Views são sincronizadas com os dados, excepto para as propriedades size/pos. As Views antigas precisam ser removidas do layout e as novas Views adicionadas.

As novas Views não são necessariamente *novas*, mas são todas as Views atualmente visíveis.

```
class kivy.uix.recycleview.views.RecycleDataViewBehavior  
Bases: object
```

Uma classe base opcional para Views de dados (`RecycleView.viewclass`). Se uma View herda dessa classe, as funções da classe serão chamadas quando a View precisa ser atualizada devido a uma alteração de dados ou atualização de layout.

refresh_view_attrs(*rv, index, data*)

Invocado por `RecycleAdapter` quando a View está sendo inicialmente preenchida com os valores do dicionário *data* para este item.

Qualquer informação de *pos* ou *size* deve ser removida porque elas são definidas posteriormente com `refresh_view_layout`.

Parameters

rv: **RecycleView** instânciaA `RecycleView` que causou a atualização.

data: dictO dict de dados usado para preencher esta View.

refresh_view_layout(*rv, index, layout, viewport*)

Invocado quando o tamanho da View é atualizado pelo gerenciador de layout, `RecycleLayoutManagerBehavior`.

Parameters

rv: **RecycleView** instânciaA `RecycleView` que causou a atualização.

viewport: 4-tupleAs coordenadas do canto inferior esquerdo e a altura da largura nas coordenadas do gerenciador de layout. Isso pode ser maior do que o item de View.

Raises`LayoutChangeException`: Se o dimensionamento ou dados alterados durante uma chamada para este método, o levantamento de uma exceção `LayoutChangeException` forçará uma atualização. Útil quando os dados foram alterados e não queremos mais o layout porque ele será sobreescrito novamente em breve.

4.15.3 View abstrata

Novo na versão 1.5.

Nota: The feature has been deprecated.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

O widget **AbstractView** tem uma propriedade de adaptadora para um adaptar mediadores de dados. O adaptador gerencia uma propriedade item_view_instance dict que mantém exibições para cada item de dados, operando como um cache.

```
class kivy.uix.abstractview.AbstractView(*args, **kwargs)
    Bases: kivy.uix.floatlayout.FloatLayout
```

View usando uma **Adapter** como um provedor de dados.

adapter

O adaptador pode ser um de diversos tipo de **adapters**. O exemplo mais comum é o **ListAdapter** usado para gerenciar itens de dados numa lista.

4.15.4 Acordeão

Novo na versão 1.0.8.



O Widget Accordion é uma forma de menu onde as opções são empilhadas verticalmente ou horizontalmente e o item em foco (quando pressionado) se abre para exibir seu conteúdo.

A classe **Accordion** deve conter um ou várias instâncias de **AccordionItem**, cada um dos quais deve conter um Widget com conteúdo principal. Você acabará com uma árvore mais ou menos assim:

- Acordeão
 - AccordionItem
 - * YourContent
 - AccordionItem
 - * BoxLayout
 - Outro conteúdo do usuário 1
 - Outro conteúdo do usuário 2

- AccordionItem
 - * Outro conteúdo do usuário

A implementação atual divide o **AccordionItem** em duas partes:

1. Um container para a barra de título
2. Um container para o conteúdo

A barra de título é feita de um template kv. Veremos como criar um novo template para customizar o desenho de uma barra de título.

Aviso: Se veres uma mensagem como essa:

```
[WARNING] [Accordion] not have enough space for displaying all_
˓→children
[WARNING] [Accordion] need 440px, got 100px
[WARNING] [Accordion] layout aborted.
```

Isso significa que também tens muitos filhos e não há mais espaço para exibir o conteúdo. Isso é “normal” e nada será feito. Tente incrementar o espaço do Accordion ou reduzir o número de filhos. Você também pode reduzir o **Accordion**. *min_space*.

Simples exemplo

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
    def build(self):
        root = Accordion()
        for x in range(5):
            item = AccordionItem(title='Title %d' % x)
            item.add_widget(Label(text='Very big content\n' * 10))
            root.add_widget(item)
        return root

if __name__ == '__main__':
    AccordionApp().run()
```

Personalizando o Accordion

Você pode incrementar o tamanho padrão de uma barra de título:

```
root = Accordion(min_space=60)
```

Ou alterar a orientação para vertical:

```
root = Accordion(orientation='vertical')
```

O AccordionItem é mais configurável e você pode definir seu próprio título de background quando os itens estiverem sendo recolhidos ou abrindo.

```
item = AccordionItem(background_normal='image_when_collapsed.png',  
                     background_selected='image_when_selected.png')
```

class kivy.uix.accordion.Accordion(kwargs)**

Bases: *kivy.uix.widget.Widget*

Classe Accordion. Veja o módulo da documentação para maiores informações.

anim_duration

Duração da animação em segundos quando um novo item do Accordion for selecionado.

anim_duration é um *NumericProperty* e o padrão é .25 (250ms).

anim_func

Função facilitadora para utilizar a animação. Veja *kivy.animation.AnimationTransition* para maiores informações disponíveis sobre funções de animação.

anim_func é um *ObjectProperty* e o padrão é ‘out_expo’. Você pode definir uma String ou uma função para usar como uma função facilitadora.

min_space

Espaço mínimo a ser utilizado para o título de cada item. Esse valor é definido automaticamente para cada filho toda vez que o evento de layout ocorre.

min_space é um *NumericProperty* e o padrão é 44 (px).

orientation

Orientação do layout.

orientation é um *OptionProperty* e o padrão é ‘horizontal’. Pode assumir os valores de ‘vertical’ ou ‘horizontal’.

class kivy.uix.accordion.AccordionItem(kwargs)**

Bases: *kivy.uix.floatlayout.FloatLayout*

A classe AccordionItem que deve ser usada em conjunto com a classe *Accordion*. Veja o módulo da documentação para maiores informações.

accordion

Instância do *Accordion* ao qual pertence um item.

accordion é um *ObjectProperty* e o padrão é *None*.

background_disabled_normal

Imagen de fundo de um item do Accordion usado como representação gráfica padrão quando o item está recolhido ou desativado.

Novo na versão 1.8.0.

background_disabled_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/button_disabled’.

background_disabled_selected

Imagen de fundo de um item do Accordion usado pela representação gráfica padrão quando o item está selecionado (não recolhido) e desativado.

Novo na versão 1.8.0.

background_disabled_selected é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/button_disabled_pressed’.

background_normal

Imagen de plano de fundo de um item do Accordion usado pela representação gráfica padrão quando um item estiver recolhido.

background_normal is a *StringProperty* and defaults to ‘atlas://data/images/defaulttheme/button’.

background_selected

Imagen de plano de fundo de um item do Accordion usado pela representação gráfica padrão quando um item estiver selecionado (não recolhido).

background_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/button_pressed’.

collapse

Boolean para indicar se o item atual está ou não recolhido.

collapse é um *BooleanProperty* e o padrão é *True*.

collapse_alpha

Valor entre 0 e 1 para indicar quantos itens estão recolhidos (1) ou se estão selecionados (0). Isso é geralmente utilizado na animação.

collapse_alpha é um *NumericProperty* e o padrão é 1.

container

(Interna) Propriedade que será definida para o contêiner de crianças dentro da representação do AccordionItem.

container_title

(Interna) Propriedade que será definida para o contêiner de título dentro da representação dp AccordionItem.

content_size

(Interno) Definido pelo *Accordion* para o tamanho alocado ao conteúdo.

min_space

Link para a propriedade [Accordion.min_space](#).

orientation

Link para a propriedade [Accordion.orientation](#).

title

String do Título do item. O título pode ser usado em conjunto com o template `AccordionItemTitle`. Se você está usando um template padrão, podes usar a propriedade como uma entrada de texto, ou não. Por padrão, ele usará o texto do título. Veja `title_template` e `the example` abaixo.

`title` é um [StringProperty](#) e o padrão é ''.

title_args

Argumento padrão que será passado ao método `kivy.lang.Builder.template()`.

`title_args` é um [DictProperty](#) e o padrão é {}.

title_template

Template usado para criar parte do título do item do Accordion. O template padrão é um simples Label, sem customizações (exceto o texto) que suporte orientações como vertical e horizontal e diferentes planos de fundo para os modo recolhidos e selecionados.

É melhor criar e usar seu próprio modelo se o template padrão não for suficiente.

`title` é um [StringProperty](#) e o padrão é 'AccordionItemTitle'. O template padrão atual está no arquivo `kivy/data/style.kv`.

Aqui está o código se você desejar construir o seu próprio template:

```
[AccordionItemTitle@Label]:
    text: ctx.title
    canvas.before:
        Color:
            rgb: 1, 1, 1
        BorderImage:
            source:
                ctx.item.background_normal \
                if ctx.item.collapse \
                else ctx.item.background_selected
            pos: self.pos
            size: self.size
        PushMatrix
        Translate:
            xy: self.center_x, self.center_y
        Rotate:
            angle: 90 if ctx.item.orientation == 'horizontal
            ↵' else 0
```

```
    axis: 0, 0, 1
    Translate:
        xy: -self.center_x, -self.center_y
    canvas.after:
        PopMatrix
```

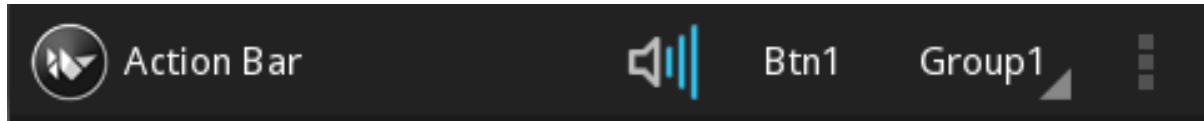
class kivy.uix.accordion.AccordionException

Bases: exceptions.Exception

Classe AccordionException.

4.15.5 Action Bar

Novo na versão 1.8.0.



O widget ActionBar é como o ActionBar do Android : <<http://developer.android.com/guide/topics/ui/actionbar.html>>, onde os itens são empilhados horizontalmente.

Um *ActionBar* contém um *ActionView* com várias *ContextualActionViews*. Um *ActionView* conterá um *ActionPrevious* tendo título, app_icon e a propriedade previous_icon. Um *ActionView* conterá subclasses de *ActionItems*. Algumas predefinições incluem um *ActionButton*, um *ActionToggleButton*, um *ActionCheck*, um *ActionSeparator* e um *ActionGroup*.

Um *ActionGroup* é usado para exibir um *ActionItems* em um grupo. Um *ActionView* sempre exibirá um *ActionGroup* antes de outro *ActionItems*. Um *ActionView* conterá um *ActionOverflow*. A um *ContextualActionView* é uma subclasse de um *ActionView*.

class kivy.uix.actionbar.ActionBarException

Bases: exceptions.Exception

Classe ActionBarException

class kivy.uix.actionbar.ActionItem

Bases: object

Classe ActionItem, uma classe abstrata para todos os Widgets ActionBar. Para criar um Widget padrão para um ActionBar, herde desta classe. Veja o módulo da documentação para maiores informações.

background_down

Imagen de plano de fundo do ActionItem utilizado como representação gráfica padrão de um ActionItem que está pressionado.

background_down é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/action_item_down’.

background_normal

Imagen de fundo do ActionItem usado pela representação gráfica padrão quando o ActionItem não está pressionado.

background_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/action_item’.

important

Determina se um ActionItem é importante ou não.

important é um *BooleanProperty* e o padrão é *False*.

inside_group

(interno) Determina se um ActionItem é mostrado dentro de um ActionGroup ou não.

inside_group é um *BooleanProperty* e o padrão é *False*.

minimum_width

Largura mínima requerida por um ActionItem.

minimum_width é um *NumericProperty* e o padrão é ‘90sp’.

mipmap

Define se a imagem/ícone exibido na parte superior do botão usa um mipmap ou não.

mipmap é um *BooleanProperty* e o padrão é *True*.

pack_width

(Somente leitura) A largura real a ser usada ao empacotar o item. Igual ao maior de minimum_width e width.

pack_width é um *AliasProperty*.

class kivy.uix.actionbarActionButton(kwargs)**

Bases: *kivy.uix.button.Button, kivy.uix.actionbar.ActionItem*

Classe ActionButton, veja o módulo da documentação para maiores informações.

A cor do texto, largura e size_hint_x são definidos manualmente através da linguagem kv. Isso cobre um monte de casos: com/sem um ícone, com/sem um grupo e cuida do preenchimento entre os elementos.

Você não possui muito controle sobre essa propriedade, então, se você deseja personalizar sua aparência, sugerimos que você crie sua própria representação. Podes fazer isso criando um classe que é uma subclasse de um Widget existente e um *ActionItem*:

```
class MyOwnActionButton(Button, ActionItem):
    pass
```

Você pode então criar seu próprio estilo usando a linguagem kv.

icon

Fonte de imagem usada quando o Button é parte de um ActionBar. Se o Button está em um grupo, o texto será preferido.

```
class kivy.uix.actionbar.ActionToggleButton(**kwargs)
```

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.togglebutton.ToggleButton*

Classe ActionToggleButton, veja o módulo da documentação para maiores informações.

icon

Fonte de imagem usada quando o Button é parte de um ActionBar. Se o Button está em um grupo, o texto será preferido.

```
class kivy.uix.actionbar.ActionCheck(**kwargs)
```

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.checkbox.CheckBox*

Classe ActionCheck, veja o módulo da documentação para maiores informações.

```
class kivy.uix.actionbar.ActionSeparator(**kwargs)
```

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.widget.Widget*

Classe ActionSeparator, veja o módulo da documentação para maiores informações.

background_image

Imagem de fundo para a representação gráfica padrão dos separadores.

background_image é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/sePARATOR’.

```
class kivy.uix.actionbar.ActionDropDown(**kwargs)
```

Bases: *kivy.uix.dropdown.DropDown*

Classe ActionDropDown, veja o módulo da documentação para maiores informações.

```
class kivy.uix.actionbar.ActionGroup(**kwargs)
```

Bases: *kivy.uix.actionbar.ActionItem*, *kivy.uix.spinner.Spinner*

Classe ActionGroup, veja o módulo da documentação para maiores informações.

dropdown_width

If non zero, provides the width for the associated DropDown. This is useful

when some items in the ActionGroup's DropDown are wider than usual and you don't want to make the ActionGroup widget itself wider.

dropdown_width is an *NumericProperty* and defaults to 0.

Novo na versão 1.10.0.

mode

Define o modo atual de um ActionGroup. Se o modo é 'normal', os filhos de ActionGroups serão mostrados normalmente se houver espaço, do contrário, eles serão exibidos em um Spinner. Se o modo for 'spinner', então, os filhos serão sempre exibidos como um Spinner.

mode é um *OptionProperty* e o padrão é 'normal'.

separator_image

Imagen de fundo para um ActionSeparator em um ActionView.

separator_image é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/separator'.

separator_width

Largura de um ActionSeparator em um ActionView.

separator_width é um *NumericProperty* e o padrão é 0.

use_separator

Determina se é para utilizar um separador antes/depois deste grupo ou não.

use_separator é um *BooleanProperty* e o padrão é *False*.

class kivy.uix.actionbar.ActionOverflow(kwargs)**

Bases: *kivy.uix.actionbar.ActionGroup*

Classe ActionOverflow, veja a documentação para maiores informações.

overflow_image

Imagen para ser utilizada como uma imagem Overflow;

overflow_image é um *ObjectProperty* e o padrão é 'atlas://data/images/defaulttheme/overflow'.

class kivy.uix.actionbar.ActionView(kwargs)**

Bases: *kivy.uix.boxlayout.BoxLayout*

Classe ActionView, veja o módulo da documentação para maiores informações.

action_previous

Botão anterior para um ActionView.

action_previous é um *ObjectProperty* e o padrão é *None*.

background_color

Imagen de fundo no formato (r, g, b, a).

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

background_image

Imagen de plano de fundo de uma representação gráfica padrão do ActionViews.

background_image é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/action_view'.

overflow_group

Widget a ser utilizado por um Overflow.

overflow_group is an *ObjectProperty* and defaults to an instance of *ActionOverflow*.

use_separator

Especifica se usar um separador antes dde cada ActionGroup ou não.

use_separator é um *BooleanProperty* e o padrão é *False*.

class kivy.uix.actionbar.ContextualActionView(kwargs)**

Bases: *kivy.uix.actionbar.ActionView*

Classe ContextualActionView, veja a documentação para maiores informações.

class kivy.uix.actionbar.ActionPrevious(kwargs)**

Bases: *kivy.uix.boxlayoutBoxLayout*, *kivy.uix.actionbar.ActionItem*

Classe ActionPrevious, veja a documentação para maiores informações.

app_icon

Ícone da aplicação para um ActionView.

app_icon é um *StringProperty* e o padrão para do ícone da janela se for definido, senão 'data/logo/kivy-icon-32.png'.

app_icon_height

Altura de uma imagem *app_icon*.

app_icon_width

Largura da imagem *app_icon*.

color

Cor do texto no formato (r, g, b, a)

color é um *ListProperty* e o padrão é [1, 1, 1, 1].

previous_image

Imagen para a representação gráfica padrão do ActionButtons anterior.

previous_image é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/previous_normal'.

previous_image_height

Altura da imagem de um *previous_image*.

previous_image_width

Largura da imagem *previous_image*.

title

Título de um ActionView.

title é um *StringProperty* e o padrão é ''.

with_previous

Determina se clicando em um ActionPrevious abrirá a tela anterior ou não. Se *True*, o 'previous_icon' será mostrado, do contrário não será.

with_previous é um *BooleanProperty* e o padrão é *True*.

class kivy.uix.actionbar.ActionBar(kwargs)**

Bases: *kivy.uix.boxlayout.BoxLayout*

Classe ActionBar, veja o módulo da documentação para maiores informações.

Events

on_previous Disparado quando um *action_previous* de um *action_view* é pressionado.

action_view

action_view de ActionBar.

action_view é um *ObjectProperty* e o padrão é uma instância de ActionView.

background_color

Cor de fundo, no forma (r, g, b, a).

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

background_image

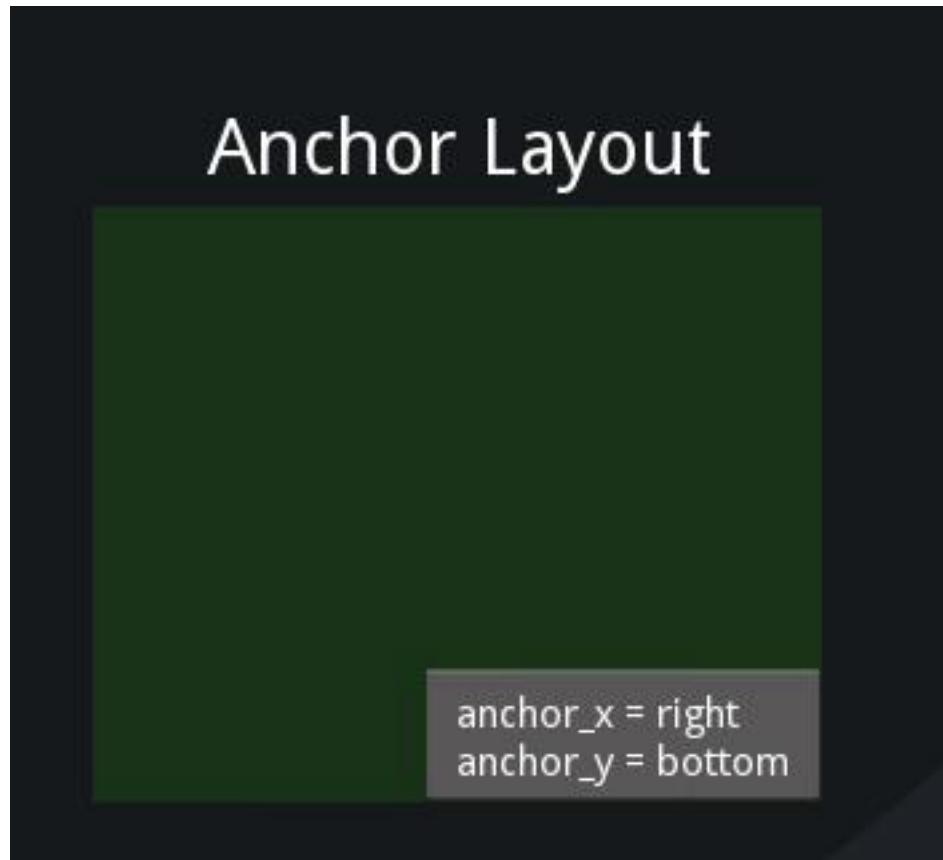
Imagen de plano de fundo da representação gráfica padrão do ActionBars.

background_image é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/action_bar'.

border

border será aplicado para um *background_image*.

4.15.6 Anchor Layout



A Classe 'AnchorLayout' alinha seus filhos a uma borda(superior,inferior,esqueda,direita) ou centro

Para desenhar um botão no canto inferior direito:

```
layout = AnchorLayout(  
    anchor_x='right', anchor_y='bottom')  
btn = Button(text='Hello World')  
layout.add_widget(btn)
```

class kivy.uix.anchorlayout.AnchorLayout(kwargs)**
Bases: *kivy.uix.layout.Layout*

Classe AnchorLayout. Veja o módulo da documentação para maiores informações.

anchor_x

Âncora Horizontal.

anchor_x é um *OptionProperty* e o padrão é 'center'. Ele aceita valores de 'left', 'center' ou 'right'.

anchor_y

Âncora Vertical

anchor_y é um *OptionProperty* e o padrão é ‘center’. Ele aceita valores como ‘top’, ‘center’ ou ‘bottom’.

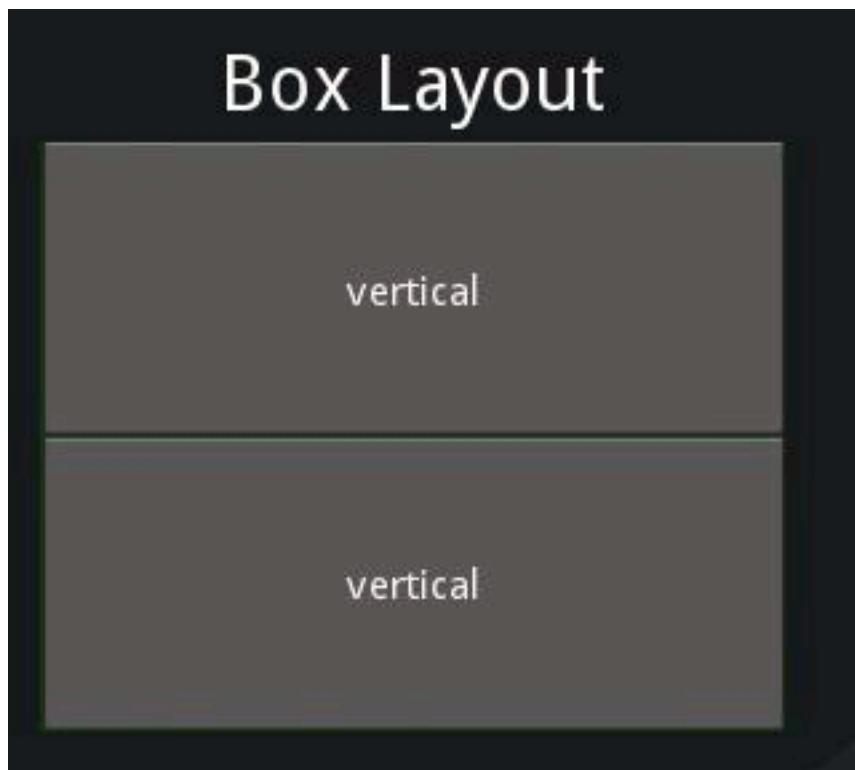
padding

Padding entre a caixa do Widget e seus filhos, em pixel: [padding_left, padding_top, padding_right, padding_bottom].

O Padding também pode ser definido numa forma que tenha dois argumentos [padding_horizontal, padding_vertical] and a one argument form [padding].

padding é um *VariableListProperty* e o padrão é [0, 0, 0, 0].

4.15.7 Box Layout



BoxLayout organiza os filhos no sentido vertical ou horizontal.

Para posicionar o Widget acima/abaixo umdo outro, utilize um BoxLayout vertical:

```
layout = BoxLayout(orientation='vertical')
btn1 = Button(text='Hello')
btn2 = Button(text='World')
layout.add_widget(btn1)
layout.add_widget(btn2)
```

Para posicionar os Widgets próximo (na sequêcia) um dos outros, utilize a propriedade horizontal do BoxLayout. Neste exemplo, utilizamos um espaçamento de 10

pixels entre os filhos; o primeiro botão utiliza 70% do espaço horizontal, o segundo utiliza 30%.

```
layout = BoxLayout(spacing=10)
btn1 = Button(text='Hello', size_hint=(.7, 1))
btn2 = Button(text='World', size_hint=(.3, 1))
layout.add_widget(btn1)
layout.add_widget(btn2)
```

As dicas de posicionamento estão funcionando parcialmente, dependendo da orientação.

- Se a orientação é *vertical*: *x*, *right* e *center_x* será utilizado.
- Se a orientação for *horizontal*: *y*, *top* e *center_y* será utilizado.

Veja os exemplos *examples/widgets(boxlayout_poshint.py* para maiores informações.

Nota: O *size_hint* utiliza o espaço disponível após subtrair o espaço de todos os Widgets definidos com tamanhos fixos. Por exemplo, se você tiver um layout de 800px de largura, e adicionar 3 botões como podemos ver a seguir:

```
btn1 = Button(text='Hello', size=(200, 100), size_hint=(None, None))
btn2 = Button(text='Kivy', size_hint=(.5, 1))
btn3 = Button(text='World', size_hint=(.5, 1))
```

O primeiro botão utilizará 200px de largura como definido, o segundo e o terceiro terão 300px cada um, por exemplo $(800-200) * 0.5$.

Alterado na versão 1.4.1: Adicionado suporte para *pos_hint*.

```
class kivy.uix.boxlayout.BoxLayout(**kwargs)
    Bases: kivy.uix.layout.Layout
```

Class Box layout. Veja o módulo na documentação para maiores informações.

minimum_height

Altura mínima necessária calculada automaticamente para conter todas as crianças.

Novo na versão 1.10.0.

minimum_height é um *NumericProperty* e o padrão é 0. Isso é somente leitura.

minimum_size

Tamanho mínimo necessário computado automaticamente para conter todas as crianças.

Novo na versão 1.10.0.

`minimum_size` é uma propriedade *ReferenceListProperty* do (`minimum_width`, `minimum_height`). Isso é somente leitura.

minimum_width

Largura mínima necessária calculada automaticamente para conter todas as crianças.

Novo na versão 1.10.0.

`minimum_width` é um *NumericProperty* e o padrão é 0. Essa propriedade é somente leitura.

orientation

Orientação do layout.

`orientation` é um *OptionProperty* e o padrão é 'horizontal'. Pode ser 'vertical' ou 'horizontal'.

padding

Padding entre o layout e os filhos: [padding_left, padding_top, padding_right, padding_bottom].

O Padding também pode ser definido numa forma que tenha dois argumentos [padding_horizontal, padding_vertical] and a one argument form [padding].

Alterado na versão 1.7.0: Substituído NumericProperty por VariableListProperty.

`padding` é um *VariableListProperty* e o padrão é [0, 0, 0, 0].

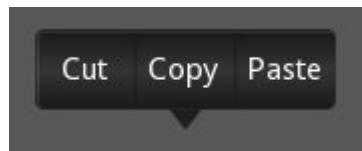
spacing

Espaço entre os filhos, em pixels.

`spacing` é uma *NumericProperty* e o padrão é 0.

4.15.8 Bubble

Novo na versão 1.1.0.



O widget Bubble é uma forma de menu ou um pequeno popup onde as opções de menu são empilhadas verticalmente ou horizontalmente.

A classe *Bubble* contém uma seta apontando na direção que você escolher.

Simples exemplo

```
'''  
Bubble  
=====  
  
Test of the widget Bubble.  
'''  
  
from kivy.app import App  
from kivy.uix.floatlayout import FloatLayout  
from kivy.uix.button import Button  
from kivy.lang import Builder  
from kivy.uix.bubble import Bubble  
  
Builder.load_string('''  
<cut_copy_paste>  
    size_hint: (None, None)  
    size: (160, 120)  
    pos_hint: {'center_x': .5, 'y': .6}  
    BubbleButton:  
        text: 'Cut'  
    BubbleButton:  
        text: 'Copy'  
    BubbleButton:  
        text: 'Paste'  
''))  
  
class cut_copy_paste(Bubble):  
    pass  
  
class BubbleShowcase(FloatLayout):  
  
    def __init__(self, **kwargs):  
        super(BubbleShowcase, self).__init__(**kwargs)  
        self.but_bubble = Button(text='Press to show bubble')  
        self.but_bubble.bind(on_release=self.show_bubble)  
        self.add_widget(self.but_bubble)  
  
    def show_bubble(self, *l):  
        if not hasattr(self, 'bubb'):  
            self.bubb = bubb = cut_copy_paste()  
            self.add_widget(bubb)  
        else:  
            values = ('left_top', 'left_mid', 'left_bottom', 'top_left',
```

```

        'top_mid', 'top_right', 'right_top', 'right_mid',
        'right_bottom', 'bottom_left', 'bottom_mid',
    ↵'bottom_right')
    index = values.index(self.bubb.arrow_pos)
    self.bubb.arrow_pos = values[(index + 1) % len(values)]


class TestBubbleApp(App):

    def build(self):
        return BubbleShowcase()

    if __name__ == '__main__':
        TestBubbleApp().run()

```

Customize o Bubble

Você pode escolher a direção na qual a seta aponta:

```
Bubble(arrow_pos='top_mid')
```

Os Widgets adicionados para o Bubble são ordenados horizontalmente por padrão, semelhante ao que temo com o BoxLayout. Você pode modificar isso por:

```
orientation = 'vertical'
```

Para adicionar itens para o bubble:

```
bubble = Bubble(orientation = 'vertical')
bubble.add_widget(your_widget_instance)
```

Para remover itens:

```
bubble.remove_widget(widget)
or
bubble.clear_widgets()
```

Para acessar a lista de filhos,. use content.children:

```
bubble.content.children
```

Aviso: Isso é o importante! Não utilize o bubble.children

Para modificar a aparência do bubble:

```
bubble.background_color = (1, 0, 0, .5) #50% translucent red
bubble.border = [0, 0, 0, 0]
background_image = 'path/to/background/image'
arrow_image = 'path/to/arrow/image'
```

class kivy.uix.bubble.Bubble(kwargs)**
Bases: *kivy.uix.gridlayout.GridLayout*

Classe Bubble. Veja o módulo da documentação para maiores informações.

arrow_image

Imagen da flecha apontando no/para o Bubble.

arrow_image is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/bubble_arrow'.

arrow_pos

Especifica a posição da seta em relação à bolha. Pode ser uma das opções: left_top, left_mid, left_bottom top_left, top_mid, top_right right_top, right_mid, right_bottom bottom_left, bottom_mid, bottom_right.

arrow_pos é a *OptionProperty* o padrão para 'bottom_mid'.

background_color

Cor de fundo, no formato (r, g, b, a). Utilize essa opção para definir uma das 2 opções *background_image* or *arrow_image* first.

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

background_image

Imagen de fondo do bubble.

background_image é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/bubble'.

border

Borda usado para as instruções gráficas *BorderImage*. Usado com *background_image*. Ele deve ser usado com fundos personalizados.

It must be a list of 4 values: (bottom, right, top, left). Read the BorderImage instructions for more information about how to use it.

border é um *ListProperty* e o padrão é (16, 16, 16, 16)

content

Este é o objeto onde o conteúdo principal da bolha é realizada.

content é um *ObjectProperty* e o padrão é 'None'.

limit_to

Especifica o Widget ao qual a posição das bolhas está restrita.

Novo na versão 1.6.0.

limit_to é uma *ObjectProperty* e o padrão é ‘None’.

orientation

Isso determina a forma na qual os filhos contidos no Bubble serão organizados. Pode ser uma das 2 opções a seguir: ‘vertical’ ou ‘horizontal’.

orientation é um *OptionProperty* e o padrão é ‘horizontal’.

show_arrow

Indica se a seta deve ser mostrada.

Novo na versão 1.8.0.

show_arrow é uma *BooleanProperty* e o padrão é *True*.

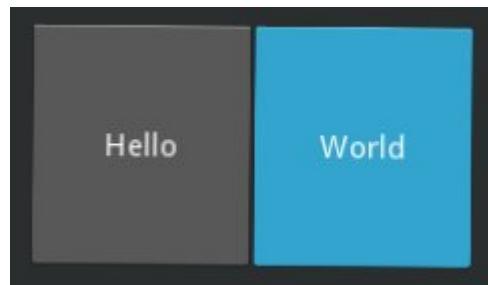
```
class kivy.uix.bubble.BubbleButton(**kwargs)
```

Bases: *kivy.uix.button.Button*

Um botão destinado a ser usado em um Widget Bubble. Você pode usar uma classe de botão “normal”, mas não ficará bem a não ser que o fundo seja alterado.

Utilize este Widget BubbleButton que já está definido e fornece um plano de fundo adequado para você.

4.15.9 Botão



A *Button* é um *Label* com ações associadas que são acionadas quando o botão é pressionado (ou liberado após um clique/toque). Para configurar o botão, as mesmas propriedades (*padding*, *font_size*, etc) e *sizing system* são utilizadas como para the *Label* class:

```
button = Button(text='Hello world', font_size=14)
```

Para anexar uma chamada de retorno quando o botão é pressionado (clicado/tocado), utilize *bind*:

```
def callback(instance):
    print('The button <%s> is being pressed' % instance.text)

btn1 = Button(text='Hello world 1')
btn1.bind(on_press=callback)
```

```
btn2 = Button(text='Hello world 2')
btn2.bind(on_press=callback)
```

Se você deseja ser notificado todas as vezes que o estado do botão é alterado, você pode vincular a propriedade `Button.state`.

```
def callback(instance, value):
    print('My button <%s> state is <%s>' % (instance, value))
btn1 = Button(text='Hello world 1')
btn1.bind(state=callback)
```

`class kivy.uix.button.Button(**kwargs)`
Bases: `kivy.uix.behaviors.button.ButtonBehavior, kivy.uix.label.Label`

Classe Button, veja o módulo da documentação para maiores informações.

Alterado na versão 1.8.0: O comportamento/lógica do botão foi movido para `ButtonBehaviors`.

background_color

Cor de fundo, no forma (r, g, b, a).

Isso age como um *multiplicador* para a cor da textura. A textura padrão é cinza, portanto, só definir a cor de fundo dará um resultado mais escuro. Para definir uma cor simples, defina o `background_normal` para ''.

Novo na versão 1.0.8.

O `background_color` é uma `ListProperty` e o padrão é [1, 1, 1, 1].

background_disabled_down

Imagem de plano de fundo do botão usado para a representação gráfica por padrão quando o botão estiver desativado e pressionado.

Novo na versão 1.8.0.

`background_disabled_down` é um `StringProperty` e o padrão é 'atlas://data/images/defaulttheme/button_disabled_pressed'.

background_disabled_normal

Imagem de plano de fundo do botão usado para a representação gráfica padrão quando o botão é desativado e não pressionado.

Novo na versão 1.8.0.

`background_disabled_normal` é um `StringProperty` e o padrão é 'atlas://data/images/defaulttheme/button_disabled'.

background_down

Imagem de plano de fundo do botão usado para a representação gráfica padrão quando o botão é pressionado.

Novo na versão 1.0.4.

background_down é um *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/button_pressed'.

background_normal

Imagen de plano de fundo do botão usado para a representação gráfica padrão quando o botão não é pressionado.

Novo na versão 1.0.4.

background_normal is a *StringProperty* and defaults to 'atlas://data/images/defaulttheme/button'.

border

Borda utilizada para as instruções gráficas *BorderImage*. Utilizado como *background_normal* e *background_down*. Pode ser usado para fundos personalizados.

It must be a list of four values: (bottom, right, top, left). Read the *BorderImage* instruction for more information about how to use it.

border é um *ListProperty* e o padrão é (16, 16, 16, 16)

4.15.10 Câmera

O Widget *Camera* é utilizado para capturar e para exibir vídeo diretamente de uma câmera. Uma vez que o Widget é criado, a textura dentro do Widget será automaticamente atualizada. Nossa implementação *CameraBase* implementation é utilizada por detrás:

```
cam = Camera()
```

Por padrão, a primeira câmera encontrada no sistema será utilizada. Para utilizar uma câmera diferente, defina a propriedade index:

```
cam = Camera(index=1)
```

Você pode selecionar a resolução da câmera:

```
cam = Camera(resolution=(320, 240))
```

Aviso: A textura da câmera não é atualizada assim que você cria o objeto. A inicialização da câmera é assíncrona, então, pode haver um atraso até que a textura seja criada.

```
class kivy.uix.camera.Camera(**kwargs)
Bases: kivy.uix.image.Image
```

Classe Câmera. Veja a documentação para maiores informações.

index

Índice de câmeras utilizadas, inicia com 0.

index é um *NumericProperty* e o padrão é -1 para permitir a auto seleção.

play

Boolean indica se a câmera está ou não ativa. Você pode iniciar/parar a câmera alterando essas propriedades:

```
# start the camera playing at creation (default)
cam = Camera(play=True)

# create the camera, and start later
cam = Camera(play=False)
# and later
cam.play = True
```

play é um *BooleanProperty* e o padrão é True.

resolution

Resolução preferida pra ser utilizada quando a câmera for invocada. Se você estiver utilizando [-1, -1], a resolução utilizada será a default:

```
# create a camera object with the best image available
cam = Camera()

# create a camera object with an image of 320x240 if_
→possible
cam = Camera(resolution=(320, 240))
```

Aviso: Dependendo da implementação, a câmera poderá não responder a essa propriedade.

resolution é um *ListProperty* e o padrão é [-1, -1].

4.15.11 Carrossel

Novo na versão 1.4.0.

O Widget *Carousel* provê um componente clássico de carrossel mobile-friendly que você pode deslizar entre os vários slides. É possível adicionar qualquer conteúdo a um carrousel e este se movimentará horizontalmente ou verticalmente. O Widget *carousel* pode mostrar páginas numa determinada sequências ou então numa sequência cíclica.

Exemplo:

```
from kivy.app import App
from kivy.uix.carousel import Carousel
from kivy.uix.image import AsyncImage

class CarouselApp(App):
    def build(self):
        carousel = Carousel(direction='right')
        for i in range(10):
            src = "http://placehold.it/480x270.png&text=slide-%d%.
→png" % i
            image = AsyncImage(source=src, allow_stretch=True)
            carousel.add_widget(image)
        return carousel

CarouselApp().run()
```

Alterado na versão 1.5.0: O carousel agora suporta a ativação de filhos, como o *ScrollView*. Isso detectará o gesto de deslizamento de acordo com as propriedades *Carousel.scroll_timeout* e *Carousel.scroll_distance*.

Além disso, o contêiner de slide não é mais exposto pela API. As propriedades impactadas são *Carousel.slides*, *Carousel.current_slide*, *Carousel.previous_slide* e *Carousel.next_slide*.

class kivy.uix.carousel.Carousel(kwargs)**
Bases: *kivy.uix.stencilview.StencilView*

Classe Carousel. Veja o módulo da documentação para maiores informações.

anim_cancel_duration

Define a duração da animação quando o movimento de deslizar não for aceito. Isso geralmente acontece quando o usuário não faz um movimento grande de deslizamento. Veja *min_move*.

anim_cancel_duration é um *NumericProperty* e o padrão é 0.3.

anim_move_duration

Define a duração da animação do Carousel entre as páginas.

anim_move_duration é um *NumericProperty* e o padrão é 0.5.

anim_type

Tipo de animação utilizada enquanto a animação para o slide seguinte/anterior. Este deve ser o nome de uma função *AnimationTransition*.

anim_type é um *StringProperty* e o padrão é 'out_quad'.

Novo na versão 1.8.0.

current_slide

O slide que está sendo mostrado.

current_slide é um *AliasProperty*.

Alterado na versão 1.5.0: A propriedade não expõe o contêiner de slides. A mesma retorna o Widget que você adicionou.

direction

Define a direção na qual o slide será organizado. Isso corresponde com a direção na qual o usuário deslizou para um ou para outro slide na sequência. Pode ser *right*, *left*, *top*, ou *bottom*. Por exemplo, com o valor padrão *right*, o segundo slide está à direita do primeiro e o usuário desliza da direita para a esquerda para chegar ao segundo slide.

direction é um *OptionProperty* e o padrão é ‘right’.

ignore_perpendicular_swipes

Ignora o movimento de deslizamento no eixo perpendicular à direção.

ignore_perpendicular_swipes é um *BooleanProperty* e o padrão é *False*.

Novo na versão 1.10.0.

index

Pega/seta o slide atual baseado no índice.

index é um *AliasProperty* e o padrão é 0 (o primeiro item).

load_next(mode='next')

Animação para o próximo slide.

Novo na versão 1.7.0.

load_previous()

Animação para o slide anterior.

Novo na versão 1.7.0.

load_slide(slide)

Animação para o slide que será passado como o argumento.

Alterado na versão 1.8.0.

loop

Permite o Carousel repetir os slides infinitamente. Se True, quando o usuário tentar deslizar para a próxima página estando na última, será retornada a primeira. Se False, ficará parado última página.

loop é um *BooleanProperty* e o padrão é *False*.

min_move

Define a distância mínima a ser coberta antes de ser considerado o gesto de

de deslizamento e o conteúdo do Carousel seja realmente alterado. Isso expressa uma fração da largura do Carousel. Se o movimento não atingir esse valor mínimo, o movimento será cancelado e o conteúdo será restaurado para o da posição atual.

min_move é um *NumericProperty* e o padrão é 0.2.

next_slide

O próximo slide no Carrossel. Será None caso o slide atual seja o último slide no Carousel. Esta ordem reflete a ordem na qual os slides são adicionados: a apresentação varia de acordo com a propriedade attr: *direction*.

next_slide é um *AliasProperty*.

Alterado na versão 1.5.0: A propriedade não expõe o contêiner de slides. A mesma retorna o Widget que você adicionou.

previous_slide

O slide anterior no Carrossel. Será None caso o slide atual seja o primeiro slide no Carousel. Esta ordem reflete a ordem na qual os slides são adicionados: a apresentação varia de acordo com a propriedade attr: *direction*.

previous_slide é um *AliasProperty*.

Alterado na versão 1.5.0: Esta propriedade não expõe mais o contêiner de slides. A mesma retorna o Widget que você adicionou.

scroll_distance

Distância a ser movida antes da rolagem *Carousel* em pixels. Assim que a distância for percorrida, *Carousel* começará a rolar, e nenhum evento de toque será enviado para os filhos. É aconselhável basear esse valor em dpi referente a tela do dispositivo em que está sendo exibido.

scroll_distance é um *NumericProperty* e o padrão é 20dp.

Novo na versão 1.5.0.

scroll_timeout

Límite de tempo para o acionamento *scroll_distance*, em milissegundos. Se o usuário não tiver movido *scroll_distance* dentro do tempo limite, nenhuma rolagem ocorrerá e o evento de toque será enviado aos filhos.

scroll_timeout é um *NumericProperty* e o padrão é 200 (millisegundos)

Novo na versão 1.5.0.

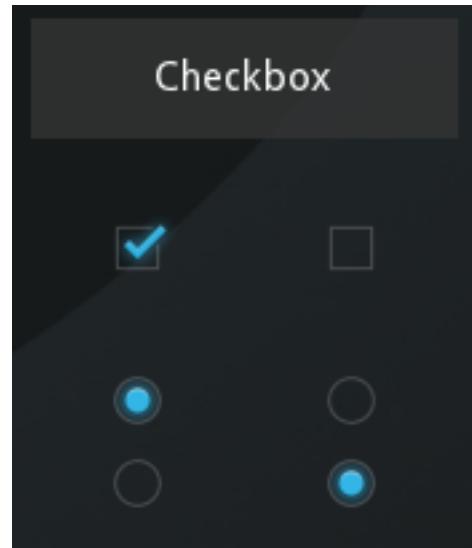
slides

Lista dos slides adicionados ao Carousel. Os slides são os Widgets adicionados ao Carousel e através da função `add_widget`.

slides é um *ListProperty* e é somente leitura.

4.15.12 CheckBox

Novo na versão 1.4.0.



CheckBox é um botão específico de dois estados que pode ser marcado ou desmarcado. Se o CheckBox estiver em um grupo, ele se tornará um RadioButton. Igual ocorre com a classe **ToggleButton**, somente um Radio Button pode ser selecionado por vez. Esse funcionamento ocorre quando `CheckBox.group` estiver definido.

Um exemplo de uso:

```
from kivy.uix.checkbox import CheckBox

# ...

def on_checkbox_active(checkbox, value):
    if value:
        print('The checkbox', checkbox, 'is active')
    else:
        print('The checkbox', checkbox, 'is inactive')

checkbox = CheckBox()
checkbox.bind(active=on_checkbox_active)
```

class kivy.uix.checkbox.CheckBox(kwargs)**
Bases: *kivy.uix.behaviors.togglebutton.ToggleButtonBehavior*,
kivy.uix.widget.Widget

Classe CheckBox, veja o módulo de documentação para maiores informações.

active

Indica se a chave está ou não ativa.

active é um **BooleanProperty** e o padrão é *False*.

background_checkbox_disabled_down

Imagen de fundo do checkbox utilizado por padrão na representação gráfica quando o checkbox está desativado ou não estiver ativo.

Novo na versão 1.9.0.

background_checkbox_disabled_down é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_disabled_on’.

background_checkbox_disabled_normal

Imagen de fundo dos checkbox utilizado por padrão na representação gráfica quando o checkbox está desativado ou não estiver ativo.

Novo na versão 1.9.0.

background_checkbox_disabled_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_disabled_off’.

background_checkbox_down

Imagen de fundo do checkbox utilizado por padrão na representação gráfica quando o checkbox está ativo.

Novo na versão 1.9.0.

background_checkbox_down é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_on’.

background_checkbox_normal

Imagen de fundo do checkbox usado por padrão na representação gráfica quando o checkbox não estiver ativo.

Novo na versão 1.9.0.

background_checkbox_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_off’.

background_radio_disabled_down

Imagen de fundo do radio button usado por padrão na representação gráfica quando o radio button estiver desativado e ativo.

Novo na versão 1.9.0.

background_radio_disabled_down é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_radio_disabled_on’.

background_radio_disabled_normal

Imagen de fundo do radio button usado por padrão na representação gráfica quando o radio button está desabilitado ou não estiver ativo.

Novo na versão 1.9.0.

background_radio_disabled_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_radio_disabled_off’.

background_radio_down

Imagen de fundo do radiobutton usado por padrão na representação gráfica quando o radiobutton está ativo.

Novo na versão 1.9.0.

background_radio_down é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_radio_on’.

background_radio_normal

Imagen de fundo do radiobutton usado por padrão na representação gráfica quando o radiobutton não estiver ativo.

Novo na versão 1.9.0.

background_radio_normal é um *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/checkbox_radio_off’.

color

Cor utiliza para matizar a representação gráfica padrão do checkbox e radiobutton (imagem).

A cor está no formato (r, g, b, a). Utilize o alpha maior do que 1 para cores brilhantes. Use alfa maior que 1 para cores mais brilhantes. Alpha maior que 4 faz com que a mistura da borda e marca de seleção se juntem.

Novo na versão 1.10.0.

color é um *ListProperty* e o padrão é ‘[1, 1, 1, 1]’.

4.15.13 Entrada de Código

Novo na versão 1.5.0.

```
if __name__ == '__main__':
    from kivy.app import App
    from kivy.uix.boxlayout import BoxLayout

    class TextInputApp(App):

        def build(self):
            root = BoxLayout(orientation='vertical')
            textinput = TextInput(multiline=True)
            textinput.text = __doc__
            root.add_widget(textinput)
            textinput2 = TextInput(text='monoline textinput',
                                  size_hint=(1, None), height=30)
            root.add_widget(textinput2)
            return root

    TextInputApp().run()
```

BoxLayout:
Double as a Tabbed Panel Demo!

TabbedPanel:
tab_pos: "top_right"
default_tab_text: "List View"
default_tab_content: list_view_tab

TabbedPanelHeader:
text: 'Icon View'
content: icon_view_tab

FileChooserListView:
id: list_view_tab

FileChooserIconView:
id: icon_view_tab
show_hidden: True

Nota: This widget requires `pygments` package to run. Install it with `pip`.

A `CodeInput` fornece uma caixa de texto editável e destacada. Veja imagem a seguir. Suporta todos os recursos fornecidos pelo `textinput`, bem como, o código destacado para [languages supported by pygments KivyLexer for `kivy.lang`](#).

Exemplo de uso

Para criar um `CodeInput` com palavras destacadas para *KV Language*:

```
from kivy.uix.codeinput import CodeInput
from kivy.extras.highlight import KivyLexer
codeinput = CodeInput(lexer=KivyLexer())
```

Para criar um `CodeInput` com palavras destacadas para *Cython*:

```
from kivy.uix.codeinput import CodeInput
from pygments.lexers import CythonLexer
codeinput = CodeInput(lexer=CythonLexer())
```

class kivy.uix.codeinput.CodeInput(kwargs)**
Bases: *kivy.uix.behaviors.codenavigation. CodeNavigationBehavior, kivy.uix.textinput.TextInput*

Classe `CodeInput`, usada para mostrar código com palavras destacadas.

lexer

Isso mantém selecionado o Lexer utilizado pela biblioteca Pygments para destacar o código.

`lexer` é um *ObjectProperty* e o padrão é *PythonLexer*.

style

Objeto de estilo Pygments utilizado na formação.

Quando ‘`style_name`’ está setado, será modificado o estilo de objeto correspondente.

`style` é um *ObjectProperty* e o padrão é `None`

style_name

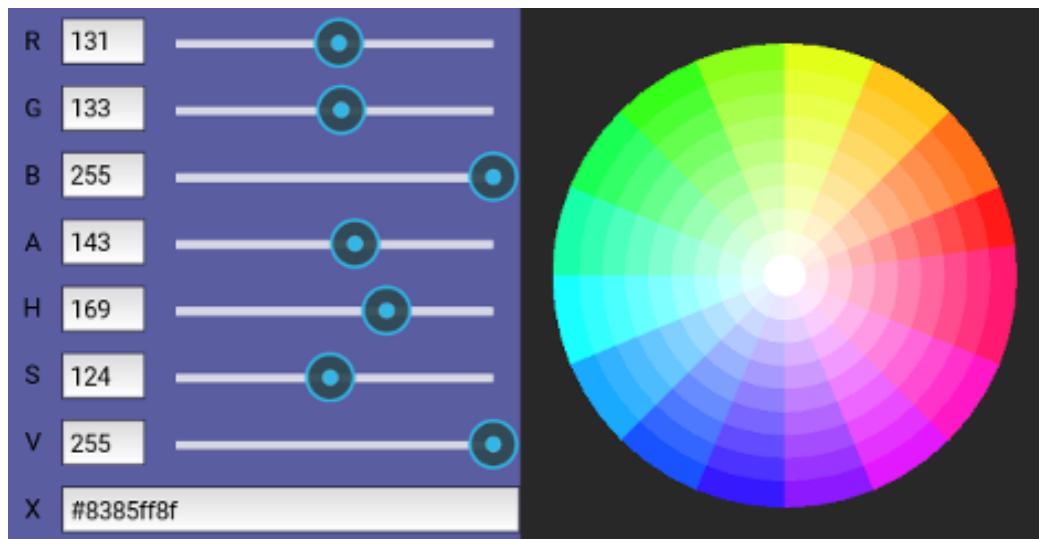
Nome do estilo do Pygments para fazer a formatação.

`style_name` é um *OptionProperty* e o padrão é ‘`default`’.

4.15.14 Color Picker

Novo na versão 1.7.0.

Aviso: Este Widget é experimental. Seu uso e API podem ser alterados a qualquer momento até que este aviso seja removido.



O widget ColorPicker permite ao usuário selecionar uma cor de uma roda cromática, onde pinça e zoom podem ser usados para mudar a saturação da roda. Sliders e entradas de Texto também são providas para entrada de valores RGBA/HSV/HEX diretamente.

Uso:

```
clr_picker = ColorPicker()  
parent.add_widget(clr_picker)  
  
# To monitor changes, we can bind to color property changes  
def on_color(instance, value):  
    print "RGBA = ", str(value) # or instance.color  
    print "HSV = ", str(instance.hsv)  
    print "HEX = ", str(instance.hex_color)  
  
clr_picker.bind(color=on_color)
```

```
class kivy.uix.colorpicker.ColorPicker(**kwargs)  
    Bases: kivy.uix.relativelayout.RelativeLayout
```

Veja o módulo da documentação.

color

O **color** tem a cor atualmente selecionada no formato RGBA.

color é uma *ListProperty* e o padrão é (1, 1, 1, 1).

font_name

Especifica a fonte usada no *ColorPicker*.

`font_name` é uma *StringProperty* e o padrão é ‘data/fonts/RobotoMono-Regular.ttf’.

hex_color

O `hex_color` contém a cor atualmente selecionada em hexadecimal.

`hex_color` é uma *AliasProperty* e o padrão é `#ffffffff`.

hsv

O `hsv` contém a cor atualmente selecionada no forma HSV.

`hsv` uma *ListProperty* e o padrão é `(1, 1, 1)`.

wheel

O `wheel` contém a roda de cores.

`wheel` é uma *ObjectProperty* e o padrão é `None`.

class kivy.uix.colorpicker.ColorWheel(kwargs)**

Bases: `kivy.uix.widget.Widget`

Roda cromática para o *ColorPicker*.

Alterado na versão 1.7.1: `font_size`, `font_name` e `foreground_color` forma removidos. O dimensionamento é agora o mesmo que outros Widget, com base em ‘sp’. A orientação também é determinada automaticamente de acordo com a relação largura/altura.

a

Valor Alpha da cor atualmente selecionada.

`a` é uma *BoundedNumericProperty* e pode ser um valor entre 0 a 1.

b

Valor Azul da cor atualmente selecionada.

`b` é uma *BoundedNumericProperty* e pode ser um valor entre 0 a 1.

color

Mantém a cor selecionada atualmente.

`color` é uma *ReferenceListProperty* e contém uma lista de valores `r`, `g`, `b`, `a`.

g

Valor Verde da cor atualmente selecionada.

`g` é uma *BoundedNumericProperty* e pode ser um valor entre 0 a 1.

r

Valor Vermelho da cor atualmente selecionada.

`r` é uma *BoundedNumericProperty* e pode ser um valor entre 0 até 1. O padrão é 0.

4.15.15 Lista Drop-Down

Novo na versão 1.4.0.

Uma versátil lista drop-down que pode ser utilizada como um Widget customizado. Ele permite exibir uma lista de Widget quando o Widget estiver sendo exibido. Diferentemente de outros toolkits, a lista de Widget pode conter qualquer tipo de Widget: simples botão, imagem e etc.

O posicionamento da lista drop-down é montada automaticamente: sempre tentaremos colocar a lista dropdown de tal forma que o usuário seja capaz de selecionar um item da lista.

Simples exemplo

Um botão com uma lista dropdown contendo 10 valores possíveis. Todos os botões contidos na lista dropdown adicionarão o método :meth:`DropDown.select`. Após ser invocado, o texto do botão principal mostrará a seleção do dropdown.

```
from kivy.uix.dropdown import DropDown
from kivy.uix.button import Button
from kivy.base import runTouchApp

# create a dropdown with 10 buttons
dropdown = DropDown()
for index in range(10):
    # When adding widgets, we need to specify the height manually
    # (disabling the size_hint_y) so the dropdown can calculate
    # the area it needs.

    btn = Button(text='Value %d' % index, size_hint_y=None,
    height=44)
    # for each button, attach a callback that will call the
    select() method
    # on the dropdown. We'll pass the text of the button as the
    data of the
    # selection.
    btn.bind(on_release=lambda btn: dropdown.select(btn.text))

    # then add the button inside the dropdown
    dropdown.add_widget(btn)

# create a big main button
mainbutton = Button(text='Hello', size_hint=(None, None))

# show the dropdown menu when the main button is released
```

```

# note: all the bind() calls pass the instance of the caller (here, ↴
# the
# mainbutton instance) as the first argument of the callback (here,
# dropdown.open.).
mainbutton.bind(on_release=dropdown.open)

# one last thing, listen for the selection in the dropdown list and
# assign the data to the button text.
dropdown.bind(on_select=lambda instance, x: setattr(mainbutton,
    ↴'text', x))

runTouchApp(mainbutton)

```

Estendendo o dropdown em kv

Você pode criar um dropdown diretamente do seu kv:

```

#:kivy 1.4.0
<CustomDropDown>:
    Button:
        text: 'My first Item'
        size_hint_y: None
        height: 44
        on_release: root.select('item1')
    Label:
        text: 'Unselectable item'
        size_hint_y: None
        height: 44
    Button:
        text: 'My second Item'
        size_hint_y: None
        height: 44
        on_release: root.select('item2')

```

E então, criar uma associação com uma classe Python e utiliza-la:

```

class CustomDropDown(DropDown):
    pass

dropdown = CustomDropDown()
mainbutton = Button(text='Hello', size_hint=(None, None))
mainbutton.bind(on_release=dropdown.open)
dropdown.bind(on_select=lambda instance, x: setattr(mainbutton,
    ↴'text', x))

```

```

class kivy.uix.dropdown.DropDown(**kwargs)
Bases: kivy.uix.scrollview.ScrollView

```

Classe DropDown. Veja o modulo da documentação para maiores informações.

Events

on_select: `data` Disparado quando uma seleção estiver pronta. O dado da seleção é passada como o primeiro argumento e é o que você passará pelo método `select()` como sendo o primeiro argumento.

on_dismiss: Novo na versão 1.8.0.

Disparado quando o DropDown é descartado, quando for selecionado ou então, quando houver um toque fora das dimensões do Widget.

attach_to

(Interna) Propriedade que será definida para o widget no qual a lista suspensa está anexada.

O método `open()` definirá automaticamente esta propriedade `dismiss()` enquanto este voltará a ser `None`.

auto_dismiss

Por padrão, a lista suspensa será automaticamente descartada quando um toque acontecer fora das dimensões da lista, essa opção permite desativar essa funcionalidade.

`auto_dismiss` é um `BooleanProperty` e o padrão é `True`.

Novo na versão 1.8.0.

auto_width

Por padrão, a largura da lista suspensa terá a mesma largura dos Widget anexados. Defina como `False` caso queiras definir a largura manualmente.

`auto_width` é um `BooleanProperty` e o padrão é `True`.

container

(interno) Propriedade que será definida o container da lista suspensa. Isso será por padrão da classe `GridLayout`.

dismiss(*args)

Remove o Widget de lista suspensa da janela e desanexa-o do Widget anexado.

dismiss_on_select

Por padrão, a lista suspensa será automaticamente descartado quando a seleção finalizar. Defina como `False` para evitar o descarte.

`dismiss_on_select` é um `BooleanProperty` e o padrão é `True`.

max_height

Indica a altura máxima que a lista suspensa pode ter. Se `None`, a mesma terá

a altura máxima disponível até que a margem superior ou inferior da tela seja alcançado.

`max_height` é um *NumericProperty* e o padrão é *None*.

min_state_time

Tempo mínimo antes do **DropDown** ser descartado. Isso é utilizado para permitir que o Widget dentro da lista suspensa exiba um estado de desativação para a **DropDown** e exiba uma animação de fechamento.

`min_state_time` é um *NumericProperty* e o padrão para *Config* é o valor *min_state_time*.

Novo na versão 1.10.0.

open(widget)

Abre a lista suspensa e anexa-a a um determinado Widget. Dependendo da posição do Widget dentro da janela e a altura da lista suspensa, o menu suspenso pode ficar acima ou abaixo do Widget.

select(data)

Chama o método para disparar o evento *on_select* como o *data selection*. O dado *data* pode ser qualquer coisa que você desejar.

4.15.16 EffectWidget

Novo na versão 1.9.0.

A classe **EffectWidget** é capaz de criar efeitos gráficos divertidos para crianças. Ela funciona renderizando uma série de instâncias da classe **Fbo** com sombreamentos personalizados da OpenGL.

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

A utilização básica é a seguinte:

```
w = EffectWidget()
w.add_widget(Button(text='Hello!'))
w.effects = [InvertEffect(), HorizontalBlurEffect(size=2.0)]
```

O equivalente na linguagem kv seria:

```
#: import ew kivy.uix.effectwidget
EffectWidget:
    effects: ew.InvertEffect(), ew.HorizontalBlurEffect(size=2.0)
    Button:
        text: 'Hello!'
```

O efeito pode ser uma lista de efeitos de qualquer tamanho, e eles serão aplicados sequencialmente.

O módulo vem com uma gama de efeitos pré-construídos, mas a interface é projetada para tornar mais fácil criar o seu próprio. Em vez de escrever um shader glsl completo, você fornece uma única função que leva algumas entradas baseadas na tela (cor de pixel atual, textura do widget atual etc.). Consulte as seções abaixo para obter mais informações.

Diretrizes de Uso

Não é eficiente redimensionar uma classe: `EffectWidget`, como a classe: `~kivy.graphics.Fbo` é recriada em cada evento de redimensionamento. Se você precisar redimensionar com frequência, considere fazer as coisas de uma maneira diferente.

Embora alguns efeitos tenham parâmetros ajustáveis, isto não é eficiente para animar estes, como todo o shader é reconstruído todo momento. Você deve usar variáveis glsl uniforme em vez disso. A classe: `AdvancedEffectBase` pode tornar isso mais fácil.

Nota: A classe: `EffectWidget` não pode desenhar fora de sua própria área de widget (pos -> pos + size). Quaisquer widgets filhos sobrepostos ao limite serão cortados neste momento.

Provedores de Efeitos

O módulo vem com vários efeitos pré-escritos. Alguns têm propriedades ajustáveis (por exemplo, raio de desfocagem). Consulte a documentação de efeitos individuais para obter mais detalhes.

- `MonochromeEffect` - faz o Widget ser preto e branco (grayscale).
- `InvertEffect` - Inverte as cores do Widget.
- `ChannelMixEffect` - troca os canais de cores.
- `ScanlinesEffect` - exibe varredura scanlines.
- `PixelateEffect` - pixelates a imagem.
- `HorizontalBlurEffect` - Gaussuan Blurs Horizontalmente
- `VerticalBlurEffect` - Gaussuan Blurs Verticalmente
- `FXAAEffect` - aplica um muito simples anti-aliasing.

Criando Cores

Os efeitos são projetados para tornar mais fácil criar e usar suas próprias transformações. Faça isso criando e usando uma instância da classe: *EffectBase* com sua própria propriedade personalizada: attr: 'EffectBase.gsls'.

A propriedade GLSL é uma String que representa parte de um Shader de fragmentos GLSL. Pode incluir tantas funções quanto quiseres (a String é simplesmente emendada em todo o Shader), mas deve implementar uma função **effect** como abaixo:

```
vec4 effect(vec4 color, sampler2D texture, vec2 tex_coords, vec2  
           coords)  
{  
    // ... your code here  
    return something; // must be a vec4 representing the new color  
}
```

O sombreador/'shader' completo calculará a cor de pixel normal em cada ponto e, em seguida, chamará sua função: *effect* para transformá-la. Os parâmetros são:

- **color**: A cor normal do pixel atual (por exemplo textura amostrada em *tex_coords*).
- **texture**: A textura normal contida no fundo dos Widgets.
- **tex_coords**: O normal *texture_coords* utilizado para acessar a textura.
- **coords**: O índice do pixel do pixel atual.

O código do Shader também tem acesso a duas variáveis uniformes **time** contendo o tempo (em segundos) desde o início do programa e **resolution** contendo a forma (*x pixels, y pixels*) do widget

Por exemplo, a seguinte simples String (pegado do *InvertEffect*) iria inverter a cor de entrada, mas definiu alfa para 1.0:

```
vec4 effect(vec4 color, sampler2D texture, vec2 tex_coords, vec2  
           coords)  
{  
    return vec4(1.0 - color.xyz, 1.0);  
}
```

Você também pode definir o GLSL carregando automaticamente a String de um arquivo, basta definir a propriedade **EffectBase.source** de um efeito.

```
class kivy.uix.effectwidget.EffectWidget(**kwargs)  
    Bases: kivy.uix.relativelayout.RelativeLayout
```

Widget com a capacidade de aplicar uma série de efeitos gráficos para seus filhos. Consulte a documentação do módulo para obter mais informações sobre como definir efeitos e como criar os seus próprios efeitos personalizados.

background_color

Isso define a cor de plano de fundo a ser usado pelo FBO no *EffectWidget*.

background_color é um *ListProperty* e o padrão é (0, 0, 0, 0)

effects

Lista de todos os efeitos a serem aplicados. Estas devem ser todas as instâncias ou subclasses de *EffectBase*.

effects é uma *ListProperty* e o padrão é [].

fbo_list

(interno) Lista de todos os FBO's que estão sendo usados para aplicar os efeitos.

fbo_list é um *ListProperty* e o padrão é [].

refresh_fbo_setup(*args)

(interno) Cria e atribui uma *Fbo* por efeito, e assegura que todos os tamanhos e etc são corretos e consistentes.

texture

A textura de saída do final *Fbo* depois de todos os efeitos tiverem sido aplicados.

textura é um *ObjectProperty* e o padrão é *None*.

class kivy.uix.effectwidget.*EffectBase*(*args, **kwargs)

Bases: *kivy.event.EventDispatcher*

A classe base para efeitos FLSL. Ele simplesmente retorna a sua entrada.

Veja o módulo da documentação para maiores informações.

fbo

O FBO usado atualmente neste efeito. A *EffectBase* trata automaticamente isso.

fbo é um *ObjectProperty* e o padrão é *None*.

glsl

A String GLSL que define sua função de efeito. Veja o módulo da documentação para maiores informações.

glsl é uma *StringProperty* e o padrão é um efeito trivial que retorna sua entrada.

set_fbo_shader(*args)

Define o Shader *Fbo*'s através do splicing da String *glsl* em um fragmento de Shader completo.

O Shader completo é composto por *shader_header* + *shader_uniforms* + *self.glsl* + *shader_footer_effect*.

source

O nome do arquivo (opcional) a partir do qual carregar a String **glsl** string.

source é um **StringProperty** e o padrão é ''.

class kivy.uix.effectwidget.AdvancedEffectBase(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Uma **EffectBase** com comportamento adicional para facilmente definir e atuar uniformemente variáveis em seus Shader.

Esta classe é fornecida por conveniência quando implementares seus próprios efeitos: ela não é usado pelas classes fornecidas com o Kivy.

Além da sua sequência GLSL básica que deve ser fornecida como normal, a **AdvancedEffectBase** tem uma propriedade extra **uniforms**, um dicionário de pares nome-valor. Sempre que um valor for alterado, o novo valor para a variável uniforme será carregado para o Shader.

Você ainda deve declarar manualmente suas variáveis uniformes no topo da sua String GLSL.

uniforms

Um dicionário de nomes de variáveis uniformes e seus valores. Estes são automaticamente carregados para o Shader fbo se for apropriado.

uniformes é **DictProperty** e o padrão é {}.

class kivy.uix.effectwidget.MonochromeEffect(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Retorna suas cor de entrada em monocromático.

class kivy.uix.effectwidget.InvertEffect(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Inverte as cores na entrada.

class kivy.uix.effectwidget.ChannelMixEffect(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Mistura os canais de cor da entrada de acordo com a propriedade de ordem. Os canais podem ser arbitrariamente reorganizados ou repetidos.

order

A nova ordem de classificação dos canais RGB.

ordem é **ListProperty** e o padrão é [1, 2, 0], correspondente ao (g, b, r).

class kivy.uix.effectwidget.ScanlinesEffect(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Adiciona scanlines à entrada.

class kivy.uix.effectwidget.PixelateEffect(*args, **kwargs)

Bases: **kivy.uix.effectwidget.EffectBase**

Pixellates a entrada de acordo com seu ***pixel_size***

pixel_size

Define o tamanho de um novo ‘pixel’ no efeito, em termos de número de pixels ‘reais’.

pixel_size é uma *NumericProperty* e o padrão é 10.

```
class kivy.uix.effectwidget.HorizontalBlurEffect(*args, **kwargs)
```

Bases: *kivy.uix.effectwidget.EffectBase*

Borra a entrada horizontalmente, com a largura dada por *size*.

size

Largura do borrão em píxels.

tamanho é um *NumericProperty* e o padrão é 4.0.

```
class kivy.uix.effectwidget.VerticalBlurEffect(*args, **kwargs)
```

Bases: *kivy.uix.effectwidget.EffectBase*

Borra a entrada vertical, com a largura dada por *size*.

size

Largura do borrão em píxels.

tamanho é um *NumericProperty* e o padrão é 4.0.

```
class kivy.uix.effectwidget.FXAAEffect(*args, **kwargs)
```

Bases: *kivy.uix.effectwidget.EffectBase*

Aplica um anti-aliasing muito simples via *fxaa*.

4.15.17 FileChooser

O módulo FileChooser fornece várias classes para descrever, exibir e navegar pelo sistema de arquivos.

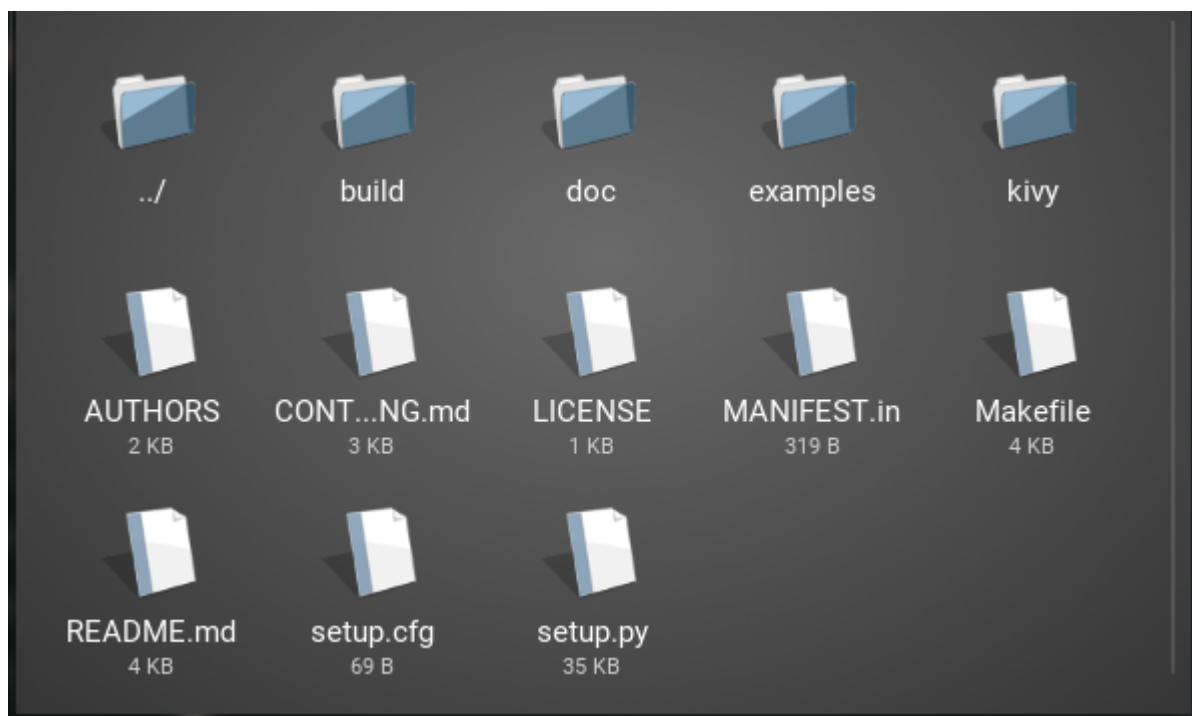
Widgets Simples

Há 2 Widget prontos para uso que fornecem visualização do sistema de arquivos. Cada um destes apresenta os arquivos e pastas em um estilo diferente.

A *FileChooserListView* exibe entrada de arquivos como itens de texto numa lista vertical, onde pastas podem ser recolhidas e expandidas.

Name	Size
../	
> build	
> doc	
> examples	
> kivy	
AUTHORS	2 KB
CONTRIBUTING.md	3 KB
LICENSE	1 KB
MANIFEST.in	319 B
Makefile	4 KB
README.md	4 KB
setup.cfg	69 B
setup.py	35 KB

A [FileChooserIconView](#) apresenta ícones e texto da esquerda para direita, envolvendo-os conforme necessário.



Ambos fornecem o deslocamento, seleção e o básico da interação com o usuário. Para maiores informações [FileChooserController](#) e detalhes do suporte a eventos e propriedades.

Composição do Widget

A classe FileChooser adotou um desenho **MVC**. Eles são expostos para que você possa estender e personalizar seu selecionador de arquivos de acordo com suas necessidades.

As classes do FileChooser podem ser categorizadas da seguinte forma:

- Modelos são representados por implementações concretas da classe **FileSystemAbstract** class, como a **FileSystemLocal**.
- Views são representadas pelas classes **FileChooserListView** e **FileChooserIconLayout**. Elas são usadas pelos Widgets **FileChooserListView** e **FileChooserIconView** respectivamente.
- Controladores são representados por implementações concretas de **FileChooserController**, classes chamadas de **FileChooser**, **FileChooserIconView** e **FileChooserListView**.

Isso significa que você pode definir suas próprias Views ou implementar provedores **FileSystemAbstract** para sistemas de arquivos diferentes para usar com estes Widgets. A classe **FileChooser** pode ser usada como um controlador por múltiplos handles, sincronizando Views no mesmo path. Pela combinação destes elementos, é possível adicionar suas próprias Views e sistemas de arquivos e então, facilmente interagir com os componentes existentes.

Exemplo de uso

main.py

```
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.factory import Factory
from kivy.properties import ObjectProperty
from kivy.uix.popup import Popup

import os

class LoadDialog(FloatLayout):
    load = ObjectProperty(None)
    cancel = ObjectProperty(None)

class SaveDialog(FloatLayout):
    save = ObjectProperty(None)
    text_input = ObjectProperty(None)
    cancel = ObjectProperty(None)
```

```

class Root(FloatLayout):
    loadfile = ObjectProperty(None)
    savefile = ObjectProperty(None)
    text_input = ObjectProperty(None)

    def dismiss_popup(self):
        self._popup.dismiss()

    def show_load(self):
        content = LoadDialog(load=self.load, cancel=self.dismiss_
        ↪popup)
        self._popup = Popup(title="Load file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()

    def show_save(self):
        content = SaveDialog(save=self.save, cancel=self.dismiss_
        ↪popup)
        self._popup = Popup(title="Save file", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()

    def load(self, path, filename):
        with open(os.path.join(path, filename[0])) as stream:
            self.text_input.text = stream.read()

        self.dismiss_popup()

    def save(self, path, filename):
        with open(os.path.join(path, filename), 'w') as stream:
            stream.write(self.text_input.text)

        self.dismiss_popup()

class Editor(App):
    pass

Factory.register('Root', cls=Root)
Factory.register('LoadDialog', cls=LoadDialog)
Factory.register('SaveDialog', cls=SaveDialog)

if __name__ == '__main__':
    Editor().run()

```

editor.kv

```
#:kivy 1.1.0

Root:
    text_input: text_input

BoxLayout:
    orientation: 'vertical'
    BoxLayout:
        size_hint_y: None
        height: 30
        Button:
            text: 'Load'
            on_release: root.show_load()
        Button:
            text: 'Save'
            on_release: root.show_save()

    BoxLayout:
        TextInput:
            id: text_input
            text: ''

    RstDocument:
        text: text_input.text
        show_errors: True

<LoadDialog>:
    BoxLayout:
        size: root.size
        pos: root.pos
        orientation: "vertical"
        FileChooserListView:
            id: filechooser

        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                on_release: root.cancel()

            Button:
                text: "Load"
                on_release: root.load(filechooser.path, filechooser.
        ↪selection)

<SaveDialog>:
```

```

text_input: text_input
BoxLayout:
    size: root.size
    pos: root.pos
    orientation: "vertical"
FileChooserListView:
    id: filechooser
    on_selection: text_input.text = self.selection and self.
    ↪selection[0] or ''
    ↪

TextInput:
    id: text_input
    size_hint_y: None
    height: 30
    multiline: False

BoxLayout:
    size_hint_y: None
    height: 30
    Button:
        text: "Cancel"
        on_release: root.cancel()

    Button:
        text: "Save"
        on_release: root.save(filechooser.path, text_input.
    ↪text)

```

Novo na versão 1.0.5.

Alterado na versão 1.2.0: No modelo de escolha, o *controller* não é mais uma referência direta, mas sim, uma referência fraca (weak-reference). Se você atualizar, deves modificar a notação de *root.controller.xxx* para *root.controller().xxx*.

class kivy.uix.filechooser.FileChooserListView(kwargs)**
 Bases: *kivy.uix.filechooser.FileChooserController*

Implementação da *FileChooserController* usando um ListView.

Novo na versão 1.9.0.

class kivy.uix.filechooser.FileChooserIconView(kwargs)**
 Bases: *kivy.uix.filechooser.FileChooserController*

Implementação de uma *FileChooserController* usando um IconView.

Novo na versão 1.9.0.

class kivy.uix.filechooser.FileChooserLayout(kwargs)**
 Bases: *kivy.uix.filechooser.FileChooserLayout*

Layout do FileChooser usando uma exibição de lista.

Novo na versão 1.9.0.

```
class kivy.uix.filechooser.FileChooserIconLayout(**kwargs)
    Bases: kivy.uix.filechooser.FileChooserLayout
```

Layout do FileChooser usando uma exibição de ícone.

Novo na versão 1.9.0.

```
class kivy.uix.filechooser.FileChooser(**kwargs)
    Bases: kivy.uix.filechooserFileChooserController
```

Implementação da *FileChooserController* que suporta a mudança entre várias visualizações de layout sincronizadas.

O FileChooser pode ser usado como no exemplo a seguir:

```
BoxLayout:
    orientation: 'vertical'

    BoxLayout:
        size_hint_y: None
        height: sp(52)

        Button:
            text: 'Icon View'
            on_press: fc.view_mode = 'icon'
        Button:
            text: 'List View'
            on_press: fc.view_mode = 'list'

    FileChooser:
        id: fc
        FileChooserIconLayout
        FileChooserListLayout
```

Novo na versão 1.9.0.

manager

Referência para a instância de *ScreenManager*.

O gerenciador é uma an *ObjectProperty*.

view_list

Lista de Views adicionada para este FileChooser.

view_list é uma *AliasProperty* do tipo *list*.

view_mode

Modo de exibição de layout atual.

view_mode é uma *AliasProperty* do tipo *str*.

```
class kivy.uix.filechooser.FileChooserController(**kwargs)
```

Bases: [kivy.uix.relativelayout.RelativeLayout](#)

Bases para implementar um FileChooser. Não use essa classe diretamente, mas prefira usar uma implementação como o [FileChooser](#), [FileChooserListView](#) ou [FileChooserIconView](#).

Events

on_entry_added: entry, parent Disparando quando uma entrada no nível da raiz é adicionada para a lista de arquivos. Se você retornar *True* neste evento, a entrada não será adicionada ao *FileChooser*.

on_entries_cleared Disparado quando a entrada na lista é desmarcada, geralmente quando a raiz é atualizada.

on_subentry_to_entry: entry, parent Disparado quando uma subentrada é adicionada a uma entrada existente ou quando uma entrada é removida desde uma entrada. Por exemplo, quando um nó é fechado.

on_submit: selection, touch Disparado quando um arquivo foi selecionado com duplo toque.

cancel(*largs)

Cancaela qualquer ação do plano de fundo iniciado pelo *FileChooser*, como a abertura de um novo diretório.

Novo na versão 1.2.0.

dirselect

Determina se os diretórios são ou não válidos.

dirselect é uma [BooleanProperty](#) e o padrão é *False*.

Novo na versão 1.1.0.

entry_released(entry, touch)

(interno) Este método deve ser chamado pelo template quando um entrada é tocada pelo usuário.

Novo na versão 1.1.0.

entry_touched(entry, touch)

(interno) Este método deve ser chamado pelo template quando um entrada é tocada pelo usuário.

file_encodings

Possíveis codificações para decodificar um nome de arquivo para unicode. No caso de o usuário ter um nome de arquivo não-ascii, undecodable sem saber que é codificação inicial, não temos outra escolha senão adivinhar.

Por favor, note que se você encontrar um erro porque um encoding está faltando, ficaremos felizes em adicioná-lo à lista.

file_encodings é uma *ListProperty* e o padrão é `['utf-8', 'latin1', 'cp1252']`.

Novo na versão 1.3.0.

Obsoleto desde a versão 1.8.0: Esta propriedade não é mais usada como o FileChooser já decodifica os nomes dos arquivos.

file_system

O objeto de sistema de arquivos usado para acessar o sistema de arquivos. Isso deve ser uma subclasse de *FileSystemAbstract*.

file_system é uma *ObjectProperty* e o padrão é *FileSystemLocal()*

Novo na versão 1.8.0.

files

A lista de arquivos no diretório especificado pelo path depois de aplicado os filtros.

files é somente leitura *ListProperty*.

filter_dirs

Indica se os filtros deve ser aplicados aos diretórios. *filter_dirs* é uma *BooleanProperty* e o padrão é *False*.

filters

filters especifica os filtros a serem aplicados ao arquivos no diretório. *filters* é uma *ListProperty* e o parão é `[]`. Isto é equivale a '*' ou seja, nada é filtrado.

Os filtros não são resetados quando o PATH é alterado. Você precisa fazer alterar você mesmo caso desejes.

Há 2 tipos de filtros: patterns e callbacks.

1.Padrões

Por exemplo `'*.png'`. Você pode usar o seguinte padrão:

Padrão	Significado
*	Combina tudo
?	Corresponde a qualquer caractere
[seq]	Corresponde a qualquer caractere em seq
[!seq]	Corresponde a qualquer caractere que não está em seq

2.Callbacks

Você pode especificar uma função que pode ser chamada para cada arquivos. O callback será passada para a pasta e o nome como o primeiro e o segundo parâmetro respectivamente. Ele

deve retornar *True* para indicar uma correspondência e *False* caso contrário.

Alterado na versão 1.4.0: Adiciona a opção para especificar o filtro como um callback.

get_nice_size(*fn*)

Passa o nome do arquivo. Retorna o tamanho na melhor formato que humanos passam lher ou “ caso seja um diretório. (Não calcula recursivamente o tamanho).

layout

Referência à instância do Widget de layout.

Layout é uma *ObjectProperty*.

Novo na versão 1.9.0.

multiselect

Determina se o usuário é ou não é capaz de selecionar múltiplos arquivos.

multiselect é uma *BooleanProperty* e o padrão é *False*.

path

path é uma *StringProperty* e o padrão para o diretório de trabalho atual como uma String Unicode. Ele especifica o path no sistema de arquivos que esse controlador deve referenciar.

Aviso: Se um PATH unicode é especificado, todos os arquivos retornados serão também Unicode, permitindo a exibição de arquivos e paths Unicode. Se um caminho de bytes for especificado, somente arquivos e paths com nomes ASCII serão mostrados corretamente: nomes de arquivos não ASCII serão exibidos e listado com um ponto de interrogação (?) ao invés dos seus caracteres Unicodes.

progress_cls

Classe usada para exibir um indicador de progresso do carregamento do *FileChooser*.

progress_cls é uma *ObjectProperty* e o padrão é *FileChooserProgress*.

Novo na versão 1.2.0.

Alterado na versão 1.8.0: Se definidos com String, a *Factory* será usada para resolver o nome da classe.

rootpath

Caminho principal a ser usado a invés do caminho da raiz do sistema. Se definido, não mostrará um diretório “..” para ir ao caminho raiz. Por exem-

plo, se você definir *rootpath* como sendo */users/foo*, o usuário não poderá ir a */users* ou para qualquer outro diretório que não comece com */users/foo*.

rootpath é uma *StringProperty* e o padrão é *None*.

Novo na versão 1.2.0.

Nota: Semelhante ao *path*, se *rootpath* é definido como bytes ou String Unicode isso determina o tipo de nome de arquivos e paths a serem lidos.

selection

Contém a lista de arquivos que estão atualmente selecionados.

select é somente leitura *ListProperty* e o padrão é *[]*.

show_hidden

Determina se arquivos ocultos e pastas deve ser mostrados.

show_hidden é uma *BooleanProperty* e o padrão é *False*.

sort_func

Fornece uma função pra ser chamado com uma lista de nomes de arquivos com o primeiro argumento e a implementação do sistema de arquivos como sendo o segundo argumento. Retorna uma lista de nomes de arquivos organizado pelo exibição na View.

sort_func é uma *ObjectProperty* e o padrão é uma função que retorna pastas com nome alfanumérico por primeiro.

Alterado na versão 1.8.0: A assinatura precisa agora de 2 argumentos: o primeiro é uma lista de arquivos e o segundo a classe de sistema de arquivos a ser utilizada.

class kivy.uix.filechooser.FileChooserProgressBase(kwargs)**

Bases: *kivy.uix.floatlayout.FloatLayout*

Base para a implementação de um Progress View. Esta View é usada quando houver muitas entradas precisando ser criada e mostrada sobre vários Frames.

Novo na versão 1.2.0.

cancel(*largs)

Cancaela qualquer ação de *FileChooserController*.

index

Índice atual das entradas *total* a serem abertas.

path

Path atual do *FileChooser*, somente leitura.

total

Número total de entrada pra carregar/abrir.

```
class kivy.uix.filechooser.FileSystemAbstract
```

Bases: object

Classe para implementação de uma Views de Sistema de Arquivos que pode ser usadas com a [FileChooser](#).

Novo na versão 1.8.0.

getsize(*fn*)

Retorna o tamanho do arquivo em bytes

is_dir(*fn*)

Retorna *True* se o argumento passado por este método é um diretório.

is_hidden(*fn*)

Retorna *True* se o arquivo está oculto.

listdir(*fn*)

Retorna a lista de arquivos no diretório *fn*.

```
class kivy.uix.filechooser.FileSystemLocal
```

Bases: [kivy.uix.filechooser.FileSystemAbstract](#)

Implementação da [FileSystemAbstract](#) para arquivos locais.

Novo na versão 1.8.0.

4.15.18 Float Layout

[FloatLayout](#) homenageia as propriedades [pos_hint](#) e as propriedades dos seus filhos [size_hint](#).



Por exemplo, um `FloatLayout` com uma largura de (300, 300) será criado:

```
layout = FloatLayout(size=(300, 300))
```

Por padrão, todos os Widgets devem ter o `size_hint=(1, 1)`, então, estes botões adotarão o mesmo tamanho do layout:

```
button = Button(text='Hello world')
layout.add_widget(button)
```

Para criar um botão com 50% de largura e 25% da altura do layout em que está contido e posicionado em (20, 20), você precisa fazer:

```
button = Button(
    text='Hello world',
    size_hint=(.5, .25),
    pos=(20, 20))
```

Se desejas criar um botão que sempre seja do tamanho do layout menos 20% em cada lado:

```
button = Button(text='Hello world', size_hint=(.6, .6),
                pos_hint={'x':.2, 'y':.2})
```

Nota: Este layout pode ser utilizado para uma aplicação. Na maior parte do tempo,

você utilizará o tamanho da janela.

Aviso: Se não estiveres utilizando pos_hint, precisarás definir o posicionamento dos seus filhos: se o floatlayout se mover, você deverá mover os filhos contidos também.

`class kivy.uix.floatlayout.FloatLayout(**kwargs)`

Bases: [kivy.uix.layout.Layout](#)

Classe FloatLayout. Veja o módulo da documentação para maiores informações.

4.15.19 Superfície de Gestos

Novo na versão 1.9.0.

Aviso: Isto é experimental e está sujeito a alterações todas as vezes que esta advertência esteja presente.

Veja `kivy/examples/demo/multistroke/main.py` para uma aplicação completa de exemplo.

`class kivy.uix.gesturesurface.GestureSurface(**kwargs)`

Bases: [kivy.uix.floatlayout.FloatLayout](#)

Superfície de gesto simples para rastrear/desenhar movimentos de toque. Normalmente usado para coletar entrada do usuário apropriada [kivy.multistroke.Recognizer](#).

Properties

`temporal_window`Tempo de espera do último evento touch_up antes de tentar reconhecer o gesto. Se você definir isso como 0, o evento `on_gesture_complete` não será disparado a menos que a condição attr: `max_strokes` seja atendida.

`temporal_window` é uma [NumericProperty](#) e o padrão é 2.0.

`max_strokes`Número máximo de traços em um único gesto; se isso for alcançado, o reconhecimento será iniciado imediatamente no evento touch_up final. Se for definido como 0, o evento `on_gesture_complete` não é acionado, a menos que: attr: `temporal_window` expire.

`max_strokes` é uma [NumericProperty](#) e o padrão é 2.0.

`bbox_margin`Margem da caixa de limite para a detecção de colisões de gestos, em pixels.

`bbox_margin` é uma *NumericProperty* e o padrão é 30

`draw_timeout`Número em segundo para manter linha/bbox sobre o Canvas após o evento *on_gesture_complete* ser disparado. Se estiver definido como 0, os gestos são imediatamente removidos da superfície quando completado.

`draw_timeout` é uma *NumericProperty* e o padrão é 3.0.

`color`Cor usada para desenhar o gesto em RGB. Esta opção não tem efeito se `use_random_color` for *True*.

`draw_timeout` é uma *ListProperty* e o padrão é [1, 1, 1] (branco).

`use_random_color`Defina como *True* para escolher uma cor aleatória para cada gesto, se você fizer isso, então `color` será ignorado. O padrão é *False*.

`use_random_color` é uma *BooleanProperty* e o padrão é *False*.

`line_width`Largura da linha utilizada para desenhar toques na superfície. Defina como 0 se você quiser apenas detectar gestos sem desenhar nada. Se você usar 1.0, OpenGL GL_LINE é usado para desenho; Valores > 1 usará um método de desenho interno baseado em triângulos (menos eficiente), veja: mod: *kivy.graphics*.

`line_width` é uma *NumericProperty* e o padrão é 2.

`draw_bbox`Defina como *True* se quiseres desenhar a caixa delimitadora por trás dos gestos. Isso só funciona se `line_width` >= 1. O padrão é *False*.

`draw_bbox` é uma *BooleanProperty* e o padrão é *True*

`bbox_alpha`Opacidade da caixa delimitadora se `draw_bbox` for *True*.

`bbox_alpha` é uma *NumericProperty* e o padrão é 0.1

Events

`on_gesture_start` *GestureContainer*Disparado quando um novo gesto é iniciado sobre a superfície, ou seja, o primeiro ‘`on_touch_down`’ que não colidir com um gesto existente na superfície.

`on_gesture_extend` *GestureContainer*Disparado quando ocorre um evento `touch_down` dentro de um gesto existente.

`on_gesture_merge GestureContainer, GestureContainer`

Disparado quando dois gestos colidem e se fundem num único gesto. O primeiro argumento é o gesto que foi mesclado (não é mais válido); O segundo é o gesto combinado (resultante).

`on_gesture_complete GestureContainer` Disparado quando um conjunto de toques for considerado um gesto completo, isso acontece quando *temporal_window* expira ou *max_strokes* é atingido. Normalmente, irás vincular a este evento e usar o método *getSets()* fornecido *GestureContainer* para coincidir com seu banco de dados gesto.

`on_gesture_cleanup GestureContainer` Disparado *draw_timeout* segundos após *on_gesture_complete*, O gesto será removido da tela (se *line_width > 0* ou *draw_bbox* for *True*) e a lista de gestos internos antes disso.

`on_gesture_discard GestureContainer` Disparado quando um gesto não atender aos requisitos mínimo de tamanho para reconhecimento (*width/height < 5*, ou consiste somente em traços de ponto único).

`find_colliding_gesture(touch)`

Verifica se um toque x/y colide com a caixa delimitadora de um gesto existente. Em caso afirmativo, devolve-o (caso contrário, devolverá *None*)

`get_gesture(touch)`

Retorna *GestureContainer* associado com determinado toque

`init_gesture(touch)`

Criar um novo gesto a partir do toque, ou seja, é o primeiro na superfície, ou não estava perto o suficiente para qualquer gesto existente (ainda)

`merge_gestures(g, other)`

Combina dois gestos juntos, o mais antigo é mantido e o mais novo tem o levantado flag *GestureContainer.was_merged*.

`on_touch_down(touch)`

Quando um novo toque é registrado, a primeira coisa que fazemos é testar se ele colide com a caixa delimitadora de outro gesto conhecido. Caso seja, presume-se ser uma continuação desse gesto.

`on_touch_move(touch)`

Quando um toque se move, adicionamos um ponto à linha na tela para que o caminho seja atualizado. Devemos também verificar se o novo ponto colide com a caixa delimitadora de outro gesto - em caso afirmativo, eles devem ser mesclados.

`class kivy.uix.gesturesurface.GestureContainer(touch, **kwargs)`

Bases: *kivy.event.EventDispatcher*

Objeto de contêiner que armazena informações sobre um gesto. O mesmom pos-sui várias propriedades que são atualizadas por *GestureSurface* a medida que o desenho avança.

Arguments

touchToque no objeto (como recebido por *on_touch_down*) usado para inicializar o contêiner de gestos. Requerido.

Properties

activeDefina como *False* quando o gesto estiver completo (conheça *max_stroke* ou 'GestureSurface.temporal_window')

active é uma *BooleanProperty*

active_strokesNúmero de traços atualmente ativos no gesto, ou seja toques simultâneos associados a este gesto.

active_strokes é uma *NumericProperty*

max_strokesNúmero máximo de traços permitidos no gesto. Isso é definido por *GestureSurface.max_strokes*, mas pode ser substi-tuído, por exemplo, de *on_gesture_start*.

max_strokes é uma *NumericProperty*

was_mergedIndica que este gesto foi mesclado com outro gesto e deve ser considerado como descartado.

was_merged é uma *BooleanProperty*

bboxDicionário com as teclas minx, miny, maxx, maxy. Representa o tamanho da caixa delimitadora da ação.

bbox é uma *DictProperty*

widthRepresenta a largura do gesto.

width é uma *NumericProperty*

heightRepresenta a altura do gesto.

height é uma *NumericProperty*

accept_stroke(*count*=1)

Retorna *True* se este contêiner puder aceitar *count* novos toques

add_stroke(*touch, line*)

Associa uma lista de pontos com um *touch.uid*; A linha propriamente dita é criada pelo caller, mas os eventos subsequentes de move/up olham isso pra gente. Isso é feito para evitar problemas durante a mesclagem.

complete_stroke()

Chamado para retocar eventos para manter o controle de quantos cursos

estão ativos no gesto (só queremos enviar evento quando o *último* golpe no gesto for liberado)

get_vectors(kwargs)**

Retorna traços num formato que seja aceitável para *kivy.multistroke.Recognizer* como um candidato ou modelo de gesto. O resultado é armazenado em cache automaticamente; O cache é invalidado no início e no final de um traço e se *update_bbox* for invocado. Se estiveres indo para analisar um gesto mid-stroke, talvez seja necessário definir o argumento *no_cache* como sendo *True*.

handles(touch)

Retorna *True* se este contêiner manipular um determinado toque

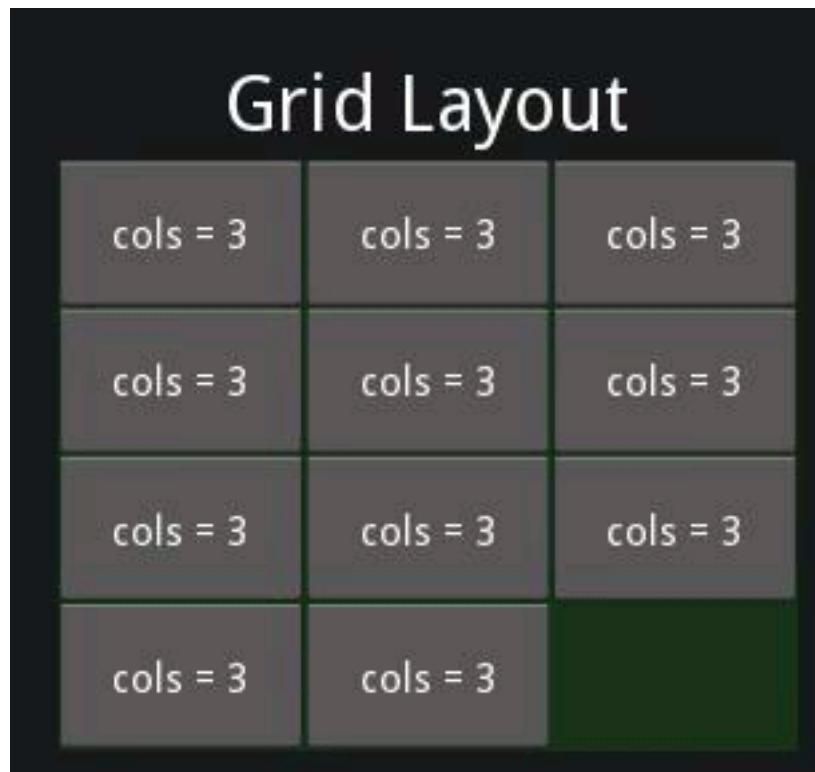
single_points_test()

Retorna *True* se o gesto consistir apenas de traços de ponto único, neste caso, devemos descartá-lo, ou uma exceção será levantada

update_bbox(touch)

Atualiza o gesto bbox a partir de uma coordenada de toque

4.15.20 Grid Layout



Novo na versão 1.0.4.

A classe **GridLayout** organiza seus filhos como sendo uma matriz. Pega-se o espaço disponível e divide-o em linha e colunas, então os Widget são adicionados às células resultantes.

Alterado na versão 1.0.7: A implementação foi alterada para uso o Widget `size_hint` para o cálculo do tamanho das linhas/colunas. `uniform_width` e `uniform_height` foram removidas e outras propriedades foram adicionadas para dar mais controle.

Background

Ao contrário de outros toolkits, você não pode explicitamente colocar um Widget numa determinada linha ou coluna. Cada filho terá sua posição determinada automaticamente pela configuração do layout e o índice do filho na lista de filhos.

Um GridLayout deve sempre ter pelo menos uma restrição de entrada: `GridLayout.cols` ou `GridLayout.rows`. Se você não determinar a quantidade de linhas ou de colunas, o Layout quando criado levantará uma exceção.

Largura da coluna e Altura da linha

A altura/largura da coluna é determinada em 3 passos:

- O tamanho inicial é determinado pelas propriedades `col_default_width` e `row_default_height`. Para personalizar a tamanho de uma única célula ou linha, utilize `cols_minimum` ou `rows_minimum`.
- O `size_hint_x`/`size_hint_y` dos filhos é considerado. Se não estiver definido uma “dica” do tamanho (`size_hint`), o tamanho máximo será usado por todos os filhos.
- É possível forçar o tamanho padrão pela configuração das propriedades `col_force_default` ou `row_force_default`. Isso forçará o layout a ignorar as propriedades `width` e `size_hint` dos filhos e utilizará o tamanho padrão.

Utilizando um GridLayout

No exemplo abaixo, todos os Widgets possuem tamanho igual. Por padrão, o `size_hint` é (1, 1), então, um Widget terá o tamanho total do seu pai:

```
layout = GridLayout(cols=2)
layout.add_widget(Button(text='Hello 1'))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2'))
layout.add_widget(Button(text='World 2'))
```



Agora, vamos corrigir o tamanho do botão Hello para 100px ao invés de utilizar `size_hint_x=1`:

```
layout = GridLayout(cols=2)
layout.add_widget(Button(text='Hello 1', size_hint_x=None,
                        width=100))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2', size_hint_x=None,
                        width=100))
layout.add_widget(Button(text='World 2'))
```



Em seguida, vamos corrigir a altura da linha, especificando o seu tamanho:

```
layout = GridLayout(cols=2, row_force_default=True, row_default_
                    height=40)
layout.add_widget(Button(text='Hello 1', size_hint_x=None,
                        width=100))
layout.add_widget(Button(text='World 1'))
layout.add_widget(Button(text='Hello 2', size_hint_x=None,
                        width=100))
layout.add_widget(Button(text='World 2'))
```

Hello 1	World 1
Hello 2	World 2

```
class kivy.uix.gridlayout.GridLayout(**kwargs)
```

Bases: [kivy.uix.layout.Layout](#)

Classe GridLayout. Veja o módulo da documentação para maiores informações.

col_default_width

O valor padrão mínimo utilizado por cada coluna.

Novo na versão 1.0.7.

col_default_width é um [NumericProperty](#) e o padrão é 0.

col_force_default

Se *True*, será ignorado a largura (width) e o *size_hint_x* do filho e será utilizado o valor padrão da largura da coluna.

Novo na versão 1.0.7.

col_force_default é um [BooleanProperty](#) e o padrão é *False*.

cols

Numero de colunas na grade.

Alterado na versão 1.0.8: Alterado de NumericProperty para BoundedNumericProperty. Não é mais permitido definir valores negativos.

cols é um [NumericProperty](#) e o padrão é 0.

cols_minimum

Dict da largura mínima para cada coluna. A chave do dicionário são os números das colunas, por exemplo, 0, 1, 2...

Novo na versão 1.0.7.

cols_minimum é um [DictProperty](#) e o padrão é {}.

minimum_height

Altura mínima necessária calculada automaticamente para conter todas as crianças.

Novo na versão 1.0.8.

minimum_height é um [NumericProperty](#) e o padrão é 0. Isso é somente leitura.

minimum_size

Tamanho mínimo necessário computado automaticamente para conter todas as crianças.

Novo na versão 1.0.8.

`minimum_size` é uma propriedade *ReferenceListProperty* do (`minimum_width`, `minimum_height`). Isso é somente leitura.

minimum_width

Largura mínima necessária calculada automaticamente para conter todas as crianças.

Novo na versão 1.0.8.

`minimum_width` é um *NumericProperty* e o padrão é 0. Essa propriedade é somente leitura.

padding

Padding entre o layout e seus filhos: [padding_left, padding_top, padding_right, padding_bottom].

O Padding também pode ser definido numa forma que tenha dois argumentos [padding_horizontal, padding_vertical] and a one argument form [padding].

Alterado na versão 1.7.0: Substituído NumericProperty por VariableListProperty.

`padding` é um *VariableListProperty* e o padrão é [0, 0, 0, 0].

row_default_height

Valor mínimo padrão a ser utilizado pela linha.

Novo na versão 1.0.7.

`row_default_height` é um *NumericProperty* e o padrão é 0.

row_force_default

Se True, ignore a altura e o size_hint_y do filho e use a altura padrão da linha.

Novo na versão 1.0.7.

`row_force_default` é um *BooleanProperty* e o padrão é *False*.

rows

Número de linha na grade.

Alterado na versão 1.0.8: Alterado de NumericProperty para BoundedNumericProperty. Não é mais possível definir valores negativos.

`rows` is a *NumericProperty* e o padrão é 0.

rows_minimum

Dict da altura mínima de cada linha. A chave do dicionário é o número da linha, por exemplo, 0, 1, 2...

Novo na versão 1.0.7.

rows_minimum é um *DictProperty* e o padrão é {}.

spacing

Espaçamento entre os filhos: [spacing_horizontal, spacing_vertical].

espaçamento também aceita uma forma de argumento [spacing].

spacing é um *VariableListProperty* e o padrão é [0, 0].

class kivy.uix.gridlayout.GridLayoutException

Bases: exceptions.Exception

Exceção para erros nas situações em que manipulação do GridLayout falhar.

4.15.21 Imagem

A classe :class:`Image` é um widget usado para mostrar uma imagem.

```
wimg = Image(source='mylogo.png')
```

Abertura Assíncrona

Para abrir uma imagem assincronamente (por exemplo, desde um WebServer externo), use a subclasse *AsyncImage*

```
aimg = AsyncImage(source='http://mywebsite.com/logo.png')
```

Isso pode ser útil, pois impede que o aplicativo aguarde até que a imagem seja carregada. Se você quiser exibir imagens grandes ou recuperá-las de URL's, usando *AsyncImage* permitirá que esses recursos sejam recuperados em um thread em background sem bloquear seu aplicativo.

Alinhamento

Por padrão, a imagem é centralizada e se encaixa dentro da caixa delimitadora do widget. Se não quiseres isso, podes definir *allow_stretch* como *True* e *keep_ratio* como *False*.

Também podes herdar de imagem e criar seu próprio estilo. Por exemplo, se desejas que sua imagem seja maior do que o tamanho do seu Widget, podes fazê-lo da seguinte maneira:

```
class FullImage(Image):
    pass
```

E no seu arquivo de linguagem Kivy:

```

<-FullImage>:
    canvas:
        Color:
            rgb: (1, 1, 1)
        Rectangle:
            texture: self.texture
            size: self.width + 20, self.height + 20
            pos: self.x - 10, self.y - 10

```

class kivy.uix.image.Image(kwargs)**

Bases: [kivy.uix.widget.Widget](#)

Classe Image, veja o módulo da documentação para maiores informações.

allow_stretch

Se *True*, o tamanho da imagem normalizada será maximizado para caber na caixa de imagem. Caso contrário, se a caixa for muito alta, a imagem não será esticada mais de 1:1 pixels.

Novo na versão 1.0.7.

allow_stretch é uma [BooleanProperty](#) e o padrão é *False*.

anim_delay

Atrasa a animação se a imagem é sequenciada (como um GIF animado). Se *anim_delay* for definido como *-1*, a animação será interrompida.

Novo na versão 1.0.8.

anim_delay é uma [NumericProperty](#) e o padrão é 0.25 (4 FPS).

anim_loop

Número de loops a serem reproduzidos para então, parar a animação. 0 significa manter animação.

Novo na versão 1.9.0.

anim_loop é uma [NumericProperty](#) e o padrão é 0.

color

Cor da imagem, no forma (r, g, b, a). Este atributo pode ser usado para ‘matizar (tint)’ uma imagem. Tenha cuidado: se a imagem de origem não for gray/white, a cor não funcionará como o esperado.

Novo na versão 1.0.6.

color é um [ListProperty](#) e o padrão é [1, 1, 1, 1].

image_ratio

Relação da imagem (largura/float(altura)).

image_ratio é uma [AliasProperty](#) e é somente leitura.

keep_data

Se True, o _coreimage subjacente armazenará os dados de imagem bruta. Isso é útil quando a detecção de colisões é baseada em pixel.

Novo na versão 1.3.0.

keep_data é uma *BooleanProperty* e o padrão é *False*.

keep_ratio

Se *False* juntamente com *allow_stretch* for *True*, o tamanho da imagem normalizada será maximizado para caber na caixa de imagem e ignorará a relação de aspecto da imagem. Caso contrário, se a caixa for muito alta, a imagem não será esticada mais do que 1:1 pixels.

Novo na versão 1.0.8.

keep_ratio é uma *BooleanProperty* e o padrão é *True*.

mipmap

Indica se desejas que o mipmapping do OpenGL seja aplicado à textura. Leia: [Mipmapping](#) para maiores informações.

Novo na versão 1.0.7.

mipmap é um *BooleanProperty* e o padrão é *False*.

nocache

Se esta propriedade estiver definida como *True*, a imagem não será adicionada ao cache interno. O cache irá simplesmente ignorar todas as chamadas tentando anexar a imagem principal.

Novo na versão 1.6.0.

nocache é uma *BooleanProperty* e o padrão é *False*.

norm_image_size

Tamanho da imagem normalizada dentro da caixa do Widget.

Esse tamanho sempre se ajustará ao tamanho do Widget e preservará a proporção da imagem.

norm_image_size é uma *AliasProperty* e é somente leitura.

reload()

Recarrega a imagem do disco. Isso facilita o re-carregamento de imagens do disco no caso de o conteúdo da imagem tiver sido alterado.

Novo na versão 1.3.0.

Uso:

```
im = Image(source = '1.jpg')
# -- do something --
im.reload()
# image will be re-loaded from disk
```

source

Nome do Arquivo/Fonte da sua Imagem.

source é uma *StringProperty* e o padrão é *None*.

texture

Textura do objeto da imagem. A textura representa a textura da imagem originalmente carregada. É esticado e posicionado durante a renderização de acordo com as propriedades *allow_stretch* and *keep_ratio*.

Dependendo da criação da textura, o valor será *Texture* ou um objeto *TextureRegion*.

texture é um *ObjectProperty* e o padrão é *None*.

texture_size

Tamanho de textura da imagem. Isso representa o tamanho original da textura da imagem carregada.

Aviso: O tamanho da textura é definido após a propriedade de textura. Então, se escutar a alteração em: attr: *texture*, a propriedade *texture_size* não estará atualizada. Use *self.texture.size* ao invés disso.

```
class kivy.uix.image.AsyncImage(**kwargs)
```

Bases: *kivy.uix.image.Image*

Classe AsyncImage. Veja a documentação para maiores informações.

Nota: O *AsyncImage* é uma forma especializada da classe *Image*. Podes querer consultar a documentação *loader* e, em particular, a *ProxyImage* para obter mais detalhes sobre como lidar com eventos através do carregamento de imagens assíncronas.

4.15.22 Rótulo



O Widget **Label** é utilizado para a renderização de texto. O mesmo suporta String (texto) com encoding ASCII ou Unicode.

```
# hello world text
l = Label(text='Hello world')

# unicode text; can only display glyphs that are available in the
# font
l = Label(text=u'Hello world ' + unichr(2764))

# multiline text
l = Label(text='Multi\nLine')

# size
l = Label(text='Hello world', font_size='20sp')
```

String e Conteúdo texto

Por padrão, o tamanho do **Label** não é afetado pelo conteúdo **text** e o texto não é afetado pelo tamanho. Para controlar o dimensionamento, você deve definir a propriedade **text_size** para controlar o texto e/ou o vinculo **size** para **texture_size** para crescer junto com o texto.

Por exemplo, o tamanho deste Label será definido para ser o tamanho do seu conteúdo (Veja mais em **padding**):

```
Label:
    size: self.texture_size
```

O texto deste Label será quebrado no ponto especificado pela sua largura e será aumentado em sua altura.

```
Label:
    text_size: cm(6), cm(4)
```

Nota: Os atributos **shorten** e **max_lines** controlam como o texto que transbordará será controlado.

Combine estes conceitos para criar um Label que pode crescer verticalmente mas que o seu texto será quebrado numa determinada largura:

```
Label:
    text_size: root.width, None
    size: self.texture_size
```

Alinhamento do Texto e seu Envólucro

A classe `Label` tem as propriedades `halign` e `valign` para controlar o alinhamento do seu texto. De qualquer forma, por padrão, a imagem do texto (`texture`) será somente da largura necessária para conter os caracteres e será posicionado no centro do Label. A propriedade `valign` não terá efeito e `halign` terá um efeito se o seu texto tiver novas linhas; uma simples linha de texto aparecerá centralizada mesmo que `halign` tenha sido setado para a esquerda (por padrão).

Para que as propriedades de alinhamento tenham efeito, defina o `text_size` que determina o tamanho das dimensões do retângulo do Label com que o texto está alinhado. Por exemplo, o código a seguir vincula esse tamanho ao tamanho do Label, então, o texto será alinhado dentro dos limites do Widget. Isso também envolverá, automaticamente o Label para que o mesmo permaneça nesta área.

`Label`:

```
text_size: self.size
halign: 'right'
valign: 'middle'
```

Marcação de Texto

Novo na versão 1.1.0.

Você pode mudar o estilo do texto usando a propriedade `Marcação de Texto`. A sintaxe é similar a sintaxe do BBCode, mas somente o estilo inline (de uma única linha) é permitido:

```
# hello world with world in bold
l = Label(text='Hello [b]World[/b]', markup=True)

# hello in red, world in blue
l = Label(text='[color=ff3333>Hello[/color][color=3333ff]World[/
    color]',
    markup = True)
```

Se você precisa escapar a marcação do texto atual, utilize `kivy.utils.escape_markup()`:

```
text = 'This is an important message [1]'
l = Label(text='[b]' + escape_markup(text) + '[/b]', markup=True)
```

As TAGs a seguir estão disponíveis:

[b] **[/b]** Define o texto em negrito

[i] **[/i]** Define o texto em itálico

[u] **[/u]** Define o texto como sublinhado

[s][/s] Define o texto como riscado

[font=<str>][/font] Altera a fonte do texto

[size=<integer>][/size] Altera o tamanho da fonte

[color=#<color>][/color] Altera a cor do texto

[ref=<str>][/ref] Adiciona zonas interativas. A referência + adiciona uma caixa delimitador dentro da referência que será disponibilizada em *Label.refs*

[anchor=<str>] Coloca um âncora no texto. Você pode pegar a posição da sua âncora dentro do texto com *Label.anchors*

[sub][/sub] Exibir o texto numa posição de subíndice em relação ao texto principal.

[sup][/sup] Exibir o texto numa posição acima em relação ao texto principal.

Se você deseja renderizar a marcação de texto com um [ou] ou o caractere &, você precisará escapa-los. Nós criamos uma simples sintaxe:

```
[ -> &bl;  
] -> &br;  
& -> &amp;
```

Então, você pode escrever:

```
"[size=24>Hello &bl;World&bt;[/size]"
```

Zona interativa no texto

Novo na versão 1.1.0.

Você pode ter agora definições de “links” usando a marcação de texto. A ideia é poder detectar quando o usuário clica numa parte do texto e reagir a este clique. A tag **[ref=xxx]** é usada para isso.

Neste exemplo, nós criamos a referência sobre a palavra “World”. Quando esta palavra for clicada, a função `print_it` será chamada com o nome da referência:

```
def print_it(instance, value):  
    print('User clicked on', value)  
widget = Label(text='Hello [ref=world]World[/ref]', markup=True)  
widget.bind(on_ref_press=print_it)
```

Para uma renderização mais bonita, você pode adicionar um cor como referência. Substitua o texto `text=` no exemplo anterior por:

```
'Hello [ref=world][color=0000ff]World[/color][/ref]'
```

Catering para linguagens Unicode

A fonte Kivy utilizada não contém todos os caracteres requeridos para a exibição de todas as línguas. Quando você usar o Widget com as configurações padrão, isso resulta em um bloco sendo desenhado onde você espera um caractere.

Se você deseja mostrar um determinado caractere, você pode escolher uma fonte que suporte-o e disponibiliza-la universalmente via kv:

```
<Label>:  
    font_name: '/<path>/<to>/<font>'
```

Observe que isso precisa ser feito antes que seus Widgets sejam carregados, pois as regras kv são aplicadas somente no momento do carregamento.

Exemplo de uso

O exemplo a seguir marca as âncoras e as referências contidas em um rótulo:

```
from kivy.app import App  
from kivy.uix.label import Label  
from kivy.clock import Clock  
from kivy.graphics import Color, Rectangle  
  
class TestApp(App):  
  
    @staticmethod  
    def get_x(label, ref_x):  
        """ Return the x value of the ref/anchor relative to the  
        canvas """  
        return label.center_x - label.texture_size[0] * 0.5 + ref_x  
  
    @staticmethod  
    def get_y(label, ref_y):  
        """ Return the y value of the ref/anchor relative to the  
        canvas """  
        # Note the inversion of direction, as y values start at the  
        # top of  
        # the texture and increase downwards  
        return label.center_y + label.texture_size[1] * 0.5 - ref_y  
  
    def show_marks(self, label):  
  
        # Indicate the position of the anchors with a red top marker  
        for name, anc in label.anchors.items():  
            with label.canvas:  
                Color(1, 0, 0)
```

```

        Rectangle(pos=(self.get_x(label, anc[0]),
                      self.get_y(label, anc[1])),
                   size=(3, 3))

    # Draw a green surround around the refs. Note the sizes y_
    ↵inversion
    for name, boxes in label.refs.items():
        for box in boxes:
            with label.canvas:
                Color(0, 1, 0, 0.25)
                Rectangle(pos=(self.get_x(label, box[0]),
                              self.get_y(label, box[1])),
                           size=(box[2] - box[0],
                                 box[1] - box[3]))


def build(self):
    label = Label(
        text='[anchor=a]a\nChars [anchor=b]b\n[ref=myref]ref[/
    ↵ref]',
        markup=True)
    Clock.schedule_once(lambda dt: self.show_marks(label), 1)
    return label

TestApp().run()

```

class kivy.uix.label.Label(kwargs)**
Bases: [kivy.uix.widget.Widget](#)

Classe Label, veja o módulo da documentação para maiores informações.

Events

on_ref_press Disparado quando o usuário clica na palavra referenciada com a tag `[ref]` tag na marcação de texto.

anchors

Novo na versão 1.1.0.

Posição de toda a marcação `[anchor = xxx]` no texto. Essas coordenadas são relativas ao canto superior esquerdo do texto, com o valor Y sendo decrementado. Os nomes de âncoras devem ser exclusivos e somente a primeira ocorrência de qualquer âncora duplicada será registrada.

Você pode colocar âncoras em seu texto de marcação como o exemplo a seguir:

```

text = """
    [anchor=title1][size=24]This is my Big title.[/size]
    [anchor=content]Hello world
"""

```

Assim, todas as referências a `[anchor=]` serão removidas e você terá todas as posições de âncoras nesta propriedade (somente depois de renderizado):

```
>>> widget = Label(text=text, markup=True)
>>> widget.texture_update()
>>> widget.anchors
{"content": (20, 32), "title1": (20, 16)}
```

Nota: Isso funciona apenas com texto utilizando marcação. Você precisa `markup` definir como sendo igual a `True`.

bold

Indica o uso da versão em negrito de sua fonte.

Nota: Dependendo da fonte, o atributo negrito pode não ter impacto na renderização do texto.

`bold` é um `BooleanProperty` e o padrão é `False`.

color

Cor do texto no formato (r, g, b, a).

`color` é um `ListProperty` e o padrão é [1, 1, 1, 1].

disabled_color

A cor do texto quando o Widget está desativado, no formato (r, g, b, a).

Novo na versão 1.8.0.

`disabled_color` é um `ListProperty` e o padrão é [1, 1, 1, .3].

disabled_outline_color

A cor do contorno do texto quando o Widget está desabilitado, no formato (r, g, b).

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

`disabled_outline_color` é um `ListProperty` e o padrão é [0, 0, 0].

ellipsis_options

Opções de fonte para a seqüência de reticências ('...') usada para dividir o texto.

Aceita um dict como nome de opção com o valor. Aplicado somente quando:`attr:markup` é verdadeiro e o texto for encurtado. Todas as opções de fontes que funcionam para `Label` funcionarão para `ellipsis_options`.

Os padrões para as opções não especificadas são retirados do texto adjacente.

Label:

```
text: 'Some very long line which will be cut'  
markup: True  
shorten: True  
ellipsis_options: {'color':(1,0.5,0.5,1), 'underline'  
→ ':True}'
```

Novo na versão 2.0.0.

ellipsis_options é um *DictProperty* e o padrão é {} (um dicionário vazio dict).

font_blended

Caso a renderização da fonte mista ou sólida deva ser usada.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

font_blended é um *BooleanProperty* e o padrão é *True*.

font_hinting

Que opção de hinting para usar na renderização de fonte. Pode ser um das seguintes opções: `'normal'`, `'light'`, `'mono'` ou `None`.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

font_hinting é um *OptionProperty* e o padrão é `'normal'`.

font_kerning

Caso o kerning está habilitado para renderização de fonte.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

font_kerning é um *BooleanProperty* e o padrão é *True*.

font_name

Nome do arquivo de fonte utilizado. O path pode ser absoluto ou relativo. Path relativo são resolvidos pela função *resource_find()*.

Aviso: Dependendo do seu provedor de texto, o arquivo de fonte pode ser ignorado. De qualquer forma, você pode utilizar isso normalmente sem maiores problemas.

Se a fonte usada não tiver os glifos para a linguagem/símbolos específicos que você está usando, verás caracteres de caixa em branco '[]' em vez dos caracteres reais. A solução é usar uma fonte que tenha os caracteres que precisas exibir. Por exemplo, para exibir CP, use uma fonte como freesans.ttf que tenha esses caracteres.

font_name is a *StringProperty* and defaults to 'Roboto'. This value is taken from *Config*.

font_size

Tamanho da fonte do texto em Pixels.

font_size é um *NumericProperty* e o padrão é 15sp.

halign

Alinhamento horizontal do texto

halign é um *OptionProperty* e o padrão é 'left'. Opções disponíveis são: left, center, right and justify.

Aviso: Isso não altera a posição da textura do texto do Label (centralizado), apenas a posição do texto nessa textura. Provavelmente irás querer vincular o tamanho do Label ao attr:*texture_size* ou definir o *text_size*.

Alterado na versão 1.6.0: Uma nova opção foi adicionada a *halign*, namely *justify*.

is_shortened

This property indicates if *text* was rendered with or without shortening when *shorten* is True.

Novo na versão 1.10.0.

is_shortened is a *BooleanProperty* and defaults to False.

italic

Indica o uso da versão em itálico da fonte.

Nota: Dependendo da fonte, o atributo itálico pode não ter nenhum impacto na renderização do texto.

italic é um *BooleanProperty* e o padrão é False.

line_height

Altura da linha do texto. Por exemplo, `line_height = 2` fará com que o espaçamento entre as linhas seja o dobro do tamanho.

`line_height` é uma [NumericProperty](#) e o parão é `1.0`.

Novo na versão 1.5.0.

markup

Novo na versão 1.1.0.

Se `True`, o texto será renderizado usando [MarkupLabel](#): você pode alterar o estilo do texto usando tags. Verifique a documentação [Marcação de Texto](#) para obter mais informações.

`markup` é um [BooleanProperty](#) e o padrão é `False`.

max_lines

Número máximo de linhas que pode ser utilizado, o padrão é `0`, o que significa ilimitado. Por favor, note que `shorten` assume esta propriedade. (Com abreviação, o texto é sempre uma linha.)

Novo na versão 1.8.0.

`max_lines` é um [NumericProperty](#) e o padrão é `0`.

mipmap

Indica se o mipmapping do OpenGL é aplicado à textura ou não. Leia [Mipmapping](#) para mais informações.

Novo na versão 1.0.7.

`mipmap` é um [BooleanProperty](#) e o padrão é `False`.

outline_color

A cor da borda do texto no formato (r, g, b).

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

`outline_color` é um [ListProperty](#) e o padrão é `[0, 0, 0]`.

outline_width

Largura em pixels para o contorno em torno do texto. Nenhum esboço será processado se o valor for `None`.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

`outline_width` é um [NumericProperty](#) e o padrão é `None`.

padding

Padding do texto no formato (padding_x, padding_y)

padding é uma propriedade *ReferenceListProperty* de (*padding_x*, *padding_y*).

padding_x

Padding horizontal do texto dentro da caixa do Widget.

padding_x é um *NumericProperty* e o padrão é 0.

Alterado na versão 1.9.0: *padding_x* foi corrigido para funcionar como esperado. No passado, o texto era preenchido por seus valores negativos.

padding_y

Vertical padding do texto dentro da caixa do Widget.

padding_y é um *NumericProperty* e o padrão é 0.

Alterado na versão 1.9.0: *padding_y* foi corrigido para funcionar como esperado. No passado, o texto era preenchido por seus valores negativos.

refs

Novo na versão 1.1.0.

Lista de itens de marcação [ref=xxx] no texto com a caixa delimitadora de todas as palavras contidas em uma referência, disponível somente após a renderização.

Por exemplo, se você escrever:

```
Check out my [ref=hello]link[/ref]
```

A referência será definida como:

```
{'hello': ((64, 0, 78, 16), )}
```

As referências marcadas “hello” possuem uma caixa delimitadora em (x1, y1, x2, y2). Essas coordenadas são relativas ao canto superior esquerdo do texto, com o valor y aumentando para baixo. Podes definir várias refs com o mesmo nome: cada ocorrência será adicionada como outra (x1, y1, x2, y2) tupla a esta lista.

A atual implementação do Label usa essas referências se elas existirem em seu texto de marcação, fazendo automaticamente a colisão com o toque e despachando um evento *on_ref_press*.

Você pode vincular um evento ref como este:

```
def print_it(instance, value):
    print('User click on', value)
```

```
widget = Label(text='Hello [ref=world]World[/ref]',  
             markup=True)  
widget.on_ref_press(print_it)
```

Nota: Isso funciona apenas com texto utilizando marcação. Você precisa *markup* definir como sendo igual a *True*.

shorten

Indica se o Label deve tentar encurtar seu conteúdo textual tanto quanto possível se o *text_size* estiver definido. Definir isto como True sem um conjunto apropriado *text_size* levará a resultados inesperados.

shorten_from e *split_str* controlam a direção a partir da qual o *text* é dividido, bem como onde *text* somos permitidos a dividir.

shorten é um *BooleanProperty* e o padrão é 'False'.

shorten_from

O lado do qual deveríamos encurtar o texto, pode ser do lado esquerdo, direito ou central.

Por exemplo, se deixado, as reticências aparecerão para no lado esquerdo e nós exibiremos o texto começando o mais direita possível. Semelhante a *shorten*, esta opção só se aplica quando *text_size`[0]* não for `None, neste caso, a String é encurtada para caber dentro da largura especificada.

Novo na versão 1.9.0.

shorten_from é um *OptionProperty* e o padrão é *center*.

split_str

A String usada para dividir o *text* enquanto a String é encutarda quando a *shorten* for *True*.

Por exemplo, se for um espaço, a seqüência será quebrada em palavras e como muitas palavras inteiras que podem caber em uma única linha será exibida. Se *split_str* for uma String vazia, "", dividimos em cada caractere ajustando tanto texto na linha o quanto possível .

Novo na versão 1.9.0.

split_str é uma *StringProperty* e o padrão é "" (uma String vazia).

strikethrough

Adiciona uma linha tachada ao texto.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

strikethrough é um *BooleanProperty* e o padrão é *False*.

strip

Se os espaços à esquerda e à direita e as novas linhas deverão ser removidas de cada linha que for exibida. Se *True*, cada linha começará na borda direita ou esquerda, dependendo de *halign*. Se *halign* for *justify* o mesmo será implicitamente *True*.

Novo na versão 1.9.0.

strip é um *BooleanProperty* e o padrão é *False*.

text

Texto do Label.

Criação de um simples Hello World:

```
widget = Label(text='Hello world')
```

Se você deseja criar o Widget com uma String Unicode, use:

```
widget = Label(text=u'My unicode string')
```

text é um *StringProperty* e o padrão é ''.

text_size

Por padrão, o Label não é limitado por qualquer caixa delimitadora. Você pode definir a restrição de tamanho do Label com esta propriedade. O texto será autofocus para as restrições. Assim, embora o tamanho da fonte não seja reduzido, o texto será organizado para caber na caixa da melhor forma possível, com qualquer texto ainda fora da caixa cortada.

Isso define e clips *texture_size* para *text_size* caso não seja *None*.

Novo na versão 1.0.4.

Por exemplo, qualquer que seja o tamanho do Widget atual, se quiseres que o Label seja criado numa caixa com largura = 200 e altura ilimitada:

```
Label(text='Very big big line', text_size=(200, None))
```

Nota: Esta propriedade *text_size* é igual à propriedade *usersize* da classe *Label*. (Isso é nomeado *size* = no construtor.)

text_size é uma *ListProperty* e o padrão é (*None*, *None*), o que significa que não há, por padrão, restrição de tamanho.

texture

Objeto de textura do texto. O texto é processado automaticamente quando

uma propriedade for alterada. A textura OpenGL criada nesta operação é armazenada nesta propriedade. Podes usar esta `texture` para quaisquer elementos gráficos.

Dependendo da criação da textura, o valor será `Texture` ou objeto class:`~kivy.graphics.texture.TextureRegion`.

Aviso: A atualização da `texture` está agendada para o próximo Frame. Se precisares da textura imediatamente após alterar uma propriedade, deverás invocar a função `texture_update()` antes de acessar a `texture`:

```
l = Label(text='Hello world')
# l.texture is good
l.font_size = '50sp'
# l.texture is not updated yet
l.texture_update()
# l.texture is good now.
```

`texture` é um *ObjectProperty* e o padrão é `None`.

texture_size

Tamanho da textura do texto. O tamanho é determinado pelo tamanho da fonte e pelo texto. Se `text_size` for `[None, None]`, a textura será do tamanho necessário para ajustar o texto, caso contrário, será cortado para caber em `text_size`.

Quando `text_size` for `[None, None]`, pode-se vincular a `texture_size` e redimensioná-la proporcionalmente para ajustar o tamanho do Label, a fim de fazer o texto preencher todo o Label.

Aviso: A `texture_size` é ajustado após a propriedade `texture`. Se escutares as `texture`, `texture_size` não estará atualizado em seu callback. Ao invés, vincule ao `texture_size`.

texture_update(*largs)

Força a recriação da textura com as propriedades atuais do Label.

Depois que essa função é chamada, o `texture` e `texture_size` serão atualizados nessa ordem.

underline

Adiciona um underline ao texto.

Nota: Esse recurso requer o provedor de texto SDL2.

Novo na versão 1.10.0.

`underline` é um *BooleanProperty* e o padrão é *False*.

unicode_errors

Como lidar com erros de decodificação Unicode. Pode ser '*strict*', '*replace*' ou '*ignore*'.

Novo na versão 1.9.0.

`unicode_errors` é um *OptionProperty* e o padrão é '*replace*'.

valign

Alinhamento vertical do texto.

`valign` é um *OptionProperty* e o padrão é '*bottom*'. Opções disponíveis são: '*bottom*', '*middle*' (ou '*center*') e '*top*'.

Alterado na versão 1.10.0: A opção '*center*' foi adicionada como um alias de '*middle*'.

Aviso: Isso não altera a posição da textura do texto do Label (centralizado), apenas a posição do texto dentro desta textura. Provavelmente desejarás vincular o tamanho do Label ao attr:`texture_size` ou definir a :attr:'`text_size`' para alterar esse comportamento.

4.15.23 Leiaute

Leiautes são utilizados para calcular e atribuir/definir posições aos Widgets

A classe `Layout` não pode ser utilizadas diretamente. Utilize umas das seguintes classes de layout:

- Anchor layout: `kivy.uix.anchorlayout.AnchorLayout`
- Box layout: `kivy.uix.boxlayout.BoxLayout`
- Float layout: `kivy.uix.floatlayout.FloatLayout`
- Grid layout: `kivy.uix.gridlayout.GridLayout`
- Page Layout: `kivy.uix.pagelayout.PageLayout`
- Relative layout: `kivy.uix.relativelayoutRelativeLayout`
- Scatter layout: `kivy.uix.scatterlayout.ScatterLayout`
- Stack layout: `kivy.uix.stacklayout.StackLayout`

Entendendo a propriedade `size_hint` do `Widget`

A propriedade :attr:`~kivy.uix.Widget.size_hint` é uma tupla de valores utilizada pelos gerenciadores de layout para definir o tamanho dos seus filhos. Isso indica o tamanho relativo dos layout's ao invés de determinar o tamanho absoluto (em pixel/cm/etc). O formato é:

```
widget.size_hint = (width_percent, height_percent)
```

O percentual é especificado como um número de ponto flutuante no intervalo de 0-1. Por exemplo, 0.5 é 50%, 1 é igual a 100%.

Se você desejar que a largura de `Widget` seja a metade da largura do seu pai e a altura seja a mesma altura dos seus pais, faça da seguinte maneira:

```
widget.size_hint = (0.5, 1.0)
```

Se você não deseja usar a propriedade `size_hint` para determinar a largura ou a altura, defina o valor igual a `None`. Por exemplo, para fazer um `Widget` ter 250px de largura e 30% da altura do seu pai, faça:

```
widget.size_hint = (None, 0.3)  
widget.width = 250
```

Sendo *Kivy properties*, essas propriedades também podem ser definidas como argumentos do construtor:

```
widget = Widget(size_hint=(None, 0.3), width=250)
```

Alterado na versão 1.4.1: O método interno `reposition_child` (definido como público erroneamente) foi removido.

class kivy.uix.layout.Layout(kwargs)**
Bases: *kivy.uix.widget.Widget*

Interface da classe Layout utilizado para implementar cada layout. Veja o módulo da documentação para maiores informações.

do_layout(*largs)

Essa função é invocada quando um layout é chamado por um gatilho. Caso seja necessário escrever uma subclasse de Layout, não invoque essa função diretamente, ao invés disso, utilize `_trigger_layout()`.

A função é por padrão chamada *antes* do próximo frame, portanto o layout não é atualizado imediatamente. Tudo dependerá do posicionamento, por exemplo, os filhos serão agendados para o próximo frame.

Novo na versão 1.0.8.

layout_hint_with_bounds(sh_sum, available_space, min_bounded_size, sh_min_vals, sh_max_vals, hint)

(Interno) Calcula a sugestão apropriada (de tamanho) para todos os widgets dados, nos limites (potenciais) mínimos ou máximos no tamanho dos widgets. A lista `hint` é atualizada com tamanhos apropriados.

Ele percorre os hints e para qualquer widgets cujo hint resultará na violação das restrições mínimas ou máximas, isso corrigirá o hint. Qualquer espaço remanescente ou ausente depois de todos os Widgets são corrigidos distribuído para os widgets, tornando-os menores ou maiores de acordo com o seu tamanho hint.

Este algoritmos não sabe nada de diferente sobre os Widgets além do que é passado através dos parâmetros de entrada, por isso que é tão genérico, para colocar as coisas de acordo com restrições usando as dicas de tamanho.

Parameters

`sh_sum: float`A soma do tamanho dos hints (basicamente `sum(size_hint)`).

`available_space: float`Quantidade de pixel disponível para todos os Widgets cujo size hint seja diferente de `None`. Não pode ser zero.

`min_bounded_size: float`Quantidade mínima de espaço requerido de acordo com o `size_hint_min` dos Widgets (basicamente `sum(size_hint_min)`).

`sh_min_vals: list or iterable`Os itens no iterável são o `size_hint_min` para cada Widget. Pode ser `None`. O comprimento pode ser o mesmo que `hint`.

`sh_max_vals: list or iterable`Itens no iterável são o `size_hint_max` para cada Widget. Pode ser `None`. O comprimento deve ser o mesmo do `hint`.

`hint: list`Uma lista cujo tamanho é o mesmo que o comprimento do `sh_min_vals` e `sh_max_vals` cujo elemento é o valor do tamanho do hint correspondente para cada elemento. Esta lista é atualizada no local com o size hint correto assegurando que as restrições não serão violadas.

RetornaNada. `hint` é atualizado no lugar.

4.15.24 List View

Novo na versão 1.5.

Nota: ListView entrou em desuso, ao invés de utiliza-lo use [RecycleView](#)

Aviso: Este código ainda é experimental e essa API está sujeita a mudança em versões futuras.

A classe ‘kivy.uix.listview.ListView’ implementa uma classe ‘kivy.uix.abstractview.AbstractView’ como uma lista vertical, rolável, apontável, cortada na caixa delimitadora do scrollview e contém instâncias de exibição do item de lista.

A AbstractView tem uma propriedade: *adapter*. O adaptador pode ser um dos seguintes: a *SimpleListAdapter*, a *ListAdapter* ou a *DictAdapter*. A Adapter pode fazer uso de *args_converters* para preparar os dados para passar para o construtor de cada instanciação de exibição de item.

Para uma visão geral de como todos esses componentes se encaixam, consulte a documentação do módulo *adapters*.

Prefácio

Listas são partes centrais de muitos projetos de software. A abordagem de Kivy para listas incluem o fornecimento de soluções para listas simples, juntamente com uma estrutura substancial para construir listas de complexidade moderada a avançada. Para um novo usuário, pode ser difícil passar de simples para avançado. Por esta razão, Kivy fornece um extenso conjunto de exemplos (com o pacote Kivy) que você pode querer executar primeiro, para obter um sabor da gama de funcionalidades oferecidas.

- [kivy/examples/widgets/lists/list_simple.py](#)
- [kivy/examples/widgets/lists/list_simple_in_kv.py](#)
- [kivy/examples/widgets/lists/list_simple_in_kv_2.py](#)
- [kivy/examples/widgets/lists/list_master_detail.py](#)
- [kivy/examples/widgets/lists/list_master_detail.py](#)
- [kivy/examples/widgets/lists/list_kv.py](#)
- [kivy/examples/widgets/lists/list_composite.py](#)
- [kivy/examples/widgets/lists/list_reset_data.py](#)
- [kivy/examples/widgets/lists/list_cascade.py](#)
- [kivy/examples/widgets/lists/list_cascade_dict.py](#)
- [kivy/examples/widgets/lists/list_cascade_images.py](#)
- [kivy/examples/widgets/lists/list_ops.py](#)

Many of the examples feature selection, some restricting selection to single selection, where only one item at a time can be selected, and others allowing multiple item selection. Many of the examples illustrate how selection in one list can be connected to actions and selections in another view or another list.

Encontre sua própria maneira de ler a documentação aqui, examinando o código-fonte para os aplicativos de exemplos e executando os exemplos. Alguns podem preferir ler a documentação com completamente primeiro, outros podem querer executar os exemplos e ver seu código. Não importa o que você faça, ir para frente e para trás provavelmente será necessário.

Exemplo Básico

De forma simples, fazemos uma listview com 100 itens.

```
from kivy.uix.listview import ListView
from kivy.base import runTouchApp

class MainView(ListView):
    def __init__(self, **kwargs):
        super(MainView, self).__init__(
            item_strings=[str(index) for index in range(100)])

if __name__ == '__main__':
    runTouchApp(MainView())
```

Ou, podemos declarar a listview usando a linguagem kivy.

```
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder
from kivy.base import runTouchApp

Builder.load_string("""
<MyListView>:
    ListView:
        item_strings: [str(index) for index in range(100)]
""")

class MyListView(BoxLayout):
    pass

if __name__ == '__main__':
    runTouchApp(MyListView())
```

Usando um Adaptador

Nos bastidores, o exemplo básico acima usa a classe 'kivy.adapters.simplelistadapter.SimpleListAdapter'. Quando o construtor para a classe 'kivy.uix.listview.ListView' vê que somente uma lista de strings é fornecida como um argumento (chamado item_strings), ele cria uma classe 'kivy.adapters.simplelistadapter.SimpleListAdapter' usando a lista de strings.

“Simple” - “Simples” na: classe: *~kivy.adapters.simplelistadapter.SimpleListAdapter* significa *sem suporte de seleção*. É uma listagem rolável de itens que não responde a eventos de toque.

To use a **SimpleListAdapter** explicitly when creating a ListView instance, do:

```
simple_list_adapter = SimpleListAdapter(  
    data=["Item #{0}".format(i) for i in range(100)],  
    cls=Label)  
  
list_view = ListView(adapter=simple_list_adapter)
```

A instância da: classe: *~kivy.adapters.simplelistadapter.SimpleListAdapter* possui um argumento de dados obrigatório que contém itens de dados a serem usados para instanciar a: classe exibida: *~kivy.uix.label.Label* para a lista de exibições (observe o argumento *cls = Label*). Os dados dos itens são strings. Cada item da string é definido pela: classe: *~kivy.adapters.simplelistadapter.SimpleListAdapter* como o argumento *text* para cada instanciação de Label.

Você pode declarar uma ListView com um adaptador em um arquivo kv com atenção especial dada à maneira como os blocos de python mais longos são indentados:

```
from kivy.uix.boxlayout import BoxLayout  
from kivy.base import runTouchApp  
from kivy.lang import Builder  
  
# Note the special nature of indentation in the adapter declaration,  
# where  
# the adapter: is on one line, then the value side must be given at  
# one  
# level of indentation.  
  
Builder.load_string("""  
#:import label kivy.uix.label  
#:import sla kivy.adapters.simplelistadapter  
  
<MyListView>:  
    ListView:  
        adapter:  
            sla.SimpleListAdapter(  
                data=["Item #{0}".format(i) for i in range(100)],
```

```

        cls=label.Label)
""")

class MyListView(BoxLayout):
    pass

if __name__ == '__main__':
    runTouchApp(MyListView())

```

ListAdapter e DictAdapter

Para a maioria dos casos de uso, seus dados são mais complexos do que uma simples lista de strings. A funcionalidade de seleção também é frequentemente necessária. As classes: `~kivy.adapters.listadapterListAdapter` e: classe: `~kivy.adapters.dictadapter.DictAdapter` cobrem essas necessidades mais elaboradas.

A classe: `~kivy.adapters.listadapterListAdapter` é a classe base para: classe: `~kivy.adapters.dictadapter.DictAdapter`, então podemos começar com esta.

Consulte os documentos: classe: `~kivy.adapters.listadapterListAdapter` para detalhes, mas aqui está uma sinopse de seus argumentos:

- ***data***: strings, class instances, dicts, etc. that form the base data for instantiating views.
- ***cls***: a Kivy view that is to be instantiated for each list item. There are several built-in types available, including ListItemLabel and ListItemButton, or you can make your own class that mixes in the required `SelectableView`.
- ***template***: the name of a Kivy language (kv) template that defines the Kivy view for each list item.

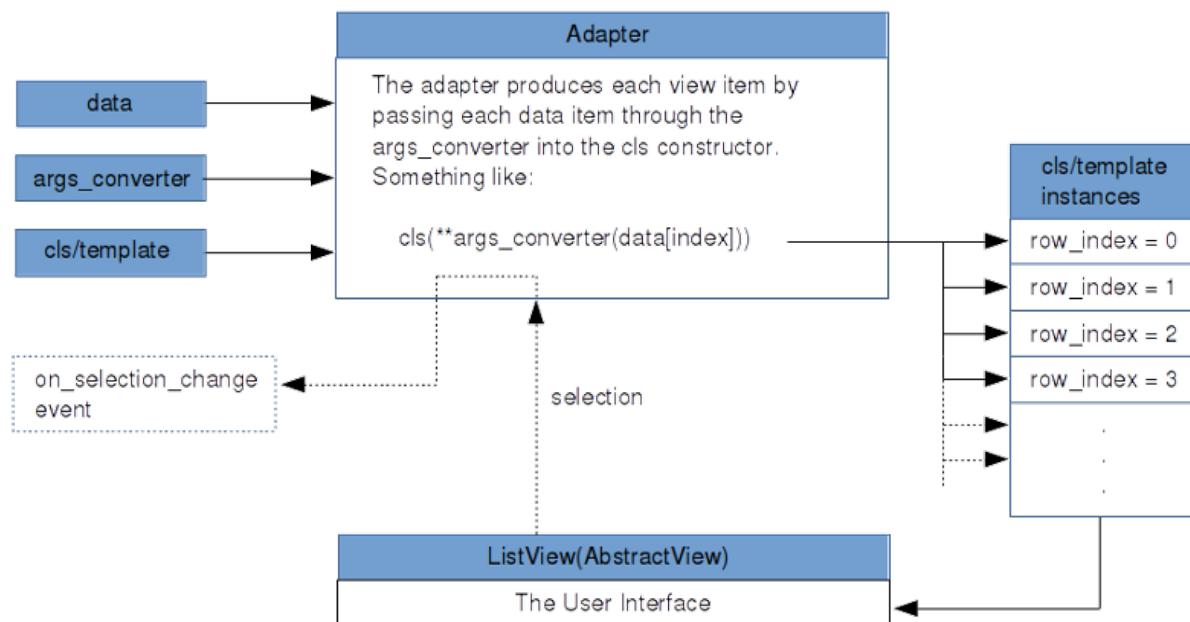
Nota: Escolha apenas um, cls ou modelo, para fornecer como um argumento.

- ***args_converters***: a function that takes a data item object as input and uses it to build and return an args dict, ready to be used in a call to instantiate item views using the item view cls or template. In the case of cls, the args dict becomes a kwargs constructor argument. For a template, it is treated as a context (ctx) but is essentially similar in form to the kwargs usage.
- ***selection_mode***: a string with the value 'single', 'multiple' or other.
- ***allow_empty_selection***: a boolean, which if False (the default), forces there to always be a selection if there is data available. If True, selection happens only as a result of user action.

Na narrativa, podemos resumir o seguinte:

O adaptador de um listview toma itens de dados e usa uma função `args_converter` para transformá-los em argumentos para criar instâncias de exibição de item de lista, usando um `cls` ou um modelo `kv`.

Em um gráfico, um resumo da relação entre uma listview e seus componentes pode ser resumido da seguinte forma:



Consulte a documentação para maiores informações [adapters](#).

Uma classe: `~kivy.adapters.dictadapter.DictAdapter` tem os mesmos argumentos e requisitos como a classe: `~kivy.adapters.listadapterListAdapter` exceto por duas coisas:

1. Existe um argumento adicional, `sorted_keys`, que deve atender aos requisitos das chaves normais do dicionário python.
2. O argumento de dados é, como seria de esperar, um dicionário. As chaves no dicionário devem incluir as chaves no argumento `sorted_keys`, mas podem formar um superconjunto de chaves em `sorted_keys`. Os valores podem ser strings, instâncias de classe, dicionários, etc. (O `args_converter` usa-o de acordo).

Usando um “Args Converter”

Uma classe: `~kivy.uix.listview.ListView` permite o uso de visualizações de item de lista embutidas, como: classe: `~kivy.uix.listview.ListItemButton`, sua própria classe de exibição de item personalizada ou um modelo `kv` personalizado. Seja qual for o tipo de exibição do item da lista, a função: doc: `args_converter <api-kivy.adapters.args_converters>` é necessária para preparar, por item de dados da lista, `kwargs` para o `cls` ou o `ctx` para o modelo.

Nota: Somente `ListItemLabel`, `ListItemButton` ou classes personalizadas como elas (e

não as simples classes Label ou Button) devem ser usadas no sistema listview.

Aviso: ListItemButton herda as propriedades `background_normal` e `background_down` do widget Button, de modo que `selected_color` e `deselected_color` não são representadas fielmente por padrão.

Aqui está um args_converter para uso com a class interna: `~kivy.uix.listview.ListItemButton` especificado como uma função normal do Python:

```
def args_converter(row_index, an_obj):
    return {'text': an_obj.text,
            'size_hint_y': None,
            'height': 25}
```

e como um lambda:

```
args_converter = lambda row_index, an_obj: {'text': an_obj.text,
                                              'size_hint_y': None,
                                              'height': 25}
```

No exemplo do conversor args acima, o item de dados é assumido como sendo um objeto (instância de classe), daí a referência `an_obj.text`.

Aqui está um exemplo de um conversor de args que funciona com dados de lista que são dicts:

```
args_converter = lambda row_index, obj: {'text': obj['text'],
                                         'size_hint_y': None,
                                         'height': 25}
```

Portanto, é responsabilidade do desenvolvedor codificar o args_converter de acordo com os dados disponíveis. O argumento `row_index` pode ser útil em alguns casos, como quando são necessários labels personalizados.

Um Exemplo com ListView

Agora, para alguns códigos de exemplo:

```
from kivy.adapters.listadapter import ListAdapter
from kivy.uix.listview import ListItemButton, ListView

data = [{'text': str(i), 'is_selected': False} for i in range(100)]

args_converter = lambda row_index, rec: {'text': rec['text'],
                                         'size_hint_y': None,
```

```

        'height': 25}

list_adapter = ListAdapter(data=data,
                           args_converter=args_converter,
                           cls=ListItemButton,
                           selection_mode='single',
                           allow_empty_selection=False)

list_view = ListView(adapter=list_adapter)

```

Esta lista exibirá 100 botões com texto de 0 a 100. A função args_converter converte os itens dict nos dados e cria instâncias de exibições ListItemButton passando esses itens convertidos em seu construtor. O listview só permitirá seleção única e o primeiro item já estará selecionado como allow_empty_selection é False. Para uma discussão completa sobre esses argumentos, consulte a documentação: class: [~kivy.adapters.listadapter.ListAdapter](#).

A: classe: [~kivy.uix.listview.ListItemLabel](#) funciona da mesma forma que a: classe: [~kivy.uix.listview.ListItemButton](#).

Usando um Custom Item View Class

Os dados usados em um adaptador podem ser qualquer um dos tipos Python normais ou classes personalizadas, como mostrado abaixo. Cabe ao programador assegurar que o args_converter executa as conversões apropriadas.

Aqui nós fazemos uma classe DataItem simples que tem o texto necessário e as propriedades is_selected:

```

from kivy.uix.listview import ListItemButton
from kivy.adapters.listadapter import ListAdapter

class DataItem(object):
    def __init__(self, text='', is_selected=False):
        self.text = text
        self.is_selected = is_selected

data_items = [DataItem(text='cat'),
              DataItem(text='dog'),
              DataItem(text='frog')]

list_item_args_converter = lambda row_index, obj: {'text': obj.text,
                                                   'size_hint_y': None,
                                                   'height': 25}

list_adapter = ListAdapter(data=data_items,

```

```

        args_converter=list_item_args_converter,
        propagate_selection_to_data=True,
        cls=ListItemButton)

list_view = ListView(adapter=list_adapter)

```

Os dados são passados para a classe: `~kivy.adapters.listadapterListAdapter` juntamente com uma função `args_converter`. A configuração de propagação significa que a propriedade `is_selected` para cada item de dados será definida e mantida em sincronia com as exibições do item de lista. Esta definição deve ser definida como True se pretender inicializar a exibição com pontos de item já selecionados.

Você também pode usar o mixin fornecido pela: classe: `~kivy.adapters.models.SelectableDataItem` para criar uma classe personalizada. Em vez da classe de `DataItem` “manualmente construída” acima, poderíamos fazer:

```

from kivy.adapters.models import SelectableDataItem

class DataItem(SelectableDataItem):
    # Add properties here.
    pass

```

`SelectableDataItem` é uma classe mixin simples que tem uma propriedade `is_selected`.

Usando um Item View Template

`SelectableView` é outra classe mixin simples que tem propriedades necessárias para um item de lista: `text`, e `is_selected`. Para fazer seu próprio template, misture-o assim:

```

from kivy.lang import Builder

Builder.load_string("""
[CustomListItem@SelectableView+BoxLayout]:
    size_hint_y: ctx.size_hint_y
    height: ctx.height
    ListItemButton:
        text: ctx.text
        is_selected: ctx.is_selected
""")

```

Uma classe chamada `CustomListItem` pode ser instanciada para cada item de lista. Observe que isto é subclasse da: classe: `~kivy.uix.boxlayout.BoxLayout` e é, portanto, um tipo de: mod:’ `~kivy.uix.layout`’. Isto contém uma instância da: classe: `~kivy.uix.listview.ListItemButton`.

Usando o poder da linguagem Kivy (kv), você pode facilmente criar itens de lista compostos: além de `ListItemButton`, você poderia ter um `ListItemLabel` ou uma classe personalizada que você definiu e registrou através da classe: `~kivy.factory.Factory`.

Um args_converter precisa ser construído junto com um modelo kv. Por exemplo, para usar o modelo kv acima:

```
list_item_args_converter = \
    lambda row_index, rec: {'text': rec['text'],
                           'is_selected': rec['is_selected'],
                           'size_hint_y': None,
                           'height': 25}

integers_dict = \
    { str(i): {'text': str(i), 'is_selected': False} for i in \
    range(100)}

dict_adapter = DictAdapter(sorted_keys=[str(i) for i in range(100)],
                            data=integers_dict,
                            args_converter=list_item_args_converter,
                            template='CustomListItem')

list_view = ListView(adapter=dict_adapter)
```

Um adaptador dict é criado com 1..100 strings inteiros como sorted_keys e um integers_dict como dados. Integers_dict tem as strings inteiras como chaves e dicts com texto e is_selected propriedades. O CustomListItem definido acima na chamada Builder.load_string () é definido como o modelo kv para as exibições do item de lista. A função lambda list_item_args_converter tomará cada dict em integers_dict e retornará um dict args, pronto para passar como o contexto (ctx) para o modelo.

Usando um CompositeListItem

A classe: `~kivy.uix.listview.CompositeListItem` é outra opção para a criação de itens avançados da lista composta. A abordagem de linguagem kv tem suas vantagens, mas aqui nós construímos uma exibição de lista composta usando Python:

```
args_converter = lambda row_index, rec: \
    {'text': rec['text'],
     'size_hint_y': None,
     'height': 25,
     'cls_dicts': [{ 'cls': ListItemButton,
                     'kwargs': { 'text': rec['text']}},
                   { 'cls': ListItemLabel,
                     'kwargs': { 'text': "Middle-{0}".format(rec['text'])},
                     'is_representing_cls': True}],
     'cls': ListItemButton,
     'kwargs': { 'text': rec['text']}}

item_strings = ["{0}".format(index) for index in range(100)]

integers_dict = \
```

```

{str(i): {'text': str(i), 'is_selected': False} for i in
range(100)}

dict_adapter = DictAdapter(sorted_keys=item_strings,
                           data=integers_dict,
                           args_converter=args_converter,
                           selection_mode='single',
                           allow_empty_selection=False,
                           cls=CompositeListItem)

list_view = ListView(adapter=dict_adapter)

```

O args_converter é um pouco complicado, então devemos passar pelos detalhes. Observe na instanciação da classe: `~kivy.adapters.dictadapter.DictAdapter` que a: classe: '`~kivy.uix.listview.CompositeListItem`' é definida como o cls a ser instanciado para cada componente de item de lista. O args_converter fará dicts args para este cls. No args_converter, os três primeiros itens, text, size_hint_y e height, são argumentos para o próprio CompositeListItem. Depois disso, você verá uma lista cls_dicts que contém conjuntos de argumentos para cada um dos widgets membros para este composto: 2: classe: `ListItemButtons <kivy.uix.listview.ListItemButton>` e a: classe: `~kivy.uix.listview.ListItemLabel`. Esta é uma abordagem semelhante à utilização de um modelo kv descrito acima.

Para obter detalhes sobre como a: classe: `~kivy.uix.listview.CompositeListItem` funciona, examine o código, procurando como a análise da lista cls_dicts e o processamento kwargs são concluídos.

Uso para Seleção

O que podemos fazer com a seleção? Combinando a seleção com o sistema de ligações em Kivy, podemos construir uma ampla gama de projetos de interface de usuário.

Podemos fazer itens de dados que contenham os nomes de raças de cães, e conectar a seleção de raça de cão para a exibição de detalhes em outra visão, que atualizaria automaticamente na seleção. Isto é feito através de uma ligação ao evento: attr: `~kivy.adapters.listadapterListAdapter.on_selection_change`:

```
list_adapter.bind(on_selection_change=callback_function)
```

Onde callback_function() recebe passado o adaptador como um argumento e faz tudo o que é necessário para a atualização. Veja o exemplo chamado `list_master_detail.py`, e imagine que a lista à esquerda pode ser uma lista de raças de cães, e a visualização de detalhes à direita pode mostrar detalhes para uma raça de cão selecionada.

In another example, we could set the selection_mode of a listview to 'multiple', and load it with a list of answers to a multiple-choice question. The question could have several correct answers. A color swatch view could be bound to selection change, as

above, so that it turns green as soon as the correct choices are made, unless the number of touches exceeds a limit, then answer session could be terminated. See the examples that feature thumbnail the images to get some ideas, e.g. list_cascade_dict.py.

Em um exemplo mais envolvido, poderíamos encadear três visualizações de lista, onde a seleção no primeiro controla os itens mostrados no segundo e a seleção no segundo controla os itens mostrados no terceiro. Se allow_empty_selection estiver definido como False para estas listas, resultaria um sistema dinâmico de seleção “em cascata” de uma lista para a próxima.

Há tantas maneiras que a funcionalidade de listviews e ligações Kivy podem ser usadas que temos apenas riscado a superfície aqui. Para obter exemplos em disco, consulte:

```
kivy/examples/widgets/lists/list_*.py
```

Vários exemplos mostram o comportamento “em cascata” descrito acima. Outros demonstram o uso de modelos kv e visualizações de lista composta.

```
class kivy.uix.listview>SelectableView(*args, **kwargs)
    Bases: object
```

A **SelectableView** é um mixin usado com itens de lista e outras classes que devem ser instanciadas em uma exibição de lista ou outras classes que usam um adaptador habilitado para seleção como ListAdapter. *select()* e *deselect()* podem ser sobreescrito com o código de exibição para marcar itens selecionados ou não, se desejado.

deselect(*args)

O item de lista é responsável por atualizar a exibição quando estiver sendo desmarcada, se desejado.

select(*args)

O item da lista é responsável por atualizar a exibição quando estiver selecionado, se desejado.

```
class kivy.uix.listview>ListItemButton(**kwargs)
    Bases:      kivy.uix.listview.ListItemReprMixin,      kivy.uix.
               selectableview.SelectableView, kivy.uix.button.Button
```

ListButtonItem mixes **SelectableView** with **Button** to produce a button suitable for use in **ListView**.

deselected_color

deselected_color é uma **ListProperty** e o padrão é [0., 1., 0., 1].

selected_color

selected_color é uma **ListProperty** e o padrão é [1., 0., 0., 1].

```
class kivy.uix.listview>ListItemLabel(**kwargs)
    Bases:      kivy.uix.listview.ListItemReprMixin,      kivy.uix.
               selectableview.SelectableView, kivy.uix.label.Label
```

ListItemLabel mistura-se com *SelectableView* with *Label* para produzir um Label adequado para uma *ListView*.

```
class kivy.uix.listview.CompositeListItem(**kwargs)
Bases:      kivy.uix.selectableview.SelectableView,    kivy.uix.
           boxlayout.BoxLayout
```

CompositeListItem mixes *SelectableView* with *BoxLayout* for a generic container-style list item, to be used in *ListView*.

background_color

ListItem é uma subclasse de *Button*, que tem *background_color*, mas para um item de lista composta, devemos adicionar esta propriedade.

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

deselected_color

deselected_color é uma *ListProperty* e o padrão é [.33, .33, .33, 1].

representing_cls

Qual classe de exibição de componente, se houver, deve representar para o item de listagem composta em *__repr__()*?

representing_cls é uma *ObjectProperty* e o padrão é *None*.

selected_color

selected_color é uma *ListProperty* e o padrão é [1., 0., 0., 1].

```
class kivy.uix.listview.ListView(*args, **kwargs)
Bases:      kivy.uix.abstractview.AbstractView,    kivy.event.
           EventDispatcher
```

ListView is a primary high-level widget, handling the common task of presenting items in a scrolling list. Flexibility is afforded by use of a variety of adapters to interface with data.

A propriedade do adaptador vem através da classe mixed: class: *~kivy.abstractview.AbstractView*.

ListView also subclasses *EventDispatcher* for scrolling. The event *on_scroll_complete* is used in refreshing the main view.

Para uma lista simples de itens de string, sem seleção, use: classe: *~kivy.adapters.simplelistadapter.SimpleListAdapter*. Para itens de lista que respondem a seleção, variando de itens simples a combinações avançadas, use: classe: *~kivy.adapters.listadapter.ListAdapter*. Para um adaptador poderoso alternativo, use: classe: *~kivy.adapters.dictadapter.DictAdapter*, completando a escolha para projetar listas altamente interativas.

Events

on_scroll_complete: (*boolean*,) Disparado quando a rolagem é concluída.

container

O contêiner é uma classe de widget: `~kivy.uix.gridlayout.GridLayout` mantido dentro de uma classe de widget: `~kivy.uix.scrollview.ScrollView` widget. (Veja o bloco kv associado na configuração `Builder.load_string()`). As instâncias de exibição de item gerenciadas e fornecidas pelo adaptador são adicionadas a este contêiner. O contêiner é desmarcado com uma chamada para `clear_widgets()` quando a lista é reconstruída pelo método `populate()`. Uma instância padding da classe: `~kivy.uix.widget.Widget` também é adicionada conforme a necessidade, dependendo dos cálculos de altura da linha.

`container` é uma *ObjectProperty* e o padrão é `None`.

divider

[TODO] Não utilizar.

divider_height

[TODO] Não utilizar.

item_strings

Se `item_strings` for fornecido, crie uma instância da classe: `~kivy.adapters.simplelistadapter.SimpleListAdapter` com esta lista de strings e use-a para gerenciar uma lista sem seleção.

`item_strings` é uma *ListProperty* e o padrão é `[]`.

row_height

A propriedade `row_height` é calculada com base na altura do contêiner e na contagem de itens.

`row_height` é uma *NumericProperty* e o padrão é `None`.

scrolling

Se o método `scroll_to()` é chamado enquanto as operações de rolagem estão acontecendo, um erro de recursão de chamada pode ocorrer. `Scroll_to()` verifica se a rolagem é `False` antes de chamar `populate()`. `Scroll_to()` envia um evento `scrolling_complete`, que define a rolagem de volta para `False`.

`scrolling` é uma *BooleanProperty* e o padrão é `False`.

class kivy.uix.listview.ListItemReprMixin(kwargs)**

Bases: `kivy.uix.label.Label`

A classe: `~kivy.uix.listview.ListItemReprMixin` fornece uma classe: `~kivy.uix.label.Label` com uma representação de string compatível com Python 2/3 (`__repr__`). É destinado para uso interno.

4.15.25 ModalView

Novo na versão 1.4.0.

A classe `ModalView` é um widget usado para criar janelas modais. Por padrão, janelas modais se sobrepõem e cobrem toda a janela “pai”.

Lembre-se de que o tamanho padrão de um Widget é `size_hint = (1, 1)`. Se você não quiser que sua visualização seja em tela cheia, use o `size hint` com valores menores que 1 (por exemplo, `size_hint = (.8, .8)`) ou desative o `size_hint` e use valores de tamanho fixo.

Exemplos

Exemplo de uma simples janela de 400x400, um Hello World:

```
view = ModalView(size_hint=(None, None), size=(400, 400))
view.add_widget(Label(text='Hello world'))
```

Por padrão, qualquer clique fora View principal fechará essa visualização. Se não quiseres isso, podes definir `ModalView.auto_dismiss` to False:

```
view = ModalView(auto_dismiss=False)
view.add_widget(Label(text='Hello world'))
view.open()
```

Para descartar/fechar manualmente a exibição, use o método `ModalView.dismiss()` da instância de `ModalView`:

```
view.dismiss()
```

Ambos `ModalView.open()` e `ModalView.dismiss()` podem ter funções vinculadas. Isso significa que podes vincular diretamente a função a uma ação, por exemplo, para `on_press` de um botão:

```
# create content and add it to the view
content = Button(text='Close me!')
view = ModalView(auto_dismiss=False)
view.add_widget(content)

# bind the on_press event of the button to the dismiss function
content.bind(on_press=view.dismiss)

# open the view
view.open()
```

Eventos do ModalView

Existem dois eventos disponíveis: `on_open` que é disparado quando a vista é aberta, e `on_dismiss` que é disparado quando a visualização é fechada. Para `on_dismiss`, você

pode impedir que a exibição feche explicitamente retornando *True* a partir da sua função de callback:

```
def my_callback(instance):
    print('ModalView', instance, 'is being dismissed, but is '
         'prevented!')
    return True
view = ModalView()
view.add_widget(Label(text='Hello world'))
view.bind(on_dismiss=my_callback)
view.open()
```

Alterado na versão 1.5.0: O *ModalView* pode ser fechado pressionando a tecla de escape (ESC) no teclado se a propriedade *ModalView.auto_dismiss* for *True* (padrão).

class kivy.uix.modalview.ModalView(kwargs)**
Bases: *kivy.uix.anchorlayout.AnchorLayout*

Classe '*ModalView*'. Veja o módulo da documentação para maiores informações.

Events

on_open: Disparado quando o *ModalView* é aberta.

on_dismiss: Disparado quando o *ModalView* é fechado. Se callback retornar *True*, o dismiss será cancelado.

attach_to

Se um Widget estiver definido em *attach_to*, a exibição será anexada à janela pai mais próxima do Widget. Se nenhuma for encontrada, ela será anexada à janela principal/global.

attach_to é uma *ObjectProperty* e o padrão é *None*.

auto_dismiss

Essa propriedade determina se a exibição é automaticamente ignorada (fechada) quando o usuário clica fora dela.

auto_dismiss é um *BooleanProperty* e o padrão é *True*.

background

Imagem de fundo da View usada pela View de fundo.

background é uma *StringProperty* e o padrão é 'atlas://data/images/defaulttheme/modalview-background'.

background_color

Imagem de fundo no formato (r, g, b, a).

background_color é uma *ListProperty* e o padrão é [0, 0, 0, .7].

border

Border usada para *BorderImage* instruções gráficas. Usado para as pro-

propriedades `background_normal` e `background_down`. Pode ser usado quanto estiver sendo utilizado fundos personalizados.

It must be a list of four values: (bottom, right, top, left). Read the `BorderImage` instructions for more information about how to use it.

`border` é uma `ListProperty` e o padrão é `(16, 16, 16, 16)`.

`dismiss(*largs, **kwargs)`

Feche a View se estiver aberta. Se realmente desejas fechar a exibição, independentemente do evento `on_dismiss` retornar, podes usar o argumento `force`:

```
view = ModalView()
view.dismiss(force=True)
```

Quando a exibição é descartada, ela será desbotada antes de ser removida do pai. Se não quiseres que essa animação seja exibida, use:

```
view.dismiss(animation=False)
```

`open(*largs)`

Mostrar a janela de visualização a partir do Widget `attach_to`. Se configurado, ele será anexado à janela mais próxima. Se o Widget não estiver anexado a nenhuma janela, a View será anexada ao global `Window`.

4.15.26 PageLayout

A classe `PageLayout` tem a função de criar um layout multi-páginas, de modo que permita mudar de páginas usando as margens.

`PageLayout` não honra atualmente o `size_hint`, `size_hint_min`, `size_hint_max`, ou `pos_hint` properties.

Novo na versão 1.8.0.

Exemplo:

```
PageLayout:
    Button:
        text: 'page1'
    Button:
        text: 'page2'
    Button:
        text: 'page3'
```

As transições de uma página para a outra são feitas deslizando das áreas de borda do lado direito ou esquerdo. Se desejas exibir vários Widgets em uma página, sugerimos

que você use um layout contendo-os. Idealmente, cada página deve consistir de um único Widget **layout** que contém os Widgets restantes daquela página.

```
class kivy.uix.pagelayout.PageLayout(**kwargs)
    Bases: kivy.uix.layout.Layout
```

Classe *PageLayout*. Veja o módulo da documentação para maiores informações.

border

A largura da borda ao redor da página atual usada para exibir a página anterior/próxima desliza pelas áreas quando necessário.

border é uma *NumericProperty* e o padrão é *50dp*.

page

A página exibida no momento.

page é uma *NumericProperty* e o padrão é *0*.

swipe_threshold

O limite usado para disparar os Swipes como porcentagem do tamanho do Widget.

swipe_threshold é uma *NumericProperty* e o padrão é *5*.

4.15.27 Popup

Novo na versão 1.0.7.



A classe **Popup** é um widget cuja função é criar popups modais. Por padrão, esse popup irá cobrir a janela principal. Quando for criado, o popup deve ser ajustado com a propriedade **Popup.title** and **Popup.content**.

Lembre-se de que o tamanho padrão de um Widget é `size_hint = (1, 1)`. Se não quiseres que seu *Popup* seja fullscreen, use `size_hint` com valores menores que 1 (por exemplo

`size_hint = (.8, .8)`) ou desative o `size_hint` e use atributos de tamanho fixo.

Alterado na versão 1.4.0: Uma classe `Popup` agora herda de `ModalView`. Um `Popup` oferece um layout padrão com um título e uma barra de separação.

Exemplos

Exemplo de um simples Poput “Olá mundo” com 400x400:

```
popup = Popup(title='Test popup',
    content=Label(text='Hello world'),
    size_hint=(None, None), size=(400, 400))
```

Por padrão, qualquer clique/toque fora do `Popup` irá desativar/fechar a janela. Se você não deseja isso, você pode definir o `auto_dismiss` como sendo `False`:

```
popup = Popup(title='Test popup', content=Label(text='Hello world'),
    auto_dismiss=False)
popup.open()
```

Para desativar/fechar manualmente o `Popup`, use `dismiss`:

```
popup.dismiss()
```

Ambos `open()` e `dismiss()` são ligáveis (podem ter funções vinculadas). Isso significa que podes vincular diretamente uma função a cada uma das ações, por exemplo, para um botão `on_press`:

```
# create content and add to the popup
content = Button(text='Close me!')
popup = Popup(content=content, auto_dismiss=False)

# bind the on_press event of the button to the dismiss function
content.bind(on_press=popup.dismiss)

# open the popup
popup.open()
```

Eventos Popup

Há dois eventos disponíveis: `on_open` que é disparado quando o `Popup` está sendo aberto, e `on_dismiss` que é disparado quando o `Popup` é fechado. Para `on_dismiss`, podes impedir que o `Popup` feche explicitamente retornando `True` desde o seu callback:

```
def my_callback(instance):
    print('Popup', instance, 'is being dismissed but is prevented!')
    return True
```

```
popup = Popup(content=Label(text='Hello world'))
popup.bind(on_dismiss=my_callback)
popup.open()
```

class kivy.uix.popup.Popup(kwargs)**
Bases: *kivy.uix.modalview.ModalView*

Classe *Popup*. Veja o módulo da documentação para maiores informações.

Events

on_open:Disparado quando o *Popup* é aberto.

on_dismiss:Disparado quando o *Popup* é fechado. Se o callback retornar *True*, o dismiss (descartar) será cancelado.

content

Conteúdo do *Popup* que é exibido logo abaixo do título.

content é uma *ObjectProperty* e o padrão é *None*.

separator_color

Cor usada pelo separador entre título e conteúdo.

Novo na versão 1.1.0.

separator_color é uma *ListProperty* e o padrão é [47 / 255., 167 / 255., 212 / 255., 1.]

separator_height

Altura do separador.

Novo na versão 1.1.0.

separator_height é uma *NumericProperty* e o padrão é *2dp*.

title

String que representa o título do *Popup*.

title é uma *StringProperty* e o padrão é 'No title'.

title_align

Alinhamento horizontal do título.

Novo na versão 1.9.0.

title_align é uma *OptionProperty* e o padrão é 'left'. Opções disponíveis são *left*, *center*, *right* e *justify*.

title_color

Cor usada pelo Título.

Novo na versão 1.8.0.

title_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

title_font

Fonte usada para renderizar o texto do título.

Novo na versão 1.9.0.

title_font is a *StringProperty* and defaults to ‘Roboto’. This value is taken from *Config*.

title_size

Representa o tamanho da fonte do título do *Popup*.

Novo na versão 1.6.0.

title_size é uma *NumericProperty* e o padrão é ‘14sp’.

class kivy.uix.popup.PopupException

Bases: *exceptions.Exception*

Exceção ‘Popup’, disparada quando vários Widgets de conteúdo são adicionados ao *Popup*.

Novo na versão 1.4.0.

4.15.28 Progress Bar

Novo na versão 1.0.8.



A classe *ProgressBar* é um widget usado para mostrar o progresso de determinadas tarefas. Só está em funcionamento o modo horizontal. O modo vertical ainda não está disponível.

A barra de progresso não tem elementos interativos e é um widget somente de exibição

Para usá-lo, basta atribuir um valor que indique o progresso atual:

```
from kivy.uix.progressbar import ProgressBar
pb = ProgressBar(max=1000)

# this will update the graphics automatically (75% done)
pb.value = 750
```

class kivy.uix.progressbar.ProgressBar(kwargs)**

Bases: *kivy.uix.widget.Widget*

Classe para criar um Widget que exiba uma Barra de Progresso.

Veja a módulo da documentação para obter mais detalhes.

max

Valor máximo permitido para `value`.

`max` é uma *NumericProperty* e o valor padrão é 100.

value

Valor atual usado pelo Slider.

`value` é um *AliasProperty* que retorna o valor da barra de progresso. Se o valor for < 0 ou > `max`, ele será normalizado para esses limites.

Alterado na versão 1.6.0: O valor é agora limitado entre 0 e `max`.

value_normalized

Valor normalizado dentro do intervalo 0-1:

```
>>> pb = ProgressBar(value=50, max=100)
>>> pb.value
50
>>> pb.value_normalized
0.5
```

`value_normalized` é uma *AliasProperty*.

4.15.29 RecycleBoxLayout

Novo na versão 1.10.0.

Aviso: Este módulo é altamente experimental, sua API pode mudar no futuro e a documentação ainda não está completa.

O RecycleBoxLayout é feito para fornecer um layout tipo *BoxLayout* quando usado com o widget *RecycleView*. Para mais informações consulte a documentação do modulo *recycleview*.

4.15.30 RecycleGridLayout

Novo na versão 1.10.0.

Aviso: Este módulo é altamente experimental, sua API pode mudar no futuro e a documentação ainda não está completa.

O RecycleGridLayout é desenhado para providenciar um layout *GridLayout* quando usado com um *RecycleView*. Por favor leia a documentação do módulo *recycleview* para mais informações.

4.15.31 RecycleLayout

Novo na versão 1.10.0.

Aviso: Este módulo é altamente experimental, sua API pode mudar no futuro e a documentação ainda não está completa.

4.15.32 Relative Layout

Novo na versão 1.4.0.

Este layout permite a você setar coordenadas relativas para os filhos. Se você quer um posicionamento absoluto, use a classe: `~kivy.uix.floatlayout.FloatLayout`.

The **RelativeLayout** classe se comporta exatamente como o regular **FloatLayout** exceto que seus Widgets filho estão posicionados em relação ao layout.

Quando um Widget com posição = (0,0) é adicionado a um RelativeLayout, o Widget filho também se moverá quando a posição do RelativeLayout for alterada. As coordenadas dos Widgets filho permanecem (0,0), pois elas são sempre relativas ao layout do pai.

Sistema de Coordenadas

Coordenadas da Janela

Por padrão, há apenas um sistema de coordenadas que define a posição dos Widgets e eventos de toques enviados para eles: o sistema de coordenadas da janela, que define (0, 0) no canto inferior esquerdo da janela. Embora existam outros sistemas de coordenadas definidos, por exemplo, local e coordenadas do Widget pai, esses sistemas de coordenadas são idênticos ao sistema de coordenadas da janela, desde que um Widget do tipo RelativeLayout não esteja na pilha do Widget pai. Quando `widget.pos` for lido ou um toque for recebido, os valores de coordenadas estarão em coordenadas do pai, porém, como mencionado, estes são idênticos as coordenadas da janela, mesmo que estejam num complexa pilha de Widget.

Por exemplo:

```
BoxLayout:  
    Label:  
        text: 'Left'  
    Button:  
        text: 'Middle'  
        on_touch_down: print('Middle: {}'.format(args[1].pos))  
BoxLayout:
```

```

on_touch_down: print('Box: {}'.format(args[1].pos))
Button:
    text: 'Right'
    on_touch_down: print('Right: {}'.format(args[1].pos))

```

Quando o botão do meio é clicado e o toque se propaga através dos diferentes sistemas de coordenadas do pai, ele imprime o seguinte:

```

>>> Box: (430.0, 282.0)
>>> Right: (430.0, 282.0)
>>> Middle: (430.0, 282.0)

```

Conforme reivindicado, o toque tem coordenadas idênticas às coordenadas da janela em cada sistema de coordenadas. `collide_point()` por exemplo, toma o ponto nas coordenadas da janela.

Coordenadas Pai

Outro `RelativeLayout` tipos de Widget são `Scatter`, `ScatterLayout`, e `ScrollView`. Se esse Widget especial estiver na pilha pai, somente então o sistema de coordenadas pai e local divergirá do sistema de coordenadas da janela. Para cada um desses elementos na pilha, é criado um sistema de coordenadas com (0, 0) desse sistema de coordenadas no canto inferior esquerdo do Widget. As **coordenadas de posição e de toque recebidas e lidas por um Widget estão no sistema de coordenadas do Widget especial mais recente em sua pilha pai (não incluindo a si mesmo) ou em coordenadas de janela se não houver nenhuma** (como no primeiro exemplo). Chamamos essas coordenadas de coordenadas do pai.

Por exemplo:

```

BoxLayout:
    Label:
        text: 'Left'
    Button:
        text: 'Middle'
        on_touch_down: print('Middle: {}'.format(args[1].pos))
    RelativeLayout:
        on_touch_down: print('Relative: {}'.format(args[1].pos))
        Button:
            text: 'Right'
            on_touch_down: print('Right: {}'.format(args[1].pos))

```

Clicando no botão do meio imprime:

```

>>> Relative: (396.0, 298.0)
>>> Right: (-137.33, 298.0)
>>> Middle: (396.0, 298.0)

```

À medida que o toque se propaga através dos Widgets, para cada Widget, o toque é recebido em coordenadas do pai. Como os Widgets relativos e intermediários não têm esses Widgets especiais em sua pilha pai, o toque é o mesmo que as coordenadas da janela. Somente o Widget direito, que tem um RelativeLayout em sua pilha pai, recebe o toque em coordenadas relativas a esse RelativeLayout que é diferente das coordenadas da janela.

Coordenadas Locais e de Widgets

Quando expressa em coordenadas do pai, a posição é expressa nas coordenadas do Widget especial mais recente em sua pilha pai, não incluindo a si mesma. Quando expressos em coordenadas locais ou coordenadas do Widget, os próprios Widgets também são incluídos.

Alterando o exemplo acima para transformar as coordenadas do pai em coordenadas locais:

```
BoxLayout:  
    Label:  
        text: 'Left'  
    Button:  
        text: 'Middle'  
        on_touch_down: print('Middle: {}'.format(self.to_  
        ↵local(*args[1].pos)))  
    RelativeLayout:  
        on_touch_down: print('Relative: {}'.format(self.to_  
        ↵local(*args[1].pos)))  
        Button:  
            text: 'Right'  
            on_touch_down: print('Right: {}'.format(self.to_  
            ↵local(*args[1].pos)))
```

Agora, clicando no botão do meio imprime:

```
>>> Relative: (-135.33, 301.0)  
>>> Right: (-135.33, 301.0)  
>>> Middle: (398.0, 301.0)
```

Isso ocorre porque agora o Widget relativo também expressa as coordenadas relativas a si mesma.

Transformações de Coordenadas

Widget fornece 4 funções para transformar coordenadas entre os vários sistemas de

coordenadas. Por agora, supomos que o *relative* palavra reservada dessa função é *False*. ***to_widget()*** toma as coordenadas expressas nas coordenadas da janela e as retorna em coordenadas locais (Widget). ***to_window()*** toma as coordenadas expressas em coordenadas locais e as retorna nas coordenadas da janela. ***to_parent()*** toma as coordenadas expressas em coordenadas locais e retorna-as em coordenadas parentes. ***to_local()*** toma as coordenadas expressas nas coordenadas de origem e as retorna em coordenadas locais.

Cada uma das 4 funções de transformação tem um parâmetro *relativo*. Quando o parâmetro relativo é True, as coordenadas são retornadas ou originadas em coordenadas relativas verdadeiras - em relação a um sistema de coordenadas com seu (0, 0) no canto inferior esquerdo do widget em questão.

Armadilhas Comuns

Como todas as posições dentro de um ***RelativeLayout*** são relativas à posição do layout propriamente dito, a posição do layout nunca deve ser usada para determinar a posição de sub-widgets ou o layout ***canvas***.

Tome o seguinte código kv como um exemplo:

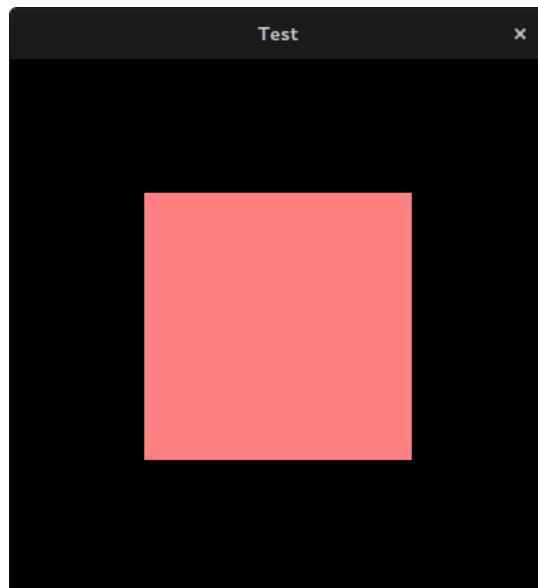


Fig. 4.1: resultado esperado

```
FloatLayout:  
    Widget:  
        size_hint: None, None  
        size: 200, 200  
        pos: 200, 200  
  
        canvas:  
            Color:
```

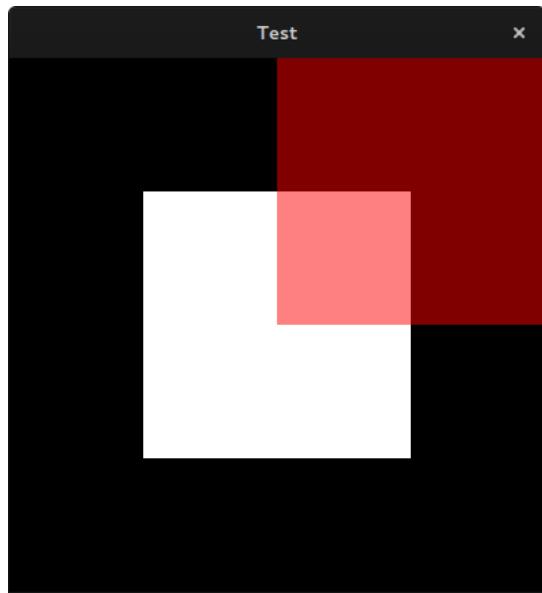


Fig. 4.2: resultado atual

```
    rgba: 1, 1, 1, 1
Rectangle:
    pos: self.pos
    size: self.size

RelativeLayout:
    size_hint: None, None
    size: 200, 200
    pos: 200, 200

    canvas:
        Color:
            rgba: 1, 0, 0, 0.5
        Rectangle:
            pos: self.pos # incorrect
            size: self.size
```

Pode esperar que este renderize um único retângulo rosa; No entanto, o conteúdo da classe `:RelativeLayout` já está transformado, então o uso de `pos: self.pos` dobrará essa transformação. Neste caso, usando `pos: 0, 0` ou omitindo `pos` completamente fornecerá o resultado esperado.

Isso também se aplica à posição de sub-widgets. Em vez de posicionar uma `Widget` com base na própria posição do layout:

```
RelativeLayout:
    Widget:
        pos: self.parent.pos
    Widget:
```

```
center: self.parent.center
```

use a propriedade pos_hint:

```
RelativeLayout:  
    Widget:  
        pos_hint: {'x': 0, 'y': 0}  
    Widget:  
        pos_hint: {'center_x': 0.5, 'center_y': 0.5}
```

Alterado na versão 1.7.0: Antes da versão 1.7.0, o *RelativeLayout* foi implementado como um *FloatLayout* dentro de uma *Scatter*. Este comportamento/Widget foi renomeado para *ScatterLayout*. A *RelativeLayout* agora só suporta posições relativas (e não pode ser rotacionado, dimensionado ou transladado em um sistema multitouch usando dois ou mais dedos). Isso foi feito para que a implementação pudesse ser otimizada e evitar os cálculos mais pesados de *Scatter* (por exemplo, matriz inversa, recalcular múltiplas propriedades etc.)

```
class kivy.uix.relativelayout.RelativeLayout(**kw)  
    Bases: kivy.uix.floatlayout.FloatLayout
```

Classe RelativeLayout, veja o módulo da documentação para maiores informações.

4.15.33 Renderizador reStructuredText

Novo na versão 1.1.0.

reStructuredText <<http://docutils.sourceforge.net/rst.html>> é um sistema de fácil leitura e visualização, estilo ‘o que você vê é o que você tem’ (what you see is what you get - WYSWYG). Usado para demonstrar sintaxes, linguagens de marcação e análise de códigos.

Nota: This widget requires the `docutils` package to run. Install it with `pip` or include it as one of your deployment requirements.

Aviso: This widget is highly experimental. The styling and implementation should not be considered stable until this warning has been removed.

Uso com texto

```
text = ""  
... _top:
```

```
Hello world
=====
This is an **emphasized text**, some ``interpreted text``.
And this is a reference to top_::

$ print("Hello world")

"""
document = RstDocument(text=text)
```

O processamento será exibido:

Hello world

This is an **emphasized text**, some interpreted text. And this is a reference to [top](#):

```
$ print "Hello world"
```

Uso com fonte

You can also render a rst file using the **source** property:

```
document = RstDocument(source='index.rst')
```

You can reference other documents using the role `:doc:`. For example, in the document `index.rst` you can write:

```
Go to my next document: :doc:`moreinfo.rst`
```

Ele irá gerar um link que, quando clicado, abrirá o documento `moreinfo.rst`.

class kivy.uix.rst.RstDocument(kwargs)**
Bases: [kivy.uix.scrollview.ScrollView](#)

Widget base usado para armazenar um documento RST. Consulte a documentação do módulo para obter mais informações.

background_color

Especifica o `background_color` a ser usado com o `RstDocument`.

Novo na versão 1.8.0.

`background_color` é uma [AliasProperty](#) para `colors['background']`.

base_font_size

Tamanho da fonte para o título principal, por padrão é usado 31. Todos os outros tamanhos de fonte são derivados a partir deste.

Novo na versão 1.8.0.

colors

Dicionário de todas as cores usadas na renderização do RST.

Aviso: Este dicionário precisa de um tratamento especial. Você também precisa invocar `RstDocument.render()` caso venha a alterá-los após o carregamento.

`colors` é uma *DictProperty*.

document_root

Root path onde :doc: irá primeiros procurar por documentos. Se nenhum caminho for fornecido, será usado o diretório do primeiro arquivo fonte carregado.

`document_root` é uma *StringProperty* e o padrão é `None`.

goto(ref, *largs)

Vá até a referência. Se não for encontrado, nada será feito.

Para este texto:

```
.. _myref:  
  
This is something I always wanted.
```

Você pode fazer:

```
from kivy.clock import Clock  
from functools import partial  
  
doc = RstDocument(...)  
Clock.schedule_once(partial(doc.goto, 'myref'), 0.1)
```

Nota: É preferível atrasar a chamada do goto se acabastes de carregar o documento, isso porque o layout pode não estar concluído ou o tamanho do RstDocument talvez ainda não tenha sido determinado. Em ambos os casos, o cálculo da rolagem seria errado.

No entanto, você pode fazer uma chamada direta se o documento já estiver carregado.

Novo na versão 1.3.0.

preload(*filename*, *encoding*=’utf-8’, *errors*=’strict’)

Pré-carrega um arquivo RST para obter seu toctree e seu título.

O resultado será armazenado em **toctrees** com o *filename* com uma chave.

render()

Força o processamento de documentos.

resolve_path(*filename*)

Obtém o caminho para este nome de arquivo. Se o nome do arquivo não existir, será retornado o *document_root* + *nome_do_arquivo*.

show_errors

Indique se os erros dos analisadores RST devem ser mostrados na tela ou não.

show_errors é uma **BooleanProperty** e o padrão é *False*.

source

Nome do arquivo do documento RST.

source é uma **StringProperty** e o padrão é *None*.

source_encoding

Encoding que será usado para o arquivo *source*.

source_encoding é uma **StringProperty** e o padrão é *utf-8*.

Nota: É sua responsabilidade garantir que o valor fornecido seja um Codec válido e suportado pelo Python.

source_error

Tratamento de erros a ser usado durante a codificação do arquivo *source*.

source_error é uma **OptionProperty** e o padrão é *strict*. Pode ser uma das seguintes opções ‘strict’, ‘ignore’, ‘replace’, ‘xmlcharrefreplace’ ou ‘backslashreplace’.

text

Marcação de texto RST para o documento.

text é uma **StringProperty** e o padrão é *None*.

title

Título do documento atual.;

title é uma **StringProperty** e o padrão é “”. Ele é somente leitura.

toctrees

O *Toctree* de todos os documentos carregados ou pré-carregados. Este dicionário é preenchido quando um primeiro documento é explicitamente carregado ou quando **preload()** for invocado.

Se o documento não tiver nome de arquivo, p. Quando o documento é carregado de um arquivo de texto, a chave será ''.

`toctrees` é uma *DictProperty* e o padrão é {}.

underline_color

Cor sublinhada dos títulos, expressa em notação de cor do formato HTML

`underline_color` é uma *StringProperty* e o padrão é '204a9699'.

4.15.34 Sandbox

Novo na versão 1.8.0.

Aviso: Isto é experimental e está sujeito a alterações todas as vezes que esta advertência esteja presente.

Este é um widget que é executado por si mesmo e todos os seus filhos em Sandbox. Isso significa que se um filho levantar uma exceção (Exception), ela será interceptada. O Sandbox possui o seu próprio Clock, Cache, etc.

O Widget SandBox ainda é experimental e necessário e utilizado pelo KivyDesign. Quando o usuário desenha seu próprio Widget, caso faça algo de errado (valor do tamanho errado, código Python inválido), o erro será capturado sem derrubar a aplicação por inteira. Como o mesmo foi projetado desta forma, ainda estamos melhorando este Widget e o módulo `kivy.context`. Não utilize este componente, salvo se souberes o que estás fazendo.

`class kivy.uix.sandbox.Sandbox(**kwargs)`

Bases: `kivy.uix.floatlayout.FloatLayout`

O Widget Sanbbox, é usado para interceptar todas as exceções levantadas pelos Widgets filhos.

on_context_created()

Substitua este método para carregar o seu arquivo kv ou para fazer alguma coisa em um novo contexto criado.

on_exception(exception, _traceback=None)

Substitua este método para capturar todas as exceções levantadas pelos filhos.

Se você retornar `True`, ele não levantará a exceção. Se retornares `False`, a exceção será lançada para o pai.

on_touch_down(*args, **kwargs)

Recebe um evento de toque.

Parameters

classe touch: *MotionEvent* Toque recebido. O toque é está nas coordenadas do pai. Veja *relativelayout* para uma discussão sobre o sistema de coordenadas.

Returns*bool* se *True*, o despacho do evento de toque será interrompido. Se *False*, o evento continuará a ser despachado para o resto da árvore de Widgets.

on_touch_move(*args, **kwargs)

Recebe um evento de movimento de toque. O toque está nas coordenadas do Widget onde o mesmo está contido (coordenadas do Widget pai).

Para maiores informações veja *on_touch_down()*.

on_touch_up(*args, **kwargs)

Recebe um fim de um evento de toque (touch up). O toque está nas coordenadas do pai.

Para maiores informações veja *on_touch_down()*.

4.15.35 Scatter

Scatter é usada para construir widgets interativos que podem ser traduzidos, rotacionados e aumentados com dois ou mais dedos em sistemas multitouch.

O Scatter tem sua própria transformação de matriz: a matriz de modelview é alterada antes que os filhos sejam desenhados e a matriz anterior seja restaurada quando o desenho estiver concluído. Isso torna possível executar rotação, dimensionamento e tradução/interpretação em toda a árvore de filhos sem alterar nenhuma propriedade de widget. Esse comportamento específico torna o scatter único, mas existem algumas vantagens / restrições que você deve considerar:

1. Os filhos são posicionadas em relação ao Scatter de forma semelhante a *RelativeLayout*. Assim, ao arrastar o Scatter, a posição dos filhos não será alterada, apenas a posição do Scatter.
2. O tamanho do Scatter não tem impacto no tamanho dos seus filhos.
3. Se desejas redimensionar o Scatter, use uma escala, não o tamanho (leia #2). A escala transforma i Scatter e seus filhos, mas não altera o tamanho.
4. O Scatter não é um layout. Você deve gerenciar o tamanho dos filhos sozinho.

Para eventos de toque, o Scatter converte da matriz pai para a matriz do Scatter automaticamente em eventos *on_touch_down/move/up*. Se estiveres fazendo as coisas manualmente, precisarás usar *to_parent()* e *to_local()*.

Utilização

Por padrão, o Scatter não tem uma representação gráfica: é apenas um container. A ideia é combinar o Scatter com outro widget, por exemplo, uma classe: `~kivy.uix.image.Image`:

```
scatter = Scatter()  
image = Image(source='sun.jpg')  
scatter.add_widget(image)
```

Interações de Controle

Por padrão, todas as interações são ativadas. Você pode desativá-las seletivamente usando as propriedades: `do_rotation`, `do_translation` e `do_scale`.

Desativar rotação:

```
scatter = Scatter(do_rotation=False)
```

Permitir somente a tradução:

```
scatter = Scatter(do_rotation=False, do_scale=False)
```

Permitir somente tradução sobre o eixo x:

```
scatter = Scatter(do_rotation=False, do_scale=False,  
                 do_translation_y=False)
```

Trazer automaticamente para frente

Se a propriedade `Scatter.auto_bring_to_front` for `True`, o Widget Scatter será removido e re-adicionado ao pai quando o mesmo for tocado (trazido para a frente, acima de todos os outros Widgets no pai). Isso é útil quando estás manipulando vários Widgets Scatter e não desejas que o ativo esteja parcialmente ocultado.

Limitação da Escala

Estamos usando uma matriz de 32 bits em dupla representação. Isso significa que temos um limite para a escala. Você não pode fazer uma escala infinita baixo/cima com nossa execução. Geralmente, você não atinge a escala mínima (porque você não vê na tela), mas a escala máxima é $9.99506983235e+19$ (2^{66}).

Você também pode limitar a escala mínima e máxima permitida:

```
scatter = Scatter(scale_min=.5, scale_max=3.)
```

Comportamento

Alterado na versão 1.1.0: Se nenhuma interação de controle estiver ativa, então o manipulador de toque nunca será *True*.

class kivy.uix.scatter.Scatter(kwargs)**

Bases: *kivy.uix.widget.Widget*

Classe Scatter. Veja o módulo da documentação para maiores informações.

Events

on_transform_with_touch: Disparado quando o Scatter tiver sido transformado pelo toque ou multitoque do usuário, como panorâmica ou zoom.

on_bring_to_front: Disparado quando a dispersão é trazida para a frente.

Alterado na versão 1.9.0: Evento *on_bring_to_front* adicionado.

Alterado na versão 1.8.0: Evento *on_transform_with_touch* adicionado.

apply_transform(trans, post_multiply=False, anchor=(0, 0))

Transforma a dispersão aplicando a matriz de transformação “trans” (em cima de seu estado de transformação atual). A matriz resultante pode ser encontrada na propriedade: attr: *~Scatter.transform*.

Parameters

trans: **Matrix**. Matriz de transformação a ser aplicada ao Widget de Scatter.

anchor: tupla, o padrão é *(0, 0)*. O ponto a ser usado como origem da transformação (usa o espaço do Widget local).

post_multiply: bool, o parâmetro padrão é *False*. Se *True*, a matriz de transformação é pós-multiplicada (como se aplicada antes da transformação atual).

Exemplo de uso:

```
from kivy.graphics.transformation import Matrix
mat = Matrix().scale(3, 3, 3)
scatter_instance.apply_transform(mat)
```

auto_bring_to_front

Se *True* ‘verdadeiro’, o widget será automaticamente pressionado no topo da lista do widgets pai para desenho.

auto_bring_to_front é uma *BooleanProperty* e o padrão é *True*.

bbox

Caixa de preenchimento do Widget no espaço do pai:

```
((x, y), (w, h))  
# x, y = lower left corner
```

bbox é uma *AliasProperty*.

do_collide_after_children

Se *True*, a detecção de colisão para limitar o toque no interior do Scatter será feita depois de distribuir o toque para os filhos. Podes colocar os filhos fora da caixa delimitadora do Scatter e ainda ser capaz de tocá-los.

Novo na versão 1.3.0.

do_rotation

Permite rotação.

do_rotation é uma *BooleanProperty* e o padrão é *True*.

do_scale

Permite escalonamento.

do_scale é uma *BooleanProperty* e o padrão é *True*.

do_translation

Permite translacão sobre os eixos X e Y.

do_translation é uma *AliasProperty* de (*do_translation_x* + *do_translation_y*)

do_translation_x

Permite translacão sobre o eixo X.

do_translation_x é uma *BooleanProperty* e o parão é *True*.

do_translation_y

Permite translacão no eixo Y.

do_translation_y é uma *BooleanProperty* e o padrão é *True*.

on_bring_to_front(*touch*)

Invocado quando um evento de toque faz com que o Scatter seja trazido para a frente do pai (somente se *auto_bring_to_front* for *True*)

Parameters

touch:O objeto de toque que trouxe o Scatter para a frente.

Novo na versão 1.9.0.

on_transform_with_touch(*touch*)

Chamado quando um evento de toque transformou o Widget Scatter. Por padrão, isso não faz nada, mas pode ser substituído por classes derivadas que precisam reagir às transformações causadas pela entrada do usuário.

Parameters

touch:O objeto de toque que disparou a transformação.

Novo na versão 1.8.0.

rotation

Valor da rotação do Scatter.

rotation é uma *AliasProperty* e o padrão é 0.0.

scale

Valor de escala do Scatter.

scale é uma *AliasProperty* e o padrão é 1.0.

scale_max

Máximo fator de escala permitido.

scale_max é uma *NumericProperty* e o padrão é 1e20.

scale_min

Fator de escala mínimo permitido.

scale_min é uma *NumericProperty* e o padrão é 0.01.

transform

Matriz de transformação.

transform é uma *ObjectProperty* e o padrão é a matriz identidade.

Nota: Esta matriz reflete o estado atual da matriz de transformação, mas defini-la diretamente irá apagar transformações aplicadas anteriormente. Para aplicar uma transformação considerando o contexto, use o método *apply_transform*.

transform_inv

Inverso da matriz de transformação.;

transform_inv é uma *ObjectProperty* e o padrão para a matriz identidade.

translation_touches

Determina se a translação foi disparada por um simples ou múltiplo toque. Isso só tem efeito quando *do_translation* = True.

translation_touches é uma *NumericProperty* e o padrão é 1.

Novo na versão 1.7.0.

class kivy.uix.scatter.ScatterPlane(kwargs)**

Bases: *kivy.uix.scatter.Scatter*

Este é essencialmente um Widget Scatter ilimitado. É uma classe de conveniência para tornar mais fácil lidar com planos infinitos.

4.15.36 Scatter Layout

Novo na versão 1.6.0.

Este layout se comporta como uma classe. class:`~kivy.uix.relativelayout.RelativeLayout`. Quando se adiciona um widget na posição (0,0) para uma classe. `ScatterLayout` o widget-filho também mudará de posição quando você mudar a posição da classe 'ScatterLayout'. As coordenadas do widget-filho permanecerão em (0,0) relativos ao layout principal.

No entanto, uma vez que `ScatterLayout` é implementado usando um Widget `Scatter`, também podes transladar, girar e dimensionar o layout usando toques ou cliques, como no caso de um Widget `Scatter` normal e os Widgets filho se comportarão conforme o esperado.

Em contraste com um `Scatter`, o layout favorece as propriedades de 'hint', como `size_hint`, `size_hint_x`, `size_hint_y` e `pos_hint`.

Nota: The `ScatterLayout` está implementado como uma `FloatLayout` dentro de um `Scatter`.

Aviso: Uma vez que o atual `ScatterLayout` é uma `Scatter`, as suas funções `add_widget` e `remove_widget` são substituídas para adicionar aos filhos à classe incorporada automaticamente `FloatLayout` (Acessível como propriedade `content` de `Scatter`) . Então, se quiseres acessar os elementos filho adicionados, precisas de `self.content.children` ao invés de `self.children`.

Aviso: A `ScatterLayout` foi introduzido na versão 1.7.0 e foi chamado de `RelativeLayout` nas versões anteriores. O `RelativeLayout` é agora uma implementação otimizada que usa apenas uma transformação posicional para evitar alguns dos cálculos mais pesados envolvidos com o `Scatter`.

`class kivy.uix.scatterlayout.ScatterLayout(**kw)`
Bases: `kivy.uix.scatter.Scatter`

Classe `ScatterLayout`, veja o módulo da documentação para maiores informações.

`class kivy.uix.scatterlayout.ScatterPlaneLayout(**kwargs)`
Bases: `kivy.uix.scatter.ScatterPlane`

Classe `ScatterPlaneLayout`, veja o módulo da documentação para maiores informações.

Semelhante ao `ScatterLayout`, mas baseado no `ScatterPlane` - portanto, a entrada não é limitada.

Novo na versão 1.9.0.

4.15.37 Gerenciador de Vídeo

Novo na versão 1.4.0.

O ScreenManager é um Widget dedicado ao gerenciamento de múltiplas telas para sua aplicação. Por padrão o **ScreenManager** mostra somente um **Screen** por vez e utiliza um **TransitionBase** para alterar entre um Screen e outro.

Múltiplas transições são suportadas com base na alteração das coordenadas / escala do Screen (tela) ou mesmo na execução de animações que fazem uso de shaders personalizados.

Uso básico

Vamos construir um ScreenManager com 3 nomes de seções. Quando você criar um Screen, **você absolutamente precisará dar um nome a ele**:

```
from kivy.uix.screenmanager import ScreenManager, Screen

# Create the manager
sm = ScreenManager()

# Add few screens
for i in range(4):
    screen = Screen(name='Title %d' % i)
    sm.add_widget(screen)

# By default, the first screen added into the ScreenManager will be
# displayed. You can then change to another screen.

# Let's display the screen named 'Title 2'
# A transition will automatically be used.
sm.current = 'Title 2'
```

O padrão **ScreenManager.transition** é um **SlideTransition** com opções **direction** e **duration**.

Por favor, note que por padrão, um **Screen** não é exibido: isso é somente um **RelativeLayout**. Você precisa utilizar essa classe como um Widget principal para a sua própria tela, a melhor forma será escrevendo uma subclasse.

Aviso: Como **Screen** é uma **RelativeLayout**, é importante entender o [Armadilhas Comuns](#).

Aqui está um exemplo com um 'Menu Screen' e um 'Settings Screen':

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager, Screen

# Create both screens. Please note the root.manager.current: this
# is how
# you can control the ScreenManager from kv. Each screen has by
# default a
# property manager that gives you the instance of the ScreenManager
# used.
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'Goto settings'
            on_press: root.manager.current = 'settings'
        Button:
            text: 'Quit'

<SettingsScreen>:
    BoxLayout:
        Button:
            text: 'My settings button'
        Button:
            text: 'Back to menu'
            on_press: root.manager.current = 'menu'
""")

# Declare both screens
class MenuScreen(Screen):
    pass

class SettingsScreen(Screen):
    pass

# Create the screen manager
sm = ScreenManager()
sm.add_widget(MenuScreen(name='menu'))
sm.add_widget(SettingsScreen(name='settings'))

class TestApp(App):

    def build(self):
        return sm

if __name__ == '__main__':
    TestApp().run()
```

Alterando a Direção

Um uso comum para *ScreenManager* envolve o uso de um *SlideTransition* que desliza da direita para a próxima tela e desliza da esquerda para a tela anterior. Com base no exemplo anterior, podemos fazer isso da seguinte maneira:

```
Builder.load_string("""
<MenuScreen>:
    BoxLayout:
        Button:
            text: 'Goto settings'
            on_press:
                root.manager.transition.direction = 'left'
                root.manager.current = 'settings'
        Button:
            text: 'Quit'

<SettingScreen>:
    BoxLayout:
        Button:
            text: 'My settings button'
        Button:
            text: 'Back to menu'
            on_press:
                root.manager.transition.direction = 'right'
                root.manager.current = 'menu'
""")
```

Uso avançando

Desde a versão 1.8.0 é possível trocar dinamicamente por uma nova tela, alterar as opções de transição e remover a anterior utilizando *switch_to()*:

```
sm = ScreenManager()
screens = [Screen(name='Title {}'.format(i)) for i in range(4)]

sm.switch_to(screens[0])
# later
sm.switch_to(screens[1], direction='right')
```

Note que este método adiciona a tela para a instância do *ScreenManager* e não deve ser utilizada se a tela já tiver sido adicionado a essa instância. Para trocar para a tela que já está adicionada, utilize a propriedade *current*.

Alterando a transição

Você tem múltiplas transições disponíveis por padrão, como exemplo:

- **NoTransition** - troca a tela instantaneamente com nenhuma transição.
- **SlideTransition** - desliza a tela pra dentro/prá fora, de qualquer direção
- **CardTransition** - new screen slides on the previous or the old one slides off the new one depending on the mode
- **SwapTransition** - implementação da transição de troca do iOS
- **FadeTransition** - shader para enfraquecer a tela na entrada/saída.
- **WipeTransition** - shader para limpar a tela da direita para a esquerda
- **FallOutTransition** - shader onde a tela antiga ‘cai’ e torna-se transparente, revelando a nova tela detrás desta.
- **RiseInTransition** - shader onde a nova tela sobre do centro da tela enquanto desaparece de transparente para opaco.

Você pode facilmente trocar a transição alterando a propriedade **ScreenManager.transition**:

```
sm = ScreenManager(transition=FadeTransition())
```

Nota: Atualmente, nenhuma transição baseada no Shader utiliza anti-aliasing. Isto porque, eles usam o FBO que não tem qualquer lógica para lidar com supersampling. Este é um problema conhecido e estamos trabalhando em uma implementação transparente que dará os mesmos resultados como se tivesse sido processado na tela.

Para ser mais preciso, se você ?afiado? durante a animação, isso é normal. (To be more concrete, if you see sharp edged text during the animation, it's normal).

```
class kivy.uix.screenmanager.Screen(**kw)
    Bases: kivy.uix.relativelayout.RelativeLayout
```

Screen é um elemento destinado a ser usado com uma **ScreenManager**. Verifique a documentação do módulo para obter mais informações.

Events

on_pre_enter: () Evento disparado quando a tela estiver prestes a ser utilizada: a animação de entrada iniciada.

on_enter: () Evento disparado quando a tela é exibida: a animação de entrada está completa.

on_pre_leave: () Evento disparado quando a tela está prestes a ser removida: a animação de saída é iniciada.

on_leave: ()Evento disparado quando a tela é removidas: a animação de saída está concluída.

Alterado na versão 1.6.0: Eventos *on_pre_enter*, *on_enter*, *on_pre_leave* e *on_leave* foram adicionados.

manager

objeto *ScreenManager*, definido quando a tela é adicionada a um gerenciador.

manager é um *ObjectProperty* e o padrão é *None*, somente leitura.

name

Nome da tela que deve ser o único dentro do *ScreenManager*. Este nome é utilizado por *ScreenManager.current*.

name é um *StringProperty* e o padrão é ''.

transition_progress

Valor que representa a conclusão da transição atual, caso exista alguma.

Se uma transição estiver em andamento, qualquer que seja o modo, o valor mudará de 0 para 1. Se você quiser saber se é uma animação de entrada ou saída, verifique *transition_state*.

transition_progress é um *NumericProperty* e o padrão é 0.

transition_state

Valor que representa o estado de transição:

- 'in' se a transição será exibida na tela
- 'out' se a transição irá esconder a sua tela

Depois que a transição for concluída, o estado reterá seu último valor (dentro ou fora).

transition_state é um *OptionProperty* e o valor padrão é 'out'.

class kivy.uix.screenmanager.ScreenManager(kwargs)**

Bases: *kivy.uix.floatlayout.FloatLayout*

ScreenManager. Esta é a classe principal que controlará a sua pilha de *Screen* e memória.

Por padrão, o gerenciador mostrará somente uma tela por vez.

current

Nome da tela exibido ou nome da tela a ser exibido.

```
from kivy.uix.screenmanager import ScreenManager, Screen

sm = ScreenManager()
sm.add_widget(Screen(name='first'))
sm.add_widget(Screen(name='second'))
```

```
# By default, the first added screen will be shown. If you  
→want to  
# show another one, just set the 'current' property.  
sm.current = 'second'
```

current é um *StringProperty* e o valor padrão é *None*.

current_screen

Contém a tela atualmente exibida. Você não pode alterar essa propriedade manualmente, ao invés disso, utilize **current**.

current_screen é um *ObjectProperty* e o valor parão é *None*, somente leitura.

get_screen(name)

Retorna o Widget da tela associado com o nome ou levanta uma exceção *ScreenManagerException* caso não seja encontrado.

has_screen(name)

Retorna *True* se um tela com o *name* for encontrada.

Novo na versão 1.6.0.

next()

Retorna o nome da próxima tela da lista de Screen's.

previous()

Retorna o nome da tela anterior da lista de Screens.

screen_names

Lista de nomes de todas os Widgets *Screen* adicionados. A lista é somente leitura.

screens_names é um *AliasProperty* e é somente leitura. Ele é atualizado se a lista de Screen for alterada ou o nome da tela for alterado.

screens

Lista de todos os Widgets *Screen* adicionados. Você não deve alterar essa lista manualmente. Ao invés disso, utilize o método **add_widget**.

screens é um *ListProperty* e o padrão é *[]*, somente leitura.

switch_to(screen, **options)

Adiciona uma nova tela ao ScreenManager e altera para este. O Screen anterior será removido dos filhos. *option* são as opções de **transition** que serão alteradas depois da animação acontecer.

Se nenhuma Screen anterior estiver disponível, a Screen que será utilizadas será a principal:

```
sm = ScreenManager()  
sm.switch_to(screen1)
```

```
# later
sm.switch_to(screen2, direction='left')
# later
sm.switch_to(screen3, direction='right', duration=1.)
```

Se nenhuma animação estiver em progresso, ele será parado e substituído por esta: evite isso porque a animação ficará estranha. Utilize a função `switch_to()` ou `current` mas não as duas.

O nome do `screen` será alterado se houver qualquer conflito com o Screen atual.

transition

Objeto de transição utilizado para animar a transição da tela atual para a próxima que será exibida.

Por exemplo, se desejas usar um `WipeTransition` entre os slides:

```
from kivy.uix.screenmanager import ScreenManager, Screen,
WipeTransition

sm = ScreenManager(transition=WipeTransition())
sm.add_widget(Screen(name='first'))
sm.add_widget(Screen(name='second'))

# by default, the first added screen will be shown. If you
# want to
# show another one, just set the 'current' property.
sm.current = 'second'
```

`transition` é um `ObjectProperty` e o padrão é `SlideTransition`.

Alterado na versão 1.8.0: Transição padrão será alterada de `SwapTransition` para `SlideTransition`.

class kivy.uix.screenmanager.ScreenManagerException

Bases: `exceptions.Exception`

Exceção para a classe `ScreenManager`.

class kivy.uix.screenmanager.TransitionBase

Bases: `kivy.event.EventDispatcher`

`TransitionBase` é utilizada para animar 2 Screen dentro da classe `ScreenManager`. Essa classe age como base para as implementações como o `SlideTransition` e `SwapTransition`.

Events

`on_progress: Objeto de transição, flutuação de progresso`

Disparado durante a animação da transição.

on_complete: Objeto de Transição Disparado quando a transição é finalizada.

add_screen(*screen*)

(interno) Utilizado para adicionar a tela ao *ScreenManager*.

duration

Duração em segundos da transição.

duration é um *NumericProperty* e o padrão é .4 (= 400ms).

Alterado na versão 1.8.0: Duração padrão foi alterada para de 700ms para 400ms.

is_active

Indica se a transição está atualmente ativa ou não.

is_active é um *BooleanProperty* e o padrão é *False*, somente leitura.

manager

objeto *ScreenManager*, definido quando a tela é adicionada a um gerenciador.

manager é um *ObjectProperty* e o padrão é *None*, somente leitura.

remove_screen(*screen*)

(interno) Utilizado para remover uma tela do *ScreenManager*.

screen_in

Propriedade que contém a tela a ser mostrada. Automaticamente definido pela classe *ScreenManager*.

screen_in é um *ObjectProperty* e o padrão é *None*.

screen_out

Propriedade que contém a tela a ser ocultada. Automaticamente definido pelo *ScreenManager*.

screen_out é um *ObjectProperty* e o padrão é *None*.

start(*manager*)

(interno) Inicia a transição. Isso é automaticamente invocado pela classe *ScreenManager*.

stop()

(interno) Para a transição. Isso é automaticamente chamado por *ScreenManager*.

class kivy.uix.screenmanager.ShaderTransition

Bases: *kivy.uix.screenmanager.TransitionBase*

Classe de transição que usa um Shader para animar a transição entre 2 telas. Por padrão, esta classe não atribui qualquer fragmento/vértice Shader. Se você quiser criar seu próprio fragmentos de Shader para a transição, precisas declarar o cabeçalho e incluir o “t”, “tex_in” and “tex_out” uniformemente:

```

# Create your own transition. This shader implements a "fading"
# transition.
fs = """$HEADER
    uniform float t;
    uniform sampler2D tex_in;
    uniform sampler2D tex_out;

    void main(void) {
        vec4 cin = texture2D(tex_in, tex_coord0);
        vec4 cout = texture2D(tex_out, tex_coord0);
        gl_FragColor = mix(cout, cin, t);
    }
"""

# And create your transition
tr = ShaderTransition(fs=fs)
sm = ScreenManager(transition=tr)

```

clearcolor

Defini a cor do Fbon *ClearColor*.

Novo na versão 1.9.0.

clearcolor é um *ListProperty* e o padrão é [0, 0, 0, 1].

fs

Fragmento de shader pra utilizar.

fs é um *StringProperty* e o padrão é *None*.

vs

Vertex shader para usar.

vs é um *StringProperty* e o padrão é *None*.

class kivy.uix.screenmanager.SlideTransition

Bases: *kivy.uix.screenmanager.TransitionBase*

O Slide Transition pode ser utilizado para mostrar uma nova tela de qualquer direção: left, right, up ou down.

direction

Direção da transição.

direction é um *OptionProperty* e o padrão é 'left'. Pode ser um das opções 'left', 'right', 'up' ou 'down'.

class kivy.uix.screenmanager.SwapTransition(kwargs)**

Bases: *kivy.uix.screenmanager.TransitionBase*

Swap transition semelhante com a transição do iOS quando uma nova janela aparece na tela.

class kivy.uix.screenmanager.FadeTransition
Bases: *kivy.uix.screenmanager.ShaderTransition*

Fade transition, com base em um fragmento Shader.

class kivy.uix.screenmanager.WipeTransition
Bases: *kivy.uix.screenmanager.ShaderTransition*

Wipe transition, com base em um fragmento Shader.

class kivy.uix.screenmanager.FallOutTransition
Bases: *kivy.uix.screenmanager.ShaderTransition*

Transição onde a nova tela ‘cai’ do centro da tela, tornando-se maior e mais transparente até desaparecer, revendo uma nova tela por detrás. Imita a transição padrão/popular do Android.

Novo na versão 1.8.0.

duration

Duração em segundos de um transição, substituindo o padrão de *TransitionBase*.

duration é um *NumericProperty* e o padrão é .15 (= 150ms).

class kivy.uix.screenmanager.RiseInTransition
Bases: *kivy.uix.screenmanager.ShaderTransition*

Transição onde a nova tela sobe do centro da tela, tornando-se maior e mudando de transparente para opaco até completar a tela. Imita a popular/padrão transição do Android.

Novo na versão 1.8.0.

duration

Duração em segundos de um transição, substituindo o padrão de *TransitionBase*.

duration é um *NumericProperty* e o padrão é .2 (= 200ms).

class kivy.uix.screenmanager.NoTransition
Bases: *kivy.uix.screenmanager.TransitionBase*

Sem transição, muda instantaneamente para a próxima tela sem delay ou animação.

Novo na versão 1.8.0.

class kivy.uix.screenmanager.CardTransition
Bases: *kivy.uix.screenmanager.SlideTransition*

Card transition that looks similar to Android 4.x application drawer interface animation.

It supports 4 directions like SlideTransition: left, right, up and down, and two modes, pop and push. If push mode is activated, the previous screen does not

move, and the new one slides in from the given direction. If the pop mode is activated, the previous screen slides out, when the new screen is already on the position of the ScreenManager.

Novo na versão 1.10.

mode

Indicates if the transition should push or pop the screen on/off the ScreenManager.

- ‘push’ means the screen slides in in the given direction
- ‘pop’ means the screen slides out in the given direction

mode is an *OptionProperty* and defaults to ‘push’.

start(*manager*)

(interno) Inicia a transição. Isso é automaticamente invocado pela classe *ScreenManager*.

4.15.38 ScrollView

Novo na versão 1.0.4.

O widget *ScrollView* fornece uma viewport de rolagem/panorâmica que é cortada na caixa delimitadora do scrollview.

Comportamento de rolagem

O ScrollView aceita apenas um filho e aplica uma janela de exibição/janela de acordo com as propriedades: attr: *~ScrollView.scroll_x* e :attr: *~ScrollView.scroll_y*. Os toques são analisados para determinar se o usuário deseja rolar ou controlar o filho de alguma outra maneira: você não pode fazer as duas coisas ao mesmo tempo. Para determinar se a interação é um ato de rolagem, essas propriedades são usadas:

- *scroll_distance*: a distância mínima para deslocar, o padrão é de 20 pixels.
- *scroll_timeout*: o período de tempo máximo, o padrão é 55 milissegundos.

Se um toque viaja *scroll_distance* pixels dentro do período *scroll_timeout*, o mesmo será reconhecido como um movimento de rolagem e a tradução (scroll / pan) começará. Se o tempo limite ocorrer, o evento TouchDown será despachado para os filhos em vez disso (sem tradução).

O valor padrão para essas configurações pode ser alterado no arquivo de configuração:

```
[widgets]
scroll_timeout = 250
scroll_distance = 20
```

Novo na versão 1.1.1: ScrollView agora anima a rolagem em Y quando a roda do mouse é usada.

Limitando ao eixo X ou Y

Por padrão, o ScrollView permite a rolagem ao longo dos eixos X e Y. Você pode explicitamente desabilitar a rolagem em um eixo definindo as propriedades: attr: `~ScrollView.do_scroll_x` ou: attr: `~ScrollView.do_scroll_y`' para False.

Gerenciando o tamanho e a posição do conteúdo

O ScrollView gerencia a posição de seus filhos de forma semelhante a um `RelativeLayout` mas não usa o `size_hint`. Deve especificar cuidadosamente com `size` do seu conteúdo para obter o efeito de rolagem/pan desejado.

Por padrão, o attr: `~kivy.uix.widget.Widget.size_hint` é (1, 1), então o tamanho do conteúdo se encaixa exatamente no ScrollView (você não terá nada para rolar). Você deve desativar pelo menos uma das instruções `size_hint` (x ou y) do filhos para habilitar a rolagem. A configuração: attr: `~kivy.uix.widget.Widget.size_hint_min` para não ser None também habilitará a rolagem para essa dimensão quando a classe: `ScrollView` for menor que o tamanho mínimo.

Para rolar uma classe: `~kivy.uix.gridlayout.GridLayout` no seu eixo Y/verticalmente, defina a largura do filho para a do ScrollView (`size_hint_x = 1`) e defina a propriedade `size_hint_y` como None:

```
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.scrollview import ScrollView
from kivy.core.window import Window
from kivy.app import runTouchApp

layout = GridLayout(cols=1, spacing=10, size_hint_y=None)
# Make sure the height is such that there is something to scroll.
layout.bind(minimum_height=layout.setter('height'))
for i in range(100):
    btn = Button(text=str(i), size_hint_y=None, height=40)
    layout.add_widget(btn)
root = ScrollView(size_hint=(1, None), size=(Window.width, Window.
    height))
root.add_widget(layout)

runTouchApp(root)
```

Efeitos de deslocamento

Novo na versão 1.7.0.

Quando a rolagem exceder os limites da `ScrollView`, a mesma usará a `ScrollEffect` para lidar com o *overscroll*. Esses efeitos podem executar ações como saltar para trás, alterar a opacidade ou simplesmente impedir a rolagem além dos limites normais. Note que os efeitos complexos podem executar muitos cálculos, que podem ser lentos em hardware mais fraco.

Você pode alterar o efeito que está sendo usado ao definir `effect_cls` para qualquer classe de efeito. As opções atuais incluem:

- `ScrollEffect`: Does not allow scrolling beyond the `ScrollView` boundaries.
- `DampedScrollEffect`: The current default. Allows the user to scroll beyond the normal boundaries, but has the content spring back once the touch/click is released.
- `OpacityScrollEffect`: Semelhante ao `DampedScrollEffect`, mas também reduz a opacidade durante overscroll.

Você também pode criar seu próprio efeito de rolagem subclassificando um destes e, em seguida, passá-lo como: attr: `~ScrollView.effect_cls` da mesma maneira.

Alternativamente, você pode definir: attr: `~ScrollView.effect_x` e/ou: attr: `~ScrollView.effect_y` para uma *instância* do efeito que você deseja usar. Isso irá substituir o efeito padrão definido em: attr: `~ScrollView.effect_cls`.

Todos os efeitos estão localizados em `kivy.effects`.

```
class kivy.uix.scrollview.ScrollView(**kwargs)
    Bases: kivy.uix.stencilview.StencilView
```

Classe ScrollView. Consulte a documentação do módulo para mais informação.

Events

`on_scroll_start` Evento genérico disparado quando começa a rolagem a partir do toque.

`on_scroll_move` Evento genérico disparado quando a rolagem move do toque.

`on_scroll_stop` Evento genérico disparado quando a rolagem pára por causa de um toque.

Alterado na versão 1.9.0: `on_scroll_start`, `on_scroll_move` e `on_scroll_stop` os eventos agora são despachados ao rolar pra manipular com ScrollViews aninhados.

Alterado na versão 1.7.0: `auto_scroll`, `scroll_friction`, `scroll_moves`, `scroll_stoptime` foram reprovadas, use: attr: `effect_cls` ao invés disso.

bar_color

Cor da barra de rolagem horizontal/vertical, no formato RGBA.

Novo na versão 1.2.0.

bar_color é uma *ListProperty* e o padrão é [.7, .7, .7, .9].

bar_inactive_color

Cor da barra de rolagem horizontal/vertical (no formato RGBA), quando nenhuma rolagem está acontecendo.

Novo na versão 1.9.0.

bar_inactive_color é uma *ListProperty* e o padrão é [.7, .7, .7, .2].

bar_margin

Margem entre o lado inferior/direito do scrollview ao desenhar a barra de rolagem horizontal/vertical.

Novo na versão 1.2.0.

bar_margin é uma *NumericProperty*, e o padrão é 0

bar_pos

Qual lado da janela de rolagem para colocar cada uma das barras.

bar_pos é uma *ReferenceListProperty* de (*bar_pos_x*, *bar_pos_y*)

bar_pos_x

Qual lado do ScrollView a barra de rolagem horizontal devia continuar. Os valores possíveis são ‘superior’ e ‘inferior’.

Novo na versão 1.8.0.

bar_pos_x é uma *OptionProperty*, o padrão é ‘inferior’.

bar_pos_y

Qual lado do ScrollView a barra de rolagem vertical devia continuar. Os valores possíveis são ‘esquerda’ e ‘direita’.

Novo na versão 1.8.0.

bar_pos_y é uma *OptionProperty* e o padrão é ‘direita’.

bar_width

Largura da barra de rolagem horizontal/vertical. A largura é interpretada como uma altura para a barra horizontal.

Novo na versão 1.2.0.

bar_width é uma *NumericProperty* e o padrão é 2.

convert_distance_to_scroll(dx, dy)

Converte uma distância em pixels para a distância em rolagem, dependendo do tamanho do conteúdo e do tamanho da scrollview.

O resultado será uma tupla da distância de rolagem que pode ser adicionada a *scroll_x* and *scroll_y*

do_scroll

Permite rolagem no eixo X ou Y.

do_scroll é uma *AliasProperty* de (*do_scroll_x* + *do_scroll_y*)

do_scroll_x

Permite rolagem no eixo X.

do_scroll_x é uma *BooleanProperty* e o padrão é True.

do_scroll_y

Permite rolagem no eixo Y.

do_scroll_y é uma *BooleanProperty* e o padrão é True.

effect_cls

Classe efeito para instanciar para o eixo X e Y.

Novo na versão 1.7.0.

effect_cls é uma *ObjectProperty* e o padrão é DampedScrollEffect.

Alterado na versão 1.8.0: Se você definir uma string, a *Factory* será usada para resolver a classe.

effect_x

Efeito para aplicar para o eixo X. Se None for definido, será criada uma instância de *effect_cls*.

Novo na versão 1.7.0.

effect_x é uma *ObjectProperty* e o padrão é None.

effect_y

Efeito para aplicar para o eixo Y. Se None for definido, será criada uma instância de *effect_cls*.

Novo na versão 1.7.0.

effect_y é uma *ObjectProperty* e o padrão é None, somente leitura.

hbar

Retorna uma tupla de (posição, tamanho) da barra de rolagem horizontal.

Novo na versão 1.2.0.

A posição e o tamanho são normalizados entre 0-1 e representam um porcentagem da altura atual do ScrollView. Esta propriedade é usada internamente para desenhar a pequena barra horizontal quando estas deslocando.

vbar é uma *AliasProperty*, somente leitura.

scroll_distance

Distância para mover antes da rolagem: classe: *ScrollView*, em pixels. Assim que a distância for percorrida, a classe: *ScrollView* começará a rolar, e

nenhum evento de toque irá para os filhos. É aconselhável basear esse valor no dpi da tela do dispositivo de destino.

scroll_distance é uma *NumericProperty* e o padrão é 20 (pixels), de acordo com o valor default na configuração do usuário.

scroll_timeout

Tempo limite permitido para acionar o *scroll_distance*, em milissegundos. Se o usuário não tiver movido *scroll_distance* dentro do tempo limite, a rolagem será desativada e o evento de toque irá para os filhos.

scroll_timeout is a *NumericProperty* and defaults to 55 (milliseconds) according to the default value in user configuration.

Alterado na versão 1.5.0: Valor padrão alterado de 250 para 55.

scroll_to (widget, padding=10, animate=True)

Rola a janela de visualização para garantir que o Widget fornecido seja visível, opcionalmente com preenchimento e animação. Se Animate for *True* (o padrão), então os parâmetros de animação padrão serão usados. Caso contrário, deve ser um dict contendo argumentos para passar para o construtor *Animation*.

Novo na versão 1.9.1.

scroll_type

Define o tipo de rolagem a ser usado para o conteúdo da visualização de rolagem. As opções disponíveis são: ['conteúdo'], ['barras'], ['barras', 'conteúdo'].

Novo na versão 1.8.0.

scroll_type é uma *OptionProperty*, o padrão é ['content'].

scroll_wheel_distance

Distância para mover quando rolar com a roda do mouse. É aconselhável basear esse valor no dpi da tela do dispositivo de destino.

Novo na versão 1.8.0.

scroll_wheel_distance é uma *NumericProperty*, o padrão é 20 pixels.

scroll_x

Valor de rolagem X, entre 0 e 1. Se 0, o conteúdo do lado esquerdo irá tocar o lado esquerdo da ScrollView. Se 1, o conteúdo do lado direito irá tocar o lado direito.

Esta propriedade é controlada por *ScrollView* apenas se *do_scroll_x* é *True*.

scroll_x é uma *NumericProperty* e o padrão é 0.

scroll_y

Valor de rolagem Y, entre 0 e 1. Se 0, o conteúdo do lado inferior irá tocar o lado inferior da ScrollView. Se 1, o conteúdo do lado superior irá tocar o lado superior.

Esta propriedade é controlada por *ScrollView* apenas se *do_scroll_y* é True.

scroll_y é uma *NumericProperty* e o padrão é 1.

update_from_scroll(*args)

Força a reposição do conteúdo, de acordo com o valor atual de *scroll_x* e *scroll_y*.

Este método é automaticamente chamado quando uma das propriedades *scroll_x*, *scroll_y*, *pos* ou *size* mudam, ou se o tamanho do conteúdo é alterado.

vbar

Retorna uma tupla de(posição, tamanho) da barra de rolagem vertical.

Novo na versão 1.2.0.

A posição e o tamanho são normalizados entre 0-1 e representam uma porcentagem da altura atual da scrollview. Essa propriedade é usada internamente para desenhar a pequena barra vertical quando estiver rolando.

vbar é uma *AliasProperty*, somente leitura.

viewport_size

(interno) Tamanho da janela de visualização. Este é o tamanho de sua única filha no scrollview.

4.15.39 SelectableView

Nota: The feature has been deprecated.

O módulo abriga a classe *SelectableView*. É usado pela classe *ListView* e é associado ao módulo *Adapters* para fornecer diferentes comportamentos de seleção ao apresentar listas grandes.

class kivy.uix.selectableview.SelectableView(*args, **kwargs)
Bases: object

A *SelectableView* é um mixin usado com itens de lista e outras classes que devem ser instanciadas em uma exibição de lista ou outras classes que usam um adaptador habilitado para seleção comoListAdapter. *select()* e *deselect()* podem ser sobreescrito com o código de exibição para marcar itens selecionados ou não, se desejado.

deselect(*args)

O item de lista é responsável por atualizar a exibição quando estiver sendo desmarcada, se desejado.

index

O índice na lista de dados subjacente ou o item de dados que essa exibição representa.

is_selected

Uma instância *SelectableView* carrega essa propriedade que deve ser mantida em sincronia com a propriedade equivalente que o item de dados representa.

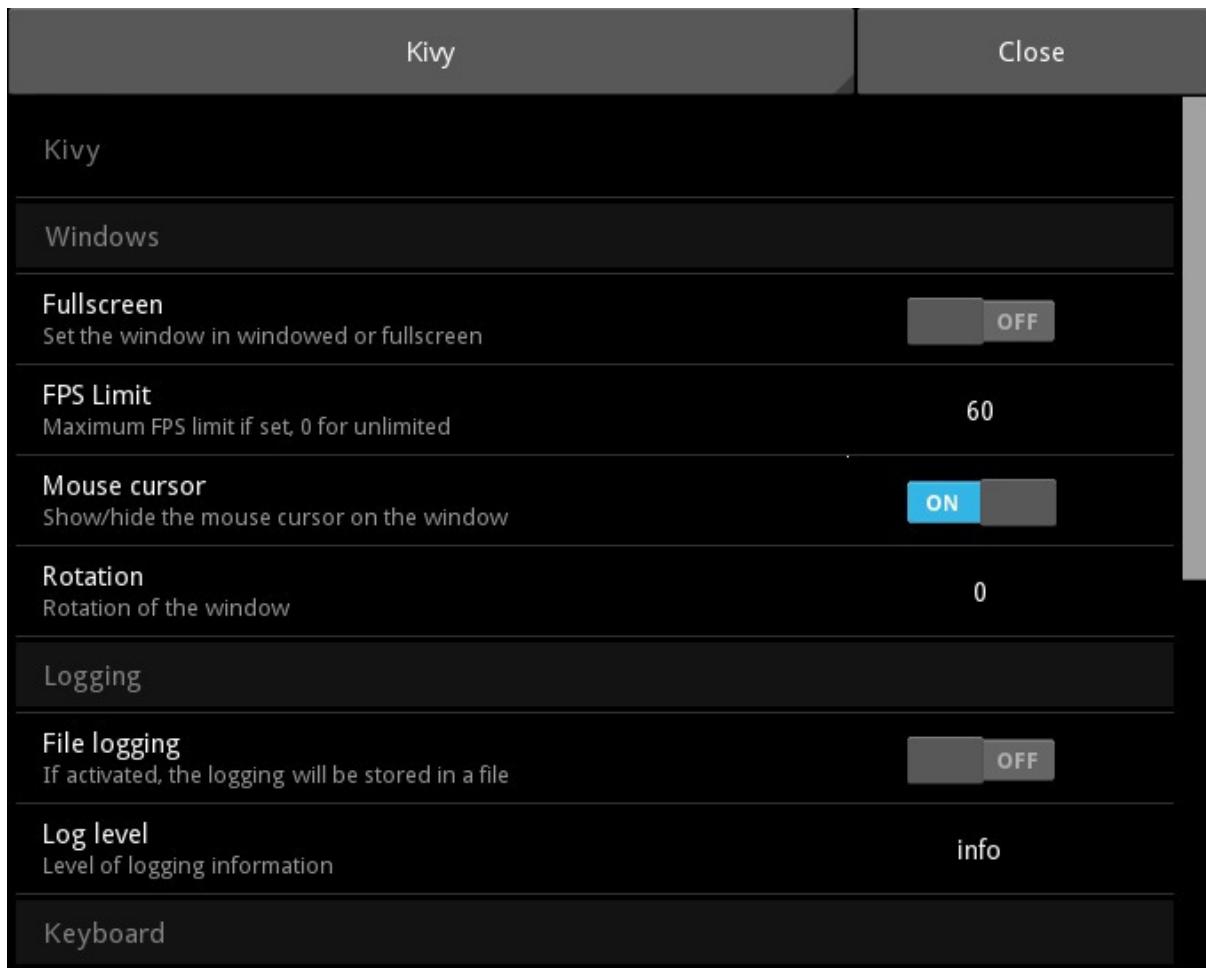
select(*args)

O item da lista é responsável por atualizar a exibição quando estiver selecionado, se desejado.

4.15.40 Configurações

Novo na versão 1.0.7.

Este módulo fornece um completo e extensível framework para adicionar uma interface para configurações de sua aplicação. Por padrão, a interface usa *SettingsWithSpinner*, que consiste em uma *Spinner* (top) para alternar entre painéis de configurações (bottom). Consulte *Diferentes layouts de painel* para conhecer algumas alternativas.



Uma *SettingsPanel* representa um grupo de opções configuráveis. A propriedade *SettingsPanel.title* é usado por *Settings* quando o painel é adicionado: ele determina o nome do botão da barra lateral. *SettingsPanel* controla uma instância de *ConfigParser*.

O painel pode ser construído automaticamente a partir de um arquivo de definição JSON: você descreve as configurações desejadas e as respectivas seções/chaves na instância de ConfigParser... e pronto!

As configurações também são classes *App* class. Utilize *Settings.add_kivy_panel()* para configurar o núcleo de configurações do painel do Kivy.

Cria um painel a partir do JSON

Para criar um painel desde um arquivo JSON, você precisa de 2 coisas:

- uma instância *ConfigParser* com os valores padrões
- um arquivo JSON

Aviso: A `kivy.config.ConfigParser` é necessária. Não podes usar o `ConfigParser` padrão das bibliotecas do Python.

Você deve criar e manipular o objeto `ConfigParser`. `SettingsPanel` irá ler os valores da instância do `ConfigParser` associada. Verifique se definistes os valores padrão (usando `setdefaults`) para todas as sections/keys em seu arquivo JSON!

O arquivo JSON contém informações estruturadas descrevendo as configurações disponíveis. Aqui temos um exemplo:

```
[  
  {  
    "type": "title",  
    "title": "Windows"  
  },  
  {  
    "type": "bool",  
    "title": "Fullscreen",  
    "desc": "Set the window in windowed or fullscreen",  
    "section": "graphics",  
    "key": "fullscreen",  
    "true": "auto"  
  }  
]
```

Cada elemento na lista raiz representa uma configuração que o usuário pode configurar. Somente a key “type” é obrigatória: uma instância da classe associada será criada e usada para a configuração - outras keys são atribuídas às propriedades correspondentes dessa classe.

Tipo	Classes Associadas
título	<code>SettingTitle</code>
bool	<code>SettingBoolean</code>
numeric	<code>SettingNumeric</code>
options	<code>SettingOptions</code>
string	<code>SettingString</code>
path	<code>SettingPath</code>

Novo na versão 1.1.0: Adiciona o tipo `SettingPath`

No exemplo JSON acima, o primeiro elemento é do tipo “title”. Ele criará uma nova instância de `SettingTitle` e aplicará o restante dos pares chave-valor às propriedades dessa classe, por exemplo, “title”: “Windows” define a propriedade `title` do painel do/para “Windows”.

Para abrir um exemplo de JSON para uma instância de `Settings`, utilize o método `Settings.add_json_panel()`. Isso instanciará automaticamente um `SettingsPanel` e adicionará o mesmo a `Settings`:

```

from kivy.config import ConfigParser

config = ConfigParser()
config.read('myconfig.ini')

s = Settings()
s.add_json_panel('My custom panel', config, 'settings_custom.json')
s.add_json_panel('Another panel', config, 'settings_test2.json')

# then use the s as a widget...

```

Diferentes layouts de painel

Uma [App](#) Kivy pode criar automaticamente e mostrar uma instância de [`Settings`](#) instance. Veja a documentação [`settings_cls`](#) para obter mais informações sobre como escolher a classe de configurações a ser exibida.

Vários Widgets de configurações pré-criados estão disponíveis. Todos exceto [`SettingsWithNoMenu`](#) incluem botões para o fechamento que acionam que invocam `on_close`.

- [`Settings`](#): Exibe as configurações com uma barra lateral à esquerda para alternar entre painéis JSON.
- [`SettingsWithSidebar`](#): uma subclasse trivial de [`Settings`](#).
- [`SettingsWithSpinner`](#): Exibe as configurações com um Spinner na parte superior, que pode ser usado para alternar entre painéis JSON. Usado [`InterfaceWithSpinner`](#) como o [`interface_cls`](#). Esse é o comportamento padrão do Kivy 1.8.0.
- [`SettingsWithTabbedPanel`](#): exibe painéis JSON como guias individuais em um [`TabbedPanel`](#). Utilize [`InterfaceWithTabbedPanel`](#) como o [`interface_cls`](#).
- [`SettingsWithNoMenu`](#): exibe um único painel JSON, sem nenhuma maneira de alternar para outros painéis e sem botão de fechamento. Isso torna impossível que o usuário saia a menos que [`close_settings\(\)`](#) seja herdado com um gatilho de fechamento diferente! Utilize [`InterfaceWithNoMenu`](#) como o [`interface_cls`](#).

Podes construir seus próprios painéis de configurações com qualquer layout que venhas a escolher, definindo [`Settings.interface_cls`](#). Este deve ser um Widget que exibe um painel de configurações JSON disponibilizando uma forma de alternar entre painéis. Uma instância será criada automaticamente por [`Settings`](#).

Interface Widgets pode ser qualquer coisa que você goste, mas *deverá* ter um método `add_panel` que recebe o JSON recém-criado com as configurações dos painéis para exibir a interface. Consulte a documentação para [`InterfaceWithSidebar`](#) obter mais

informações. Eles podem, opcionalmente, invocar o evento `on_close`, por exemplo, caso o botão de fechamento for clicado. Este evento é usado pela `Settings` para acionar seu próprio evento `on_close`.

Para um exemplo completo, por favor veja `kivy/examples/settings/main.py`.

```
class kivy.uix.settings.Settings(*args, **kargs)
    Bases: kivy.uix.boxlayout.BoxLayout
```

Configuração da UI. Veja o módulo da documentação para maiores informações de como usar essa classe.

Events

`on_config_change`: instância de `ConfigParser`, `section`, `key`, `value`

Disparado quando o par key-value da seção de um `ConfigParser` é alterado.

`on_close` Disparado quando o botão Close for pressionado.

`add_interface()`

(Internal) cria uma instância de `Settings.interface_cls`, e define-a como `interface`. Quando os painéis JSON forem criados, estes serão adicionados a esta relação que os exibirá ao usuário.

`add_json_panel(title, config, filename=None, data=None)`

Cria e adiciona uma nova classe:`SettingsPanel` usando a configuração `config` com a definição JSON `filename`.

Verifica a seção `Cria um painel a partir do JSON` na documentação para obter mais informações sobre o formato JSON e como utilizar esta função.

`add_kivy_panel()`

Adiciona um painel para configurar o Kivy. Este painel age diretamente na configuração do Kivy. Sinta-se livre para incluir ou excluí-lo das suas configurações.

Veja o módulo `use_kivy_settings()` para mais informações sobre ativação/desativação automática do painel Kivy.

`create_json_panel(title, config, filename=None, data=None)`

Cria uma nova `SettingsPanel`.

Novo na versão 1.5.0.

Veja a documentação de `add_json_panel()` para maiores informações.

`interface`

(Interno) Referência ao Widget que irá conter, organizará e exibirá os Widgets do painel de configuração.

`interface` é uma `ObjectProperty` e o padrão é `None`.

`interface_cls`

A classe de Widget que será usada para exibir a interface gráfica do painel

de configurações. Por padrão, será exibido um único painel Configurações por vez com uma barra lateral para alternar entre os demais painéis.

`interface_cls` é uma `ObjectProperty` e o padrão é `InterfaceWithSidebar`.

Alterado na versão 1.8.0: Se você definir uma string, a `Factory` será usada para resolver a classe.

`register_type(tp, cls)`

Registra um novo tipo que pode ser usado na definição JSON.

```
class kivy.uix.settings.SettingsPanel(**kwargs)
```

Bases: `kivy.uix.gridlayout.GridLayout`

Esta classe é usada para construir as configurações do painel, para uso com a instância ou subclasse de `Settings`.

`config`

Uma instância de `kivy.config.ConfigParser`. Veja o módulo da documentação para maiores informações.

`get_value(section, key)`

Retorna o valor da seção/chave da instância de `config` ConfigParser. Esta função é usada por `SettingItem` para obter o valor de uma determinada seção/chave.

Se você não quiser usar uma instância do `ConfigParser`, talvez queira substituir essa função.

`settings`

Uma instância de: class: `Settings` que será usada para disparar o evento `on_config_change`.

`title`

Título do painel. O título será reutilizado pela class: `Settings` na barra lateral.

```
class kivy.uix.settings.SettingItem(**kwargs)
```

Bases: `kivy.uix.floatlayout.FloatLayout`

Classe base para configurações individuais (dentro de um painel). Esta classe não pode ser usada diretamente; ela é usada para implementar as outras classes de configuração. Ela constrói uma linha com um título/descrição (à esquerda) e um controle de configuração (à direita).

Olhe para `SettingBoolean`, `SettingNumeric` e `SettingOptions` exemplos de utilização.

Events

`on_release` Disparado quando o item tocado é liberado.

`content`

(Interno) Referência ao Widget que contém a configuração real. Assim que

o objeto de conteúdo for definido, qualquer chamada adicional ao método `add_widget` invocará o `content.add_widget`. Isso é definido automaticamente.

`content` é uma `ObjectProperty` e o padrão é `None`.

desc

Descrição da configuração, exibida na linha abaixo do título.

`desc` é uma `StringProperty` e o padrão é `None`.

disabled

Indica se esta configuração está desativada. Se `True`, todos os toques no item de configuração serão descartados.

`disabled` é uma `BooleanProperty` e o padrão é `False`.

key

Chave do Token dentro da `section` na instância `ConfigParser`.

`key` é uma `StringProperty` e o padrão é `None`.

panel

(Interno) Referência para o `SettingsPanel` para esta configuração. Você não precisa usar isso.

`panel` é uma `ObjectProperty` e o padrão é `None`.

section

Seção do Token dentro da instância `ConfigParser` instance.

`section` é uma `StringProperty` e o padrão é `None`.

selected_alpha

(Interno) Valor flutuante no intervalo entre 0 a 1, usado para animar o plano de fundo quando o usuário toca no item.

`selected_alpha` é uma `NumericProperty` e o padrão é `0`

title

Título da configuração, por padrão será ‘<No title set>’.

`title` é uma `StringProperty` e o padrão é ‘<No title set>’.

value

Valor do token de acordo com a instância `ConfigParser`. Qualquer alteração neste valor desencadeará o evento `Settings.on_config_change()`.

`value` é uma `ObjectProperty` e o padrão é `None`.

class kivy.uix.settings.`SettingString`(kwargs)**

Bases: `kivy.uix.settings.SettingItem`

Implementação de uma configuração de string em cima de um `SettingItem`. Isso é visualizado como um Widget que `Label`, quando clicado, abrirá uma

Popup com uma `TextInput` para que o usuário possa inserir um valor personalizado.

popup

(Interno) Usado para armazenar o *Poput* atual quando o mesmo for mostrado.

popup é uma *ObjectProperty* e o padrão é *None*.

textinput

(Interno) Usado para armazenar o *TextInput* atual do *Popup* e para ouvir as alterações.

textinput é uma *ObjectProperty* e o padrão é *None*.

```
class kivy.uix.settings.SettingPath(**kwargs)
```

Bases: *kivy.uix.settings.SettingItem*

Implementação de uma configuração Path em cima de uma *SettingItem*. É visualizado com um Widget que *Label*, quando clicado, abrirá um *Popup* com uma *FileChooserListView* Para que o usuário possa inserir um valor personalizado.

Novo na versão 1.1.0.

dirselect

Whether to allow selection of directories.

dirselect is a *BooleanProperty* and defaults to True.

Novo na versão 1.10.0.

popup

(Interno) Usado para armazenar o atual *Poput* quando o mesmo for mostrado.

popup é uma *ObjectProperty* e o padrão é *None*.

show_hidden

Whether to show 'hidden' filenames. What that means is operating-system-dependent.

show_hidden is an *BooleanProperty* and defaults to False.

Novo na versão 1.10.0.

textinput

(Interno) Usado para armazenar o *TextInput* atual do *Poput* e para ouvir as alterações.

textinput é uma *ObjectProperty* e o padrão é *None*.

```
class kivy.uix.settings.SettingBoolean(**kwargs)
```

Bases: *kivy.uix.settings.SettingItem*

Implementação de um configuração booleana em cima de uma [SettingItem](#). É visualizado com um Widget [Switch](#). Por padrão, 0 e 1 são usados por valores: podes alterá-los definindo [values](#).

values

Valores usados para representar o estado da configuração. Se quiseres usar “yes” e “no” na instância do [ConfigParser](#):

```
SettingBoolean(..., values=['no', 'yes'])
```

Aviso: Você precisa de um mínimo de dois valores, o índice 0 será usado como *False* e o índice 1 como *True*.

[values](#) é uma [ListProperty](#) e o padrão é ['0', '1']

```
class kivy.uix.settings.SettingNumeric(**kwargs)
```

Bases: [kivy.uix.settings.SettingString](#)

Implementação de uma configuração numérica em cima de uma [SettingString](#). É visualizado como um Widget que [Label](#), quando clicado, abrirá um [Popup](#) com um [TextInput](#) para que o usuário possa inserir um valor personalizado.

```
class kivy.uix.settings.SettingOptions(**kwargs)
```

Bases: [kivy.uix.settings.SettingItem](#)

Implementação de uma lista de opções com uma [SettingItem](#). É visualizado como um Widget que [Label](#), quando clicado, abrirá um [Popup](#) com uma lista de opções a partir da qual o usuário pode selecionar.

options

Lista de todas as opções. Esta lista deve ser uma lista de itens “string”, isso vai falhar :)

[options](#) é uma [ListProperty](#) e o padrão é [].

popup

(Interno) Usado para armazenar o atual [Popup](#) quando o mesmo for mostrado.

[popup](#) é uma [ObjectProperty](#) e o padrão é *None*.

```
class kivy.uix.settings.SettingTitle(**kwargs)
```

Bases: [kivy.uix.label.Label](#)

Um simples Label de título é usado para organizar as configurações em seções.

```
class kivy.uix.settings.SettingsWithSidebar(*args, **kwargs)
```

Bases: [kivy.uix.settings.Settings](#)

Um Widget de configurações que exibe painéis de configurações com uma barra lateral para alternar entre elas. Esse é o comportamento padrão de **Settings**, e esse Widget é uma subclasse trivial de Wrapper.

```
class kivy.uix.settings.SettingsWithSpinner(*args, **kwargs)
    Bases: kivy.uix.settings.Settings
```

Um Widget de configurações que exibe um painel de configurações de cada vez com um Spinner na parte superior para alternar entre ambos.

```
class kivy.uix.settings.SettingsWithTabbedPanel(*args, **kwargs)
    Bases: kivy.uix.settings.Settings
```

Um Widget de configurações que exibe painéis de configurações como **TabbedPanel**.

```
class kivy.uix.settings.SettingsWithNoMenu(*args, **kwargs)
    Bases: kivy.uix.settings.Settings
```

Um Widget de configurações que exibe um único painel de configurações *sem* o botão fechar. Não é aceito mais de um painel Configurações. Destina-se a ser utilizado em programas com poucas definições onde não há razão para ter um alternador entre vários painéis.

Aviso: Este painel de Configurações não *fornecce* um botão de fechamento e, portanto, é impossível deixar a tela de configurações, a menos que você também adicione outro comportamento ou substitua **display_settings()** e **close_settings()**.

```
class kivy.uix.settings.InterfaceWithSidebar(*args, **kwargs)
    Bases: kivy.uix.boxlayout.BoxLayout
```

A interface de classe padrão de Settings. Exibe um menu da barra lateral com nomes de painéis de configurações disponíveis, que podem ser usados para alternar entre qual deve ser exibido em cada momento.

Consulte **add_panel()** para obter mais informações sobre o método que você deve implementar se criar sua própria interface.

Esta classe também invoca o evento ‘on_close’, que é disparado quando o botão de fechamento do menu da barra lateral for solto. Se estiveres criando seu próprio Widget de interface, ele também deve despachar um evento que será automaticamente capturado por **Settings** e usado para disparar seu próprio evento ‘on_close’.

add_panel(panel, name, uid)

Este método é usado por *Settings* para adicionar novos painéis das possíveis exibições. Qualquer substituição com ContentPanel *deverá* implementar este método.

Parameters

panel: **SettingsPanel** Deve ser armazenado e a interface deve fornecer uma maneira de alternar entre painéis.

name: O nome do painel como sendo uma String. Pode ser usado para representar o painel, mas não será necessariamente o único.

uid: Um int único identificando o painel. Ele deve ser usado para identificar e alternar entre os painéis.

content

(Interno) Uma referência ao Widget de exibição do painel (uma **ContentPanel**).

content é uma **ObjectProperty** e o padrão é *None*.

menu

(Interno) Uma referência ao Widget do menu da barra lateral.

menu é uma **ObjectProperty** e o padrão é *None*.

class kivy.uix.settings.ContentPanel(kwargs)**

Bases: **kivy.uix.scrollview.ScrollView**

Uma classe para exibir os painéis de configurações. Ele exibe um único painel de configurações de cada vez, ocupando o tamanho completo e a forma do **ContentPanel**. É usado por **InterfaceWithSidebar** e **InterfaceWithSpinner** para exibir as configurações.

add_panel(panel, name, uid)

Este método é usado por *Settings* para adicionar novos painéis das possíveis exibições. Qualquer substituição com ContentPanel *deverá* implementar este método.

Parameters

panel: **SettingsPanel** Deve ser armazenado e exibido quando solicitado.

name: O nome do painel como uma string. Pode ser usado para representar o painel.

uid: Um int único identificando o painel. Deve ser armazenado e utilizado para identificar os painéis durante a alteração.

container

(Internal) Uma referência para o *GridLayout* que contém o painel de configurações.

container é uma **ObjectProperty** e o padrão é *None*.

current_panel

(Interno) Uma referência ao painel de configurações atual.

current_panel é uma *ObjectProperty* e o parão é *None*.

current_uid

(Interno) Uma referência Ao uid do painel de configurações atual.

current_uid é uma *NumericProperty* e o padrão é *0*.

on_current_uid(*args)

O uid do painel exibido no momento. Alterar isso alterará automaticamente o painel exibido.

Parameters

uid:Um painel *uid*. Ele deve ser usado para recuperar e exibir um painel de configurações que foi adicionado anteriormente com *add_panel()*.

panels

(Interno) Armazena um dicionário mapeando os painéis de configurações para seus uids.

panels é uma *DictProperty* e o padrão é {}.

class kivy.uix.settings.MenuSidebar(kwargs)**

Bases: *kivy.uix.floatlayout.FloatLayout*

O menu utilizado por *InterfaceWithSidebar*. Ele fornece uma barra lateral com uma entrada para cada painel de configurações, que o usuário pode clicar para selecionar.

add_item(name, uid)

Este método é utilizado para adicionar novos painéis ao menu.

Parameters

name:O nome (uma string) do painel. Ele deve ser usado para representar o painel no menu.

uid:O nome (um int) do painel. Ele deve ser usado internamente para representar o painel e usado para definir *self.selected_uid* quando o painel for alterado.

buttons_layout

(Interno) Referência ao *GridLayout* que contém botões individuais de menu do painel de configurações.

buttons_layout é uma *ObjectProperty* e o parão é *None*.

close_button

(Interno) Referência ao botão Fechar do widget.

buttons_layout é uma *ObjectProperty* e o parão é *None*.

on_selected_uid(*args)

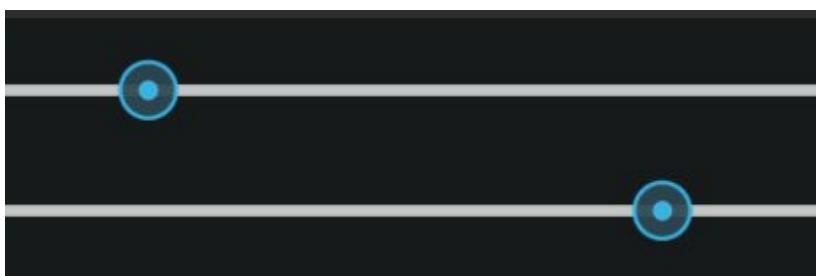
(Interno) desmarca qualquer botão de menu atualmente selecionado, a menos que eles representem o painel atual.

selected_uid

O *uid* do painel atualmente selecionado. Isto pode ser utilizado para alternar entre painéis exibidos, por exemplo, vinculando-o ao *current_uid* de *ContentPanel*.

selected_uid é uma *NumericProperty* e o padrão é 0.

4.15.41 Slider



O Widget *Slider* parece uma barra de rolagem. Ele suporta as orientações horizontal e vertical, valores min/max e um valor padrão.

Para criar um Slider deslizante de - 100 até 100 a partir de 25:

```
from kivy.uix.slider import Slider
s = Slider(min=-100, max=100, value=25)
```

Para criar um Slider vertical:

```
from kivy.uix.slider import Slider
s = Slider(orientation='vertical')
```

Para criar um Slider com umlinha vermelha rastreando o valor:

```
from kivy.uix.slider import Slider
s = Slider(value_track=True, value_track_color=[1, 0, 0, 1])
```

class kivy.uix.slider.Slider(kwargs)**

Bases: *kivy.uix.widget.Widget*

Classe para a criação do Widget Slider.

Veja o módulo da documentação para maiores informações.

background_disabled_horizontal

Fundo do controle deslizante desativado usado na orientação horizontal.

Novo na versão 1.10.0.

background_disabled_horizontal é uma *StringProperty* e o padrão é *atlas://data/images/defaulttheme/sliderh_background_disabled*.

background_disabled_vertical

Fundo do Slider desativado usado na orientação vertical.

Novo na versão 1.10.0.

background_disabled_vertical é uma *StringProperty* e o padrão é *atlas://data/images/defaulttheme/sliderv_background_disabled*.

background_horizontal

Fundo do Slider usado na orientação horizontal.

Novo na versão 1.10.0.

background_horizontal é uma *StringProperty* e o padrão é *atlas://data/images/defaulttheme/sliderh_background*.

background_vertical

Fundo do controle deslizante usado na orientação vertical.

Novo na versão 1.10.0.

background_vertical é uma *StringProperty* e o padrão é *atlas://data/images/defaulttheme/sliderv_background*.

background_width

Largura do fundo do Slider (espessura), usada em orientações horizontais e verticais.

background_width é uma *NumericProperty* e o padrão é *36sp*.

border_horizontal

Borda usada para desenhar o fundo do Slider na orientação horizontal.

border_horizontal é uma *ListProperty* e o padrão é *[0, 18, 0, 18]*.

border_vertical

Borda usada para desenhar o fundo do Slider na orientação vertical.

border_horizontal é uma *ListProperty* e o padrão é *[18, 0, 18, 0]*.

cursor_disabled_image

Caminho da imagem usada para desenhar o Cursor deslizante desativado.

cursor_image é uma *StringProperty* e o padrão é *atlas://data/images/defaulttheme/slider_cursor_disabled*.

cursor_height

q

cursor_height é uma *NumericProperty* e o padrão é *32sp*.

cursor_image

Caminho da imagem usada para desenhar o cursor do Slider.

`cursor_image` é uma *StringProperty* e o padrão é `atlas://data/images/defaulttheme/slider_cursor`.

cursor_size

Tamanho da imagem do Cursor.

`cursor_size` é uma propriedade *ReferenceListProperty* de (`cursor_width, cursor_height`).

cursor_width

Largura da imagem do Cursor.

`cursor_width` é uma *NumericProperty* e o padrão é `32sp`.

max

Valor máximo permitido para `value`.

`max` é uma *NumericProperty* e o valor padrão é `100`.

min

Valor mínimo permitido para `value`.

`min` é uma *NumericProperty* e o padrão é `0`.

orientation

Orientação do Slider

`orientation` é um *OptionProperty* e o padrão é ‘horizontal’. Pode assumir os valores de ‘vertical’ ou ‘horizontal’.

padding

Padding do controle deslizante. O preenchimento é usado para representação gráfica e interação. Evita que o cursor saia dos limites da caixa delimitadora do limitador.

Por padrão, o padding é `16sp`. O alcance do controle deslizante é reduzido de padding *2 na tela. Ele permite desenhar o cursor padrão de largura `32sp` sem que o cursor tenha que sair do Widget.

`padding` é uma *NumericProperty* e o padrão é `16sp`.

range

Faixa do Slider no formato (valor mínimo, valor máximo):

```
>>> slider = Slider(min=10, max=80)
>>> slider.range
[10, 80]
>>> slider.range = (20, 100)
>>> slider.min
20
>>> slider.max
100
```

`range` é uma propriedade *ReferenceListProperty* de (`min, max`).

step

Tamanho do passo do Slider.

Novo na versão 1.4.0.

Determina o tamanho de cada intervalo ou etapa que o controle deslizante leva entre *min* e *max*. Se o intervalo de valores não puder ser divisível em etapas, o último passo será limitado pelo *slider.max*.

step é uma *NumericProperty* e o padrão é 1.

value

Valor atual usado pelo Slider.

value é uma *NumericProperty* e o padrão é 0.

value_normalized

Valor normalizado dentro do *range* (min/max) no intervalo 0-1

```
>>> slider = Slider(value=50, min=0, max=100)
>>> slider.value
50
>>> slider.value_normalized
0.5
>>> slider.value = 0
>>> slider.value_normalized
0
>>> slider.value = 100
>>> slider.value_normalized
1
```

Você também pode usar ele para configura o valor real sem conhecer o mínimo e o máximo:

```
>>> slider = Slider(min=0, max=200)
>>> slider.value_normalized = .5
>>> slider.value
100
>>> slider.value_normalized = 1.
>>> slider.value
200
```

value_normalized é uma *AliasProperty*.

value_pos

Posição interna do Cursor, com base no valor normalizado.

value_pos é uma *AliasProperty*.

value_track

Decide se o controle deslizante deve desenhar a linha indicando o espaço entre os valores das propriedades *min* e *value*.

value_track é uma *BooleanProperty* e o padrão é *False*.

value_track_color

Cor do *value_line* no formato RGBA.

value_track_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

value_track_width

Largura da linha de rastreamento.

value_track_width é uma *NumericProperty* e o valor padrão é 3dp.

4.15.42 Spinner

Novo na versão 1.4.0.



Spinner é um widget que provê uma maneira rápida de selecionar um valor de um conjunto. Por padrão, um spinner apresenta seu valor atualmente selecionado. Tomando no spinner é apresentado um menu dropdown com todos os valores disponíveis para que o usuário selecione um.

Exemplo:

```
from kivy.base import runTouchApp
from kivy.uix.spinner import Spinner

spinner = Spinner(
    # default value shown
    text='Home',
    # available values
    values=('Home', 'Work', 'Other', 'Custom'),
    # just for positioning in our example
    size_hint=(None, None),
```

```

        size=(100, 44),
        pos_hint={'center_x': .5, 'center_y': .5})

def show_selected_value(spinner, text):
    print('The spinner', spinner, 'have text', text)

spinner.bind(text=show_selected_value)

runTouchApp(spinner)

```

class kivy.uix.spinner.Spinner(**kwargs)

Bases: [kivy.uix.button.Button](#)

Classe Spinner, veja o módulo da documentação para maiores informações.

dropdown_cls

Classe usada para exibir a lista suspensa quando o Spinner for pressionado.

dropdown_cls é uma [ObjectProperty](#) e o padrão é [DropDown](#).

Alterado na versão 1.8.0: Se você definir uma string, a [Factory](#) será usada para resolver a classe.

is_open

Por padrão, o Spinner não está aberto. Defina como *True* para abri-lo.

is_open é um [BooleanProperty](#) e o padrão é *False*.

Novo na versão 1.4.0.

option_cls

Classe usada para exibir as opções dentro da lista suspensa mostrada sob o *Spinner*. A propriedade *text* da classe será usada para representar o valor.

A classe de opções requer:

- um propriedade *text*, usada para exibir o valor.
- Um evento *on_release*, usado para acionar a opção quando for pressionado/tocado.
- um *size_hint_y* de *None*.
- o *height* para ser definido.

option_cls é uma [ObjectProperty](#) e o padrão é [SpinnerOption](#).

Alterado na versão 1.8.0: Se você definir uma string, a [Factory](#) será usada para resolver a classe.

sync_height

Cada elemento na lista suspensa usa uma altura padrão/"fornecida pelo usuário". Defina como *True* para propagar o valor da altura do *Spinner* para cada elemento da lista suspensa.

Novo na versão 1.10.0.

sync_height é uma *BooleanProperty* e o padrão é *False*.

text_autoupdate

Indica se o Spinner *text* deve ser atualizado automaticamente com o primeiro valor da propriedade *values*. Configurá-lo para *True* fará com que o *Spinner* atualize sua propriedade *text* toda vez que *attr:values* for alterados.

Novo na versão 1.10.0.

text_autoupdate é uma *BooleanProperty* e o padrão é *False*.

values

Valores que podem ser selecionados pelo usuário. Deve ser uma lista de Strings.

values é uma *ListProperty* e o padrão é *[]*.

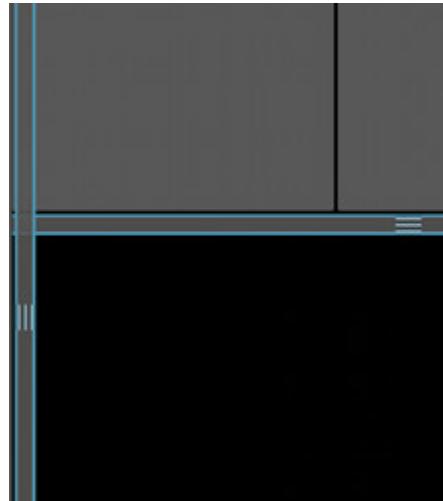
`class kivy.uix.spinner.SpinnerOption(**kwargs)`

Bases: *kivy.uix.button.Button*

Especial Button usado no DropDown List *Spinner*. Por padrão, isso é somente uma *Button* com uma *size_hint_y* de *None* e a altura igual a *48dp*.

4.15.43 Splitter

Novo na versão 1.5.0.



A classe *Splitter* é um widget que ajuda a redimensionar o seu próprio widget/layout-filho. O redimensionamento se dá arrastando ou dando um toque duplo nas margens. Este widget é similar ao *ScrollView*, na medida que permite apenas um widget-filho

Uso:

```
splitter = Splitter(sizable_from = 'right')
splitter.add_widget(layout_or_widget_instance)
splitter.min_size = 100
splitter.max_size = 250
```

Para alterar o tamanho da faixa/borda utilizada para redimensionar:

```
splitter.strip_size = '10pt'
```

Para alterar sua aparência:

```
splitter.strip_cls = your_custom_class
```

Você também pode alterar a aparência do *strip_cls*, que assume como padrão **SplitterStrip**, substituindo a regra *kv* no seu aplicativo:

```
<SplitterStrip>:
    horizontal: True if self.parent and self.parent.sizable_from[0]_
    ↵in ('t', 'b') else False
    background_normal: 'path to normal horizontal image' if self.
    ↵horizontal else 'path to vertical normal image'
    background_down: 'path to pressed horizontal image' if self.
    ↵horizontal else 'path to vertical pressed image'
```

class kivy.uix.splitter.Splitter(kwargs)**
Bases: *kivy.uix.boxlayout.BoxLayout*

Veja o módulo da documentação.

Events

on_press: Disparado quando o Splitter é pressionado.

on_release: Disparado quando o Splitter é liberado.

Alterado na versão 1.6.0: Adicionado os eventos *on_press* e *on_release*.

border

Borda usada para a instrução de gráficos:
class ~kivy.graphics.vertex_instructions.BorderImage

This must be a list of four values: (bottom, right, top, left). Read the BorderImage instructions for more information about how to use it.

border é uma *ListProperty* e o padrão é (4, 4, 4, 4).

keep_within_parent

Se True, irá limitar o Splitter para ficar dentro do seu Widget pai.

keep_within_parent é uma *BooleanProperty* e o padrão é *False*.

Novo na versão 1.9.0.

max_size

Especifica o tamanho máximo além do qual o Widget não é redimensionável.

max_size é uma *NumericProperty* e o padrão é *500pt*.

min_size

Especifica o tamanho mínimo além do qual o Widget não é redimensionável.

min_size é uma *NumericProperty* e o padrão é *100pt*.

rescale_with_parent

Se *True*, automaticamente mudará de tamanho para ocupar a mesma proporção do Widget pai quando ele for redimensionado, enquanto permanece dentro de *min_size* and *max_size*. Contanto que esses atributos possam ser satisfeitos, isso interrompe a classe *Splitter* ao exceder o tamanho do pai durante o reescalonamento.

rescale_with_parent é uma *BooleanProperty* e o padrão é *False*.

Novo na versão 1.9.0.

sizable_from

Especifica se o Widget é redimensionável. As opções são: *left*, *'right'*, *top* ou *'bottom'*.

sizable_from é uma *OptionProperty* e o padrão é *left*.

strip_cls

Especifica a classe da faixa de redimensionamento.

strip_cls é uma *kivy.properties.ObjectProperty* e o padrão é *SplitterStrip*, que é do tipo *Button*.

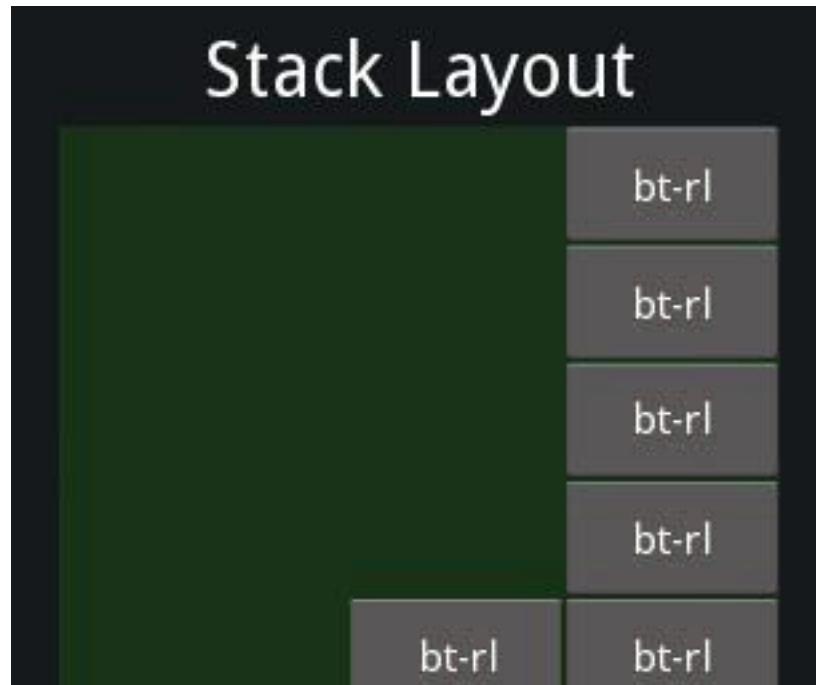
Alterado na versão 1.8.0: Se você definir uma string, a *Factory* será usada para resolver a classe.

strip_size

Especifica o tamanho da faixa de redimensionamento

strip_size é uma *NumericProperty* e o padrão é *10pt*

4.15.44 Stack Layout

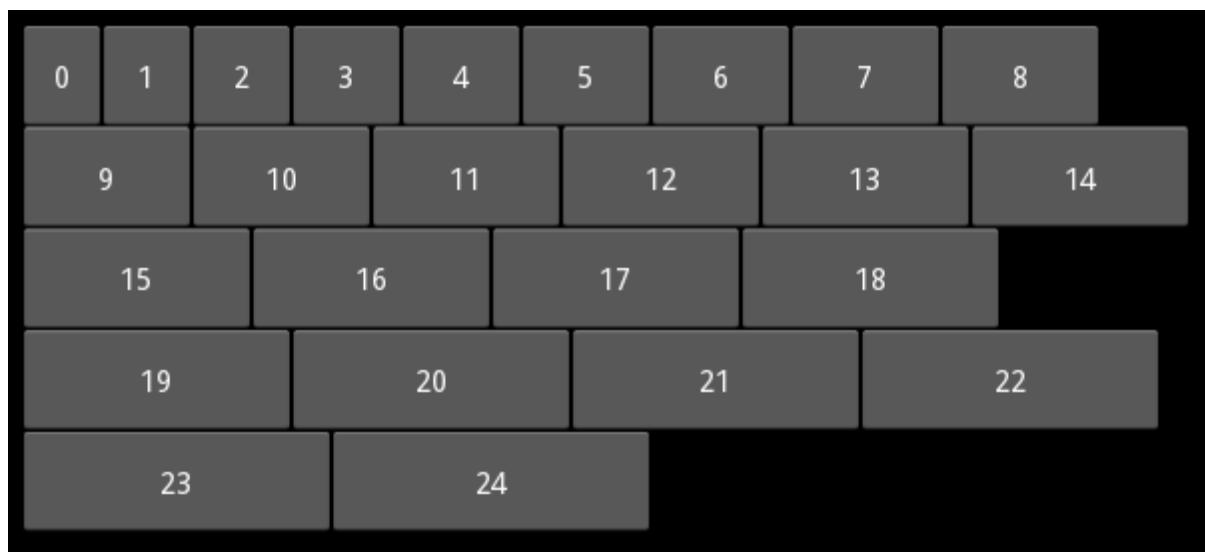


Novo na versão 1.0.5.

A **StackLayout** organiza os filhos verticalmente ou horizontalmente, a quantidade que é possível até completar. O tamanho de um Widget filho individual não será uniforme.

Por exemplo, para exibir Widgets que ficam progressivamente maiores em largura:

```
root = StackLayout()
for i in range(25):
    btn = Button(text=str(i), width=40 + i * 5, size_hint=(None, 0.
    ↴15))
    root.add_widget(btn)
```



```
class kivy.uix.stacklayout.StackLayout(**kwargs)
```

Bases: [kivy.uix.layout.Layout](#)

Classe StackLayout. Veja a documentação para maiores informações.

minimum_height

Altura mínima necessária para conter todos os filhos. Isso é automaticamente definido pelo layout.

Novo na versão 1.0.8.

minimum_height é uma [kivy.properties.NumericProperty](#) e o padrão é 0.

minimum_size

Tamanho mínimo necessário para conter todos os filhos. Isso é automaticamente definido pelo layout.

Novo na versão 1.0.8.

minimum_size é uma propriedade [ReferenceListProperty](#) de (*minimum_width*, *minimum_height*).

minimum_width

Largura mínima necessária para conter todos os filhos. Isso é automaticamente definido pelo layout.

Novo na versão 1.0.8.

minimum_width é uma [kivy.properties.NumericProperty](#) e o padrão é 0.

orientation

Orientação do layout.

orientation é uma [OptionProperty](#) e o padrão é 'lr-tb'.

Orientações válidas são 'lr-tb', 'tb-lr', 'rl-tb', 'tb-rl', 'lr-bt', 'bt-lr', 'rl-bt' and 'bt-rl'.

Alterado na versão 1.5.0: *orientation* agora está tratando todas as combinações validas de 'lr', 'rl', 'tb', 'bt'. A versão anterior somente suportava 'lr-tb' and 'tb-lr' e 'tb-lr' havia sido mal nomeada e colocava Widgets de baixo para cima e da direita para esquerda (ordem invertida em comparação com o que era esperado).

Nota: 'lr' significa Left to Right (esquerda para direita). 'rl' significa Right to Left (direita para esquerda). 'tb' significa Top to Bottom (cima para baixo). 'bt' significa Botttom to Top (baixo para cima).

padding

Padding entre a caixa de layout e seus filhos: [padding_left, padding_top,

`padding_right, padding_bottom]`.

`padding` também aceita uma forma de dois argumentos [`padding_horizontal, padding_vertical`] e uma forma simples [`padding`].

Alterado na versão 1.7.0: Substituído o `NumericProperty` pelo `VariableListProperty`.

`padding` é um `VariableListProperty` e o padrão é [0, 0, 0, 0].

spacing

Espaçamento entre os filhos: [`spacing_horizontal, spacing_vertical`].

`spacing` também aceita um forma simples de argumentos [`spacing`].

`spacing` é um `VariableListProperty` e o padrão é [0, 0].

4.15.45 Stencil View

Novo na versão 1.0.4.

`StencilView` limita os widgets ao tamanho da caixa `StencilView`. Qualquer elemento que exceder esses limites será eliminado.

O `StencilView` usa as instruções gráficas do `Stencil` debaixo do capô. Ele fornece uma maneira eficiente para grampear a área de desenho de crianças.

Nota: Como com as instruções gráficas do `Stencil`, não é possível empilhar mais de 128 Widgets compatíveis com o `Wtencil`.

Nota: `StencilView` não é um layout. Conseqüentemente, você tem que controlar o tamanho e a posição dos Widgets filhos. Podes combinar (ambos são subclasse) um `StencilView` e um Layout para conseguires o comportamento de um layout. Por exemplo:

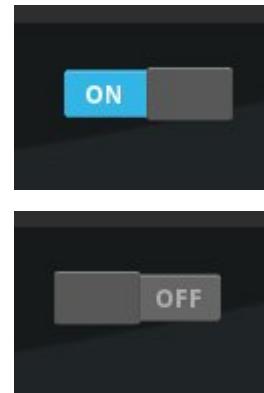
```
class BoxStencil(BoxLayout, StencilView):  
    pass
```

```
class kivy.uix.stencilview.StencilView(**kwargs)  
    Bases: kivy.uix.widget.Widget
```

Classe `StencilView`. Consulte a documentação do módulo para maiores informações.

4.15.46 Switch

Novo na versão 1.0.7.



A classe: **Switch** é um widget que funciona como um interruptor de luz. É um widget que mostra o estado ativo/inativo. Para ativar/desativar, o usuário irá deslizá-lo.

```
switch = Switch(active=True)
```

Para anexar um callback que escuta o estado de ativação:

```
def callback(instance, value):
    print('the switch', instance, 'is', value)

switch = Switch()
switch.bind(active=callback)
```

Por padrão, a representação do widget é estática. O tamanho mínimo necessário é de 83x32 pixels (definido pela imagem de plano de fundo). A imagem é centrada dentro do widget.

O widget inteiro está ativo, não apenas a parte com gráficos. Contanto que você deslize sobre a caixa delimitadora do widget, ele funcionará.

Nota: Se você quiser controlar o estado com um único toque em vez de um swipe, use a **ToggleButton** em vez disso.

```
class kivy.uix.switch.Switch(**kwargs)
    Bases: kivy.uix.widget.Widget
```

Classe *Switch*. Veja o módulo da documentação para maiores informações.

active

Indicar se o Switch está ativo ou inativo.

active é um **BooleanProperty** e o padrão é *False*.

active_norm_pos

(Interno) Contém a posição normalizada do elemento móvel dentro do interruptor, na faixa 0-1.

active_norm_pos é uma *NumericProperty* e o padrão é 0.

touch_control

(Interno) Contém o toque que atualmente interage com o Switch.

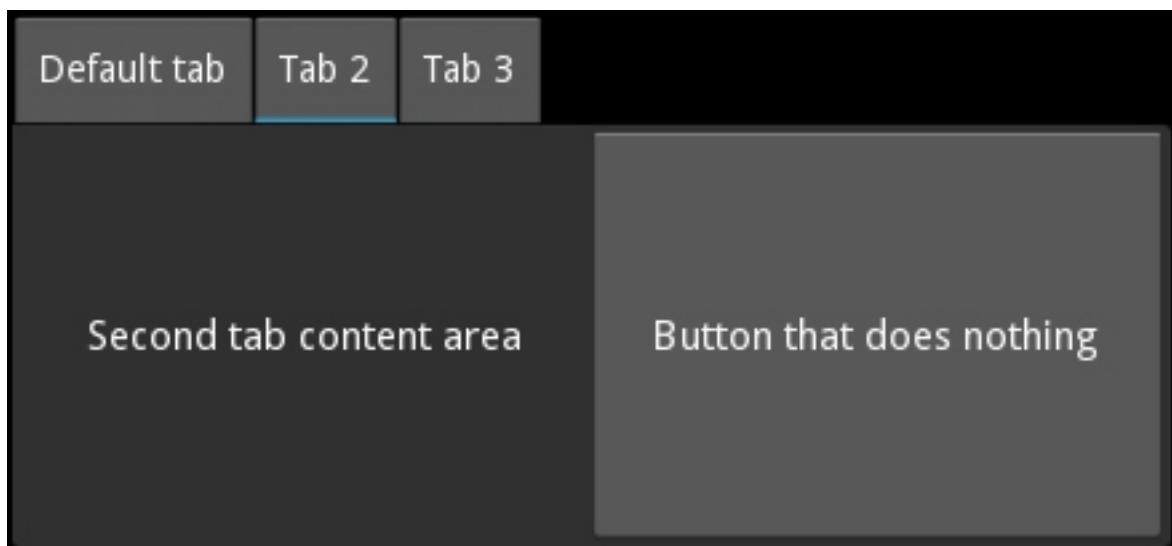
touch_control é uma *ObjectProperty* e o padrão é *None*.

touch_distance

(Interno) Contém a distância entre a posição inicial do toque e a posição atual para determinar se o deslize é do lado esquerdo ou direito.

touch_distance é uma *NumericProperty* e o padrão é 0.

4.15.47 TabbedPanel



Novo na versão 1.3.0.

O widget ‘TabbedPanel’ gerencia os widgets constantes nas abas, com um cabeçalho para os botões e uma área de conteúdo para mostrar o conteúdo da aba atual.

A classe :class:‘TabbedPanel’ fornece uma Tab padrão.

Simples exemplo

```
...
TabbedPanel
=====
Test of the widget TabbedPanel.
...  
...
```

```

from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.lang import Builder

Builder.load_string("""
<Test>:
    size_hint: .5, .5
    pos_hint: {'center_x': .5, 'center_y': .5}
    do_default_tab: False

    TabbedPanelItem:
        text: 'first tab'
        Label:
            text: 'First tab content area'
    TabbedPanelItem:
        text: 'tab2'
        BoxLayout:
            Label:
                text: 'Second tab content area'
            Button:
                text: 'Button that does nothing'
    TabbedPanelItem:
        text: 'tab3'
        RstDocument:
            text:
                '\n'.join(("Hello world", "-----",
                           "You are in the third tab."))
""")

class Test(TabbedPanel):
    pass

class TabbedPanelApp(App):
    def build(self):
        return Test()

if __name__ == '__main__':
    TabbedPanelApp().run()

```

Nota: Uma nova classe **TabbedPanelItem** foi introduzida em 1.5.0 por conveniência. Portanto, agora é possível simplesmente adicionar um **TabbedPanelItem** a uma **TabbedPanel** e *content* para a **TabbedPanelItem** como no exemplo fornecido

acima.

Customize o Tabbed Panel

Você pode escolher a posição na qual as Tabs são exibidas:

```
tab_pos = 'top_mid'
```

Uma Tab individual é chamada `TabbedPaneHeader`. É um botão especial que contém uma propriedade `content`. Você adiciona o `TabbedPaneHeader` primeiro e define sua propriedade `content` separadamente:

```
tp = TabbedPane()  
th = TabbedPaneHeader(text='Tab2')  
tp.add_widget(th)
```

Uma Tab individual, representada por um `TabbedPaneHeader`, precisa do seu conjunto de conteúdo. Este conteúdo pode ser qualquer Widget. Poderia ser um layout com uma grande hierarquia de Widgets, ou poderia ser um simples Widget, como um Label ou um Button:

```
th.content = your_content_instance
```

Há uma área de conteúdo principal “compartilhada” ativa a qualquer momento, para todas as Tabs. Seu aplicativo é responsável por adicionar o conteúdo de Tabs individuais e para gerenciá-las, mas não é responsável pela troca de conteúdo. O painel com Abas controla a troca do objeto de conteúdo principal de acordo com a ação do usuário.

Há uma Aba padrão adicionada quando o painel com Abas é instanciado. As Abas que você adiciona individualmente como acima, são adicionadas além da Aba padrão. Assim, dependendo de suas necessidades e design, você vai querer personalizar a Aba padrão

```
tp.default_tab_text = 'Something Specific To Your Use'
```

A maquinaria padrão da Tab requer uma consideração especial e um gerenciamento. Consequentemente, o evento `on_default_tab` é fornecido para associar um callback:

```
tp.bind(default_tab = my_default_tab_callback)
```

É importante notar que, por padrão, `default_tab_cls` é do tipo `TabbedPaneHeader` e, portanto, tem as mesmas propriedades de outras guias.

Desde a versão 1.5.0, é possível desabilitar a criação do `default tab` através da configuração `:attr:`do_default_tab`` definindo como `False`.

Tabs e conteúdo podem ser removidos de várias maneiras:

```
tp.remove_widget(widget/tabbed_panel_header)
or
tp.clear_widgets() # to clear all the widgets in the content area
or
tp.clear_tabs() # to remove the TabbedPanelHeaders
```

Para acessar as crianças de um painel de Tabs, use `content.children`:

```
tp.content.children
```

Para acessar a lista de Tabs:

```
tp.tab_list
```

Para alterar a aparência do conteúdo do painel com as Tabs principais:

```
background_color = (1, 0, 0, .5) #50% translucent red
border = [0, 0, 0, 0]
background_image = 'path/to/background/image'
```

Para alterar o fundo de uma Tab individual, use estas duas propriedades:

```
tab_header_instance.background_normal = 'path/to/tab_head/img'
tab_header_instance.background_down = 'path/to/tab_head/img_pressed'
```

Um TabbedPanelStrip contém os cabeçalhos de tabulação individuais. Para alterar a aparência desta faixa de tabulação, substitua o Canvas de TabbedPanelStrip. Por exemplo, com a linguagem kv:

```
<TabbedPanelStrip>
    canvas:
        Color:
            rgba: (0, 1, 0, 1) # green
        Rectangle:
            size: self.size
            pos: self.pos
```

Por padrão, a faixa de painel com Tabs toma sua imagem de plano de fundo e a cor de `background_image` e `background_color` do painel com Tabs.

```
class kivy.uix.tabbedpanel.StripLayout(**kwargs)
    Bases: kivy.uix.gridlayout.GridLayout
```

O layout principal é usado para abrigar toda a faixa TabbedPanel incluindo as áreas em branco no caso de as Tabs não cobrirem toda a largura/altura.

Novo na versão 1.8.0.

background_image

Imagen de plano de fundo pra ser usada com o layout Strip do `TabbedPanel`.

background_image é uma *StringProperty* e o padrão é uma imagem transparente.

border

Propriedade de borda para o *background_image*.

border é uma *ListProperty* e o padrão é [4, 4, 4, 4].

```
class kivy.uix.tabbedpanel.TabbedPanel(**kwargs)
```

Bases: *kivy.uix.gridlayout.GridLayout*

Classe *TabbedPanel*. Veja o módulo da documentação para maiores informações.

background_color

Cor de fundo, no forma (r, g, b, a).

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

background_disabled_image

Imagen de plano de fundo do objeto de conteúdo compartilhado principal quando desabilitado.

Novo na versão 1.8.0.

background_disabled_image é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/tab’.

background_image

Imagen de plano de fundo do objeto principal de conteúdo compartilhado.

background_image é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/tab’.

border

Borda utilizada pelas instrução gráfica da `:classe:'~kivy.graphics.vertex_instructions.BorderImage'`, usada pela *background_image*. Pode ser alterado por um fundo personalizado.

It must be a list of four values: (bottom, right, top, left). Read the BorderImage instructions for more information.

border é um *ListProperty* e o padrão é (16, 16, 16, 16)

content

Este é o objeto que mantém (o conteúdo de *current_tab* é adicionado a isso) o conteúdo da Tab atual. Para Listen as alterações do conteúdo da Tab atual, deverás vincular a propriedade do propriedade *current_tabs content*.

content é uma *ObjectProperty* e o padrão é ‘None’.

current_tab

Links para a Tab atualmente selecionada ou ativa.

Novo na versão 1.4.0.

current_tab é uma *AliasProperty*, somente leitura.

default_tab

Mantém a Tab padrão.

Nota: Por conveniência, a Aba padrão fornecida automaticamente é excluída quando você altera `default_tab` para outra coisa. A partir da versão 1.5.0, esse comportamento foi estendido para cada `default_tab` para consistência e não apenas o fornecido automaticamente.

`default_tab` é uma *AliasProperty*.

default_tab_cls

Especifica a classe a ser usada para o estilo da Aba padrão.

Novo na versão 1.4.0.

Aviso: `default_tab_cls` deve ser uma subclasse de `TabbedPanelHeader`

`default_tab_cls` é um *ObjectProperty* e o padrão é `TabbedPanelHeader`. Se você definir uma String, a *Factory* será usada para definir a classe.

Alterado na versão 1.8.0: A *Factory* resolverá a classe se uma string estiver definida.

default_tab_content

Contém o conteúdo da Aba padrão.

`default_tab_content` é uma *AliasProperty*.

default_tab_text

Especifica o texto exibido no cabeçalho da Aba padrão.

`default_tab_text` é uma *StringProperty* e o padrão é 'default tab'.

do_default_tab

Especifica se um cabeçalho `default_tab` será fornecida.

Novo na versão 1.5.0.

`do_default_tab` é uma *BooleanProperty* e o padrão é 'True'.

strip_border

Borda a ser usada com `strip_image`.

Novo na versão 1.8.0.

`strip_border` é uma *ListProperty* e o padrão é [4, 4, 4, 4].

strip_image

Imagem de fundo da tira com as Abas.

Novo na versão 1.8.0.

`strip_image` é uma *StringProperty* e o padrão é uma imagem vazia.

switch_to(*header, do_scroll=False*)

Alternar para um cabeçalho de painel específico.

Alterado na versão 1.10.0.

Se usado com *do_scroll=True*, ele também rolará para a Aba do cabeçalho.

tab_height

Determina a altura da Aba de cabeçalho.

`tab_height` é uma *NumericProperty* e o padrão é 40.

tab_list

Lista de todos os cabeçalhos de tabulação.

`tab_list` é uma *AliasProperty* e é somente leitura.

tab_pos

Especifica a posição das Abas em relação ao conteúdo. Pode ser uma das seguintes opções: *left_top, left_mid, left_bottom, top_left, top_mid, top_right, right_top, right_mid, right_bottom, bottom_left, bottom_mid, bottom_right*.

`tab_pos` é uma *OptionProperty* e o padrão é 'top_left'.

tab_width

Especifica a largura do cabeçalho da Tab.

`tab_width` é uma *NumericProperty* e o padrão é 100.

class kivy.uix.tabbedpanel.TabbedPanelContent(kwargs)**

Bases: *kivy.uix.floatlayout.FloatLayout*

A classe TabbedPanelContent.

class kivy.uix.tabbedpanel.TabbedPanelHeader(kwargs)**

Bases: *kivy.uix.togglebutton.ToggleButton*

Uma base para a implementação de Tabbed Panel Head. Um botão destinado a ser usado como Título/Aba para o Widget TabbedPanel.

Você pode usar este Widget *TabbedPanelHeader* para adicionar uma nova Tab a um *TabbedPanel*.

content

Conteúdo a ser carregado quando este cabeçalho da Tab for selecionado.

`content` é uma *ObjectProperty* e o padrão é *None*.

class kivy.uix.tabbedpanel.TabbedPanelItem(kwargs)**

Bases: *kivy.uix.tabbedpanel.TabbedPanelHeader*

Esta é uma classe de conveniência que fornece um cabeçalho do tipo TabbedPanelHeader e vincula-o com o conteúdo automaticamente. Isso facilita a você fazer simplesmente o seguinte com a linguagem kv:

```
<TabbedPanel>:  
    # ...other settings  
    TabbedPanelItem:  
        BoxLayout:  
            Label:  
                text: 'Second tab content area'  
            Button:  
                text: 'Button that does nothing'
```

Novo na versão 1.5.0.

```
class kivy.uix.tabbedpanel.TabbedPanelStrip(**kwargs)  
Bases: kivy.uix.GridLayout.GridLayout
```

Uma tira destinada a ser usada como fundo para Cabeçalho/Aba. Isso não cobre as áreas em branco no caso das Abas não cobrirem toda a largura/altura do TabbedPanel (use *StripLayout* for that).

tabbed_panel

Link para o painel que a tira da Aba é uma parte de.

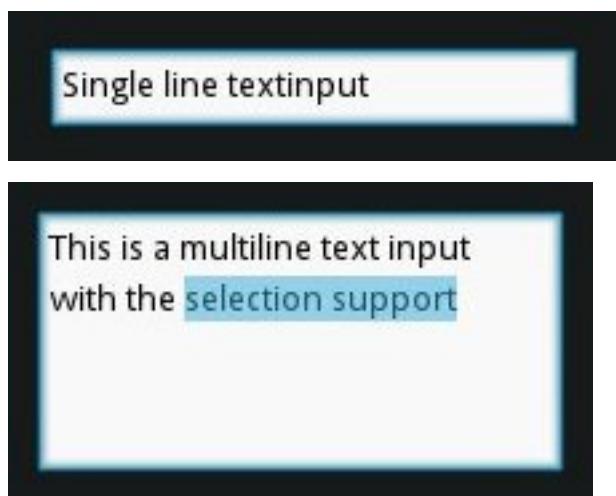
tabbed_panel é uma *ObjectProperty* e o padrão é *None*.

```
class kivy.uix.tabbedpanel.TabbedPanelException  
Bases: exceptions.Exception
```

A classe TabbedPanelException.

4.15.48 Entrada de Texto

Novo na versão 1.0.4.



A *TextInput* widget fornece uma caixa de texto editável.

Os recursos unicode, múltiplas linhas, navegação do cursor, seleção e cópia para área de transferência são suportados.

A *TextInput* utiliza dois tipos diferentes de sistema de coordenadas:

- (x, y) - coordenadas em pixels, mais utilizado para renderizar na tela.
- (row, col) - índice do cursor em caracteres / linhas, utilizado para seleção e movimento do cursor.

Exemplo de uso

Para criar multiline *TextInput* (Ao pressionar a tecla ‘enter’ será adicionada uma nova linha):

```
from kivy.uix.textinput import TextInput  
textinput = TextInput(text='Hello world')
```

Para criar sigleline *TextInput*, setar a propriedade *TextInput.multiline* para False (Ao pressionar a tecla ‘enter’ é disparado o evento ‘on_text_validate’):

```
def on_enter(instance, value):  
    print('User pressed enter in', instance)  
  
textinput = TextInput(text='Hello world', multiline=False)  
textinput.bind(on_text_validate=on_enter)
```

As entradas de texto são armazenadas na propriedade *TextInput.text*. Para executar um retorno de chamado enquanto o texto é alterado:

```
def on_text(instance, value):  
    print('The widget', instance, 'have:', value)  
  
textinput = TextInput()  
textinput.bind(text=on_text)
```

Você precisa setar na *focus* para a Texinput, isso significa que a caixa de texto será realçada e o foco do teclado será solicitado.

```
textinput = TextInput(focus=True)
```

O foco do textinput é retirado quando pressionamos a tecla ‘esc’, ou se outro widget é requisitado pelo teclado. Se você ligar uma chamada de retorno a propriedade focus pega a notificação do focus alterado:

```
def on_focus(instance, value):  
    if value:  
        print('User focused', instance)  
    else:  
        print('User defocused', instance)
```

```
textinput = TextInput()  
textinput.bind(focus=on_focus)
```

Olhe a *FocusBehavior*, de qual *TextInput* herda, para mais detalhes.

Seleção

A seleção é automaticamente atualizada quando a posição do cursor muda. Você pode pegar o texto atualmente selecionado pela propriedade *TextInput.selection_text*.

Filtragem

Você pode controlar qual texto pode ser adicionado ao *TextInput* através da sobrescrita do método *TextInput.insert_text()*. Cada String que é digitada, colada ou inserida por outros meios em um *TextInput* é passada através desta função. Pela sobrescrita do método é possível rejeitar ou modificar caracteres.

Por exemplo, para escrever somente caractere maiúsculos:

```
class CapitalInput(TextInput):  
  
    def insert_text(self, substring, from_undo=False):  
        s = substring.upper()  
        return super(CapitalInput, self).insert_text(s, from_  
undo=from_undo)
```

Ou para permitir somente número com ponto flutuante (0 - 9 e um período simples):

```
class FloatInput(TextInput):  
  
    pat = re.compile('^[0-9]+')  
    def insert_text(self, substring, from_undo=False):  
        pat = self.pat  
        if '.' in self.text:  
            s = re.sub(pat, '', substring)  
        else:  
            s = ('.').join([re.sub(pat, '', s) for s in substring.  
split('.', 1)])  
        return super(FloatInput, self).insert_text(s, from_  
undo=from_undo)
```

Teclas de Atalho Padrões

Teclas de Atalho	Descrição
Esquerda	Move o Cursor para a Esquerda
Direita	Move o Cursor para a Direita
Cima	Move o Cursor pra Cima
Baixo	Move o Cursor pra Baixo
Home	Move o Cursor para o início da linha
End	Move o Cursor para o Final da Linha
PageUp	Move o Cursor a 3 linhas atrás
PageDown	Move o Cursor a 3 Linhas a Frente
Backspace	Deleta a seleção ou o caractere após o cursor
Del	Deleta a seleção ou o caractere após o Cursor
Shift + <dir>	Inicia o texto selecionado. <dir> pode ser 'Up', 'Down', 'Left' ou 'Right'
Control + C	Copia a seleção
Control + X	Recorta a seleção
Control + P	Cola a seleção
Controle + A	Seleciona todo o conteúdo
Control + Z	Desfaz
Control + R	Refaz

Nota: Para ativar as teclas de atalho do estilo Emacs, podes utilizar [EmacsBehavior](#).

```
class kivy.uix.textinput.TextInput(**kwargs)
    Bases:      kivy.uix.behaviors.focus.FocusBehavior, kivy.uix.widget.Widget
```

Class TextInput. Veja o módulo da documentação para maiores informações.

Events

on_text_validate Disparado somente no modo *multiline=False* quando o usuário pressiona o 'Enter'. Isso também desfoca o TextInput.

on_double_tap Disparado quando um duplo toque acontecer num TextInput. O comportamento padrão seleciona o texto ao redor da posição do Cursor. Para mais informações veja [on_double_tap\(\)](#).

on_triple_tap Disparado quando um triplo toque acontecer dentro do TextInput. O comportamento padrão seleciona a linha ao redor da posição do Cursor. Mais informações em [on_triple_tap\(\)](#).

on_quad_touch Disparado quando quatro dedos estão tocando o

`TextInput`. O comportamento padrão é selecionar todo o texto.
Para mais informações veja [on_quad_touch\(\)](#).

Aviso: Ao alterar uma propriedade `TextInput` que requer o re-desenho, por exemplo, modificando `text`, as atualizações ocorrem no próximo ciclo de Clock e não instantaneamente. Isso pode causar alterações na classe `TextInput` que ocorrem entre a modificação e o próximo ciclo a serem ignorados, ou usar valores anteriores. Por exemplo, depois de uma atualização do `text`, mudar o cursor no mesmo frame do Clock irá movê-lo usando o texto anterior e provavelmente acabará em uma posição incorreta. A solução é programar todas as atualizações pra ocorrerem no próximo ciclo do Clock usando `schedule_once()`.

Nota: Seleção é cancelada quando o `TextInput` está focado. Se você precisar mostrar a seleção quando o `TextInput` está focado, você deve atrasar (usar `Clock.schedule`) a chamada da função para selecionar o texto (`select_all`, `select_text`).

Alterado na versão 1.10.0: `background_disabled_active` foi removido.

Alterado na versão 1.9.0: `:classe:'TextInput'` agora herda da classe: `~kivy.uix.behaviors.FocusBehavior`.
`keyboard_mode`, `meth:~kivy.uix.behaviors.FocusBehavior.hide_keyboard`,
`:meth:~kivy.uix.behaviors.FocusBehavior.focus` e `:attr:~kivy.uix.behaviors.FocusBehavior.inp` tem sido removido desde que eles são herdados da classe: `~kivy.uix.behaviors.FocusBehavior`.

Alterado na versão 1.7.0: `on_double_tap`, `on_triple_tap` and `on_quad_touch` eventos adicionados.

allow_copy

Decide se é permitido copiar o texto.

Novo na versão 1.8.0.

`allow_copy` é uma `BooleanProperty` e o padrão é `True`.

auto_indent

Texto Multiline automaticamente Indentado.

Novo na versão 1.7.0.

`auto_indent` é uma `BooleanProperty` e o padrão é `False`.

background_active

Plano de Fundo da imagem do `TextInput` quando o mesmo está com o foco.

Novo na versão 1.4.1.

`background_active` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/textinput_active’.

background_color

Cor atual do plano de fundo no formato (r, g, b, a).

Novo na versão 1.2.0.

`background_color` é uma *ListProperty* e o padrão é [1, 1, 1, 1] (branco).

background_disabled_normal

Imagem de fundo do TextInput quando desativado.

Novo na versão 1.8.0.

`background_disabled_normal` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/textinput_disabled’.

background_normal

Imagem do plano de fundo do TextInput quando ele não estiver focado.

Novo na versão 1.4.1.

`background_normal` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/textinput’.

border

Borda usada pela *BorderImage* instrução gráfica. Usado com `background_normal` e `background_active`. Pode ser usado para um fundo personalizado.

Novo na versão 1.4.1.

It must be a list of four values: (bottom, right, top, left). Read the BorderImage instruction for more information about how to use it.

`border` é uma *ListProperty* e o padrão é (4, 4, 4, 4).

cancel_selection()

Canca a Seleção Atual (caso exista).

copy(*data*=“”)

Copia o valor fornecido no argumento *data* dentro da área de transferência atual. Se os dados não forem do tipo String eles serão convertido para String. Se nenhum dado for fornecido, então a seleção atual, se presente, será copiada.

Novo na versão 1.8.0.

cursor

Tupla com valores (linha, coluna) indicando a posição atual do Cursor. Podes definir uma nova tupla (linha, coluna) se desejas mover o Cursor. A área de rolagem será automaticamente atualizada para assegurar que o Cursor esteja visível dentro da janela de visualização (ViewPort).

cursor é uma *AliasProperty*.

cursor_blink

Esta propriedade é usada para piscar o Cursor gráfico. O valor de *cursor_blink* é automaticamente calculado. Definir um valor aqui não terá qualquer impacto.

cursor_blink é uma *BooleanProperty* e o padrão é *False*.

cursor_col

Coluna atual do Cursor.

cursor_col é uma *AliasProperty* para cursor[0], somente leitura.

cursor_color

Cor atual do Cursor no formato (r, g, b, a).

Novo na versão 1.9.0.

cursor_color é uma *ListProperty* e o padrão é [1, 0, 0, 1].

cursor_index(cursor=None)

Retorna o índice do curso no texto/valor.

cursor_offset()

Obtém o deslocamento em X do Cursor na linha atual.

cursor_pos

Posição atual do cursor em (x, y).

cursor_pos é uma *AliasProperty*, somente leitura.

cursor_row

Linha atual do Cursor.

cursor_row é uma *AliasProperty* para cursor[1], somente leitura.

cursor_width

Current width of the cursor.

Novo na versão 1.10.0.

cursor_width is a *NumericProperty* and defaults to '1sp'.

cut()

Copia a seleção atual para a área de transferência e então deleta o texto do TextInput.

Novo na versão 1.8.0.

delete_selection(from_undo=False)

Deleta o texto atualmente selecionado (caso tenha).

disabled_foreground_color

Cor atual do plano quando desabilitado no formato (r, g, b, a).

Novo na versão 1.8.0.

disabled_foreground_color é uma *ListProperty* e o padrão é [0, 0, 0, 5] (50% transparência preta).

do_backspace(*from_undo=False, mode='bkspc'*)

Faz a operação de Backspace na posição atual do Cursor. Esta ação pode fazer coisas diferentes:

- Remoção da seleção atual se disponível.
- Removendo o caractere anterior e movendo o cursor de volta.
- Não faz nada, se nós estivermos no início.

do_cursor_movement(*action, control=False, alt=False*)

Move o Cursor relativo para a posição atual. Esta ação pode ser uma de:

- *cursor_left*: move o Cursor para a esquerda
- *cursor_right*: move o Cursor para a direita
- *cursor_up*: move o Cursor para a linha anterior
- *cursor_down*: move o Cursor para a próxima linha
- *cursor_home*: move o Cursor para o início da linha atual
- *cursor_end*: move o Cursor para o final da linha atual
- *cursor_pgup*: move uma “página” anterior
- *cursor_pgdown*: move uma “page” a frente

Além disso, o comportamento de certas ações podem ser modificadas:

- *control + cursor_left*: move o Cursor uma palavra para a esquerda
- *Control + cursor_right*: move o Cursor uma palavra para a direita
- *control + cursor_up*: rola a uma linha acima
- *control + cursor_down*: rola para uma linha abaixo
- *control + cursor_home*: vá para o início do texto
- *control + cursor_end*: vá para o final do texto
- *alt + cursor_up*: move linha(s) pra cima
- *alt + cursor_down*: move linha(s) pra baixo

Alterado na versão 1.9.1.

do_redo()

Refaz a operação.

Novo na versão 1.3.0.

Esta ação refaz qualquer comando que tenha sido desfeita pelo uso de `do_undo()`/`Ctrl + Z`. Esta função é automaticamente chamada quando as teclas `Ctrl + R` forem pressionadas.

do_undo()

Volta a operação anterior.

Novo na versão 1.3.0.

Esta ação refaz qualquer edição que tenha sido feita desde a última chamada para `reset_undo()`. Esta função é automaticamente chamada quando as teclas `Ctrl + Z` são pressionadas.

font_name

Nome do arquivo de fonte utilizado. O path pode ser absoluto ou relativo. Path relativo são resolvidos pela função `resource_find()`.

Aviso: Dependendo do seu provedor de texto, o arquivo de Fonte pode ser ignorado. No entanto, você pode usar isso sem problemas.

Se a fonte usada não tiver os caracteres/ícone para o idioma/símbolos específicos que você está usando, será exibido o caracteres caixa em branco '[]' ao invés dos glifos reais. A solução é usar uma fonte que tenha os símbolos necessários para a exibição. Por exemplo, para exibir , use uma fonte como a freesans.ttf que tenha determinado símbolo.

`font_name` é uma *StringProperty* e o padrão é 'Roboto'. This value is taken from `Config`.

font_size

Tamanho da fonte do texto em pixels.

`font_size` é uma *NumericProperty* e o padrão é `15sp`.

foreground_color

Cor atual do primeiro plano no formato (r, g, b, a).

Novo na versão 1.2.0.

`foreground_color` é uma *ListProperty* e o padrão é [0, 0, 0, 1] (preto).

get_cursor_from_index(index)

Retorna o (linha, coluna) do Cursor para o índice do texto.

get_cursor_from_xy(x, y)

Retorna o (linha, coluna) do Cursor para uma posição (x, y).

handle_image_left

Imagem usada para exibir a ?alça? (handle) esquerdo da seleção do TextInput.

Novo na versão 1.8.0.

`handle_image_left` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/selector_left’.

handle_image_middle

Imagen usada para exibir a metade do handle do TextInput para o posicionamento do Cursor.

Novo na versão 1.8.0.

`handle_image_middle` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/selector_middle’.

handle_image_right

Imagen usada para exibir a handle a direita do TextInpt para a seleção.

Novo na versão 1.8.0.

`handle_image_right` é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/selector_right’.

hint_text

Exibe o texto do Widget, mostrando se o texto é ‘’.

Novo na versão 1.6.0.

Alterado na versão 1.10.0: A propriedade é agora um AliasProperty e os valores em bytes são decodificados para o tipo String. O text hint ficará visível quando o Widget estiver focado.l

`hint_text` é uma *AliasProperty* e o padrão é ‘’.

hint_text_color

Cor atual do texto text_hint no formato (r, g, b, a).

Novo na versão 1.6.0.

`hint_text_color` é uma *ListProperty* e predefinido para [0.5, 0.5, 0.5, 1.0] (cinza).

input_filter

Filtrar a entrada de acordo com o modo especificado, caso não seja *None*. Se *None*, nenhum filtro é aplicado.

Novo na versão 1.9.0.

`input_filter` é uma *ObjectProperty* e o padrão é *None*. Pode assumir uns dos seguintes valores *None*, ‘int’ (string), ou ‘float’ (string), ou um callable. Caso seja um ‘int’, somente números serão aceitos. Caso seja um ‘float’, também só será aceito períodos simples. Finalmente, se for um callable será chamado com dois parâmetros; a String a ser adicionada e um *bool* indicando se a String é um resultado de undo (True). O callable deve retornar um nova substring que será ao usada ao invés da String recebida.

insert_text(*substring*, *from_undo=False*)

Insere novo texto na posição atual do Cursor. Substitua essa função para pré-processar o texto na validação de texto.

keyboard_suggestions

Se *True* fornece sugestões de auto no topo do teclado. Isso só funcionará se *input_type* estiver definido como *text*.

Novo na versão 1.8.0.

keyboard_suggestions é uma *BooleanProperty* e o padrão é *True*.

line_height

Altura da linha. Essa propriedade é automaticamente computada desde o *font_name*, *font_size*. Alteração em *line_height* não haverá impacto.

Nota: *line_height* é a altura de uma simples linha de texto. Use *minimum_height*, que também inclui o padding, para pegar a altura necessária para exibir o texto corretamente.

line_height é uma *NumericProperty*, somente leitura.

line_spacing

Espaço ocupado entre as linhas.

Novo na versão 1.8.0.

line_spacing é uma *NumericProperty* e o padrão é *0*.

minimum_height

Altura mínima do conteúdo dentro do TextInput.

Novo na versão 1.8.0.

minimum_height é uma classe somente leitura *AliasProperty*.

Aviso: *minimum_width* é calculado com base no *width*, portanto, código como este conduzirá a um loop infinito.

```
<FancyTextInput>:  
    height: self.minimum_height  
    width: self.height
```

multiline

Se *True*, o Widget estará apto a exibir múltiplas linhas de texto. Se *False*, a tecla “Enter” irá desfocar o TextInput ao invés de adicionar uma nova linha.

multiline é uma *BooleanProperty* e o padrão é *True*.

on_double_tap()

Este evento é despachado quando um duplo toque acontecer dentro do TextInput. O comportamento padrão é o de selecionar uma palavra próxima a posição do Cursor. Substitua isso para fornecer um comportamento diferente. Como alternativa, você pode vincular a este evento e fornecer funcionalidades adicionais.

on_quad_touch()

Este evento é despachado quando quatro dedos estão tocando dentro do TextInput. O comportamento padrão é selecionar o texto. Substitua isso para fornecer funcionalidades adicionais. Como alternativa, você pode vincular a este evento e fornecer funcionalidades adicionais.

on_triple_tap()

Este evento é despachado quando triplo toque ocorrer dentro do TextInput. O comportamento padrão é selecionar o texto. Substitua isso para fornecer funcionalidades adicionais. Como alternativa, você pode vincular a este evento e fornecer funcionalidades adicionais.

padding

Padding do texto: [padding_left, padding_top, padding_right, padding_bottom].

O Padding também pode ser definido numa forma que tenha dois argumentos [padding_horizontal, padding_vertical] and a one argument form [padding].

Alterado na versão 1.7.0: Substituído AliasProperty por VariableListProperty.

padding é uma *VariableListProperty* e o padrão é [6, 6, 6, 6].

padding_x

Padding horizontal do texto: [padding_left, padding_right].

padding_x também aceita uma forma de argumento [padding_horizontal].

padding_x é uma *VariableListProperty* e o padrão é [0, 0]. Isso pode ser alterado pelo tema atual.

Obsoleto desde a versão 1.7.0: Ao invés, utilize *padding*.

padding_y

Padding vertical do texto: [padding_top, padding_bottom].

padding_y também aceita uma forma de argumento: [padding_vertical].

padding_y é uma *VariableListProperty* e o padrão é [0, 0]. Isso pode ser modificado pelo tema atual.

Obsoleto desde a versão 1.7.0: Ao invés, utilize *padding*.

password

Se *True*, o Widget exibirá os caracteres como como o caractere definido em

password_mask

Novo na versão 1.2.0.

password é uma *BooleanProperty* e o padrão é *False*.

password_mask

Define o caractere usado pra mascarar o texto quando *password* é *True*.

Novo na versão 1.10.0.

password_mask é uma *StringProperty* e o padrão é '*'.

paste()

Insere texto desde o sistema Clipboard dentro do *TextInput* na posição atual do Cursor.

Novo na versão 1.8.0.

readonly

Se *True*, o usuário não será capaz de alterar o conteúdo do *TextInput*.

Novo na versão 1.3.0.

readonly é uma *BooleanProperty* e o padrão é *False*.

replace_crlf

Automaticamente substitui o *CRLF* por *LF*.

Novo na versão 1.9.1.

replace_crlf é uma *BooleanProperty* e o padrão é *True*.

reset_undo()

Redefine o “desfazer” e “refazer” das listas na memória.

Novo na versão 1.3.0.

scroll_x

Valor X de rolagem do *ViewPort*. A rolagem é atualizada automaticamente quando o cursor é movido ou o texto é alterado. Se não houver entrada do usuário, as propriedades *scroll_x* e *scroll_y* poderão ser alteradas.

scroll_x é uma *NumericProperty* e o padrão é 0.

scroll_y

Valor Y de rolagem do ViewPort. Veja *scroll_x* para maiores informações.

scroll_y é uma *NumericProperty* e o padrão é 0.

select_all()

Seleciona todo o texto exibido dentro do *TextInput*.

Novo na versão 1.4.0.

select_text(start, end)

Seleciona uma porção de texto exibido dentro do *TextInput*.

Novo na versão 1.4.0.

Parameters

start Índice do *TextInput.text* desde onde inicia a seleção

end Índice do *TextInput.text* até que a seleção seja exibida.

selection_color

Cor atual da seleção no formato (r, g, b, a).

Aviso: A cor deve sempre ter um componente “alfa” inferior a 1 uma vez que a seleção é desenhada após o texto.

selection_color é uma *ListProperty* e o padrão é [0.1843, 0.6549, 0.8313, .5].

selection_from

Se uma seleção está em progresso ou completa, esta propriedade representará o índice do Cursor onde a seleção inicia.

Alterado na versão 1.4.0: *selection_from* é uma *AliasProperty* e o padrão é *None*, somente leitura.

selection_text

Seleção do Conteúdo Atual

selection_text é uma *StringProperty* e o padrão é “”, somente leitura.

selection_to

Se uma seleção está em progresso ou completa, esta propriedade representará o índice do Cursor onde a seleção inicia.

Alterado na versão 1.4.0: *selection_to* é uma *AliasProperty* e o padrão é *None*, somente leitura.

suggestion_text

Mostra um texto de sugestão no final da linha atual. O recurso é utilizado no sistema de autocompletar e não há implementação de validação (aceitando o texto sugerido ao entrar e etc.). Isso também pode ser usado pelo IME para instalar a palavra atual que está sendo editada.

Novo na versão 1.9.0.

suggestion_text é uma *StringProperty* e o padrão é “”.

tab_width

Por padrão, cada Tab será substituído por quatro espaços no Widget *TextInput*. É possível definir uma quantidade de espaços maior ou menor do que a padrão.

tab_width é uma *NumericProperty* e o padrão é 4.

text

Texto do Widget

Criação de um simples Hello World:

```
widget = TextInput(text='Hello world')
```

Se você deseja criar o Widget com uma String Unicode, use:

```
widget = TextInput(text=u'My unicode string')
```

text é uma *AliasProperty*.

use_bubble

Indica se o balão de recortar/copiar/colar é usado.

Novo na versão 1.7.0.

use_bubble é uma *BooleanProperty* e o padrão é *True* em SO's mobile, *False* em SO's Desktops.

use_handles

Indica se o handle de seleção está sendo exibido.

Novo na versão 1.8.0.

use_handles é uma *BooleanProperty* e o padrão é *True* em SO mobile, *False* se estiver em SO's Desktop.

write_tab

Se a tecla Tab fornecer o foco para o próximo Widget ou se ele deve inserir o caractere no *TextInput*. Se *True* um Tab será escrito, senão, o foco será movido para o próximo Widget.

Novo na versão 1.9.0.

write_tab é uma *BooleanProperty* e o padrão é *True*.

4.15.49 Botão de alternar

O Widget da classe *ToggleButton* funciona como um *CheckBox*. Quando você tocar/clicar nele, o estado é alterado entre 'normal' e 'down' (como oposto a *Button* que somente está 'down' quando realmente estiver pressionado).

O *ToggleButton* pode também ser agrupado como os *RadioButton* - somente um Botão no grupo pode estar no modo 'down'. O nome do grupo pode ser uma String ou qualquer outro valor de objeto Python hashable.

```
btn1 = ToggleButton(text='Male', group='sex')
btn2 = ToggleButton(text='Female', group='sex', state='down')
btn3 = ToggleButton(text='Mixed', group='sex')
```

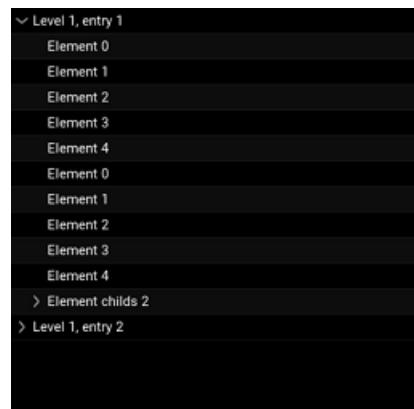
Somente um botão pode estar ao mesmo tempo 'down'/checked.

Para configurar o ToggleButton, você pode utilizar a mesma propriedade que utilizastes na classe [Button](#).

```
class kivy.uix.togglebutton.ToggleButton(**kwargs)
    Bases: kivy.uix.behaviors.togglebutton.ToggleButtonBehavior,
kivy.uix.button.Button
```

Classe ToggleButton, veja o módulo da documentação para maiores informações.

4.15.50 Tree View



Novo na versão 1.0.4.

[TreeView](#) é um widget que representa o conteúdo em formato de árvore. É um recurso bastante básico, com recursos mínimos.

Prefácio

Uma: classe: [TreeView](#) é preenchido com intâncias da: classe: [TreeViewNode](#), mas você não pode usar uma: classe: [TreeViewNode](#) diretamente. Você deve combiná-lo com outro widget, como a: classe: [~kivy.uix.label.Label](#), :classe:' [~kivy.uix.button.Button](#)' ou até mesmo seu próprio widget. O TreeView sempre cria um nó raiz 'root' padrão, baseado na: classe: [TreeViewLabel](#).

[TreeViewNode](#) é um objeto de classe contendo as propriedades necessárias para servir como um nó da árvore. Estender [TreeViewNode](#) para criar tipos de nó personalizados para uso com a [TreeView](#).

Para construir a sua própria subclasse, siga o padrão do [TreeViewLabel](#) que combina um [Label](#) com um [TreeViewNode](#), produzindo uma [TreeViewLabel](#) para uso direto

numa instância de TreeView.

Para usar a classe TreeViewLabel, você pode criar dois nós conectados diretamente ao root:

```
tv = TreeView()
tv.add_node(TreeViewLabel(text='My first item'))
tv.add_node(TreeViewLabel(text='My second item'))
```

Ou, crie dois nós conectados a um primeiro:

```
tv = TreeView()
n1 = tv.add_node(TreeViewLabel(text='Item 1'))
tv.add_node(TreeViewLabel(text='SubItem 1'), n1)
tv.add_node(TreeViewLabel(text='SubItem 2'), n1)
```

Se tiveres uma grande estrutura hierárquica, talvez precisaras de uma função utilitária para preencher a exibição da árvore:

```
def populate_tree_view(tree_view, parent, node):
    if parent is None:
        tree_node = tree_view.add_node(TreeViewLabel(text=node[
            'node_id'],
                                                       is_open=True))
    else:
        tree_node = tree_view.add_node(TreeViewLabel(text=node[
            'node_id'],
                                                       is_open=True),
                                       parent)

    for child_node in node['children']:
        populate_tree_view(tree_view, tree_node, child_node)

tree = {'node_id': '1',
        'children': [{ 'node_id': '1.1',
                      'children': [{ 'node_id': '1.1.1',
                                    'children': [{ 'node_id': '1.1.1.
                                         1',
                                         'children': []}],
                                    'children': []},
                                   { 'node_id': '1.1.2',
                                     'children': []},
                                   { 'node_id': '1.1.3',
                                     'children': []}]},
                    { 'node_id': '1.2',
                      'children': []}]}

class TreeWidget(FloatLayout):
```

```

def __init__(self, **kwargs):
    super(TreeWidget, self).__init__(**kwargs)

    tv = TreeView(root_options=dict(text='Tree One'),
                  hide_root=False,
                  indent_level=4)

    populate_tree_view(tv, None, tree)

    self.add_widget(tv)

```

O widget raiz / 'root' na exibição em árvore é aberto por padrão e tem o texto definido como 'Root'. Se você quiser alterar isso, você pode usar a propriedade: attr: *TreeView.root_options*. Isso irá passar opções para o widget raiz/'root':

```
tv = TreeView(root_options=dict(text='My root label'))
```

Criando seu próprio Widget de nó

Para nó tipo Button, combine uma *Button* e uma *TreeViewNode* como pode ser visto a seguir:

```

class TreeViewButton(Button, TreeViewNode):
    pass

```

Você deve saber que, para um determinado nó, somente o: attr: *~kivy.uix.widget.Widget.size_hint_x* será honrado. A largura alocada para o nó dependerá da largura atual do TreeView e do nível do nó. Por exemplo, se um nó estiver no nível 4, a largura alocada será:

*treeview.width - treeview.indent_start - treeview.indent_level *' node.level'*

Podes ter algum problema com isso. É responsabilidade do desenvolvedor lidar corretamente com a adaptação dos nós da representação gráfica, quando necessário.

class kivy.uix.treeview.TreeView(kwargs)**
 Bases: *kivy.uix.widget.Widget*

Classe TreeView. Veja o módulo da documentação para maiores informações.

Events

*on_node_expand: (node,)*Acionado quando um nó está sendo expandido

*on_node_collapse: (node,)*Acionado quando um nó está sendo recolhido

add_node(node, parent=None)

Adiciona um novo nó para a árvore.

Parameters

node: instância de uma **TreeViewNode** Nó para adicionar dentre da árvore

parent: instância de uma **TreeViewNode**, o padrão é *None*
Nó pai pra anexar o novo nó. Se *None*, ele é adicionado ao nó **root**.

Retorna um nó *node*.

deselect_node(*args)

Desmarca qualquer nó selecionado.

Novo na versão 1.10.0.

get_node_at_pos(pos)

Obtém o nó na posição (x, y).

hide_root

Use esta propriedade para mostrar/ocultar o nó da raiz inicial. Se *True*, o nó raiz será exibido como um nó fechado.

hide_root é uma **BooleanProperty** e o padrão é *False*.

indent_level

Largura utilizada para a indentação de cada nível, exceto o primeiro.

O cálculo da indentação para cada nível da árvore é:

```
indent = indent_start + level * indent_level
```

indent_level é uma **NumericProperty** e o padrão é 16.

indent_start

Largura de indentação do nó de nível 0 / raiz 'root'. Este é principalmente o tamanho inicial para acomodar um ícone de árvore (recolhido / expandido). Consulte: attr: *indent_level* para obter mais informações sobre o cálculo de recuo de nível.

indent_start é uma **NumericProperty** e o padrão é 24.

iterate_all_nodes(node=None)

Generator para iterar sobre todos os nós desde *node* para baixo caso esteja expandido ou não. Se *node* for *None*, o Generator começará com **root**.

iterate_open_nodes(node=None)

Gerador para iterar sobre todos os nós gastos a partir de *nó* e baixo. Se *node* for *None*, o gerador começará com: attr: *root*.

Para obter todos os nós abertos

```
treeview = TreeView()  
# ... add nodes ...
```

```
for node in treeview.iterate_open_nodes():
    print(node)
```

load_func

Callback pra ser usado no carregamento assíncrono. Se definido, o carregamento assíncrono será feito automaticamente. O callback deve atuar como uma função generator do Python, usando o rendimento para enviar dados de volta para a árvore.

O callback deve estar no formato:

```
def callback(treeview, node):
    for name in ('Item 1', 'Item 2'):
        yield TreeViewLabel(text=name)
```

load_func é uma *ObjectProperty* e o padrão é *None*.

minimum_height

Altura mínima necessária para conter todas os filhos.

Novo na versão 1.0.9.

minimum_height é uma *kivy.properties.NumericProperty* e o padrão é 0.

minimum_size

Tamanho mínimo necessário para conter todas os filhos.

Novo na versão 1.0.9.

minimum_size é uma propriedade *ReferenceListProperty* de (*minimum_width*, *minimum_height*).

minimum_width

Largura mínima necessária para conter todas os filhos.

Novo na versão 1.0.9.

minimum_width é uma *kivy.properties.NumericProperty* e o padrão é 0.

remove_node(*node*)

Remove todos os nós da árvore.

Novo na versão 1.0.7.

Parameters

node: instância de uma *TreeViewNode*Nó para removido da árvore. Se 'node' é *root*, ele não será removido.

root

Nó da raiz.

Por padrão, o Widget de nó raiz é *TreeViewLabel* com o texto ‘Root’. Se quiseres alterar as opções padrão passadas na criação do Widget, use a propriedade *root_options*:

```
treeview = TreeView(root_options={  
    'text': 'Root directory',  
    'font_size': 15})
```

root_options Irá alterar as propriedades da instância do *TreeViewLabel*. No entanto, você não pode alterar a classe usada para o nó raiz ainda.

root é uma *AliasProperty* e o padrão é *None*. Ele é somente leitura. No entanto, o conteúdo do Widget pode ser alterado.

root_options

Opções de raiz padrão para passar pelo Widget raiz. Consulte a propriedade *root* para obter mais informações sobre o uso de *root_options*.

root_options é uma *ObjectProperty* e o padrão é {}.

select_node(node)

Seleciona um nó na árvore.

selected_node

Nó selecionado pelo *TreeView.select_node()* ou pelo toque.

selected_node é uma *AliasProperty* e o padrão é *None*. Ele é somente leitura.

toggle_node(node)

Alternar o estado do nó (aberto/recolhido).

class kivy.uix.treeview.TreeViewException

Bases: exceptions.Exception

Exceção para erros no *TreeView*.

class kivy.uix.treeview.TreeViewLabel(kwargs)**

Bases: *kivy.uix.label.Label, kivy.uix.treeview.TreeNode*

Combina uma *Label* e uma *TreeNode* para criar um *TreeViewLabel* que pode ser usado como um nó de texto na árvore.

Veja o módulo da documentação para maiores informações.

class kivy.uix.treeview.TreeNode(kwargs)**

Bases: *object*

Classe *TreeNode*, usada para criar uma classe de *node* (nó) para um objeto *TreeView*.

color_selected

Cor do plano de fundo do nó quando o mesmo estiver selecionado.

`color_selected` é uma *ListProperty* e o padrão é [.1, .1, .1, 1].

even_color

Cor de fundo dos nós mesmo quando o nó não está selecionado.

`bg_color` é uma *ListProperty* e o padrão é [.5, .5, .5, 1].

is_leaf

Boolean que indica se este nó é uma folha ou não. Usado para ajustar a representação gráfica.

`is_leaf` é uma *BooleanProperty* e o padrão é *True*. Ele é automaticamente definido como *False* quando um filho é adicionado.

is_loaded

Booleano para indicar se este nó já está carregado ou não. Esta propriedade é usada somente se: classe: *TreeView* usa carregamento assíncrono.

`is_loaded` é uma *BooleanProperty* e o padrão é *False*.

is_open

Booleano para indicar se este nó está aberto ou não, no caso de haver nós filhos. Isso é usado para ajustar a representação gráfica.

Aviso: Esta propriedade é automaticamente definida pela *TreeView*. Você pode lê-la, mas não escreve nela.

`is_open` é um *BooleanProperty* e o padrão é *False*.

is_selected

Boolean para indicar se este nó está selecionado ou não. Isso é usado para ajustar a representação gráfica.

Aviso: Esta propriedade é automaticamente definida pela *TreeView*. Você pode lê-la, mas não escreve nela.

`is_selected` é uma *BooleanProperty* e o padrão é *False*.

level

Nível do nó.

`level` é uma *NumericProperty* e o padrão é -1.

no_selection

Booleano é usado para indicar se a seleção do nó é permitida ou não.

`no_selection` é uma *BooleanProperty* e o padrão é *False*.

nodes

Lista de nós. A lista de nós é diferente da lista de filhos. Um nó na lista de nós representa um nó na árvore. Um item na lista de filhos representa o widget associado ao nó.

Aviso: Esta propriedade é automaticamente definida pela *TreeView*. Você pode lê-la, mas não escreve nela.

nodes é uma *ListProperty* e o padrão é '[]P'.

odd

Essa propriedade é definida pelo Widget *TreeView* automaticamente e é somente leitura.

odd é uma *BooleanProperty* e o padrão é *False*.

odd_color

Cor de fundo de nós estranhos quando o nó não está selecionado.

odd_color é uma *ListProperty* e o padrão é [1., 1., 1., 0.].

parent_node

Nó pai. Esse atributo é necessário porque o *parent* pode ser *None* quando o nó não é exibido.

Novo na versão 1.0.7.

parent_node é uma *ObjectProperty* e o padrão é *None*.

4.15.51 Vídeo

O widget *Video* é usado para reproduzir arquivos ou streams de vídeo. Dependendo do seu dispositivo, plataforma e plugins, poderá ser reproduzido vários formatos. Por exemplo, usando pygame video será suportado apenas o formato MPEG1 no Linux e no OSX. Gstreamer possui mais opções, sendo possível reproduzir videos em MKV, OGV, AVI, MOV, FLV (se tiver os plugins necessários instalados). A implementação de *VideoBase* é feita de forma implícita.

O vídeo é carregado assincronamente - muitas propriedades não estão disponíveis enquanto o vídeo não for carregado (isso é, quando a textura for criada):

```
def on_position_change(instance, value):
    print('The position in the video is', value)
def on_duration_change(instance, value):
    print('The duration of the video is', value)
video = Video(source='PandaSneezes.avi')
video.bind(position=on_position_change,
           duration=on_duration_change)
```

```
class kivy.uix.video.Video(**kwargs)
```

Bases: [kivy.uix.image.Image](#)

Classe *Video*. Consulte a documentação do módulo para obter mais informações.

duration

Duração do vídeo. A duração padrão é -1 e é definida como uma duração real quando o vídeo é carregado.

duration é uma [NumericProperty](#) e o padrão é -1.

eos

Boolean, indica se o vídeo terminou ou não a reprodução (chegou ao final do fluxo).

eos é uma [BooleanProperty](#) e o padrão é *False*.

loaded

Boolean, indica se o vídeo está carregado e pronto ou não para a reprodução

Novo na versão 1.6.0.

loaded é uma [BooleanProperty](#) e o padrão é *False*.

options

Opções a ser passada na criação do objeto de Video principal.

Novo na versão 1.0.4.

options é uma [kivy.properties.ObjectProperty](#) e o padrão é {}.

play

Obsoleto desde a versão 1.4.0: Ao invés, use [state](#).

Boolean, indica se o vídeo está sendo reproduzido ou não. Pode iniciar/parar o vídeo definindo esta propriedade:

```
# start playing the video at creation
video = Video(source='movie.mkv', play=True)

# create the video, and start later
video = Video(source='movie.mkv')
# and later
video.play = True
```

play é uma [BooleanProperty](#) e o padrão é *False*.

Obsoleto desde a versão 1.4.0: Ao invés, use [state](#).

position

Posição do vídeo entre 0 e [duration](#). A posição padrão é -1 e é definida como uma posição real quando o vídeo é carregado.

position é uma [NumericProperty](#) e o padrão é -1.

seek(*percent*)

Altera a posição para a duração percentual. A porcentagem deve ser um valor entre 0-1.

Aviso: Invocar a função *seek()* antes que o vídeo esteja carregado não terá qualquer efeito.

Novo na versão 1.2.0.

state

String, indica se deve reproduzir, pausar ou parar o vídeo:

```
# start playing the video at creation
video = Video(source='movie.mkv', state='play')

# create the video, and start later
video = Video(source='movie.mkv')
# and later
video.state = 'play'
```

state é uma *OptionProperty* e o padrão é ‘stop’.

unload()

Descarrega o vídeo. A reprodução será interrompida.

Novo na versão 1.8.0.

volume

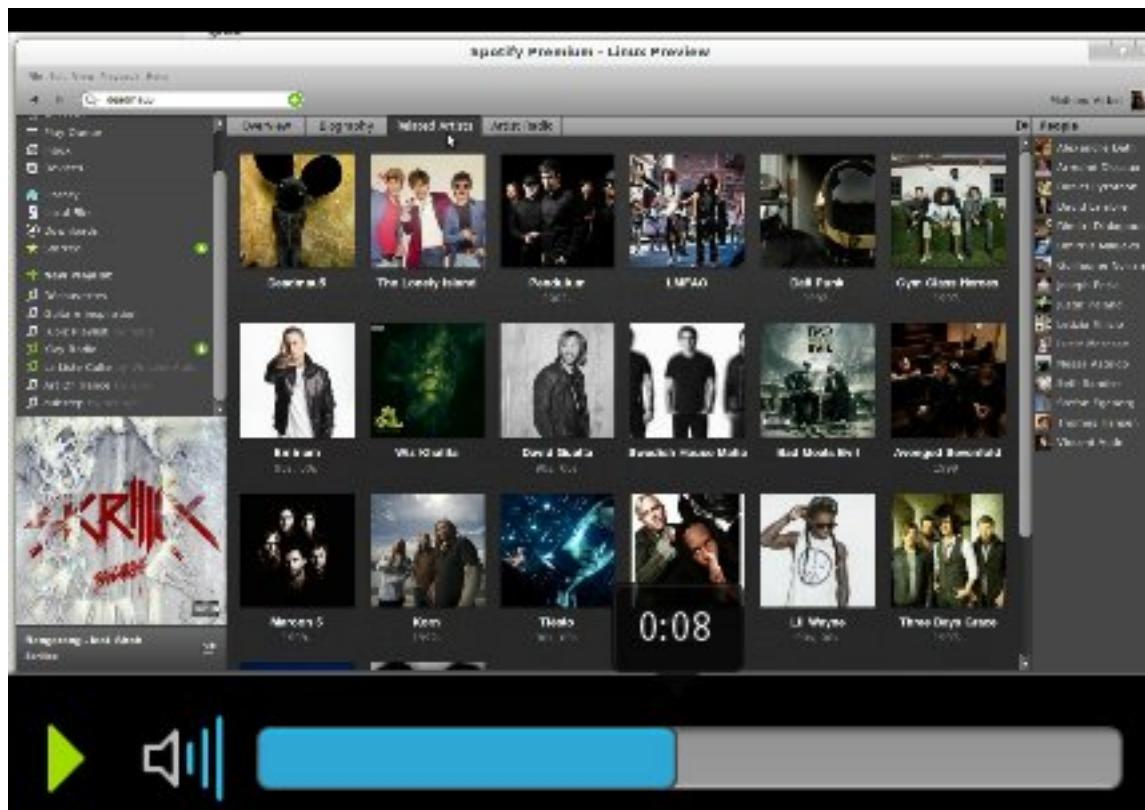
Volume do vídeo, no intervalo de 0-1. 1 significa volume total, 0 significa mudo.

volume é uma *NumericProperty* e o padrão é 1.

4.15.52 Player de Vídeo

Novo na versão 1.2.0.

O widget video player pode ser usado para reproduzir vídeos e permitir que o usuário controle pausa, reprodução, volume e posição. Esse widget não pode ser alterado devido à complexidade de sua construção.



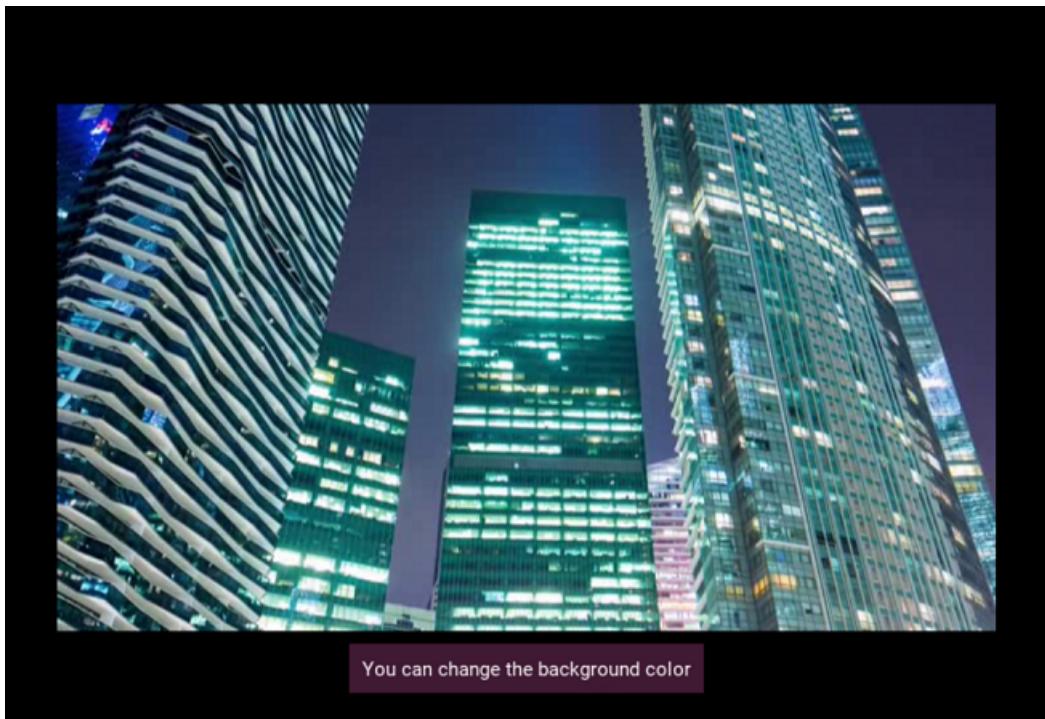
Anotações

Se quiseres exibir texto num momento específico e por um determinado período, considere as anotações. Um arquivo de anotação possui extensão “*.jsa”. O Player carregará automaticamente o arquivo de anotação associado, caso exista.

An annotation file is JSON-based, providing a list of label dictionary items. The key and value must match one of the *VideoPlayerAnnotation* items. For example, here is a short version of a jsa file that you can find in *examples/widgets/cityCC0.jsa*:

```
[  
  {"start": 0, "duration": 2,  
   "text": "This is an example of annotation"},  
  {"start": 2, "duration": 2,  
   "bgcolor": [0.5, 0.2, 0.4, 0.5],  
   "text": "You can change the background color"}  
]
```

For our cityCC0.mpg example, the result will be:



Se quiseres experimentar com arquivos de anotação, teste com:

```
python -m kivy.uix.videoplayer examples/widgets/cityCC0.mpg
```

Fullscreen (Tela Cheia)

O Player de vídeo pode reproduzir o vídeo em tela cheia, se `VideoPlayer.allowFullscreen` for ativado por um duplo toque no vídeo. Por padrão, se o vídeo for menor que a janela, o mesmo não será redimensionado.

Pode permitir o redimensionamento enviando uma opções personalizadas para a instância de `VideoPlayer`:

```
player = VideoPlayer(source='myvideo.avi', state='play',
                      options={'allow_stretch': True})
```

Comportamento do final da execução

Pode especificar o que acontece quando o vídeo terminar de reproduzir, passando uma diretiva `eos` (end of stream (fim de fluxo)) para a classe subjacente `VideoBase`. `Eos` pode ser uma das opções a seguir: 'stop', 'pause' ou 'loop' e o padrão é 'stop'. Por exemplo, para fazer um loop no vídeo faça o seguinte:

```
player = VideoPlayer(source='myvideo.avi', state='play',
                      options={'eos': 'loop'})
```

Nota: A propriedade `eos` da classe `VideoBase` é uma String especificando o comportamento de fim de fluxo (end-of-stream). Esta propriedade difere das propriedades `eos` das classes `VideoPlayer` e `Video`, cuja propriedade `eos` é simplesmente um booleano indicando que o final do arquivo foi atingido.

class kivy.uix.videoplayer.VideoPlayer(kwargs)**
Bases: `kivy.uix.gridlayout.GridLayout`

Classe `VideoPlayer`. Consulte a documentação do módulo para maiores informações.

allowFullscreen

Por padrão, podes dar 2 toques no vídeo para fazê-lo ser exibido em tela cheia. Defina essa propriedade como `False` para evitar esse comportamento.

`allowFullscreen` é uma `BooleanProperty` e o padrão é `True`.

annotations

Se definido, ele será usado na caixa de leitura das anotações.

`annotations` é uma `StringProperty` e o padrão é ''.

duration

Duração do vídeo. A duração padrão é -1 e é definida como a duração real quando o vídeo é carregado.

`duration` é uma `NumericProperty` e o padrão é -1.

fullscreen

Altera para a visualização de tela cheia. Isso deve ser usado com cuidado. Quando ativado, o Widget será removido do seu Widget pai, remover todas os filhos da janela e irá se adicionar ao mesmo. Quando a tela cheia for desativada, todas as crianças anteriores são restauradas e o Widget será restaurado da mesma forma que estava antes.

Aviso: A operação de re-adicionar não se preocupa com a posição do índice de seus filhos dentro do Widget pai.

`fullscreen` é uma `BooleanProperty` e o padrão é `False`.

image_loading

Nome do arquivo de imagem usado quando o vídeo está sendo carregado.

`image_loading` é uma `StringProperty` e o padrão é 'data/images/image-loading.gif'.

image_overlay_play

Nome do arquivo de imagem usado para mostrar uma sobreposição de "Play" antes do vídeo ser exibido.

image_overlay_play é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/player-play-overlay’.

image_pause

Nome do arquivo de imagem usado pelo botão “Pause”.

image_pause é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/media-playback-pause’.

image_play

Nome do arquivo de imagem usado para o botão “Play”.

image_play é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/media-playback-start’.

image_stop

Nome do arquivo de imagem usado para o botão “Stop”.

image_stop é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/media-playback-stop’.

image_volumehigh

Nome do arquivo de imagem usado para o ícone de volume quando o volume está alto.

image_volumehigh é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/audio-volume-high’.

image_volumelow

Nome do arquivo de imagem usado para o ícone de volume quando o volume está baixo.

image_volumelow é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/audio-volume-low’.

image_volumemedium

Nome do arquivo de imagem usado para o ícone de volume quando o volume está medio.

image_volumemedium é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/audio-volume-medium’.

image_volumemuted

Nome do arquivo da imagem usada como o ícone de volume quando estiver mudo.

image_volumemuted é uma *StringProperty* e o padrão é ‘atlas://data/images/defaulttheme/audio-volume-muted’.

options

Os parâmetros opcionais podem ser passados para uma instância: class: ~kivy.uix.video.Video com esta propriedade.

options uma *DictProperty* e o padrão é {}.

play

Obsoleto desde a versão 1.4.0: Ao invés, use **state**.

Boolean, indica se o vídeo está sendo reproduzido ou não. Pode iniciar/parar o vídeo definindo esta propriedade:

```
# start playing the video at creation
video = VideoPlayer(source='movie.mkv', play=True)

# create the video, and start later
video = VideoPlayer(source='movie.mkv')
# and later
video.play = True
```

play é uma **BooleanProperty** e o padrão é *False*.

position

Posição do vídeo entre 0 e: attr: *duration*. A posição padrão é -1 e é definida como a posição real quando o vídeo é carregado.

position é uma **NumericProperty** e o padrão é -1.

seek(percent)

Altera a posição pela porcentagem da duração. A porcentagem deve ser um valor entre 0-1.

Aviso: Invocar *seek()* antes que o vídeo seja carregado não terá qualquer efeito.

source

Fonte do vídeo para leitura.

source é um **StringProperty** e o padrão é “”.

Alterado na versão 1.4.0.

state

String, indica se deve reproduzir, pausar ou parar o vídeo:

```
# start playing the video at creation
video = VideoPlayer(source='movie.mkv', state='play')

# create the video, and start later
video = VideoPlayer(source='movie.mkv')
# and later
video.state = 'play'
```

state é uma **OptionProperty** e o padrão é ‘stop’.

thumbnail

Miniatura do vídeo a ser exibido. Se *None*, o VideoPlayer tentará encontrar a miniatura a partir de *source* + '.png'.

thumbnail uma *StringProperty* e o padrão é ''.

Alterado na versão 1.4.0.

volume

Volume do vídeo no intervalo entre 0-1. 1 significa volume máximo e 0 significa mudo.

volume é uma *NumericProperty* e o padrão é 1.

```
class kivy.uix.videoplayer.VideoPlayerAnnotation(**kwargs)
```

Bases: *kivy.uix.label.Label*

Classe de anotação usada para criar os Labels com as anotações.

Estão disponíveis chaves adicionais:

- *bgcolor*: [r, g, b, a] - cor de fundo da caixa de texto
- *bgsource*: 'filename' - Imagem de fundo usada pela caixa de texto em segundo plano
- *border*: (n, e, s, w) - borda usada pela imagem de fundo

duration

Duração de uma anotação.

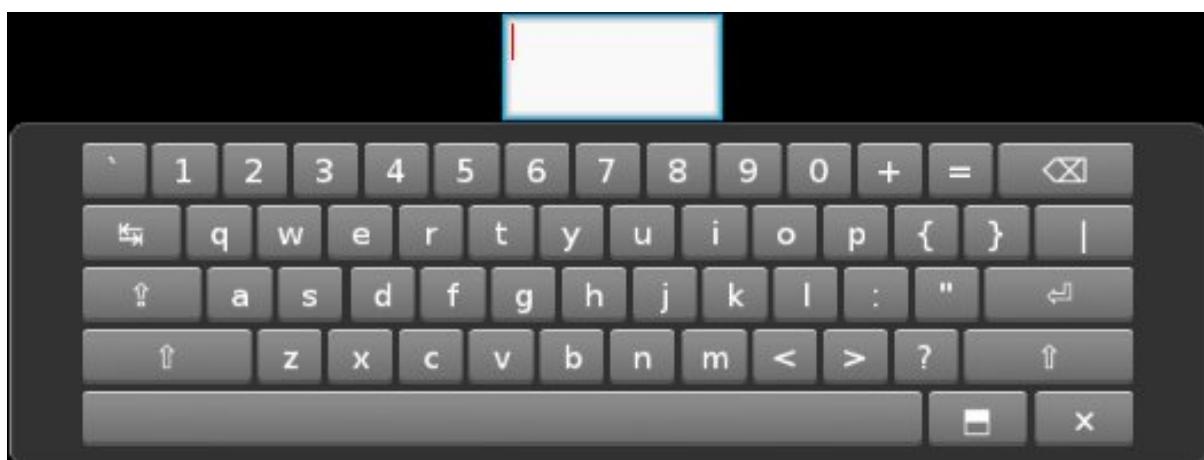
duration é uma *NumericProperty* e o padrão é 1.

start

Tempo de início da anotação.

start é uma *NumericProperty* e o padrão é 0.

4.15.53 VKeyboard



Novo na versão 1.0.8.

VKeyboard é o teclado virtual do kivy. Seu uso é dedicado a ser transparente ao usuário. Não é recomendado usá-lo diretamente. Leia a seção *Request keyboard* para esclarecimentos.

Modos

Este teclado virtual tem um modo ancorado e livre:

- docked mode (*VKeyboard.docked* = True) Geralmente usado quando apenas uma pessoa está usando o computador, como um tablet ou computador pessoal etc.
- free mode: (*VKeyboard.docked* = False) Na maior parte das vezes para superfícies MultiTouch. Este modo permite que vários teclados virtuais sejam usados na tela.

Se o modo fixado alterar, você precisará chamar manualmente *VKeyboard.setup_mode()* caso contrário a mudança não terá impacto. Durante essa chamada, o VKeyboard, implementado em cima de *Scatter*, mudará o comportamento do Scatter e posicionará o teclado perto do alvo (se o destino e o modo fixado estiver definido).

Layouts

O teclado virtual é capaz de carregar um layout personalizado. Se criares um novo layout e colocar o JSON em <kivy_data_dir>/keyboards/<layoutid>.json, poderás carregá-lo configurando :attr:`VKeyboard.layout` para o seu layoutid.

O JSON deve estar estruturado como este:

```
{  
    "title": "Title of your layout",  
    "description": "Description of your layout",  
    "cols": 15,  
    "rows": 5,  
    ...  
}
```

Então, precisarás descrever as Keys em cada linha, para o modo “normal”, “shift” ou um modo “especial” (adicionado na versão 1.9.0). As Keys para esses dados de linha devem ser chamadas *normal_<row>*, *shift_<row>* e *special_<row>*. Substitua *row* pelo número da linha. Dentro de cada linha, você descreverá a Key. Uma Key é uma lista de 4 elementos no formato:

```
[ <text displayed on the keyboard>, <text to put when the key is  
pressed>,  
<text that represents the keycode>, <size of cols> ]
```

Aqui estão exemplos chaves:

```
# f key  
["f", "f", "f", 1]  
# capslock  
["\u21B9", " ", "tab", 1.5]
```

Finalmente, JSON completo:

```
{  
    ...  
    "normal_1": [  
        ["`", "`", "`", 1],      ["1", "1", "1", 1],      ["2", "2", "2",  
         1],  
        ["3", "3", "3", 1],      ["4", "4", "4", 1],      ["5", "5", "5",  
         1],  
        ["6", "6", "6", 1],      ["7", "7", "7", 1],      ["8", "8", "8",  
         1],  
        ["9", "9", "9", 1],      ["0", "0", "0", 1],      ["+", "+", "+",  
         1],  
        [=, =, =, 1],          ["\u232b", null, "backspace", 2]  
    ],  
  
    "shift_1": [ ... ],  
    "normal_2": [ ... ],  
    "special_2": [ ... ],  
    ...  
}
```

Solicitador de Teclado

A instanciação do teclado virtual é controlada pela configuração. Verifique as opções `keyboard_mode` e `keyboard_layout` no arquivo [Objeto de Configuração](#).

Se pretendes criar um Widget que irá requerer o teclado, não use o teclado virtual diretamente, prefira utilizar uma forma disponível que seja melhor. Verifique o método `request_keyboard()` no [Janela](#).

Se quiseres um layout específico quando solicitares o teclado, deves escrever algo como isto (a partir da versão 1.8.0, `numeric.json` pode estar no mesmo diretório que seu arquivo `main.py`):

```
keyboard = Window.request_keyboard(  
    self._keyboard_close, self)
```

```
if keyboard.widget:  
    vkeyboard = self._keyboard.widget  
    vkeyboard.layout = 'numeric.json'
```

```
class kivy.uix.vkeyboard.VKeyboard(**kwargs)
```

Bases: *kivy.uix.scatter.Scatter*

VKeyboard é um teclado na tela com suporte multitouch. Esse layout é totalmente personalizável e poderás alternar entre os layouts disponíveis usando um botão na parte inferior direita do Widget.

Events

on_key_down: keycode, internal, modifiers Disparado quando o teclado recebeu um evento de que um tecla foi pressionada (tecla pressionada).

on_key_up: keycode, internal, modifiers Disparado quando o teclado recebeu um evento key up (liberação de teclas).

available_layouts

Dicionário de todos os Layouts disponíveis. As chaves são o ID do layout e o valor é o JSON (traduzido para um objeto Python).

available_layouts é uma *DictProperty* e o padrão é {}.

background

Nome do arquivo da imagem de plano de fundo.

background is a *StringProperty* and defaults to `atlas://data/images/defaulttheme/vkeyboard_background`.

background_border

Borda da imagem de fundo. Usado para controlar a propriedade *border* de plano de fundo.

background_border é uma *ListProperty* e o padrão é [16, 16, 16, 16]

background_color

Cor de fundo, no formato (r, g, b, a). Se um fundo estiver definido, a cor será combinada com a textura de fundo.

background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

background_disabled

Filename of the background image when the vkeyboard is disabled.

Novo na versão 1.8.0.

background_disabled é uma *StringProperty* e o padrão é `atlas://data/images/defaulttheme/vkeyboard_disabled_background`.

callback

O callback pode ser definido para uma função que será chamada caso o VKeyboard for fechado pelo usuário.

target é uma instância da *ObjectProperty* e o padrão é *None*.

collide_margin(x, y)

Faça um teste de colisão, e retorne True se o(x,y) estiver dentro da margem do vkeyboard

docked

Indique se o VKeyboard está ou não fixado na tela. Se você alterá-lo, deve-rás chamar manualmente *setup_mode()* caso contrário não terá nenhum impacto. Se o VKeyboard for criado pela Janela, o modo fixado será automaticamente definido pela configuração, usando o Token *keyboard_mode* na seção '[kivy]'.

docked é uma *BooleanProperty* e o padrão é *False*.

font_size

font_size, specifies the size of the text on the virtual keyboard keys. It should be kept within limits to ensure the text does not extend beyond the bounds of the key or become too small to read.

Novo na versão 1.10.0.

font_size is a *NumericProperty* and defaults to 20.

key_background_color

Cor do plano de fundo, no formato (r, g, b, a). Se uma Key de plano de fundo estiver definido, a cor será combinada com a textura de fundo dessa Key.

key_background_color é uma *ListProperty* e o padrão é [1, 1, 1, 1].

key_background_down

Nome do arquivo da imagem de fundo chave para uso quando um toque está ativo no Widget.

key_background_down is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/vkeyboard_key_down*.

key_background_normal

Nome do arquivo da imagem de fundo Key pra ser usada quando nenhum toque estiver ativo no Widget.

key_background_normal is a *StringProperty* and defaults to *atlas://data/images/defaulttheme/vkeyboard_key_normal*.

key_border

Límite da imagem Key. Usado para controlar a propriedade da Key *border*.

key_border é uma *ListProperty* e o padrão é [16, 16, 16, 16]

key_disabled_background_normal

Nome do arquivo da imagem de fundo da Key para uso quando não houver toques ativos no Widget e VKeBoard estiver desabilitado.

Novo na versão 1.8.0.

key_disabled_background_normal is a *StringProperty* and defaults to atlas://data/images/defaulttheme/vkeyboard_disabled_key_normal.

key_margin

Margem da Key, usada para criar espaço entre as Keys. A margem é composta por quatro valores, em pixels:

```
key_margin = [top, right, bottom, left]
```

key_margin é uma *ListProperty* e o padrão é [2, 2, 2, 2]

layout

Layout a ser usado pelo VKeyBoard. Por padrão, será o layout configurado na configuração, de acordo com a seção *keyboard_layout* na seção *[kivy]*.

Alterado na versão 1.8.0: Se o layout for um nome de arquivo .json, ele será carregado e adicionado ao *available_layouts*.

layout é uma *StringProperty* e o padrão é *None*.

layout_path

Caminho a partir do qual os layouts são lidos.

layout é uma *StringProperty* e o padrão é <kivy_data_dir>/keyboards/

margin_hint

Margem Hint, usada como espaçamento entre fundo de teclado e o conteúdo das Keys. A margem é composta por quatro valores, entre 0 e 1:

```
margin_hint = [top, right, bottom, left]
```

As margens hints serão multiplicadas pela largura e altura, de acordo com sua posição.

margin_hint é uma *ListProperty* e o padrão é [.05, .06, .05, .06]

refresh(*force=False*)

(Interno) Recriar todo o widget e gráficos de acordo com o layout selecionado.

setup_mode(args*)**

Chame este método quando quiser reajustar o teclado de acordo com as opções: *docked* ou não, com anexoado *target* ou não:

- Se *docked* é *True*, ele chamará o *setup_mode_dock()*

- Se o `docked` é `False`, ele chamará o `:meth:'setup_mode_free'`

Sinta-se livre para sobrecarregar esses métodos e criar um novo comportamento de posicionamento.

`setup_mode_dock(*args)`

Configure o teclado no modo docked.

O modo Dock irá reiniciar a rotação, desativar a tradução, rotação e escala. A escala e a posição serão ajustadas automaticamente para conectar o teclado à parte inferior da tela.

Nota: Não invoque este método diretamente, ao invés, utilize `setup_mode()`.

`setup_mode_free()`

Configure o teclado no modo livre.

O modo livre é projetado para permitir que o usuário controle a posição e a orientação do teclado. O único uso real é para um ambiente multiusuário, mas podes encontrar outras formas de usá-lo. Se a `target` estiver definido, colocará o VKeyBoard sob o alvo.

Nota: Não invoque este método diretamente, ao invés, utilize `setup_mode()`.

`target`

Widget alvo associado ao VKeyboard. Se definido, ele será usado para enviar eventos de teclado. Se o modo VKeyboard estiver “livre”, ele também será usado para definir a posição inicial.

`target` é uma instância da `ObjectProperty` e o padrão é `None`.

4.15.54 Classe Widget

A classe `Widget` é a classe base requerida para a criação dos Widgets. Esta classe Widget foi desenhada com 2 princípios em mente:

- *Event Driven*

A interação com Widgets é construída sobre os eventos que ocorrerem. Se uma propriedade for alterada, o Widget poderá responder a alteração no evento de callback ‘`on_<propname>`’. Se nada for modificado, nada acontecerá. Este é o objetivo principal da classe `Property`.

- *Separação de Responsabilidades (o Widget e sua representação gráfica)*

Os Widgets não possuem um método `draw()`. Isso foi feito de propósito: a ideia é permitir que você crie as suas representações gráficas fora da classe do Widget. Obviamente, você pode utilizar todas as propriedades disponíveis pra fazer isso, para que a sua representação reflita o corretamente o estado do seu Widget. Cada Widget possui a sua própria classe `Canvas` que você pode utilizar para desenhar. Essa separação permite o Kivy rodar sua aplicação de uma forma muito eficiente.

- *Caixa de Contorno / Colisão*

Muitas vezes você precisará saber se um certo ponto está dentro dos limites de seu Widget. Um exemplo seria um Widget botão em que você deseja acionar uma ação quando o mesmo for pressionado. Para isso, você pode usar o método `collide_point()` que retornará `True` se o ponto que você passar estiver dentro dos limites da caixa alinhada ao eixo definido pela posição e tamanho do Widget. Se um simples AABB não for suficiente, poderás substituir (overrrides) o método para executar as verificações de colisão com formas mais complexas, por exemplo, com polígonos. Também é possível verificar se um Widget colide com outro widget através do método `collide_widget()`.

Nós também temos alguns valores default e comportamentos que você deve ter atenção:

- A classe `Widget` não é uma `Layout`: isso não modificará sua posição ou o tamanho dos seus filhos. Se quiseres controlar o posicionamento ou o dimensionamento, utilize `Layout`.
- O tamanho padrão dos Widgets é (100, 100). Isso só poder ser alterado se a classe pai for um `Layout`. Por exemplo, se você adicionar um `Label` dentro de um `Button`, o Label não herdará o tamanho do botão ou posicionamento porque o botão não é uma `Layout`. Ele é somente um outro `Widget`.
- O padrão é `size_hint (1, 1)`. Se o pai for um `Layout`, então tamanho do Widget será o tamanho do layout de seu pai.
- `on_touch_down()`, `on_touch_move()`, `on_touch_up()` não faz qualquer tipo de colisão. Se você deseja saber se o toque foi em cima do seu Widget, utilize `collide_point()`.

Propriedades utilizadas

Quando você ler a documentação, todas as propriedades descritas estarão no formato:

<name> **is** a <property class> **and** defaults to <default value>.

Por exemplo.

`text` é um `StringProperty` e o padrão é “”.

Se desejas ser notificado quando o atributo da posição for alterada, por exemplo, quando o Widget se mover, você pode vincular uma função de callback semelhante ao

que podemos ver a seguir:

```
def callback_pos(instance, value):
    print('The widget', instance, 'moved to', value)

wid = Widget()
wid.bind(pos=callback_pos)
```

Leia mais sobre [Propriedades](#).

Desenho básico

Os Widgets suportam uma série de instruções gráficas que você pode utilizar para personalizar o visual de seus Widgets e Layouts. Por exemplo, para desenha uma imagem como plano de fundo para seu Widgets, você pode fazê-lo da seguinte forma:

```
def redraw(self, args):
    self.bg_rect.size = self.size
    self.bg_rect.pos = self.pos

widget = Widget()
with widget.canvas:
    widget.bg_rect = Rectangle(source="cover.jpg", pos=self.pos,
                                size=self.size)
widget.bind(pos=redraw, size=redraw)
```

Para desenhar o fundo com a linguagem kv:

```
Widget:
    canvas:
        Rectangle:
            source: "cover.jpg"
            size: self.size
            pos: self.pos
```

Este exemplo apenas esboça a superfície. Veja a documentação para maiores informações [kivy.graphics](#).

Bubbling do Evento de Toque do Widget

Quando você pegar um evento de toque entre vários Widgets, muitas vezes precisarás estar ciente da ordem em que esses eventos são propagados. No Kivy, os eventos se propagam desde o primeiro filho até os outros Widgets. Se um Widgets tem filhos, o evento é passado através de seus filhos antes de ser passada ao mesmo.

Como o método [on_touch_up\(\)](#) adiciona Widgets com índice igual a 0 por padrão, isso significa que o evento vai desde o Widget adicionado mais recentemente em direção ao primeiro Widget que foi adicionado. Considere o seguinte:

```
box = BoxLayout()
box.add_widget(Label(text="a"))
box.add_widget(Label(text="b"))
box.add_widget(Label(text="c"))
```

O Label com o texto “c” obtém o primeiro evento, “b” obtém o segundo e “a” obtém o último. Você pode inverter essa ordem especificando o índice manualmente.

```
box = BoxLayout()
box.add_widget(Label(text="a"), index=0)
box.add_widget(Label(text="b"), index=1)
box.add_widget(Label(text="c"), index=2)
```

Agora a ordem seria “a”, “b” e então “c”. Uma coisa que deves ter em mente ao usar kv é que declarar um Widget invoca-se o método `add_widget()` para inserção. Assim, quando

```
BoxLayout:
    MyLabel:
        text: "a"
    MyLabel:
        text: "b"
    MyLabel:
        text: "c"
```

resultaria na ordem de evento for “c”, “b” e então “a”, como o “c” foi na verdade o último Widget adicionado. Ele terá, portanto, índice igual a 0 o “b” terá índice igual a 1 e o a índice igual a 2. Efetivamente, a ordem dos filhos é o inverso da ordem listada.

Essa ordenação é a mesma para os eventos `on_touch_move()` e `on_touch_up()`.

Para parar essa propagação de evento (event bubbling), um método pode retornar *Trie*. Isso diz ao Kivy que o evento foi tratado, e a propagação deve parar. Por exemplo:

```
class MyWidget(Widget):
    def on_touch_down(self, touch):
        if <some_condition>:
            # Do stuff here and kill the event
            return True
        else:
            return super(MyWidget, self).on_touch_down(touch)
```

Essa abordagem fornece um bom controle sobre como os eventos são despachados e gerenciados. Às vezes, entretanto, podes desejar deixar o evento ser propagado completamente antes de fazer alguma coisa. Podes usar a classe `Clock` para ajudá-lo aqui:

```
class MyWidget(Label):
    def on_touch_down(self, touch, after=False):
```

```

if after:
    print "Fired after the event has been dispatched!"
else:
    Clock.schedule_once(lambda dt: self.on_touch_down(touch,
    ↵ True))
return super(MyWidget, self).on_touch_down(touch)

```

Utilização de `Widget.center`, `Widget.right`, e `Widget.top`

Um erro comum ao usarmos uma das propriedades computadas como `Widget.right` é usa-los para fazer o Widget seguir seus pais com uma regra kv tal como `right: self.parent.right`. Considere o exemplo seguinte:

```

FloatLayout:
    id: layout
    width: 100
    Widget:
        id: wid
        right: layout.right

```

A expectativa (equivocada) é que esta regra garanta que o direito de `wid` sempre será o que o `layout` é certo - no caso, que `wid.right` e `layout.right` sempre seja idêntico. Na realidade, esta regra apenas diz que “sempre que o layout `right` mudar, o `right` do `wid` será ajustado pra esse valor”. A diferença é que enquanto `layout.right` não mudar, `wid.right` poderá ser qualquer coisa, mesmo um valor que seja diferente um do outro.

Especificamente, para o código KV acima, considere o seguinte exemplo:

```

>>> print(layout.right, wid.right)
(100, 100)
>>> wid.x = 200
>>> print(layout.right, wid.right)
(100, 300)

```

Como pode ser visto, inicialmente eles estão em sincronia, no entanto, quando mudamos `wid.x` eles ficam fora de sincronia porque `layout.right` não é alterado e a regra não é acionada.

A maneira correta de fazer o Widget seguir o seu pai corretamente é usar `Widget.pos_hint`. Se em vez de `right: layout.right` fizermos `pos_hint: {'right': 1}`, então o lado direito do Widgets sempre será definido para estar no lado direito do seu pai em cada atualização de layout.

```

class kivy.uix.widget.Widget(**kwargs)
    Bases: kivy.uix.widget.WidgetBase

```

Classe Widget. Veja o módulo da documentação para maiores informações.

Events

on_touch_down: Disparado quando um novo evento de toque ocorrer

on_touch_move: Disparado quando um movimento de toque existir

on_touch_up: Disparado quando um toque existente desaparecer

Aviso: Adicionar o método `__del__` para a classe derivada de Widget com Python antes da versão 3.4 desabilitará a coleta automática de lixo para as instâncias dessa classe. Isso ocorre porque a classe Widget cria ciclos de referência (reference cycles), impedindo a coleta de lixo, veja mais em <<https://docs.python.org/2/library/gc.html#gc.garbage>>_.

Alterado na versão 1.0.9: Tudo relacionado às propriedades do evento foi movido para a *EventDispatcher*. As propriedades de evento agora podem ser usadas quando for construir uma classe simples sem subclasse *Widget*.

Alterado na versão 1.5.0: O construtor agora aceita `on_*` argumentos para automaticamente vincular callbacks a propriedades ou eventos, como na linguagem kv.

add_widget(widget, index=0, canvas=None)

Adiciona um novo Widget como sendo filho deste Widget.

Parameters

widget: *Widget* Widget adicionará para a lista de filhos.

index: int, padrão é 0 Índice para inserir o Widget na lista.

Observe que o padrão 0 significa que o Widget será inserido no início da lista e, assim, será desenhado em cima dos outros Widgets. Para uma discussão completa da hierarquia de índice e Widget, consulte o *Widgets Programming Guide*.

Novo na versão 1.0.5.

canvas: str, padrão é None Canvas para adicionar a tela do Widget a. Pode ser 'before', 'after' ou None para o padrão Canvas.

Novo na versão 1.9.0.

```
>>> from kivy.uix.button import Button
>>> from kivy.uix.slider import Slider
>>> root = Widget()
>>> root.add_widget(Button())
```

```
>>> slider = Slider()  
>>> root.add_widget(slider)
```

canvas = *None*

Canvas do Widget

O Canvas é um objeto gráfico que contém todas as instruções de desenho para uma representação gráfica do Widget.

Não há propriedades gerais para a classe Widget, como cor de plano de fundo, para manter o design simples e magra. Algumas classes derivadas, como *Button*, adiciona essas propriedades de conveniência, mas geralmente o desenvolvedor é responsável pela implementação da representação gráfica de um Widget personalizado a partir do zero. Veja as classes derivadas de Widget para os padrões a serem seguidos e estendidos.

Veja [Canvas](#) para maiores informações sobre como utilizar.

center

Posição central do Widget.

center é uma propriedade [ReferenceListProperty](#) de (*center_x*, *center_y*).

center_x

Posição central X do Widget.

center_x é uma propriedade [AliasProperty](#) de (*x* + *width* / 2.).

center_y

Posição central Y do Widget.

center_y é uma propriedade [AliasProperty](#) de (*y* + *height* / 2.).

children

Lista de filhos deste Widget.

children é um [ListProperty](#) e o padrão é uma lista vazia.

Utilize *add_widget()* e *remove_widget()* para manipular a lista de filhos. Não manipule a lista de filhos diretamente a não ser que você saiba o que estejas fazendo.

clear_widgets(*children=None*)

Remove todos (ou os especificados) *children* deste Widget. Se o argumento ‘*children*’ for especificado, ele será uma lista (ou lista filtrada) de filhos do Widget atual.

Alterado na versão 1.8.0: O argumento *children* pode ser utilizado para especificar os filhos que você deseja remover.

cls

Classe de Widget, utilizada para estilizar.

collide_point(x, y)

Cheque se o ponto (x, y) está alinhado dentro dos eixos do retângulo do Widget's

Parameters

x: numericx position of the point (in parent coordinates)

y: numericy position of the point (in parent coordinates)

ReturnsUm *bool*. Se *True* o ponto está dentro das dimensões do Widget, do contrário será 'False'.

```
>>> Widget(pos=(10, 10), size=(50, 50)).collide_point(40, _  
    ↪40)  
True
```

collide_widget(wid)

Checa se outro Widget colide com este Widget. Essa função realiza um teste de interseção de caixa delimitadora alinhado pelo eixo por padrão.

Parameters

Classe wid: *Widget*Widget to test collision with.

Returns*bool*. Se *True* há outro Widget colidindo com o Widget, do contrário será *False*.

```
>>> wid = Widget(size=(50, 50))  
>>> wid2 = Widget(size=(50, 50), pos=(25, 25))  
>>> wid.collide_widget(wid2)  
True  
>>> wid2.pos = (55, 55)  
>>> wid.collide_widget(wid2)  
False
```

disabled

Indica se este Widget pode interagir com a entrada ou não.

Nota:

- 1.Widgets filhos, quando adicionado a um Widget desativado, será desativado automaticamente.
 - 2.Habilitar ou desabilitar um Widget pai (Widget que contém outros Widgets) ativará ou desativará todos os seus filhos.
-

Novo na versão 1.8.0.

disabled é uma *BooleanProperty* e o padrão é *False*.

export_to_png(filename, *args)

Salva uma imagem do Widget e seus filhos em formato PNG no nome do arquivo especificado. Funciona removendo a tela do Widget do seu pai, renderizanod para uma *Fbo*, e invocando *save()*.

Nota: A imagem inclui apenas este Widget e seus filhos. Se quiseres incluir Widgets em outro lugar da árvore, você deve invocar *export_to_png()* de seu pai, ou usar *screenshot()* para capturar toda a janela .

Nota: A imagem será salva no formato PNG, é necessário incluir a extensão no nome do arquivo.

Novo na versão 1.9.0.

get_parent_window()

Retorna a janela pai.

ReturnsInstância da janela pai. Pode ser um *WindowBase* ou um *Widget*.

get_root_window()

Retorna a janela principal.

ReturnsInstância da janela pai. Pode ser um *WindowBase* ou um *Widget*.

get_window_matrix(x=0, y=0)

Calcula a matriz de conversão para converter entre a janela e a coordenada do Widget.

Parameters

x: float, o padrão é 0Traduz a matriz no eixo X.

y: float, padrão é 0Traduz a matriz no eixo Y.

height

Altura da janela.

height é um *NumericProperty* e o padrão é 100.

Aviso: Lembre-se de que a propriedade *height* está sujeita à lógica de layout e que isso ainda não aconteceu no momento do método *_init_* do Widget. (as propriedades *height* e *width* NÃO ESTÃO calculadas no eventos de inicialização de classe).

id

Identificador único do Widget na árvore.

id é um *StringProperty* e o padrão é *None*.

Aviso: Se o *id* estiver em uso na árvore, uma exceção será levantada.

ids

Este é uma dicionário de ids definido em seu arquivo kv. Isso só será populado se você usar ids em seus arquivo definido com a linguagem kv.

Novo na versão 1.7.0.

ids é um *DictProperty* e o padrão é um dicionário vazio {}.

Os *ids* será populado para cada novo nível de Widget definido. Por exemplo:

```
# in kv
<MyWidget@Widget>:
    id: my_widget
    Label:
        id: label_widget
        Widget:
            id: inner_widget
            Label:
                id: inner_label
    TextInput:
        id: text_input
    OtherWidget:
        id: other_widget

<OtherWidget@Widget>
    id: other_widget
    Label:
        id: other_label
    TextInput:
        id: other_textinput
```

Então, em Python:

```
>>> widget = MyWidget()
>>> print(widget.ids)
{'other_widget': <weakproxy at 041CFED0 to OtherWidget at 041BEC38>,
 'inner_widget': <weakproxy at 04137EA0 to Widget at 04138228>,
 'inner_label': <weakproxy at 04143540 to Label at 04138260>,
```

```
'label_widget': <weakproxy at 04137B70 to Label at 040F97A0>
    ↵,
'text_input': <weakproxy at 041BB5D0 to TextInput at
    ↵041BEC00>}
>>> print(widget.ids['other_widget'].ids)
{'other_textinput': <weakproxy at 041DBB40 to TextInput at
    ↵041BEF48>,
'other_label': <weakproxy at 041DB570 to Label at 041BEEA0>}
>>> print(widget.ids['label_widget'].ids)
{}
```

on_touch_down(*touch*)

Recebe um evento de toque.

Parameters

classe touch: *MotionEvent* Toque recebido. O toque é está nas coordenadas do pai. Veja [relativelayout](#) para uma discussão sobre o sistema de coordenadas.

Returns *bool* se *True*, o despacho do evento de toque será interrompido. Se *False*, o evento continuará a ser despachado para o resto da árvore de Widgets.

on_touch_move(*touch*)

Recebe um evento de movimento de toque. O toque está nas coordenadas do Widget onde o mesmo está contido (coordenadas do Widget pai).

Para maiores informações veja [on_touch_down\(\)](#).

on_touch_up(*touch*)

Recebe um fim de um evento de toque (touch up). O toque está nas coordenadas do pai.

Para maiores informações veja [on_touch_down\(\)](#).

opacity

Opacidade do Widget e todos seus filhos.

Novo na versão 1.4.1.

O atributo opacity controla a opacidade do Widget e seus filhos. Tenha cuidado, isso é um atributo cumulativo: o valor é multiplicado pela atual opacidade global e o resultado é aplicado a atual cor do contexto.

Por exemplo, se o pai tiver uma opacidade de 0.0 e o filho tem uma opacidade de 0.2, a opacidade real do filho será $0.5 * 0.2 = 0.1$.

Então, a opacidade é aplicada pelo Shader como:

```
frag_color = color * vec4(1.0, 1.0, 1.0, opacity);
```

opacity é uma *NumericProperty* e o padrão é 1.0.

parent

Pai deste Widget. O pai de um Widget é definido quando o Widget é adicionado a outro Widget e desconfigurado quando o Widget é removido deste Widget (do Widget em que estava contido).

parent é uma *ObjectProperty* e o padrão é *None*.

pos

Posição do Widget.

pos é uma propriedade *ReferenceListProperty* de (*x*, *y*).

pos_hint

Dica da posição. Essa propriedade permite a você definir a posição do Widget dentro do layout do seu pai, em porcentagem (similar a propriedade *size_hint*).

Por exemplo, desejas definir o *top* do Widget como sendo de 90% da altura do layout do seu pai, pode escrever da seguinte forma:

```
widget = Widget(pos_hint={'top': 0.9})
```

As keys 'x', 'direito' e 'center_x' usarão a largura pai. As keys 'y', 'top' e 'center_y' usarão a altura pai.

Consulte *Float Layout* para referência.

Nota: *pos_hint* não é usado por todos os layout. Veja a documentação do layout em questão para ver se o mesmo suporta a propriedade *pos_hint*.

attr:pos_hint é uma *ObjectProperty* contém um *dict*.

proxy_ref

Retorna uma referência de proxy para o Widget, ou seja, sem criar uma referência para o Widget. Veja *weakref.proxy* para mais informações.

Novo na versão 1.7.2.

remove_widget(widget)

Remover um Widget dos filhos deste Widget.

Parameters

widget: **Widget** Widget para remover da nossa lista de filhos.

```
>>> from kivy.uix.button import Button  
>>> root = Widget()  
>>> button = Button()
```

```
>>> root.add_widget(button)
>>> root.remove_widget(button)
```

right

Posição direita do Widget.

right é uma *AliasProperty* de (*x + width*).

size

Tamanho de uma Widget.

size é uma propriedade *ReferenceListProperty* de (*width, height*).

size_hint

Tamanho da dica.

size_hint é uma propriedade *ReferenceListProperty* de (*size_hint_x, size_hint_y*).

Veja a documentação para maiores informações *size_hint_x*.

size_hint_max

Tamanho máximo quando ao usar *size_hint*.

size_hint_max é uma propriedade *ReferenceListProperty* de (*size_hint_max_x, size_hint_max_y*).

Novo na versão 1.10.0.

size_hint_max_x

When not None, the x-direction maximum size (in pixels, like *width*) when *size_hint_x* is also not None.

Semelhante a *size_hint_min_x*, exceto que ele define a largura máxima.

size_hint_max_x é uma *NumericProperty* e o padrão é *None*.

Novo na versão 1.10.0.

size_hint_max_y

When not None, the y-direction maximum size (in pixels, like *height*) when *size_hint_y* is also not None.

Semelhante a *size_hint_min_y*, exceto que ele define a altura máxima.

size_hint_max_y é uma *NumericProperty* e o padrão é *None*.

Novo na versão 1.10.0.

size_hint_min

Tamanho mínimo ao usar *size_hint*.

size_hint_min é uma propriedade *ReferenceListProperty* de (*size_hint_min_x, size_hint_min_y*).

Novo na versão 1.10.0.

size_hint_min_x

When not None, the x-direction minimum size (in pixels, like *width*) when *size_hint_x* is also not None.

Quando *size_hint_x* não for *None*, será a largura mínima que o Widget poderá ser definido devido a *size_hint_x*. Por exemplo, quando um tamanho menor deve ser definido, *size_hint_min_x* será o valor usado em vez disso para a largura do Widget. Quando *None*, ou quando *size_hint_x* for *None*, *size_hint_min_x* não faz nada.

Somente as classes *Layout* e a *Window* fazem uso do “hint”.

size_hint_min_x é uma *NumericProperty* e o padrão é *None*.

Novo na versão 1.10.0.

size_hint_min_y

When not None, the y-direction minimum size (in pixels, like *height*) when *size_hint_y* is also not None.

Quando *size_hint_y* não for *None*, será a largura mínima que o Widget poderá ser definido devido a *size_hint_y*. Por exemplo, quando um tamanho menor deve ser definido, *size_hint_min_y* será o valor usado em vez disso para a largura do Widget. Quando *None*, ou quando *size_hint_y* for *None*, *size_hint_min_y* não faz nada.

Somente as classes *Layout* e a *Window* fazem uso do “hint”.

size_hint_min_y é uma *NumericProperty* e o padrão é *None*.

Novo na versão 1.10.0.

size_hint_x

x size hint. Represents how much space the widget should use in the direction of the x axis relative to its parent's width. Only the *Layout* and *Window* classes make use of the hint.

O *size_hint* é usado pelos layouts para dois propósito:

- Quando o layout considera os Widgets por conta própria, em vez de em relação aos seus outros filhos, o *size_hint_x* será uma proporção direta da largura do pai, normalmente entre 0,0 e 1,0. Por exemplo, um Widget com *size_hint_x* = 0,5 em um BoxLayout vertical ocupará metade da largura do BoxLayout ou um Widget em um FloatLayout com *size_hint_x* = 0,2 ocupará 20% da largura do FloatLayout. Se a *size_hint* for maior que 1, o Widget será maior que o pai.
- Quando vários Widgets podem compartilhar uma linha de um layout, como em um BoxLayout horizontal, suas larguras serão sua *size_hint_x* como uma fração da soma dos *size_hints* do Widget. Por exemplo, se os

`size_hint_xs` forem $(0,5, 1,0, 0,5)$, o primeiro Widget terá uma largura de 25% da largura do pai.

`size_hint_x` é uma *NumericProperty* e o padrão é 1.

size_hint_y

y size hint.

`size_hint_y` é uma *NumericProperty* e o padrão é 1.

Veja a documentação `size_hint_x` para mais informações, mas com a altura e a largura trocadas.

to_local(x, y, relative=False)

Transforma as coordenadas do pai em coordenadas locais. Veja a documentação *relativelayout* para mais detalhes sobre o sistema de coordenadas.

Parameters

`relative: bool`, o padrão é *False*Mude para *True* se desejas traduzir as coordenadas para coordenadas relativas do Widget.

to_parent(x, y, relative=False)

Transforma coordenadas locais para coordenadas dos pais. Veja a documentação *relativelayout* para mais informações sobre o sistema de coordenadas.

Parameters

`relative: bool`, o padrão é *False*Mude para *True* se desejas traduzir posições relativas de um Widget para as coordenadas do seu pai.

to_widget(x, y, relative=False)

Converte as coordenadas dadas desde a janela para as coordenadas relativas do Widget. Veja a documentação *relativelayout* para mais informações sobre o sistema de coordenadas.

to_window(x, y, initial=True, relative=False)

Transforma coordenadas locais para coordenadas da janela. Veja a documentação *relativelayout* para mais informações sobre o sistema de coordenadas.

top

Posição superior do widget.

`top` é uma *AliasProperty* de (`y + height`).

walk(restrict=False, loopback=False)

Iterador que caminha pela árvore de Widgets começando pelo Widget e vai retornando os Widgets na ordem em que o layout os exibe.

Parameters

restrict: bool, e o padrão é False Se *True* iterá somente através do Widget e seus filhos (ou filho dos filhos e etc). O padrão é *False*.

loopback: bool, e o padrão é False Se *True*, quando o último Widget na árvore for atingido, ele volta para a raiz mais alta e começa a caminhar até atingir este Widget novamente. Naturalmente, ele só pode voltar quando *restrict* for *False*. O padrão é *False*.

Retorna Um gerador que caminha na árvore, retornando Widgets na ordem de layout para a frente.

Por exemplo, dado uma árvore com a seguinte estrutura:

```
GridLayout:  
    Button  
BoxLayout:  
    id: box  
    Widget  
    Button  
    Widget
```

Andando nesta árvore:

```
>>> # Call walk on box with loopback True, and restrict=False  
>>> [type(widget) for widget in box.walk(loopback=True)]  
<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>,  
    <class 'Widget'>, <class 'GridLayout'>, <class 'Button'>  
>>> # Now with loopback False, and restrict False  
>>> [type(widget) for widget in box.walk()]  
<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>,  
    <class 'Widget'>  
>>> # Now with restrict True  
>>> [type(widget) for widget in box.walk(restrict=True)]  
<class 'BoxLayout'>, <class 'Widget'>, <class 'Button'>
```

Novo na versão 1.9.0.

walk_reverse(*loopback=False*)

Iterator que caminha para trás na árvore de Widgets começando com o Widget anterior, e indo para trás retornando Widgets na ordem inversa em que os Layouts os exibem.

Isso caminha na direção oposta de `walk()`, então uma lista da árvore gerada com `walk()` estará na ordem inversa em relação à lista gerada com isso, desde que *loopback* seja *True*.

Parameters

loopback: bool, e o padrão é *False*Se *True*, quando a raiz mais alta na árvore for atingida, o loop volta para o último Widget e começar a andar de volta até atingirmos o Widget novamente. O padrão é *False*.

RetornaUm gerador que caminha pela árvore, retornando Widgets na ordem inversa do layout.

Por exemplo, dado uma árvore com a seguinte estrutura:

```
GridLayout:  
    Button  
    BoxLayout:  
        id: box  
        Widget  
        Button  
    Widget
```

Andando nesta árvore:

```
>>> # Call walk on box with loopback True  
>>> [type(widget) for widget in box.walk_  
     ↪reverse(loopback=True)]  
[<class 'Button'>, <class 'GridLayout'>, <class 'Widget'>,  
 <class 'Button'>, <class 'Widget'>, <class 'BoxLayout'>]  
>>> # Now with loopback False  
>>> [type(widget) for widget in box.walk_reverse()]  
[<class 'Button'>, <class 'GridLayout'>]  
>>> forward = [w for w in box.walk(loopback=True)]  
>>> backward = [w for w in box.walk_reverse(loopback=True)]  
>>> forward == backward[::-1]  
True
```

Novo na versão 1.9.0.

width

Largura do Widget.

width é uma *NumericProperty* e o padrão é 100.

Aviso: Tenha em mente que a propriedade *width* está sujeita a lógica de layout e que a função que determina essa lógica ainda não foi calculada/executada no método *__init__* do Widget.

x

Posição X do Widget.

x é uma *NumericProperty* e o padrão é 0.

y

Posição Y do Widget.

y é uma *NumericProperty* e o padrão é 0.

class kivy.uix.widget.WidgetException

Bases: exceptions.Exception

Disparado quando o Widget obtém uma exceção.

Parte IV

APENDICE

O apêndice contém informações de licenciamento e uma enumeração de todos os diferentes módulos, classes, funções e variáveis disponíveis no Kivy.

Licença

Kivy é liberado e distribuído sob os termos da licença MIT a partir da versão 1.7.2. Versões mais antigas ainda estão sob a licença GPLv3.

Receberás uma cópia da licença MIT no mesmo diretório da sua distribuição Kivy. Consulte o arquivo LICENSE na pasta raiz do Kivy. Uma versão on-line da licença pode ser encontrada em:

<https://github.com/kivy/kivy/blob/master/LICENSE>

Em poucas palavras, a licença permite-lhe usar o Kivy em seus projetos pessoais, independentemente de serem de código aberto, sem a liberação dos fontes, seja em aplicações comerciais ou gratuitas. Mesmo que a licença não o exija, gostaríamos realmente de saber quando fizeres alterações no código fonte da biblioteca **“Kivy”**, compartilhe essas mudanças conosco!

Para obter uma lista de autores, consulte o arquivo AUTHORS que acompanha a distribuição do código-fonte Kivy (junto ao arquivo da LICENÇA).

Kivy – Copyright 2010-2016, Os autores do Kivy.

Índice de Módulos Python

k
kivy, 193
kivy.adapters, 315
kivy.adapters.adapter, 317
kivy.adapters.args_converters, 320
kivy.adapters.dictadapter, 318
kivy.adapters.listadapter, 321
kivy.adapters.models, 324
kivy.adapters.simplelistadapter, 325
kivy.animation, 195
kivy.app, 206
kivy.atlas, 224
kivy.base, 269
kivy.cache, 228
kivy.clock, 230
kivy.compat, 242
kivy.config, 243
kivy.context, 251
kivy.core, 326
kivy.core.audio, 327
kivy.core.camera, 329
kivy.core.clipboard, 330
kivy.core.gl, 331
kivy.core.image, 331
kivy.core.spelling, 336
kivy.core.text, 337
kivy.core.text.markup, 345
kivy.core.text.text_layout, 341
kivy.core.video, 347
kivy.core.window, 348
kivy.deps, 364
kivy.effects, 364
kivy.effects.dampedscroll, 365
kivy.effects.kinetic, 366
kivy.effects.opacityscroll, 368
kivy.effects.scroll, 368
kivy.event, 251
kivy.factory, 262
kivy.garden, 369
kivy.geometry, 262
kivy.gesture, 263
kivy.graphics, 370
kivy.graphics.cgl, 403
kivy.graphics.compiler, 414
kivy.graphics.context, 409
kivy.graphics.context_instructions, 404
kivy.graphics.fbo, 409
kivy.graphics.gl_instructions, 413
kivy.graphics.instructions, 397
kivy.graphics.opengl, 415
kivy.graphics.opengl_utils, 425
kivy.graphics.scissor_instructions, 427
kivy.graphics.shader, 429
kivy.graphics.stencil_instructions, 431
kivy.graphics.svg, 433
kivy.graphics.tesselator, 434
kivy.graphics.texture, 437
kivy.graphics.transformation, 445

kivy.uix.behaviors.drag, 558
kivy.uix.behaviors.emacs, 560
kivy.uix.behaviors.focus, 561
kivy.uix.behaviors.knspase, 566
kivy.uix.behaviors.togglebutton, kivy.uix.switch, 746
573
kivy.uix.boxlayout, 599
kivy.uix.bubble, 601
kivy.uix.button, 605
kivy.uix.camera, 607
kivy.uix.carousel, 608
kivy.uix.checkbox, 612
kivy.uix.codeinput, 614
kivy.uix.colorpicker, 615
kivy.uix.dropdown, 618
kivy.uix.effectwidget, 621
kivy.uix.filechooser, 626
kivy.uix.floatlayout, 637
kivy.uix.gesturesurface, 639
kivy.uix.gridlayout, 643
kivy.uix.image, 648
kivy.uix.label, 651
kivy.uix.layout, 665
kivy.uix.listview, 667
kivy.uix.modalview, 680
kivy.uix.pagelayout, 683
kivy.uix.popup, 684
kivy.uix.progressbar, 687
kivy.uix.recycleboxlayout, 688
kivy.uix.recyclegridlayout, 688
kivy.uix.recyclelayout, 689
kivy.uix.recycleview, 575
kivy.uix.recycleview.datamodel,
580
kivy.uix.recycleview.layout, 581
kivy.uix.recycleview.views, 583
kivy.uix.relativelayout, 689
kivy.uix.rst, 694
kivy.uix.sandbox, 698
kivy.uix.scatter, 699
kivy.uix.scatterlayout, 704
kivy.uix.screenmanager, 705
kivy.uix.scrollview, 715
kivy.uix.selectableview, 721
kivy.uix.settings, 722
kivy.uix.slider, 734
kivy.uix.spinner, 738
kivy.uix.splitter, 740
kivy.uix.stacklayout, 743
kivy.uix.stencilview, 745
kivy.uix.tabbedpanel, 747
kivy.uix.textinput, 754
kivy.uix.togglebutton, 768
kivy.uixtreeview, 769
kivy.uix.video, 776
kivy.uix.videoplayer, 778
kivy.uix.vkeyboard, 784
kivy.uix.widget, 790
kivy.utils, 307
kivy.vector, 310
kivy.weakmethod, 314
kivy.weakproxy, 315