

Tkinter Moderno

para desenvolvedores Python ocupados

Mark Roseman

Conteúdo

Introdução.....	7 A quem se destina este tutorial.....	7 Melhores
Práticas Modernas.....	7 Extensões	
Tk.....	8 O melhor caminho	
a seguir.....	8 Como	
usar	9	
Convenções	9	
Instalando o Tk.....	10	
Instalando o Tk no Mac OS X.....	10	
Instalar o ActiveTcl.....	10 Instalar o	
Python.....	10 Instalando o Tk no	
Windows.....	11 Instalando o Tk no	
Linux.....	12 Opção 1. O gerenciador de pacotes da sua distribuição Linux.....	12 Opção 2. Instalar o
ActivePython.....	12 Opção 3. Instalar Tcl/Tk e Compilar o Padrão Distribuição do Python.....	13 Verificando sua
e Compilar o Padrão Distribuição do Python.....	13 Verificando sua	13 O Primeiro Programa
instalação.....	14 Um Primeiro Exemplo	
Obrigatório.....	14	
(Real)	14	
Desenho.....	14	
Código.....	15 Uma	
nota sobre o estilo de codificação.....	16	
Passo a passo passo a passo.....	17 O	
que está faltando.....	19	
Conceitos de Tk	19	
Widgets.....	19 Classes de	
widgets.....	20 Hierarquia de	
Janelas.....	20 Criando e usando	
widgets.....	21 Opções de	
configuração.....	21 Gerenciamento de	
Geometria.....	22 O	
Problema.....	23	
Como Funciona.....	23	
Manipulação de Eventos.....	24	
Chamadas de retorno de comando.....	24	
Ligações de Eventos.....	25 Virtuais	
Eventos.....	25 Ligações	
Múltiplas.....	26 Widgets	
Básicos.....	26	
Moldura.....	26	
Tamanho solicitado	27	
Preenchimento.....	27	
Fronteiras.....	27 Mudando	
estilos	28	
Rótulo.....	28 Exibindo	
Texto	29 Exibindo	
imagens.....	29	
Esquema.....	29	

Fontes, Cores e Mais.....	30
Botão.....	31 Texto
ou Imagem.....	31 O retorno
de chamada do comando.....	32 Estado
do botão.....	32 Botão de
verificação.....	33 Valor do
Widget.....	33 Botão de
opção.....	34
Entrada.....	34
Senhas.....	35 Estados
do Widget.....	35
Validação.....	36 Caixa
de combinação.....	36
Valores predefinidos.....	37 O
gerenciador de geometria da grade.....	37
Colunas e Linhas.....	38 Abrangendo
Múltiplas Células.....	38 Layout dentro da
célula.....	39 Manipulando o
redimensionamento.....	40
Preenchimento	40
Recursos Adicionais da Grade.....	42
Consultando e Alterando Opções de Grade.....	42
Interno Preenchimento.....	43
Esquecer e Remover.....	43
Layouts aninhados.....	44
Mais Widgets	44 Caixa
de listagem.....	44
Preenchendo os itens da caixa de listagem.....	45
Selecionando Itens	46
Estilizando a lista.....	46
Mantendo dados de itens extras.....	46
Exemplo	47 Barra
de rolagem	49
Exemplo	51
SizeGrip.....	51
Texto.....	52
Conteúdo.....	53 Barra de
Progresso.....	53 Progresso
Determinado.....	54 Progresso
Indeterminado.....	54
Escala.....	55 Caixa
giratória.....	56
Menus.....	57 Barras
de menu.....	58 Widgets
de Menu e Hierarquia.....	58 Antes de
começar.....	59 Criando uma
barra de menu.....	59 Adicionando
Menus.....	60 Adicionando
Itens de Menu.....	60 Tipos de Itens
de Menu.....	60

Teclas Aceleradoras.....	61
Mais sobre opções de itens.....	62
Sublinhado.....	62
Imagens	62
Estado	62
Consultando e Alterando Opções de Item.....	62
Menus da plataforma.....	62
63 Mac OS X.....	63
O Menu do Aplicativo.....	63
Lidando com o item do menu Preferências.....	64
Fornecendo um Menu de Ajuda.....	64
Fornecimento de um menu de janela.....	65
Outros manipuladores de menu.....	65
Janelas.....	66
X11	66
Menus Contextuais.....	67 Janelas e
Diálogos.....	67 Criando e
Destruindo Janelas.....	68 Mudando o
Comportamento e os Estilos da Janela.....	68 Título
da Janela.....	68 Tamanho e
Localização.....	68 Ordem de
Empilhamento.....	69
Comportamento de	
redimensionamento.....	70
Iconificando e Retirando.....	70 Caixas de
Diálogo Padrão.....	70
Selecionando arquivos e diretórios.....	70
Selecionando Cores.....	72
Caixas de diálogo de alerta e confirmação.....	73
Organizando Interfaces Complexas.....	74
Múltiplas janelas.....	74
Espaço em branco.....	74
Separador.....	75
Molduras de etiquetas.....	75
Janelas em painel.....	76
Caderno.....	77
Fontes, Cores, Imagens.....	78
Fontes.....	78
Fontes Padrão.....	79
Fontes Específicas da Plataforma.....	79
Fontes Nomeadas.....	80
Descrições de Fontes.....	80
Cores.....	81
Imagens.....	81
Problemas com PIL /Travesseiro no Mac OS X?.....	82
Canv como.....	82
Criando itens.....	83
Atributos do Item.....	84
Ligações.....	85
Etiquetas.....	85 Modifica

Rolagem.....	87	Outros tipos de
itens.....	89	
Texto.....	90	O
básico	90	Fornecimento
de Conteúdo Inicial.....	91	
Rolagem	91	Controlando
o Empacotamento.....	91	Desativando o
Widget.....	91	Recuperando o
Texto.....	91	Modificando o texto no
código.	92	Posições e Índices de
Texto.....	92	
texto.....	93	Exemplo: Janela de
registro.	93	Formatação com
tags.....	94	Adicionando Marcas ao
Texto.....	94	Aplicando formatação a
tags	94	Mais manipulações de
tags.....	95	Diferenças entre Tags em Tela e
Widgets de Texto.....	95	Eventos e
Ligações.....	96	Selecionando o
texto.....	96	
Marcas.....	96	Imagens e
Widgets.....	97	Ainda
mais.	98	
Pesquisar.....	98	
Modificações, Desfazer e Refazer.....	98	Elimine
o texto.....	98	
Introspecção.....	98	
Emparelhamento.....	98	
Árvore	99	
Adicionando itens à árvore.....	99	
Reorganizando itens.....	100	Exibindo
informações para cada item.....	100	Aparência de Item e
Eventos.....	101	Personalizando a
Tela.....	102	Estilos e
Temas.....	102	
Definições.....	102	Classe de
Widget.....	102	Estado do
Widget....	103	
Estilo.....	104	
Temas.....	104	Usando estilos e
temas..	104	Nomes de
estilo.....	104	Usando um
estilo.....	105	Usando
Temas.....	105	O que há dentro de
um estilo?.....	106	
Elementos.....	106	
Esquema.....	107	Opções de
Elemento.....	108	Alterando opções de
estilo.....	108	Modificando uma opção de
estilo.....	108	

Criando um novo estilo derivado	109
Opções de estilo específico do estado.....	109
Parece difícil para você?.....	110
Avançado: Mais sobre Elementos.....	112
Estudo de Caso: Modernização IDLE.....	
113 Objetivos do Projeto.....	115
Menus.....	116
Janela Principal.....	117
Fonte padrão	118
Ao redor do widget de texto.....	120 Caixa
de Diálogo de Preferências.....	122
Guias.....	125
Atualizando Widgets	126
Esquema.....	
.126 Outro Exemplo.....	128
Outros Diálogos.....	130
Diálogo Localizar.....	130
Sobre o Diálogo.....	133
Ajuda on-line.....	136 Diálogos
de consulta.....	137
Posicionamento do Diálogo.....	139
Integração de Janelas.....	140
Editor com guias.....	141
Depurador.....	142
Shell integrado e depurador.....	144 Soluções
alternativas, hacks e muito mais.....	144 Dicas
de ferramentas.....	144
Menus de Contexto.....	144
Widgets de texto de mesmo nível	144
Janelas Modais.....	144

Introdução

Este tutorial foi projetado para ajudar as pessoas a se familiarizarem rapidamente com a criação de interfaces gráficas de usuário de desktop convencionais com o Tk e, em particular, o Tk 8.5, que é um lançamento de marco incrivelmente significativo e um afastamento significativo das versões mais antigas do Tk que a maioria das pessoas conhece e amor reconhecer.

A desvantagem é que, a menos que você saiba uma ou duas coisas em particular, na verdade não é um lançamento tão significativo; Por motivos de compatibilidade com versões anteriores, a menos que os programas existentes façam algumas alterações simples, eles não parecerão muito diferentes. Portanto, embora este tutorial certamente beneficie os recém-chegados ao Tk, ele também ajudará os desenvolvedores de Tk existentes a atualizar seus conhecimentos. É um clichê, mas não acredito no quanto aprendi ao escrever este tutorial e uso o Tk há mais de quinze anos.

O estado geral da documentação Tk (fora a documentação de referência orientada a Tcl, que é excelente) infelizmente não está em um ponto alto atualmente. Isso é particularmente verdadeiro para desenvolvedores que usam Tk de linguagens diferentes de Tcl e desenvolvedores que trabalham em várias plataformas.

Portanto, este tutorial será, tanto quanto possível, direcionado aos desenvolvedores nas três plataformas principais (Windows, Mac, Linux) e também será neutro em termos de linguagem. Inicialmente, o tutorial abordará Tcl, Ruby, Perl e Python. Com o tempo, outros idiomas podem ser adicionados. Mesmo que seu próprio idioma não esteja incluído, é provável que você ainda se beneficie; como todas as linguagens usam a mesma biblioteca Tk subjacente, obviamente há muita sobreposição.

Este também não é um guia de referência, não vai cobrir tudo, apenas o essencial que você precisa em 95% dos aplicativos. O resto você pode encontrar na documentação de referência.

Para quem é este tutorial

Este tutorial foi desenvolvido para desenvolvedores que criam ferramentas e aplicativos em Tk. Ele também se preocupa com interfaces gráficas de usuário razoavelmente convencionais, com botões, listas, caixas de seleção, edição de rich text, gráficos 2D e assim por diante. Portanto, se você deseja hackear o código C interno do Tk ou criar a próxima grande interface de jogo imersiva em 3D, provavelmente este não é o material para você.

Este tutorial também não ensina a linguagem de programação subjacente (Tcl, Ruby, Perl, Python, etc.), então você já deve ter uma compreensão básica disso. Da mesma forma, você deve ter uma familiaridade básica com aplicativos de desktop em geral e, embora não precise ser um designer de interface do usuário, alguma apreciação do design da GUI é sempre útil.

Melhores práticas modernas

Este tutorial é sobre a construção de interfaces de usuário Tk modernas usando as ferramentas atuais que o Tk tem a oferecer. É tudo sobre as melhores práticas que você precisa saber para fazer isso.

Para a maioria das ferramentas, você não pensaria que teria que dizer algo assim, mas para Tk não é o caso. Tk teve uma evolução muito longa (ver [Tk Backgrounder](#)), e qualquer evolução tende a deixá-lo com um pouco de lixo; junte isso com o quanto as plataformas e padrões de interface gráfica do usuário evoluíram naquele tempo, e você pode ver onde manter algo tão grande e complexo como uma biblioteca GUI atualizada, bem como compatível com versões anteriores, pode ser um desafio.

Tk, nos últimos anos, ganhou uma má reputação, para dizer o mínimo. Parte disso foi bem merecido, a maior parte nem tanto. Como qualquer ferramenta GUI, ela pode ser usada para criar interfaces de usuário desatualizadas e de aparência terrível, mas com o devido cuidado e atenção, ela também pode ser usada para criar interfaces espetacularmente boas. A maioria das pessoas conhece os ruins; a maioria dos bons que as pessoas nem conhecem são feitos em Tk. Neste tutorial, vamos nos concentrar no que você precisa para criar boas interfaces de usuário, o que não é tão difícil quanto costumava ser antes do Tk 8.5.

Interfaces gráficas de usuário de desktop tão modernas, usando convenções modernas e senso de design, usando as ferramentas modernas fornecidas pelo Tk 8.5.

Extensões Tk

Quando se trata de práticas recomendadas modernas, as extensões Tk merecem uma observação especial. Ao longo dos anos, vários grupos forneceram todos os tipos de complementos ao Tk, por exemplo, adicionando novos widgets não disponíveis no núcleo (ou pelo menos não no momento). Algumas extensões Tk conhecidas e bastante populares incluem BLT, Tix, iWidgets, BWidgents; há muitos, muitos outros.

Muitas dessas extensões foram criadas anos atrás. Como o core Tk sempre foi altamente compatível com versões anteriores, essas extensões geralmente continuam a funcionar com versões mais recentes. No entanto, muitos não foram atualizados, ou não foram atualizados significativamente, há muito tempo. Eles podem não refletir as convenções ou estilos atuais da plataforma e, portanto, embora "funcionem", podem fazer com que seu aplicativo pareça extremamente desatualizado ou deslocado.

Se você decidir usar extensões Tk, é altamente recomendável que você investigue e analise suas escolhas com cuidado.

O melhor caminho a seguir

Tk também oferece muitas opções. Existem pelo menos seis maneiras diferentes de layout de widgets na tela, geralmente vários widgets diferentes que podem realizar a mesma coisa, especialmente se você contar a enorme variedade de extensões Tk como Tix, BLT, BWidgents, Itk e outros. A maioria deles também é mais antiga, a maioria não atualizada e, portanto, de aparência ruim e, em muitos casos, as instalações que eles fornecem foram obsoletas por instalações mais novas e modernas construídas recentemente no próprio Tk. Mas, por motivos de compatibilidade com versões anteriores, a maioria dessas formas antigas de fazer as coisas ainda continua funcionando, ano após ano. Isso não significa necessariamente que as pessoas ainda devam usar alguns deles.

Portanto, há muitas opções em Tk, mas, francamente, todas essas opções atrapalham. Se você quer aprender e usar Tk, você não precisa de todas as escolhas, você precisa da escolha *certa*, então você não precisa fazer toda a pesquisa e fazer essa escolha sozinho. Isso é o que este tutorial lhe dará. Pense nisso como o

documentação equivalente a [software opinativo](#). Portanto, frequentemente usaremos maneiras diferentes de fazer as coisas em outras documentações ou exemplos; muitas vezes, é porque quando eles foram escritos, as melhores maneiras ainda nem existiam. Mais tarde, quando você se tornar um especialista e se deparar com alguma situação maluca em que a escolha normal não se encaixa, você pode procurar alternativas.

Como usar

Embora o tutorial seja projetado para ser usado linearmente, sinta-se à vontade para pular como achar melhor. Frequentemente, forneceremos links onde você pode obter mais informações, sejam links para outra documentação neste site, como nosso "resumo de widget", fornecendo informações de uso em cada widget Tk, ou para documentação externa, como a referência completa para um determinado comando.

O tutorial também permite selecionar qual idioma (Tcl, Ruby, Perl ou Python) mostrar. Você pode alterar isso no menu pop-up "Mostrar:" localizado na barra lateral, próximo ao canto superior direito de cada página do tutorial. Mas também permite que você veja como o Tk é usado por todos os diferentes idiomas, o que pode ser bastante interessante e útil.

Convenções

Como normalmente é feito, listagens de código, comandos de interpretador ou shell e respostas serão indicados com uma fonte de largura fixa. Ao mostrar uma sessão interativa com o intérprete, as partes que você inserir também estarão em **negrito de largura fixa**.

Ao descrever chamadas de procedimento ou método, as partes literais (por exemplo, o nome do método) estarão em fonte simples de largura fixa, os parâmetros onde você deve preencher o valor real adicionarão itálico e os parâmetros opcionais serão cercados por '?', por exemplo "definir variável *valor*?".

Vários ícones que aparecem à esquerda do texto são usados, como segue:

Este parágrafo consiste em material específico para a ligação do Python ao Tk.

Este parágrafo ajudará a apontar erros comuns que as pessoas cometem ou sugerir soluções úteis, mas não necessariamente óbvias, relacionadas ao tópico.

Isso indica uma nova maneira de fazer as coisas em Tk 8.5, muito diferente da maneira como as coisas eram feitas anteriormente. Pessoas familiarizadas com versões mais antigas do Tk, ou trabalhando em programas desenvolvidos com versões mais antigas do Tk, devem prestar muita atenção.

Este parágrafo fornece algumas informações básicas adicionais, não estritamente necessárias para entender o tópico em questão, mas que podem ajudá-lo a entender um pouco mais sobre como ou por que as coisas são feitas da maneira que são.

Isso indica algum erro no próprio tutorial que ainda não foi corrigido ou uma seção que foi excluída, mas ainda não foi substituída.

Isso indica uma área em Tk que poderia ser caridosaamente descrita como uma "borda áspera". Isso pode indicar uma API ruim ou ausente, exigindo que você use uma solução alternativa em seu código. Porque essas coisas

tendem a ser corrigidos com o tempo, vale a pena marcá-los em seu código com um "TODO" para que você possa se lembrar de voltar mais tarde e ver se uma API mais recente resolve o problema de forma limpa.

Instalando Tk

Neste capítulo, você instalará o Tk em sua máquina, verificará se funciona e verá um exemplo rápido de como é um programa Tk.

Embora praticamente todas as máquinas Mac OS X e Linux já venham com o Tk instalado, geralmente é uma versão mais antiga (normalmente 8.4.x). Você quer ter certeza de que tem pelo menos a versão 8.5 (ou possivelmente 8.6) para usar o novo conjunto de widgets, portanto, se ainda não estiver disponível, instale a versão mais recente.

Embora existam várias maneiras de instalar o Tk, a mais fácil é baixar e instalar uma das versões fornecidas pelo ActiveState (www.activestate.com).

ActiveState é uma empresa que vende ferramentas de desenvolvimento profissional para linguagens dinâmicas. Eles também fornecem (gratuitamente) distribuições com controle de qualidade de algumas dessas linguagens e empregam vários desenvolvedores principais dessas linguagens.

Instalando o Tk no Mac OS X

Tkinter (e, desde Python 3.1, ttk) estão incluídos em todas as distribuições Python padrão no Mac OS X. No entanto, distribuições mais recentes nem **sempre incluem as bibliotecas Tcl e Tk subjacentes e arquivos de suporte**, que devem ser instalados separadamente.

Embora existam várias maneiras diferentes de obter Tcl e Tk em sua máquina, a mais fácil e recomendada é usar a distribuição ActiveTcl.

Se você é um masoquista e quer ler sobre outras opções e variações de Tcl/Tk e como elas interagem com Python, veja o Mac Tcl/Tk página em python.org.

Instalar ActiveTcl

Em seu navegador da Web, acesse www.activestate.com, e siga os links para baixar a Community Edition do ActiveTcl, disponível como um binário universal. Certifique-se de que está baixando uma versão 8.5.x, não uma versão 8.4.x mais antiga.

Execute o instalador para obter Tcl e Tk carregados em sua máquina.

Instalar Python

É importante que você use uma versão do Python compatível com Tk 8.5 ou superior e ttk. Recomendamos o uso do instalador padrão Python 3.x para Mac, que pode ser baixado em python.org.

As versões anteriores deste tutorial recomendavam o uso do ActivePython do ActiveState. As versões mais antigas costumavam incluir bibliotecas Tcl/Tk e arquivos de suporte, mas as versões mais recentes não o fazem mais e exigem

baixá-los separadamente, por exemplo, via ActiveTcl. Portanto, a distribuição padrão do Python funciona tão bem quanto, e o resultado final é o mesmo.

Execute o instalador e acompanhe. Você terminará com uma nova instalação do ActivePython em / Library/Frameworks, juntamente com links para os binários do Python com versão colocados em /usr/local/bin (por exemplo, 'python3.4' se você baixou o ActivePython 3.4.x). A partir de uma janela do Terminal, você poderá executar um shell Python:

```
% /usr/local/bin/python3.4
```

Isso deve fornecer o prompt de comando do Python. No prompt, insira estes dois comandos:

```
>>> importar tkinter  
>>> tkinter._test()
```

Isso deve abrir uma pequena janela; a primeira linha no topo da janela deve dizer "This is Tcl/Tk version 8.5"; certifique-se de que não é 8.4!

Você também pode obter a versão exata do Tcl/Tk que está sendo usado com:

```
>>> tkinter.Tcl().eval('info patchlevel')
```

que deve retornar algo como '8.5.18'.

Instalação verificada usando ActiveTcl 8.5.18.0 e Python 3.4.3 de python.org no Mac OS X 10.10.3.

Instalando Tk no Windows

Tkinter (e, desde o Python 3.1, ttk) estão incluídos em todas as distribuições padrão do Python. É importante que você use uma versão do Python compatível com Tk 8.5 ou superior e ttk. Recomendamos o uso do instalador padrão do Python 3.x para Windows, que pode ser baixado em python.org.

As versões anteriores deste tutorial recomendam o uso da distribuição ActivePython do ActiveState; isso também funcionará. Ambos contêm Python, bem como as bibliotecas Tcl/Tk subjacentes.

Execute o instalador e siga em frente. Você terminará com uma nova instalação do ActivePython, localizada em, por exemplo, C:\python34. A partir de um prompt de comando do Windows ou do comando "Executar..." do menu Iniciar, você poderá executar um shell Python por meio de:

```
% C:\python34\python
```

Isso deve fornecer o prompt de comando do Python. No prompt, insira estes dois comandos:

```
>>> importar tkinter  
>>> tkinter._test()
```

Isso deve abrir uma pequena janela; a primeira linha no topo da janela deve dizer "This is Tcl/Tk version 8.5" (ou "8.6"); certifique-se de que não é 8.4!

Você também pode obter a versão exata do Tcl/Tk que está sendo usado com:

```
>>> tkinter.Tcl().eval('info patchlevel')
```

que deve retornar algo como '8.5.18'.

Instalação verificada usando Python 3.4.3 de python.org (contendo Tcl/Tk 8.6.1) no Windows 8.1.

Instalando Tk no Linux

Tkinter (e, desde o Python 3.1, ttk) estão incluídos em todas as distribuições padrão do Python. É importante que você use uma versão do Python compatível com Tk 8.5 ou superior e ttk. Muitas distribuições do Python não instalarão automaticamente as bibliotecas Tcl e Tk subjacentes.

Você tem várias opções diferentes para colocar Python e Tkinter em sua máquina.

Opção 1. Gerenciador de pacotes da sua distribuição Linux

As distribuições Linux suportadas atualmente geralmente já incluem uma versão recente do Python 3.x (ou têm um pacote .deb, .rpm, etc. disponível para instalação). Se assim for, esta é geralmente a maneira mais fácil de ir.

No entanto, depois de terminar, inicie um shell Python (por exemplo, `/usr/bin/python3.4`) e verifique a instalação (veja abaixo). Você pode descobrir que, ao tentar 'importar tkinter', obtém um erro informando que precisa instalar outro pacote. Nesse caso, siga as instruções e tente novamente.

Por exemplo, executando o Ubuntu 14.04 LTS, o Python 3.4.0 já está incluído. No entanto, você também precisa instalar um pacote separado, `python3-tk`, para usar o Tkinter, por exemplo

```
% sudo apt-get install python3-tk
```

Nesse caso, esse pacote fornece bibliotecas Tcl/Tk 8.6.1 para serem usadas com Python.

Opção 2. Instalar o ActivePython

A distribuição "ActivePython" do ActiveState inclui Python e as bibliotecas Tcl/Tk necessárias.

Em seu navegador da Web, acesse www.activestate.com, e siga os links para baixar o Community Edition do ActivePython para Linux. Verifique se você está baixando uma versão 3.1 ou mais recente.

Descompacte-o, execute o instalador (`sudo ./install.sh`) e siga em frente. Você terminará com uma nova instalação do ActivePython, localizada em, por exemplo, /opt/ActivePython-3.4. A partir de uma janela do Terminal, você poderá executar um shell Python por meio de:

```
% /opt/ActivePython-3.4/bin/python3.4
```

Veja abaixo para verificar sua instalação.

Opção 3. Instalar Tcl/Tk e compilar a distribuição Python padrão

Se você quiser usar a distribuição de fonte padrão de python.org, você certamente pode fazer isso.

Mas para fazer isso, você precisará obter os arquivos Tcl e Tk include e as bibliotecas carregadas em sua máquina primeiro. Embora existam várias maneiras de fazer isso, a mais fácil é baixar e instalar o ActiveTcl.

Em seu navegador da Web, acesse www.activestate.com, e siga os links para baixar a Community Edition do ActiveTcl para Linux. Verifique se você está baixando uma versão 8.5 ou mais recente.

Descompacte-o, execute o instalador (`./install.sh`) e siga em frente. Você terminará com uma nova instalação do ActiveTcl, localizada em, por exemplo, `/opt/ActiveTcl-8.5`.

Em seguida, baixe a distribuição de origem do Python 3.x atual em python.org, e descompacte-o. Na sua linha de configuração, você precisará informar como encontrar a versão do Tcl/Tk que você instalou. Em seguida, construa como de costume:

```
% ./configure --with-tcltk-includes='-I/opt/ActiveTcl-8.5/include'
              --with-tcltk-libs='/opt/ActiveTcl-8.5/lib/libtcl8.5.so /opt/ActiveTcl
8.5/lib/libtk8.5.so' % ./make
% ./make install
```

Certifique-se de verificar sua instalação (veja abaixo).

Não funcionou? Pode ter ocorrido um erro ao compilar o código tkinter do Python. Para verificar, no diretório fonte principal do Python, tente "**touch Modules/_tkinter.c**" (**observe** o sublinhado) e depois "**make**" para recompilá-lo. Observe atentamente as mensagens de erro.

O mais comum é que a forma como você especificou o include Tcl/Tk e as bibliotecas precisa ser alterada de alguma forma. Ou se você receber mensagens de que certos arquivos de inclusão não podem ser encontrados (por exemplo "X11/Xlib.h") você pode precisar instalar pacotes adicionais em sua distribuição Linux (por exemplo, "**apt-get install libx11-dev**"). Depois de conseguir compilar sem erros, não se esqueça de "**fazer instalação**".

Verificando sua instalação

No prompt de comando do Python, insira estes dois comandos:

```
>>> importar tkinter
>>> tkinter._test()
```

Isso deve abrir uma pequena janela; a primeira linha no topo da janela deve dizer "This is Tcl/Tk version 8.5"; certifique-se de que não é 8.4!

Se ocorrer um erro ao tentar 'importar tkinter' (por exemplo, "Se isso falhar, seu Python pode não estar configurado para Tk"), algo não foi configurado corretamente. Se você mesmo compilou o Python, veja acima para verificar se há erros de compilação.

Você também pode obter a versão exata do Tcl/Tk que está sendo usado com:

```
>>> tkinter.Tcl().eval('info patchlevel')
```

que deve retornar algo como '8.5.18'.

Instalação verificada usando ActiveTcl 8.5.18, Python 3.4.3 de python.org no Ubuntu 14.04 LTS

O primeiro programa obrigatório

Para ter certeza de que tudo realmente funcionou, vamos tentar executar um programa "Hello World" em Tk.

Embora para algo tão curto você possa simplesmente digitá-lo diretamente no interpretador, use seu editor de texto favorito para colocá-lo em um arquivo.

```
from tkinter import *  
from  
tkinter import ttk  
root = Tk()  
ttk.Button(root, text="Hello  
World").grid()  
root.mainloop()
```

Salve isso em um arquivo chamado 'hello.py'. Em um prompt de comando, digite:

```
% python hello.py
```

Não foi possível encontrar hello.py? Você pode estar procurando no diretório errado. Tente fornecer o caminho completo para hello.py.



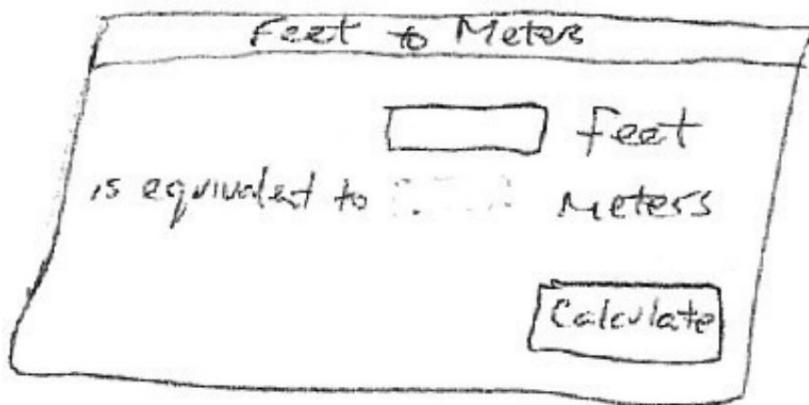
Nosso Primeiro Programa. Algum trabalho ainda a ser feito antes do IPO.

Um Primeiro Exemplo (Real)

Com isso fora do caminho, vamos tentar um exemplo um pouco mais útil, que lhe dará uma ideia inicial de como é o código por trás de um programa Tk.

Projeto

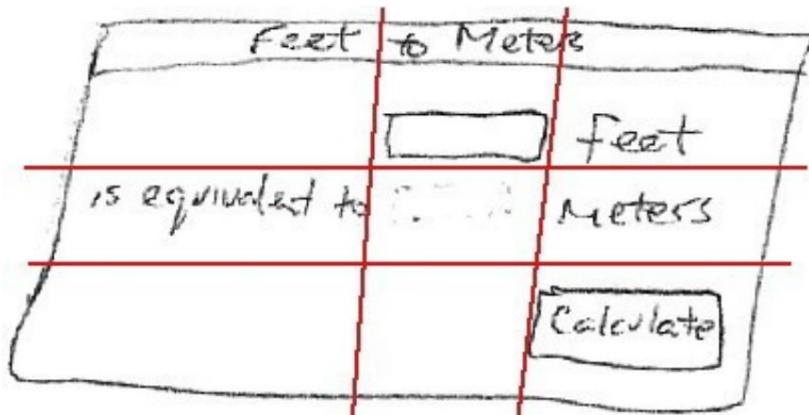
O exemplo que usaremos é uma ferramenta GUI simples que converterá um número de pés no número equivalente de metros. Se fôssemos esboçar isso, poderia ser algo assim:



Um esboço do nosso programa de conversão de pés para metros.

Parece que temos um pequeno widget de entrada de texto que nos permitirá digitar o número de pés e um botão 'Calcular' que obterá o valor dessa entrada, realizará o cálculo e, em seguida, colocará o número resultante de metros na tela logo abaixo de onde está a entrada. Também temos três rótulos estáticos ("pés", "equivalente a" e "metros") que ajudam nosso usuário a descobrir como usar a interface.

Em termos de layout, as coisas parecem se dividir naturalmente em três colunas e três linhas:



O layout da nossa interface de usuário, que segue uma grade 3 x 3.

Código

Agora aqui está o código Python para criar este programa.

```
da importação tkinter * da
importação tkinter ttk

def calcular(*args): tente:
    valor = float(feet.get())
    metros.set((0.3048 * valor * exceto
    ValueError: pass
                                10.000,0 + 0,5)/10.000,0)

root = Tk()
root.title("Pés para metros")
```

```

mainframe = ttk.Frame(root, padding="3 3 12 12")
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.columnconfigure(0, weight=1) mainframe.rowconfigure(0,
peso=1)

péss = StringVar() metros
= StringVar()

péss_entry = ttk.Entry(mainframe, largura=7, textvariable=péss) péss_entry.grid(coluna=2,
linha=1, sticky=(W, E))

ttk.Label(mainframe, textvariable=meters).grid(column=2, row=2, sticky=(W, E)) ttk.Button(mainframe,
text="Calcular", command=calcular).grid(coluna= 3, linha=3, pegajoso=W)

ttk.Label(mainframe, text="péss").grid(coluna=3, linha=1, sticky=W) ttk.Label(mainframe,
text="é equivalente a").grid(coluna=1, linha= 2, sticky=E) ttk.Label(mainframe, text="metros").grid(coluna=3,
linha=2, sticky=W)

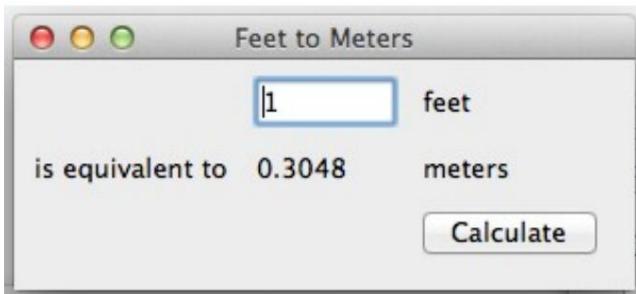
para filho em mainframe.winfo_children(): child.grid_configure(padx=5, pady=5)

foot_entry.focus()
root.bind('<Return>', calcular)

root.mainloop()

```

E a interface de usuário resultante:



Captura de tela de nossa interface de usuário concluída de pés a metros (no Mac OS X, Windows e Linux).

Uma observação sobre estilo de

codificação Cada uma das linguagens incluídas neste tutorial tem uma variedade de estilos e convenções de codificação disponíveis para escolha, que ajudam a determinar as convenções para nomes de variáveis e funções, estilos procedurais, funcionais ou orientados a objetos e assim por diante.

Como o foco deste tutorial é Tk, este tutorial manterá as coisas o mais simples possível, geralmente usando um estilo de codificação muito direto, em vez de agrupar a maior parte do nosso código em procedimentos, módulos, objetos, classes e assim por diante. Na medida do possível, você também verá os mesmos nomes para objetos, variáveis etc. usados nas linguagens de cada exemplo.

Passo a passo

Vamos dar uma olhada mais de perto nesse código, parte por parte. Por enquanto, tudo o que estamos tentando fazer é obter uma compreensão básica dos tipos de coisas que precisamos fazer para criar uma interface de usuário no Tk e, aproximadamente, como essas coisas se parecem. Entraremos em detalhes mais tarde.

```
da importação tkinter * da
importação ttk
```

Essas duas linhas informam ao Python que nosso programa precisa de dois módulos. A primeira, "tkinter", é a ligação padrão para Tk, que quando carregada também faz com que a biblioteca Tk existente em seu sistema seja carregada. O segundo, "ttk", é a ligação do Python aos "widgets temáticos" mais recentes que foram adicionados ao Tk na versão 8.5.

Observe que importamos tudo do módulo tkinter, para que possamos chamar funções tkinter etc. sem prefixá-las, o que é uma prática padrão do Tkinter. No entanto, como importamos apenas o próprio "ttk", isso significa que precisaremos prefixar qualquer coisa dentro desse módulo. Assim, por exemplo, chamar "Entry(...)" invocaria a função dentro do módulo tkinter, enquanto precisaríamos de "ttk.Entry(...)" para invocar a função dentro do ttk. Como você verá, várias funções são definidas em ambos os módulos e, às vezes, você precisará de ambas, dependendo do contexto. Tornar as chamadas ttk explícitas facilita isso e será o estilo usado neste tutorial.

Uma das primeiras coisas que você descobrirá se estiver migrando o novo código é que o nome do módulo Tkinter agora está em letras minúsculas, ou seja, "tkinter", em vez de "Tkinter". Isso foi alterado a partir do Python 3.0.

```
root = Tk()
root.title("Pés em metros")
mainframe = ttk.Frame(root, padding="3 3 12 12")
mainframe.grid(column=0, row=0)
mainframe.columnconfigure(0, weight=1)
mainframe.rowconfigure(0, weight=1)
```

Sim, a função "calcular" apareceu antes disso. Vamos descrevê-lo abaixo, mas precisamos incluí-lo próximo ao início porque o referenciamos em outras partes do programa.

Em seguida, as linhas acima configuram a janela principal, dando-lhe o título "Pés a Metros". Em seguida, criamos um widget de quadro, que conterá todo o conteúdo de nossa interface de usuário e o colocamos em nossa janela principal. Os bits "columnconfigure"/"rowconfigure" apenas informam a Tk que se a janela principal for redimensionada, o quadro deve se expandir para ocupar o espaço extra.

Estritamente falando, poderíamos apenas colocar as outras partes de nossa interface diretamente na janela raiz principal, sem o quadro de conteúdo interventor. No entanto, a janela principal não faz parte dos widgets "temáticos", então sua cor de fundo não corresponderia aos widgets temáticos que colocaremos dentro dela.

O uso de um widget de quadro "temático" para manter o conteúdo garante que o plano de fundo esteja correto.

```
pés = StringVar()
metros = StringVar()
pés_entry = ttk.Entry(mainframe, width=7, textvariable=pés)
pés_entry.grid(column=2, row=0)
sticky=(W, E)
tk.Label(mainframe, textvariable=metros).grid(column=2, row=1)
sticky=(W, E)
```

```
ttk.Button(mainframe, text="Calcular", comando=calcular).grid(coluna=3, linha=3, sticky=W)
```

As linhas anteriores criam os três widgets principais em nosso programa: a entrada onde digitamos o número de pés, um rótulo onde colocamos o número de metros resultante e o botão calcular que pressionamos para realizar o cálculo.

Para cada um dos três widgets, precisamos fazer duas coisas: criar o próprio widget e colocá-lo na tela. Todos os três widgets, que são 'filhos' de nossa janela de conteúdo, são criados como instâncias de uma das classes temáticas de widgets de Tk. Ao mesmo tempo em que os criamos, damos a eles certas opções, como a largura da entrada, o texto a ser colocado dentro do Button, etc. Cada entrada e rótulo recebem uma misteriosa "variável de texto"; veremos o que isso faz em breve.

Se os widgets acabaram de ser criados, eles não aparecerão automaticamente na tela, porque Tk não sabe como você deseja que eles sejam colocados em relação a outros widgets. É isso que a parte "grade" faz.

Lembrando a grade de layout de nosso aplicativo, colocamos cada widget na coluna apropriada (1, 2 ou 3) e linha (também 1, 2 ou 3). A opção "aderente" diz como o widget se alinharia dentro da célula da grade, usando as direções da bússola. Portanto, "w" (oeste) significa ancorar o widget no lado esquerdo da célula, "nós" (oeste-leste) significa ancorá-lo nos lados esquerdo e direito e assim por diante.

```
ttk.Label(mainframe, text="pés").grid(coluna=3, linha=1, sticky=W) ttk.Label(mainframe,
text="é equivalente a").grid(coluna=1, linha= 2, sticky=E) ttk.Label(mainframe, text="metros").grid(coluna=3,
linha=2, sticky=W)
```

As três linhas acima fazem exatamente a mesma coisa para os três rótulos de texto estático em nossa interface de usuário; crie cada um e coloque-o na tela na célula apropriada na grade.

```
para filho em mainframe.winfo_children(): child.grid_configure(padx=5, pady=5) feet_entry.focus()
root.bind('<Return>', calcular)
```

As três linhas anteriores ajudam a dar alguns retoques finais em nossa interface de usuário.

A primeira linha percorre todos os widgets que são filhos de nosso quadro de conteúdo e adiciona um pouco de preenchimento ao redor de cada um, para que não fiquem tão amontoados. Poderíamos ter adicionado essas opções a cada chamada de "grade" quando colocamos os widgets na tela pela primeira vez, mas esse é um bom atalho.

A segunda linha diz a Tk para colocar o foco em nosso widget de entrada. Dessa forma, o cursor começará naquele campo, para que o usuário não precise clicar nele antes de começar a digitar.

A terceira linha diz a Tk que se o usuário pressionar a tecla Return (Enter no Windows) em qualquer lugar dentro da janela raiz, ele deve chamar nossa rotina de cálculo, da mesma forma como se o usuário pressionasse o botão Calcular.

```
def calcular(*args): tente:
    valor = float(feet.get())
    metros.set((0.3048 * valor * exceto
    ValueError: pass
                                10.000,0 + 0,5)/10.000,0)
```

Aqui definimos nosso procedimento de cálculo, que é chamado quando o usuário pressiona o botão Calcular ou pressiona a tecla Enter. Ele executa o cálculo de pés para metros, pegando o número de pés de nosso widget de entrada e colocando o resultado em nosso widget de rótulo.

Dizer o que? Parece que não estamos fazendo nada com esses widgets! Aqui é onde as opções mágicas de "variável de texto" que especificamos ao criar os widgets entram em ação. Especificamos a variável global "pés" como a variável de texto para a entrada, o que significa que sempre que a entrada for alterada, Tk atualizará *automaticamente* a variável global pés. Da mesma forma, se alterarmos explicitamente o valor de uma variável de texto associada a um widget (como estamos fazendo para "metros" que está anexado ao nosso rótulo), o widget será atualizado automaticamente com o conteúdo atual da variável. Liso.

```
root.mainloop()
```

Esta linha final diz a Tk para entrar em seu loop de eventos, que é necessário para fazer tudo rodar.

O que está a faltar

Também vale a pena examinar o que *não* precisávamos incluir em nosso programa Tk para fazê-lo funcionar. Por exemplo:

- não precisávamos nos preocupar em redesenhar a tela conforme as coisas mudavam • não precisávamos nos preocupar em analisar e despachar eventos, detecção de ocorrências ou manipulação de eventos em cada widget
- não precisávamos fornecer muitas opções ao criar os widgets; os padrões pareciam cuidar da maioria das coisas, então só tivemos que mudar coisas como o texto que o botão exibia
- não precisávamos escrever código complexo para obter e definir os valores de widgets simples; Nós apenas anexá-los a variáveis
- não precisávamos nos preocupar com o que aconteceria quando o usuário fechasse a janela ou a redimensionasse • não precisávamos escrever código extra para que tudo isso funcionasse entre plataformas

Conceitos Tk

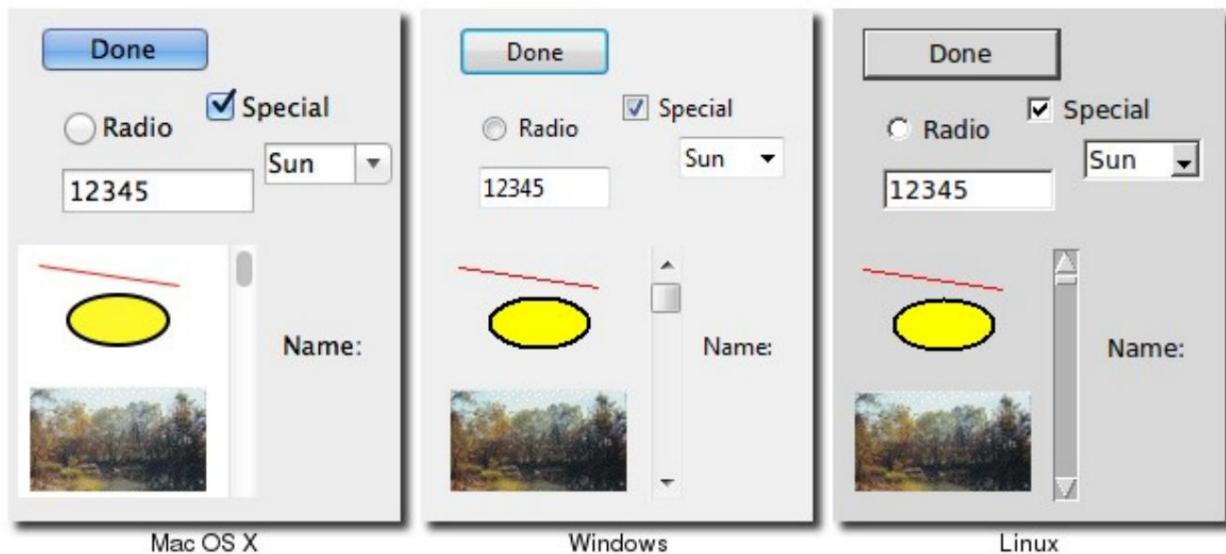
Com um primeiro exemplo atrás de você, você deve ter uma ideia básica de como um programa Tk pode parecer e os tipos de tarefas que ele precisa realizar. Voltaremos um pouco e examinaremos três conceitos amplos que você precisa conhecer para entender o Tk: widgets, gerenciamento de geometria e manipulação de eventos.

Widgets

Widgets são todas as coisas que você vê na tela. Em nosso exemplo, tínhamos um botão, uma entrada, alguns rótulos e um quadro. Outros são itens como caixas de seleção, exibições em árvore, barras de rolagem, áreas de texto e assim por diante.

Widgets são frequentemente chamados de "controles"; você também os verá freqüentemente referidos como "janelas", particularmente na documentação do Tk, um resquício de suas raízes X11 (portanto, sob essa terminologia, tanto uma janela de nível superior quanto coisas como um botão seriam chamadas de janelas).

Aqui está um exemplo mostrando alguns dos widgets do Tk, que abordaremos individualmente em breve.



Vários Tk Widgets.

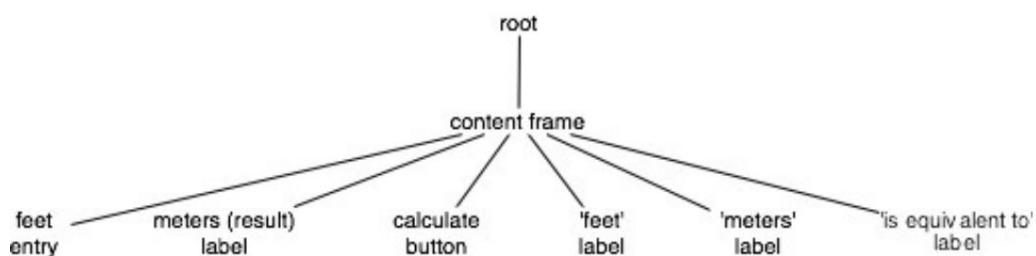
Classes de widgets

Widgets são objetos, instâncias de classes que representam botões, quadros e assim por diante. Portanto, a primeira coisa que você precisa fazer é identificar a classe específica do widget que deseja instanciar. Este tutorial e o resumo do [widget](#) vai ajudar com isso.

Hierarquia de janelas

A outra coisa que você precisa saber é o *pai* da instância do widget que deseja criar. Em Tk, todos os widgets fazem parte de uma *hierarquia de janela*, com uma única raiz no topo da hierarquia. Essa hierarquia pode ser arbitrariamente profunda; então você pode ter um botão em um quadro em outro quadro dentro da janela raiz. Mesmo uma nova janela de nível superior faz parte dessa mesma hierarquia, com ela e todo o seu conteúdo formando uma subárvore da hierarquia geral da janela.

Em nosso exemplo de conversão métrica, tínhamos um único quadro criado como filho da janela raiz e esse quadro tinha todos os outros controles como filhos. A janela raiz era um *container* para o quadro e, portanto, o *pai do quadro*. A hierarquia completa para o exemplo ficou assim:



A hierarquia da janela do exemplo de conversão de métrica.

Criando e usando widgets

Cada widget separado é um objeto Python. Ao criar um widget, você deve passar seu pai como parâmetro para a função de criação do widget. A única exceção é a janela "raiz", que é a janela de nível superior que conterá todo o resto. Isso é criado automaticamente e não tem um pai. Por exemplo:

```
root = Tk()
conteúdo = ttk.Frame(root)
botão = ttk.Button(conteúdo)
```

Se você salva ou não o objeto do widget em uma variável, depende inteiramente de você e depende, é claro, se você precisará consultá-lo mais tarde. Como o objeto é inserido na hierarquia do widget, ele não será coletado como lixo, mesmo que você não mantenha sua própria referência a ele.

Se você der uma olhada em como o Tcl gerencia os widgets, verá que cada widget tem um nome de caminho específico; você também verá esse nome de caminho mencionado na documentação de referência Tk. Tkinter escolhe e gerencia todos esses nomes de caminho para você nos bastidores, então você nunca deve se preocupar com eles. Se você fizer isso, poderá obter o nome do caminho de um widget chamando "str(widget)".

Opções de configuração

Todos os widgets também possuem várias *opções de configuração diferentes*, que geralmente controlam como são exibidos ou como se comportam.

As opções disponíveis dependem da classe do widget, é claro. Há muita consistência entre diferentes classes de widgets, então opções que fazem praticamente a mesma coisa tendem a ter o mesmo nome. Portanto, tanto um botão quanto um rótulo têm uma opção de "texto" para ajustar o texto exibido pelo widget, enquanto uma barra de rolagem, por exemplo, não teria uma opção de "texto", pois não é necessária. Da mesma forma, o botão tem uma opção de "comando" que diz o que fazer quando pressionado, enquanto um rótulo, que contém apenas texto estático, não.

As opções de configuração podem ser definidas quando o widget é criado pela primeira vez, passando os nomes e valores das opções como parâmetros opcionais. Você pode verificar posteriormente qual é o valor dessas opções e, com um número muito pequeno de exceções, alterá-las a qualquer momento. Se você não tiver certeza de quais são todas as diferentes opções para um widget, peça ao widget para fornecê-las. Isso fornece uma longa lista de todas as opções e, para cada opção, você pode ver o nome da opção e seu valor atual (junto com três outros atributos com os quais você normalmente não precisa se preocupar).

Tudo isso é melhor ilustrado com o seguinte diálogo interativo com o intérprete.

```
% python
>>> from tkinter import *
from tkinter import ttk
root = Tk()
crie um botão, passando duas
opções: >>> button = ttk.Button(root,
text="Hello", command ="botão pressionado")
>>> button.grid()
```

verifique o valor atual da opção de texto: >>> botão['texto']

'Olá'

altere o valor da opção de texto: >>> botão['texto']

= 'adeus' outra forma de fazer a mesma coisa:

>>> botão.configure(texto='adeus') verifique o valor atual de a opção de texto: >>> button['text']

'goodbye' obtenha todas as informações sobre a opção de

texto: >>> button.configure('text') ('text', 'text', 'Text', "",

'tchau') obtenha informações sobre todas as opções para

este widget: >>> button.configure() {'cursor': ('cursor',

'cursor', 'Cursor', "", ""), 'estilo': ('estilo', 'estilo', 'Estilo', "", ""),

'padrão': ('padrão', 'padrão', 'Padrão', <objeto de índice em

0x00DFFD10>, <objeto de índice em 0x00DFFD10 >), 'texto':

('texto', 'texto', 'Texto', "", 'adeus'), 'imagem': ('imagem', 'imagem',

'Imagen', "",), 'class': ('class', "", "", ""), 'padding': ('padding', 'padding', 'Pad', "", ""), 'largura ': ('largura',

'largura', 'Largura', "",), 'estado': ('estado', 'estado', 'Estado', <objeto de índice em 0x0167FA20>, <objeto

de índice em 0x0167FA20>), 'comando': ('comando', 'comando' 'variável de texto': ('variável de texto', 'te

xtVariable', 'Variable', "",), 'compound': ('compound', 'compound', 'Compound', <index object at

0x0167FA08>, <index object at 0x0167FA08>), 'underline': ('sublinhado', 'sublinhado', 'Sublinhado', -1, -1),

'takefocus': ('takefocus', 'takeFocus', 'TakeFocus', "", 'ttk::takefocus')}

, 'Comando', "", 'botão pressionado'),

Gerenciamento de Geometria

Se você já brincou com a criação de widgets, provavelmente notou que só de criá-los eles não apareciam na tela. Colocar as coisas na janela da tela e exatamente *onde* elas aparecem na janela é uma etapa separada chamada *gerenciamento de geometria*.

Em nosso exemplo, esse posicionamento foi realizado pelo comando "grid", onde passamos a coluna e linha que queríamos que cada widget entrasse, como as coisas deveriam ser alinhadas dentro da grade e assim por diante. Grid é um exemplo de *gerenciador de geometria* (existem vários em Tk, sendo grid o mais útil). Falaremos sobre grade em detalhes em um capítulo posterior, mas, por enquanto, veremos o gerenciamento de geometria em geral.

O trabalho de um gerente de geometria é descobrir exatamente onde esses widgets serão colocados.

Isso acaba sendo um problema de otimização muito difícil, e um bom gerenciador de geometria depende de algoritmos bastante complexos. Um bom gerenciador de geometria fornece flexibilidade, poder e facilidade de uso que deixa os programadores felizes, e a "grade" do Tk é sem dúvida uma das melhores. Um péssimo gerenciador de geometria... bem, todos os programadores Java que sofreram com o "GridBagLayout", por favor, levantem suas mãos.

O problema

O problema para um gerenciador de geometria é pegar todos os diferentes widgets que o programa cria, mais as instruções para onde na janela o programa gostaria que as coisas fossem (explicitamente, ou com mais frequência, em relação a outros widgets) e, em seguida, colocá-los. na janela.

Ao fazer isso, o gerenciador de geometria precisa equilibrar várias restrições diferentes:

- Os widgets podem ter um tamanho "natural" (por exemplo, a largura natural de um rótulo normalmente seria determinado pelo texto e fonte nele), mas o nível superior em que todos esses diferentes widgets estão tentando se encaixar não é grande o suficiente para acomodá-los; o gerenciador de geometria deve decidir quais widgets encolher para caber, quanto, etc. • Se a janela de nível superior for maior que o tamanho natural de todos os widgets, como o espaço extra é usado? Ele é usado apenas para espaço extra entre os widgets e, em caso afirmativo, como esse espaço é distribuído? Ele é usado para tornar certos widgets maiores do que normalmente gostariam de ser? • Se a janela de nível superior for redimensionada, como o tamanho e a posição dos widgets serão alterados?

Certas áreas (por exemplo, uma área de entrada de texto) serão expandidas ou reduzidas, enquanto outras partes permanecerão do mesmo tamanho ou a área será distribuída de forma diferente? Alguns widgets têm um tamanho mínimo (ou máximo) que você deseja evitar que fique abaixo (mais)?

- Como os widgets em diferentes partes da interface do usuário podem ser alinhados entre si, para apresentar um layout limpo e corresponder às diretrizes da plataforma relacionadas ao espaçamento entre widgets? • Para uma interface de usuário complexa, que pode ter muitos quadros aninhados em outros quadros aninhados na janela (etc.), como todos os itens acima podem ser realizados, compensando as demandas conflitantes de diferentes partes de toda a interface do usuário?

Como funciona

O gerenciamento de geometria em Tk depende do conceito de widgets *mestre* e *escravo*. Um mestre é um widget, geralmente uma janela de nível superior ou um quadro, que conterá outros widgets, chamados de escravos. Você pode pensar em um gerenciador de geometria como assumindo o controle do widget mestre e decidindo o que será exibido dentro dele.

O gerenciador de geometria pedirá a cada widget escravo seu tamanho natural, ou quanto grande ele gostaria de ser exibido. Em seguida, ele pega essas informações e as combina com quaisquer parâmetros fornecidos pelo programa quando solicita ao gerenciador de geometria para gerenciar esse widget escravo específico. Em nosso exemplo, passamos à grade um número de "coluna" e "linha" para cada widget, que indicava a posição relativa do widget em relação aos outros, e também um parâmetro "adesivo" para sugerir como o widget deve ser alinhado ou possivelmente esticado. Também usamos "columnconfigure" e "rowconfigure" para indicar as colunas e linhas que gostaríamos de expandir se houver espaço extra disponível na janela. Claro, todos esses parâmetros são específicos da grade; outros gerenciadores de geometria usariam outros diferentes.

O gerenciador de geometria pega todas as informações sobre os escravos, bem como as informações sobre o tamanho do mestre, e usa seus algoritmos internos para determinar a área que cada escravo será alocado (se houver!). O escravo é então responsável por desenhar etc. dentro desse retângulo específico.

E, claro, sempre que o tamanho do mestre mudar (por exemplo, porque a janela de nível superior foi redimensionada), o tamanho natural de um escravo mudar (por exemplo, porque alteramos o texto em um rótulo) ou qualquer um dos parâmetros do gerenciador de geometria mudança (por exemplo, como "row", "column" ou "sticky"), repetimos a coisa toda.

Isso tudo funciona recursivamente também. Em nosso exemplo, tínhamos um quadro de conteúdo dentro da janela de nível superior e vários outros controles no quadro de conteúdo. Portanto, tínhamos um gerenciador de geometria trabalhando em dois mestres diferentes. No nível externo, a janela de nível superior era a mestre e o quadro de conteúdo era a escrava. No nível interno, o quadro de conteúdo era o mestre, com cada um dos outros widgets sendo escravos. Portanto, o mesmo widget pode ser mestre e escravo. Essa hierarquia também pode, é claro, ser aninhada muito mais profundamente.

Embora cada mestre possa ter apenas um gerenciador de geometria (por exemplo, grade), é totalmente possível que diferentes mestres tenham diferentes gerenciadores de geometria; embora a grade seja geralmente usada, outras podem fazer sentido para um layout específico usado em uma parte da interface do usuário. Além disso, estamos assumindo que os widgets escravos são os filhos imediatos de seu mestre na hierarquia de widgets.

Embora este seja geralmente o caso, e principalmente não há uma boa razão para fazer isso de outra maneira, também é possível (com algumas restrições) contornar isso.

Manipulação de eventos

Em Tk, como na maioria dos outros toolkits de interface com o usuário, há um *loop de eventos* que recebe eventos do sistema operacional. São coisas como pressionar botões, pressionar teclas, mover o mouse, redimensionar janelas e assim por diante.

Geralmente, Tk cuida do gerenciamento desse loop de eventos para você. Ele descobrirá a qual widget o evento se aplica (o usuário clicou neste botão? se uma tecla foi pressionada, qual caixa de texto teve o foco?) E o enviará de acordo. Widgets individuais sabem como responder a eventos, então, por exemplo, um botão pode mudar de cor quando o mouse passa sobre ele e voltar quando o mouse sai.

Chamadas de retorno de comando

Muitas vezes, você deseja que seu programa lide com eventos específicos, por exemplo, fazendo algo quando um botão é pressionado. Para aqueles eventos que são praticamente essenciais para personalizar (de que serve um botão sem que algo aconteça quando você o pressiona?), o widget fornecerá um *retorno de chamada* como uma opção de configuração do widget. Vimos isso no exemplo com a opção "comando" do botão.

Callbacks em Tk tendem a ser mais simples do que em toolkits usados com linguagens compiladas (onde um callback geralmente deve ser direcionado a um procedimento com um certo conjunto de parâmetros ou um método de objeto com uma certa assinatura). Em vez disso, o retorno de chamada é apenas um pedaço normal de código que o interpretador avalia. Embora possa ser tão complexo quanto você deseja, na maioria das vezes, você só deseja que seu retorno de chamada chame algum outro procedimento.

Vinculações de eventos

Para eventos que não possuem um retorno de chamada de comando associado a eles, você pode usar o "bind" de Tk para capturar qualquer evento e, em seguida, (como nos retornos de chamada) executar um código arbitrário.

Aqui está um exemplo (bobó) que mostra como um rótulo pode ter ligações configuradas para responder a diferentes eventos, o que ele faz apenas alterando o que é exibido no rótulo.

```
from tkinter import * from
tkinter import ttk root = Tk() l
=ttk.Label(root, text="Iniciando...")
l.grid() l.bind('<Enter>', lambda e: l.configure(text='Rato
movido para dentro')) l.bind('<Leave>', lambda e:
l.configure(text='Rato movido para fora')) l.bind('<1>', lambda e: l.configure(text='Botão
esquerdo do mouse clicado')) l.bind('<Double-1>', lambda e: l.configure(text='Clique duplo'))
l.bind('<B3-Motion >', lambda e: l.configure(text='botão direito arraste para %d,%d' % (ex, ey)))
root.mainloop()
```

As três primeiras associações de eventos são bastante diretas, apenas olhando para eventos simples. A ligação de clique duplo apresenta a ideia de um *modificador de evento*; neste caso, queremos acionar o evento em um clique esquerdo do mouse (o "1"), mas somente quando for um clique duplo (o "Double-").

A última ligação também usa um modificador: captura o movimento do mouse ("Motion"), mas somente quando o botão direito do mouse ("B3") é pressionado. Essa ligação também mostra um exemplo de como usar os *parâmetros do evento*, por meio do uso de *substituições percentuais*. Muitos eventos, como cliques do mouse ou movimento, têm como parâmetros informações adicionais, como a posição atual do mouse. Essas substituições percentuais permitem capturá-las para que possam ser usadas em seu script.

O Tkinter espera que você forneça uma função como retorno de chamada do evento, cujo primeiro argumento é um objeto de evento que representa o evento que acionou o retorno de chamada. Geralmente não vale a pena definir funções nomeadas regulares para retornos de chamada únicos, como neste exemplo, então usamos apenas as funções anônimas do Python via lambda. O exemplo anterior de pés para metros usava uma função definida regular (calcular).

Para obter uma descrição completa de todos os diferentes nomes de evento, modificadores e os diferentes parâmetros de evento disponíveis em cada um, o melhor lugar para procurar é a [referência de comando "bind"](#).

Eventos Virtuais

Além dos eventos de sistema operacional de baixo nível, como cliques do mouse e redimensionamento de janelas, muitos widgets geram eventos de nível superior chamados de *eventos virtuais*. Por exemplo, um widget de caixa de listagem gerará um evento virtual "ListboxSelect" sempre que a seleção for alterada, independentemente de o usuário ter clicado em um item, movido para ele com as teclas de seta ou qualquer outra coisa. Isso evita o problema de configurar várias ligações de evento, possivelmente específicas da plataforma, para capturar a mudança.

Eventos virtuais para um widget, se houver, serão listados na documentação do widget.

Ligações Múltiplas

Na verdade, os widgets podem ter vários acionadores de associações de eventos diferentes para um único evento.

Normalmente, os eventos podem ser configurados para: o próprio widget individual, todos os widgets de uma determinada classe (por exemplo, botões), a janela de nível superior que contém o widget e todos os widgets no aplicativo. Cada um deles disparará em seqüência.

Vimos isso em nosso exemplo quando configuramos um vínculo para a tecla Return na janela de nível superior, e isso se aplica a todos os widgets dessa janela.

O comportamento padrão de cada classe de widget em Tk é definido com ligações de evento em nível de script e, portanto, pode ser introspectivo e modificado para alterar o comportamento de todos os widgets de uma determinada classe. Você pode até modificar completamente a manipulação dessa sequência múltipla de eventos para cada widget; consulte a [referência de comando "bindtags"](#) se você está curioso.

Widgets Básicos

Este capítulo apresenta os widgets Tk básicos que você encontrará em praticamente qualquer interface de usuário: frames, labels, botões, checkbuttons, radiobuttons, entradas e caixas de combinação. No final, você saberá como usar todos os widgets necessários para um tipo de interface de usuário típico de formulário de preenchimento.

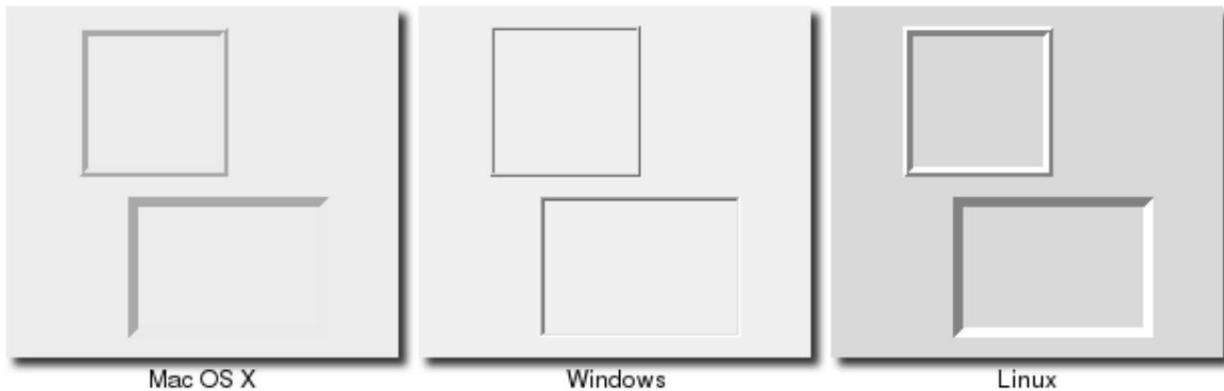
Este capítulo (e os seguintes que discutem mais widgets) devem ser lidos na ordem. Como há muita semelhança entre muitos widgets, introduziremos certos conceitos em um widget anterior que também se aplicarão a um posterior. Em vez de passar pelo mesmo terreno várias vezes, vamos apenas nos referir a quando o conceito foi introduzido pela primeira vez.

Ao mesmo tempo, cada widget também se referirá ao [resumo do widget](#) [página para](#) o widget específico, bem como a [página do manual de referência](#), então sinta-se livre para pular um pouco também.

Quadro

- [Resumo de widgets](#) •
- [Manual de Referência](#)

Um **quadro** é um widget exibido como um retângulo simples. Os quadros são usados principalmente como um contêiner para outros widgets, que estão sob o controle de um gerenciador de geometria, como grade.



Widgets de quadro

Os quadros são criados usando a função **ttk.Frame** :

```
quadro = ttk.Frame(pai)
```

Os quadros podem ter várias opções de configuração diferentes que podem alterar a forma como são exibidos.

Tamanho solicitado

Como qualquer outro widget, após a criação, ele é adicionado à interface do usuário por meio de um gerenciador de geometria (pai). Normalmente, o tamanho que o quadro irá solicitar ao gerenciador de geometria será determinado pelo tamanho e layout de quaisquer widgets contidos nele (que estão sob o controle do gerenciador de geometria que gerencia o conteúdo do próprio quadro).

Se, por algum motivo, você quiser um quadro vazio que não contenha outros widgets, defina explicitamente o tamanho que o quadro solicitará de seu gerenciador de geometria pai usando as opções de configuração "largura" e/ou "altura" (caso contrário, você vai acabar com um quadro muito pequeno, de fato).

Normalmente, distâncias como largura e altura são especificadas apenas como um número de pixels na tela.

Você também pode especificá-los por meio de um dos vários sufixos. Por exemplo, "350" significa 350 pixels, "350c" significa 350 centímetros, "350i" significa 350 polegadas e "350p" significa 350 pontos da impressora (1/72 polegadas).

Preenchimento A opção de configuração "preenchimento" é usada para solicitar espaço extra dentro do widget; desta forma, se você estiver colocando outros widgets dentro do quadro, haverá um pouco de margem ao redor. Um único número especifica o mesmo preenchimento em toda a volta, uma lista de dois números permite especificar o preenchimento horizontal e depois o vertical, e uma lista de quatro números permite especificar o preenchimento esquerdo, superior, direito e inferior, nessa ordem.

```
frame['preenchimento'] = (5,10)
```

Fronteiras

Você pode exibir uma borda ao redor do widget de quadro; você vê muito isso onde você pode ter uma parte

a interface do usuário parecendo "afundada" ou "elevada" em relação ao ambiente. Para fazer isso, você precisa definir a opção de configuração "borderwidth" (que tem como padrão 0, portanto sem borda), bem como a opção "relief", que especifica a aparência visual da borda: "flat" (padrão), "elevado", "afundado", "sólido", "cume" ou "ranhura".

```
frame['borderwidth'] = 2
frame['relief'] = 'afundado'
```

Mudando estilos

Há também uma opção de configuração de "estilo", que é comum a todos os widgets temáticos, que permite controlar praticamente qualquer aspecto de sua aparência ou comportamento. Isso é um pouco mais avançado, então não vamos entrar nisso agora.

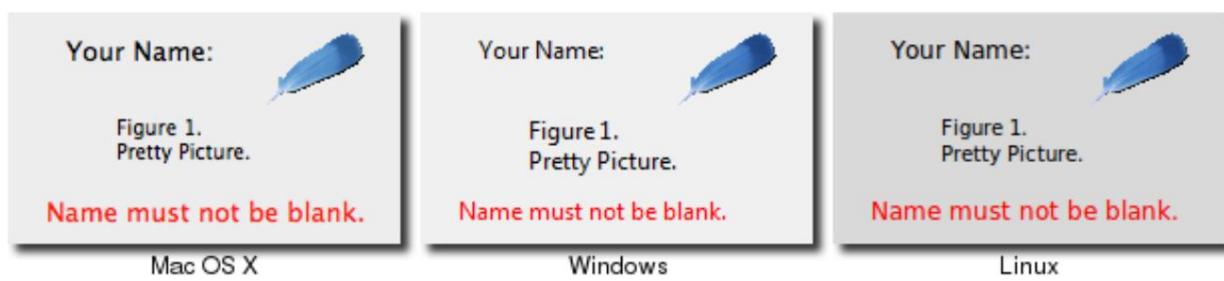
Os estilos marcam um afastamento acentuado da maneira como a maioria dos aspectos da aparência visual de um widget são alterados nos widgets Tk "clássicos". Enquanto no clássico Tk você pode fornecer uma ampla gama de opções para controlar com precisão todos os aspectos do comportamento (cor de primeiro plano, cor de fundo, fonte, espessura de realce, cor de primeiro plano selecionada, preenchimento etc.), nos novos widgets temáticos essas alterações são feitas alterando estilos, não adicionando opções a cada widget.

Como tal, muitas das opções com as quais você pode estar familiarizado em determinados widgets não estão presentes em sua versão temática. Dado que o uso excessivo de tais opções foi um fator chave para minar a aparência dos aplicativos Tk, especialmente quando movidos entre plataformas, a transição para widgets temáticos fornece um momento oportuno para revisar e refiná-los se e como essas alterações de aparência são feitas.

Rótulo

- [Resumo de widgets](#) •
- [Manual de Referência](#)

Um **rótulo** é um widget que exibe texto ou imagens, normalmente que o usuário apenas visualizará, mas não interagirá de outra forma. Os rótulos são usados para identificar controles ou outras partes da interface do usuário, fornecer feedback ou resultados textuais, etc.



Widgets de etiquetas

Os rótulos são criados usando a função **ttk.Label** e normalmente seus conteúdos são configurados no mesmo tempo:

```
label = ttk.Label(pai, text='Nome completo:')
```

Como os quadros, os rótulos podem ter várias opções de configuração diferentes que podem alterar a forma como são exibidos.

Exibição de Texto

A opção de configuração "texto" mostrada acima na hora de criar a etiqueta é a mais utilizada, principalmente quando a etiqueta é meramente decorativa ou explicativa. É claro que você pode alterar essa opção a qualquer momento, não apenas ao criar o rótulo pela primeira vez.

Você também pode fazer com que o widget monitore uma variável em seu script, para que sempre que a variável mudar, o rótulo exiba o novo valor da variável; isso é feito com a opção "textvariable":

```
resultsContents = StringVar()
label['textvariable'] = resultsContents
resultsContents.set('Novo valor a ser exibido')
```

Tkinter apenas permite que você se conecte a uma instância da classe "StringVar", que contém toda a lógica para observar mudanças, comunicá-las entre a variável e Tk, e assim por diante. Você precisa ler ou escrever o valor atual usando os métodos "get" e "set".

Exibindo Imagens

Você também pode exibir uma imagem em um rótulo em vez de texto; se você deseja apenas uma imagem em sua interface, normalmente é essa a maneira de fazer isso. Veremos as imagens com mais detalhes em um capítulo posterior, mas, por enquanto, vamos supor que você queira exibir uma imagem GIF que está em um arquivo em disco. Este é um processo de duas etapas, primeiro criando um "objeto" de imagem e, em seguida, informando ao rótulo para usar esse objeto por meio de sua opção de configuração "imagem":

```
image = PhotoImage(file='myimage.gif') label['image']
= image
```

Você pode usar tanto uma imagem quanto um texto, como verá frequentemente nos botões da barra de ferramentas, por meio da opção de configuração "composta". O valor padrão é "none", o que significa exibir apenas a imagem se presente, caso contrário, o texto especificado pelas opções "text" ou "textvariable". Outras opções são "texto" (somente texto), "imagem" (somente imagem), "centro" (texto no centro da imagem), "superior" (imagem acima do texto), "esquerda", "inferior" e "direita".

Layout

Enquanto o layout geral do rótulo (ou seja, onde ele está posicionado na interface do usuário e qual o seu tamanho) é determinado pelo gerenciador de geometria, existem várias opções que podem ajudá-lo a controlar como o rótulo será exibido na caixa o gerenciador de geometria fornece.

Se a caixa fornecida ao rótulo for maior do que o rótulo exige para seu conteúdo, você pode usar a opção "âncora" para especificar a qual borda ou canto o rótulo deve ser anexado, o que deixaria qualquer espaço vazio na borda ou canto oposto. Os valores possíveis são especificados como bússola

direções: "n" (norte ou borda superior), "ne", (nordeste ou canto superior direito), "e", "se", "s", "sw", "w", "nw" ou "centro".

Os rótulos podem ser usados para exibir mais de uma linha de texto. Isso pode ser feito incorporando retornos de carro ("\\n") na string "text"/"textvariable". Você também pode permitir que o rótulo envolva a string em várias linhas que não ultrapassem um determinado comprimento (com o tamanho especificado como pixels, centímetros, etc.), usando a opção "comprimento da quebra".

Os rótulos de várias linhas são um substituto para os widgets de "mensagem" mais antigos no clássico Tk.

Você também pode controlar como o texto é justificado, usando a opção "justificar", que pode ter os valores "esquerda", "centro" ou "direita". Se você tiver apenas uma única linha de texto, isso é praticamente o mesmo que usar apenas a opção "âncora", mas é mais útil com várias linhas de texto.

Fontes, cores e muito mais

Assim como nas molduras, normalmente você não quer tocar em coisas como a fonte e as cores diretamente, mas se precisar alterá-las (por exemplo, para criar um tipo especial de etiqueta), isso pode ser feito criando um novo estilo, que é então usado pelo widget com a opção "estilo".

Ao contrário da maioria dos widgets temáticos, o widget de rótulo também fornece opções explícitas específicas do widget como alternativa; novamente, você usaria isso apenas em casos pontuais especiais, quando usar um estilo não necessariamente faz sentido.

Você pode especificar a fonte usada para exibir o texto da etiqueta usando a opção de configuração "fonte".

Embora entraremos em fontes com mais detalhes em um capítulo posterior, aqui estão os nomes de algumas fontes predefinidas que você pode usar:

TkDefaultFont	O padrão para todos os itens GUI não especificados de outra forma.
TkTextFont	Usado para widgets de entrada, caixas de listagem, etc.
TkFixedFont	Uma fonte padrão de largura fixa.
TkMenuFont	A fonte usada para itens de menu.
TkHeadingFont	A fonte normalmente usada para cabeçalhos de coluna em listas e tabelas.
TkCaptionFont	Uma fonte para janelas e barras de legenda de diálogo.
TkSmallCaptionFont	Uma fonte de legenda menor para subjanelas ou caixas de diálogo de ferramentas
TkIconFont	Uma fonte para legendas de ícones.
TkTooltipFont	Uma fonte para dicas de ferramentas.
Como a escolha das fontes é tão específica da plataforma, tenha cuidado ao codificá-las (famílias de fontes, tamanhos, etc.);	isso é outra coisa que você verá em muitos programas Tk mais antigos que podem torná-los feios.

As cores do primeiro plano (texto) e do plano de fundo também podem ser alteradas por meio das opções "primeiro plano" e "plano de fundo". As cores são abordadas em detalhes posteriormente, mas você pode especificá-las como nomes de cores (por exemplo, "vermelho") ou códigos RGB hexadecimais (por exemplo, "#ff340a").

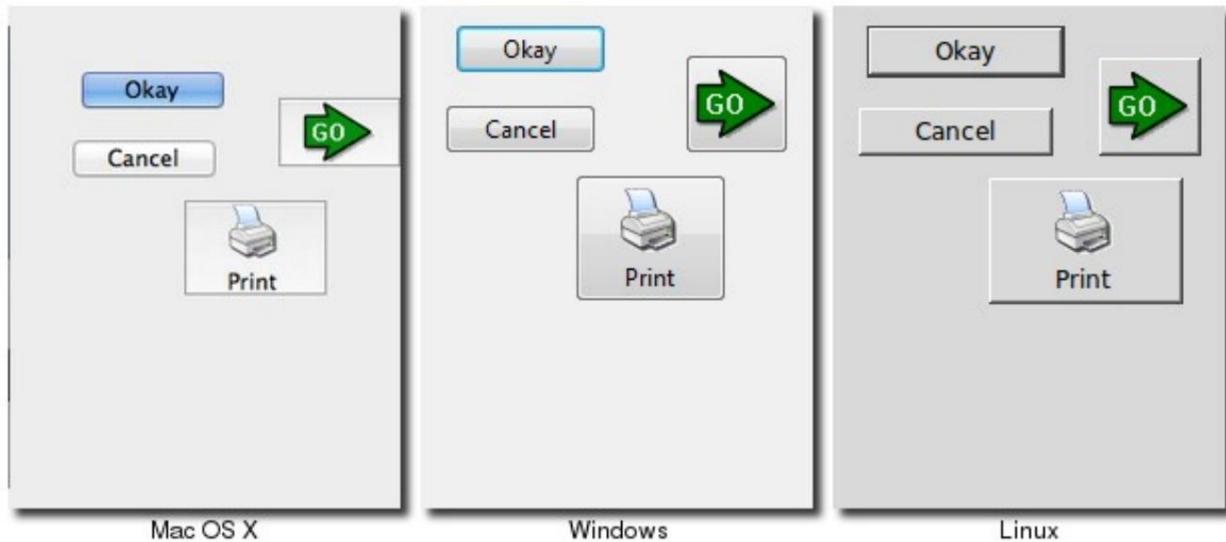
Os rótulos também aceitam a opção de "*relevo*" que foi discutida para quadros.

Botão

• [Resumo de widgets](#) •

[Manual de Referência](#)

Um **botão**, ao contrário de um quadro ou rótulo, é muito projetado para o usuário interagir e, em particular, pressionar para executar alguma ação. Assim como os rótulos, eles podem exibir texto ou imagens, mas também possuem toda uma gama de novas opções usadas para controlar seu comportamento.



Widgets de botão

Os botões são criados usando a função **ttk.Button** e, normalmente, seu conteúdo e retorno de chamada de comando são configurados ao mesmo tempo:

```
botão = ttk.Button(parent, text='Ok', command=submitForm)
```

Como em outros widgets, os botões podem ter várias opções de configuração diferentes que podem alterar sua aparência e comportamento.

Botões de texto

ou imagem usam as mesmas opções de configuração "texto", "variável de texto" (raramente usada), "imagem" e "composta" como rótulos, que controlam se o botão exibe texto e/ou uma imagem.

Os botões têm uma opção "padrão", que diz a Tk que o botão é o botão padrão na interface do usuário (ou seja, aquele que será invocado se o usuário pressionar Enter ou Return). Algumas plataformas e estilos desenharão isso com uma borda ou destaque diferente. Defina a opção como "ativo" para especificar que este é um botão padrão; o estado regular é "normal". Observe que definir essa opção não cria uma associação de evento que fará com que a tecla Return ou Enter ative o botão; que você mesmo tem que fazer.

O retorno de chamada de comando

A opção "comando" é usada para fornecer uma interface entre a ação do botão e seu aplicativo. Quando o usuário clica no botão, o script fornecido pela opção é avaliado pelo interpretador.

Você também pode pedir ao botão para invocar o retorno de chamada do comando de seu aplicativo. Isso é útil para que você não precise repetir o comando a ser invocado várias vezes em seu programa; para que você saiba que, se alterar a opção no botão, não precisará alterá-la em outro lugar também.

```
botão.invoke()
```

Estado do botão

Botões e muitos outros widgets podem estar em um estado normal onde podem ser pressionados, mas também podem ser colocados em um estado desativado, onde o botão fica acinzentado e não pode ser pressionado. Isso é feito quando o comando do botão não é aplicável em um determinado momento.

Todos os widgets temáticos carregam consigo um estado interno, que é uma série de sinalizadores binários. Você pode definir ou limpar esses sinalizadores diferentes, bem como verificar a configuração atual usando os métodos "state" e "instate". Os botões usam o sinalizador "desativado" para controlar se o usuário pode ou não pressionar o botão. Por exemplo:

```
button.state(['desabilitado']) botão           # definir o sinalizador desativado, desativando o
button.state(['!desabilitado'])                 # limpe o sinalizador desabilitado
button.instate(['desabilitado']) else           # retorne true se o botão estiver desabilitado,
false button.instate(['!disabled'])             # retorna verdadeiro se o botão não estiver
desativado, else false botão .instate(['!        # executa 'cmd' se o botão não estiver
disabled']), cmd) desativado                  #
```

Observe que esses comandos aceitam uma **matriz** de sinalizadores de estado como argumento.

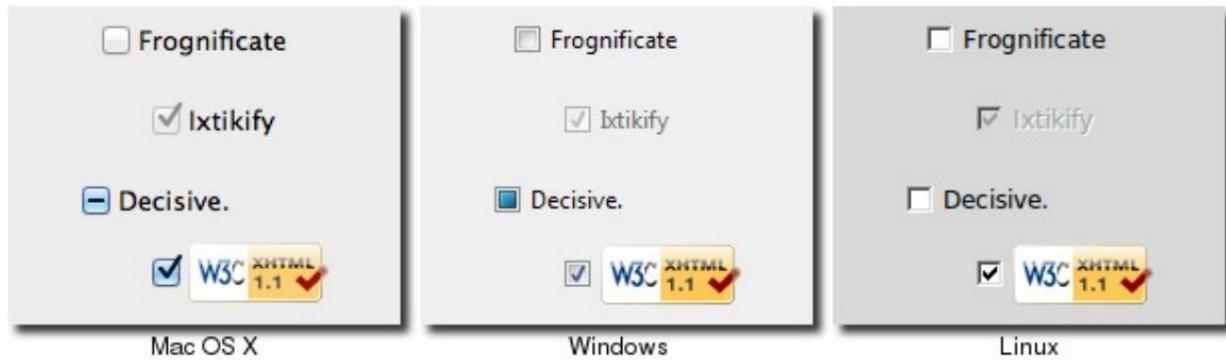
O uso de "state"/"instate" substitui a opção de configuração "state" mais antiga (que usava os valores "normal" ou "disabled"). Essa opção de configuração ainda está disponível no Tk 8.5, mas "somente gravação", o que significa que alterar a opção chama o comando "estado" apropriado, mas outras alterações feitas usando o comando "estado" não são refletidas na opção. Isso é apenas por motivos de compatibilidade; você deve alterar seu código para usar o novo vetor de estado.

A lista completa de sinalizadores de estado disponíveis para widgets temáticos é: "ativo", "desativado", "foco", "pressionado", "selecionado", "plano de fundo", "somente leitura", "alternativo" e "inválido". Eles são descritos na [referência do widget temático](#); nem todos os estados são significativos para todos os widgets. Também é possível obter fantasia nos métodos "state" e "instate" e especificar vários sinalizadores de estado ao mesmo tempo.

botão de seleção

- [Resumo de widgets](#)
- [Manual de Referência](#)

Um **botão de seleção** é como um botão normal, exceto que o usuário não apenas pode pressioná-lo, o que invocará um retorno de chamada de comando, mas também contém um valor binário de algum tipo (ou seja, uma alternância). Os botões de verificação são usados o tempo todo quando um usuário é solicitado a escolher entre, por exemplo, dois valores diferentes para uma opção.



Widgets de botão de seleção

Checkbuttons são criados usando a função `ttk.Checkbutton` e normalmente configurados ao mesmo tempo:

```
measureSystem = StringVar() check
= ttk.Checkbutton(parent, text='Use Metric', command=metricChanged,
variable=measureSystem, onvalue='metric', offvalue='imperial')
```

Os botões de verificação usam muitas das mesmas opções que os botões normais, mas adicionam mais algumas. As opções "text", "textvariable", "image" e "compound" controlam a exibição do rótulo (ao lado da própria caixa de seleção), e os métodos "state" e "instate" permitem que você manipule o "disabled" sinalizador de estado para habilitar ou desabilitar o botão de seleção. Da mesma forma, a opção "comando" permite especificar um script a ser chamado toda vez que o usuário alternar o botão de seleção, e o método "invocar" também executará o mesmo retorno de chamada.

Valor do widget

Ao contrário dos botões, os botões de verificação também possuem um valor. Vimos antes como a opção "textvariable" pode ser usada para vincular o rótulo de um widget a uma variável em seu programa; a opção "variável" para botões de seleção se comporta de maneira semelhante, exceto que é usada para ler ou alterar o valor atual do widget e atualiza sempre que o widget é alterado. Por padrão, os botões de seleção usam um valor de "1" quando o widget está marcado e "0" quando não está marcado, mas eles podem ser alterados para praticamente qualquer coisa usando as opções "onvalue" e "offvalue".

O que acontece quando a variável vinculada não contém nem o valor on nem o valor off (ou mesmo não existe)? Nesse caso, o botão de verificação é colocado em um modo especial "tristate" ou indeterminado; às vezes, você verá isso em interfaces de usuário em que a caixa de seleção contém um único traço em vez de

estar vazio ou segurando uma marca de seleção. Quando neste estado, o sinalizador de estado "alternate" é definido, então você pode verificá-lo com o método "instate":

```
check.instate(['alternativo'])
```

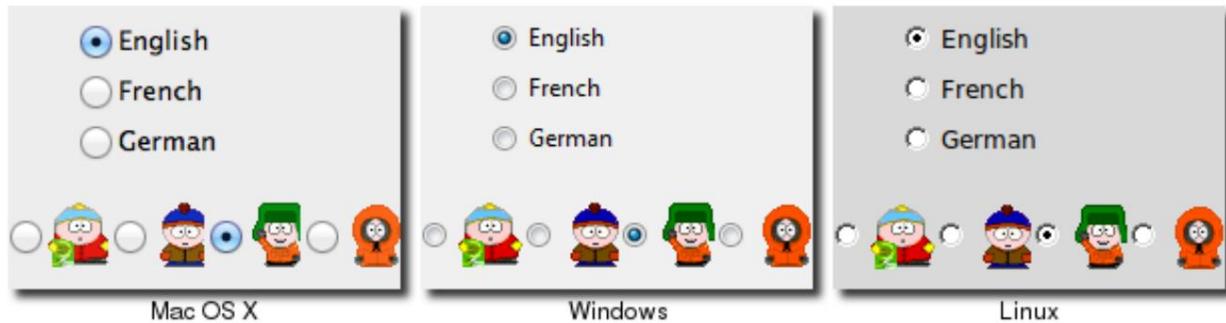
Como o botão de seleção não definirá (ou criará) automaticamente a variável vinculada, seu programa precisa garantir que define a variável com o valor inicial apropriado.

Botão de radio

- [Resumo de widgets](#) •

[Manual de Referência](#)

Um **botão de opção** permite que você escolha entre várias opções mutuamente exclusivas; ao contrário de um botão de seleção, não se limita a apenas duas opções. Os botões de opção são sempre usados juntos em um conjunto e são uma boa opção quando o número de opções é bastante pequeno, por exemplo, 3-5.



Widgets de botão de opção

Radiobuttons são criados usando a função **ttk.Radiobutton** e normalmente como um conjunto:

```
telefone = StringVar()
home = ttk.Radiobutton(pai, text='Casa', variável=telefone, valor='casa')
escritório = ttk.Radiobutton(pai, text='Escritório', variável=telefone, valor=' escritório')
celular = ttk.Radiobutton(parent, text='Celular', variável=telefone, valor='celular')
```

Radiobuttons compartilham a maioria das mesmas opções de configuração que checkbuttons. Uma exceção é que as opções "onvalue" e "offvalue" são substituídas por uma única opção "value". Cada um dos radiobuttons do conjunto terá a mesma variável vinculada, mas um valor diferente; quando a variável tiver o valor dado, o radiobutton será selecionado, caso contrário, desmarcado. Quando a variável vinculada não existe, os botões de opção também exibem um "tristate" ou indeterminado, que pode ser verificado por meio do sinalizador de estado "alternativo".

Entrada

- [Resumo de widgets](#) •

- Manual de Referência

Uma **entrada** apresenta ao usuário um campo de texto de linha única que ele pode usar para digitar um valor de string. Podem ser praticamente qualquer coisa: nome, cidade, senha, número do seguro social e assim por diante.



Widgets de entrada

As entradas são criadas usando a função **ttk.Entry** :

```
nome de usuário = StringVar()
nome = ttk.Entry(pai, textvariable=nome de usuário)
```

Uma opção de configuração de "largura" pode ser especificada para fornecer o número de caracteres que a entrada deve ter, permitindo, por exemplo, fornecer uma entrada mais curta para um CEP ou código postal.

Vimos como os widgets checkbutton e radiobutton têm um valor associado a eles. As entradas também funcionam e esse valor é normalmente acessado por meio de uma variável vinculada especificada pela opção de configuração "textvariable". Observe que, ao contrário dos vários botões, as entradas não possuem um texto ou imagem separada ao lado delas para identificá-las; use um widget de rótulo separado para isso.

Você também pode obter ou alterar o valor do widget de entrada diretamente, sem passar pela variável vinculada. O método "get" retorna o valor atual, e os métodos "delete" e "insert" permitem alterar o conteúdo, por exemplo

```
print('valor atual é %s' % name.get())
# exclui entre dois índices, baseado em 0'someime')#Insere novo texto em um determinado índice
```

Observe que os widgets de entrada não têm uma opção de "comando" que invocará um retorno de chamada sempre que a entrada for alterada. Para observar as alterações, você deve observar as alterações na variável vinculada.

Veja também "Validação", abaixo.

senhas

As entradas podem ser usadas para senhas, onde o conteúdo real é exibido como um marcador ou outro símbolo.

Para fazer isso, defina a opção de configuração "mostrar" para o personagem que você deseja exibir, por exemplo

"*".

Widget States

Como os vários botões, as entradas também podem ser colocadas em um estado desativado por meio do comando "state" (e consultadas com "instate"). As entradas também podem usar o sinalizador de estado "somente leitura"; se definido, usuários

não podem alterar a entrada, embora ainda possam selecionar o texto nela (e copiá-lo para a área de transferência).

Há também um estado "inválido", definido se o widget de entrada falhar na validação, o que nos leva a...

Validação

validar (controla o comportamento geral de validação) - nenhum (padrão), chave (em cada pressionamento de tecla, executa antes - pré-validação), focus/focusin/focusout (executa após... revalidação), todos * script de comando de validação (o script deve retornar 1 ou 0) * script de comando inválido (executado quando o comando validar retorna 0) - várias substituições em scripts. excluir ou modificar -textvariable, o que significa que a edição em andamento é rejeitada em qualquer caso (já que substituiria o que acabamos de definir)

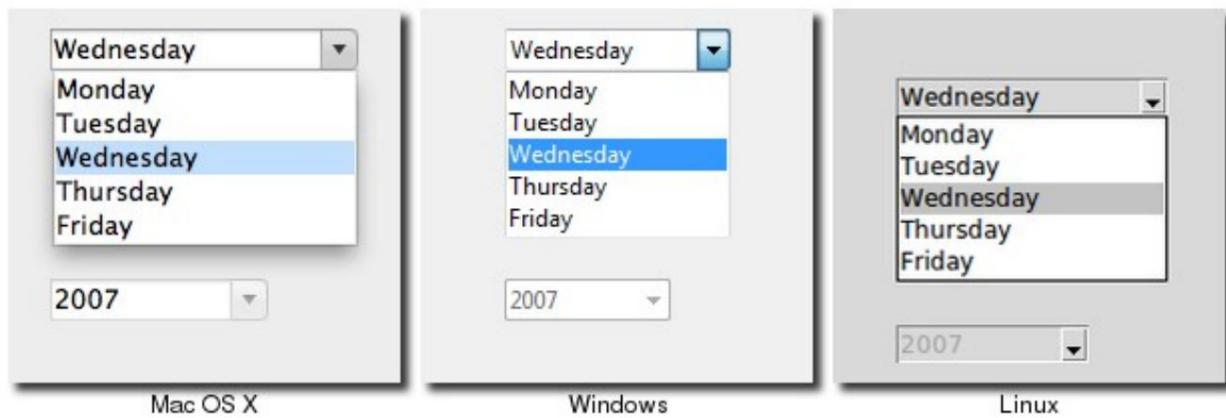
* .e validar para forçar a validação agora

Caixa combo

• [Resumo de widgets](#) •

[Manual de Referência](#)

Uma **caixa de combinação** combina uma entrada com uma lista de opções disponíveis para o usuário. Isso permite que eles escolham um conjunto de valores que você forneceu (por exemplo, configurações típicas), mas também coloque seu próprio valor (por exemplo, para casos menos comuns que você não deseja incluir na lista).



Widgets de caixa de combinação

As caixas de combinação são criadas usando a função **ttk.Combobox** :

```
countryvar = StringVar()
país =
ttk.Combobox(pai, textvariable=countryvar)
```

Assim como as entradas, a opção "textvariable" vincula uma variável em seu programa ao valor atual da caixa de combinação.

Como em outros widgets, você deve inicializar a variável vinculada em seu próprio código.

Você também pode obter o valor atual usando o método "get" e alterar o valor atual usando o método "set" (que usa um único argumento, o novo valor).

Uma caixa de combinação gerará um evento virtual "<ComboboxSelected>" ao qual você pode vincular sempre que seu valor for alterado.

```
country.bind('<<ComboboxSelected>>', função)
```

Valores predefinidos

Você pode fornecer uma lista de valores que o usuário pode escolher usando a opção de configuração "valores":

```
país['valores'] = ('EUA', 'Canadá', 'Austrália')
```

Se definido, o sinalizador de estado "somente leitura" restringirá o usuário a fazer escolhas apenas na lista de valores predefinidos, mas não será capaz de inserir seus próprios valores (embora, se o valor atual da caixa de combinação não estiver na lista, não será possível ser alterado).

Se você estiver usando a caixa de combinação no modo "somente leitura", recomendo que, quando o valor for alterado (ou seja, em um evento ComboboxSelected), você chame o método "limpar seleção". Parece um pouco estranho visualmente sem fazer isso.

Como complemento aos métodos "get" e "set", você também pode usar o método "current" para determinar qual item na lista de valores predefinidos está selecionado (chamar "current" sem argumentos, ele retornará um valor baseado em 0 index na lista ou -1 se o valor atual não estiver na lista) ou selecione um dos itens na lista (chame "atual" com um único argumento de índice baseado em 0).

Deseja associar algum outro valor a cada item da lista, para que seu programa possa se referir a algum valor significativo real, mas seja exibido na caixa de combinação como outra coisa? Você vai querer dar uma olhada na seção intitulada "Mantendo dados de itens extras" quando chegarmos à discussão sobre caixas de listagem em alguns capítulos a partir de agora.

O gerenciador de geometria da grade

Faremos uma pausa ao falar sobre diferentes widgets (o que colocar na tela) e focaremos no gerenciamento de geometria (onde colocá-lo). Apresentamos a ideia geral de gerenciamento de geometria no capítulo "Tk Concepts"; aqui, nos concentramos em um gerenciador de geometria específico: grade.

Como você viu, a grade permite que você organize os widgets em colunas e linhas. Se você estiver familiarizado com o uso de tabelas HTML para fazer layout, você se sentirá em casa aqui. Este capítulo descreve as várias maneiras de ajustar a grade para fornecer todo o controle necessário para a interface do usuário.

O Grid é um dos vários gerenciadores de geometria disponíveis no Tk, mas sua combinação de poder, flexibilidade e facilidade de uso, juntamente com seu ajuste natural aos layouts atuais (que dependem do alinhamento de widgets) o tornam a melhor escolha para uso geral. Existem outros gerenciadores de geometria: "pack" também é bastante poderoso, mas mais difícil de usar e entender; "lugar" dá a você controle total do posicionamento de cada elemento; veremos que até mesmo widgets como janelas panorâmicas, cadernos, telas e texto podem atuar como gerenciadores de geometria.

O Grid foi apresentado ao Tk pela primeira vez em 1996, vários anos depois que o Tk se tornou popular, e demorou um pouco para

pegar. Antes disso, os desenvolvedores sempre usaram "pack" para fazer o gerenciamento de geometria baseado em restrições. Quando o grid foi lançado, muitos desenvolvedores continuaram usando pack, e você ainda o encontrará usado em muitos programas e documentações do Tk. Embora não haja nada tecnicamente errado com ele, o comportamento do algoritmo costuma ser difícil de entender. Mais importante, como a ordem em que os widgets são compactados é significativa na determinação do layout, modificar os layouts existentes pode ser mais difícil.

O Grid tem todo o poder do pacote, geralmente produz layouts mais agradáveis (porque facilita o alinhamento de widgets horizontal e verticalmente) e é mais fácil de aprender e usar. Por causa disso, achamos que o grid é a escolha certa para a maioria dos desenvolvedores na maioria das vezes. Inicie seus novos programas usando grade e alterne os antigos para grade enquanto faz alterações em uma interface de usuário existente.

A [documentação de referência para grade](#) fornece uma descrição exaustiva da grade, seus comportamentos e todas as opções.

Colunas e Linhas

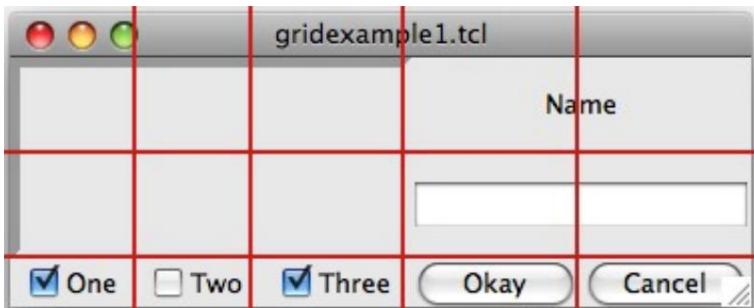
Usando a grade, os widgets recebem um número de "coluna" e um número de "linha", que indica sua posição relativa entre si. Todos os widgets na mesma coluna estarão, portanto, acima ou abaixo uns dos outros, enquanto os da mesma linha estarão à esquerda ou à direita uns dos outros.

Os números das colunas e linhas devem ser inteiros, com a primeira coluna e linha começando em 0. Você pode deixar lacunas nos números das colunas e linhas (por exemplo, coluna 0, 1, 2, 10, 11, 12, 20, 21), o que é útil se você planeja adicionar mais widgets no meio da interface do usuário posteriormente.

A largura de cada coluna (ou altura de cada linha) depende da largura ou altura dos widgets contidos na coluna ou linha. Isso significa que, ao esboçar sua interface de usuário e dividi-la em linhas e colunas, você não precisa se preocupar com a mesma largura de cada coluna ou linha.

Abrangendo várias células Widgets podem ocupar mais de uma única célula na grade; para fazer isso, você usará as opções "columnspan" e "rowspan" ao criar a grade do widget. Estes são análogos aos atributos "colspan" e "rowspan" das tabelas HTML.

Aqui está um exemplo de criação de uma interface de usuário que possui vários widgets, alguns que ocupam mais de uma única célula.



Grid vários widgets

```
da importação tkinter *
da
importação tkinter ttk
```

```
raiz = Tk()
```

```
content = ttk.Frame(root) frame =
ttk.Frame(conteúdo, borderwidth=5, relevo="afundado", largura=200, altura=100) namelbl =
ttk.Label(content, text="Nome") name = ttk.Entry(conteúdo)
```

```
onevar = BooleanVar()
twovar = BooleanVar()
threevar = BooleanVar()
onevar.set(True)
twovar.set(False)
threevar.set(True)
```

```
um = ttk.Checkbutton(conteúdo, text="Um", variável=onevar, onvalue=True) dois =
ttk.Checkbutton(conteúdo, text="Dois", variável=doisvar, onvalue=True) três = ttk.Checkbutton( content,
text="Three", variable=threevar, onvalue=True) ok = ttk.Button(content, text="Ok") cancel = ttk.Button(content,
text="Cancelar")
```

```
content.grid(column=0, row=0)
frame.grid(column=0, row=0, columnspan=3, rowspan=2)
namelbl.grid(column=3, row=0, columnspan=2) nome. grade(coluna=3,
linha=1, extensão de colunas=2) um.grid(coluna=0, linha=3)
dois.grid(coluna=1, linha=3) três.grid(coluna=2, linha=3) ok.grid(coluna=3,
linha=3) cancel.grid(coluna=4, linha=3)
```

```
root.mainloop()
```

Layout dentro da célula

Como a largura de uma coluna (e a altura de uma linha) depende de todos os widgets que foram adicionados a ela, as chances são de que pelo menos alguns widgets tenham uma largura ou altura menor do que a alocada para a célula que foi colocada. Assim, a questão é: onde exatamente ele deve ser colocado dentro da célula?

Por padrão, se uma célula for maior que o widget contido nela, o widget será centralizado nela,

tanto na horizontal quanto na vertical, com o plano de fundo do mestre aparecendo no espaço vazio ao seu redor. A opção "adesiva" pode ser usada para alterar esse comportamento padrão.

O valor da opção "sticky" é uma string de 0 ou mais das direções da bússola "nsew", especificando em quais bordas da célula o widget deve ser "fixado". Assim, por exemplo, um valor de "n" (norte) irá prender o widget contra o lado superior, com qualquer espaço vertical extra na parte inferior; o widget ainda estará centralizado horizontalmente. Um valor de "nw" (noroeste) significa que o widget ficará preso no canto superior esquerdo, com espaço extra na parte inferior e direita.

No Tkinter, você também pode especificar isso como uma lista, contendo qualquer N, S, E e W.

Especificando duas bordas opostas, como "nós" (oeste, leste) significa que o widget será esticado, neste caso, ele ficará preso tanto na borda esquerda quanto na direita. Portanto, o widget será mais largo do que seu tamanho "ideal". A maioria dos widgets tem opções que podem controlar como são exibidos se forem maiores do que o necessário. Por exemplo, um widget de rótulo tem uma opção de "âncora" que controla onde o texto do rótulo será posicionado.

Se você deseja que o widget se expanda para preencher toda a célula, marque-o com um valor fixo de "nsew" (norte, sul, leste, oeste), o que significa que ele ficará preso a todos os lados.

Lidando com o redimensionamento

Se você deu uma olhada abaixo e adicionou as opções "adesivas" extras ao nosso exemplo, ao experimentá-lo, você notará que as coisas ainda não parecem corretas (a entrada é mais baixa na tela do que gostaríamos), e as coisas ficam ainda piores se você tentar redimensionar a janela — nada se move!

Parece que "sticky" pode dizer a Tk *como* reagir se a linha ou coluna da célula for redimensionada, mas na verdade não diz que a linha ou colunas *devem* ser redimensionadas se houver espaço extra disponível. Vamos consertar isso.

Cada coluna e linha tem uma opção de grade de "peso" associada a ela, que informa quanto deve crescer se houver espaço extra no mestre para preencher. Por padrão, o peso de cada coluna ou linha é 0, o que significa que não expande para preencher o espaço.

Para que a interface do usuário seja redimensionada, precisaremos atribuir um peso positivo às colunas que desejamos expandir. Isso é feito usando os métodos "columnconfigure" e "rowconfigure" do grid.

Se duas colunas tiverem o mesmo peso, elas se expandirão na mesma taxa; se um tiver um peso de 1, outro de 3, este último expandirá três pixels para cada pixel adicionado ao primeiro.

Ambos "columnconfigure" e "rowconfigure" também aceitam uma opção de grade "minsize", que especifica um tamanho mínimo que você realmente não deseja que a coluna ou linha diminua além.

Preenchimento

Normalmente, cada coluna ou linha estará diretamente adjacente à próxima, de modo que os widgets estarão corretos

ao lado uns dos outros. Às vezes, isso é o que você deseja (pense em uma caixa de listagem e sua barra de rolagem), mas geralmente você deseja algum espaço entre os widgets. Em Tk, isso é chamado de preenchimento e há várias maneiras de adicioná-lo.

Na verdade, já vimos uma maneira, que é usar as próprias opções de um widget para adicionar espaço extra ao seu redor. Nem todos os widgets têm isso, mas um que tem é um quadro; isso é útil porque os quadros são usados com mais frequência como o mestre para gradear outros widgets. A opção "preenchimento" do quadro permite especificar um pouco de preenchimento extra dentro do quadro, seja a mesma quantidade para cada um dos quatro lados ou até mesmo diferente para cada um.

Uma segunda maneira é usar as opções de grade "padx" e "pady" ao adicionar o widget. Como seria de esperar, "padx" coloca um pouco de espaço extra à esquerda e à direita do widget, enquanto "pady" adiciona espaço extra na parte superior e inferior. Um único valor para a opção coloca o mesmo preenchimento à esquerda e à direita (ou superior e inferior), enquanto uma lista de dois valores permite colocar valores diferentes à esquerda e à direita (ou superior e inferior). Observe que esse preenchimento extra está dentro da célula da grade que contém o widget.

Se você deseja adicionar preenchimento em torno de uma linha ou coluna inteira, os métodos "columnconfigure" e "rowconfigure" aceitam uma opção "pad", que fará isso para você.

Vamos adicionar o comportamento extra de aderência, redimensionamento e preenchimento ao nosso exemplo (acréscimos em negrito).

```
da importação tkinter * da
importação tkinter ttk
```

```
raiz = Tk()
```

```
content = ttk.Frame(raiz, preenchimento=(3,3,12,12)) frame =
ttk.Frame(conteúdo, borderwidth=5, relevo="afundado", largura=200, altura=100) namelbl = ttk.
Label(conteúdo, text="Nome") nome = ttk.Entry(conteúdo)
```

```
uma var = BooleanVar()
duas var = BooleanVar()
tresvar = BooleanVar()
```

```
onevar.set(True)
twovar.set(False)
threevar.set(True)
```

```
um = ttk.Checkbutton(conteúdo, text="Um", variável=onevar, onvalue=True) dois =
ttk.Checkbutton(conteúdo, text="Dois", variável=doisvar, onvalue=True) três = ttk.Checkbutton( content,
text="Three", variable=threevar, onvalue=True) ok = ttk.Button(content, text="Ok") cancel = ttk.Button(content,
text="Cancelar")
```

```
content.grid(column=0, row=0, sticky=(N, S, E, W))
frame.grid(column=0, row=0, columnspan=3, rowspan=2, sticky=(N, S , E, W)) namelbl.grid(coluna=3,
linha=0, columnspan=2, sticky=(N, W), padx=5) name.grid(coluna=3, linha=1, columnspan=2,
sticky=(N, E, W), pady=5, padx=5) one.grid(column=0, row=3) two.grid(column=1, row=3)
three.grid(column=2, linha=3) ok.grid(coluna=3, linha=3) cancelar.grid(coluna=4, linha=3)
```

```

root.columnconfigure(0, peso=1)
root.rowconfigure(0, peso=1)
content.columnconfigure(0, peso=3)
content.columnconfigure(1, peso=3)
content.columnconfigure(2, peso=3)
content.columnconfigure(3, peso=1)
content.columnconfigure(4, peso=1)
content.rowconfigure(1, peso=1)

root.mainloop()

```

Isso parece mais promissor. Brinque com o exemplo para ter uma ideia do comportamento do redimensionamento.



Exemplo de grade, manipulação de layout e redimensionamento na célula.

Você notará o pequeno gadget de redimensionamento na parte inferior direita da janela; enquanto estamos apenas seguindo o caminho mais fácil e evitando-o com o preenchimento extra, veremos mais tarde como melhor levá-lo em consideração usando um widget "sizegrip".

Recursos de grade adicionais

Como você pode ver na [referência da grade](#), há muitas outras coisas que você pode fazer com a grade. Aqui estão alguns dos mais úteis.

Consultando e alterando opções de grade

Como os próprios widgets, é fácil examinar as várias opções de grade, bem como alterá-las; definir as opções quando você gradear o widget pela primeira vez é apenas uma conveniência e certamente você pode alterá-las a qualquer momento que desejar.

O método "escravos" informará todos os widgets que foram agrupados em grade dentro de um mestre ou, opcionalmente, aqueles em apenas uma determinada coluna ou linha. O método "info" fornecerá uma lista de todas as opções de grade para um widget e seus valores. Por fim, o método "configurar" permite alterar uma ou mais opções de grade em um widget.

Estes são ilustrados nesta sessão interativa:

```

>>> content.grid_slaves() <objeto
de mapa em 0x00C3F470> >>> para
w em content.grid_slaves(): print(w)
...
.14597008.14622128
.14597008.14622096
.14597008.14622064
.14597008.14622032
.14597008.14622000
.14597008.14621872
.14597008.14621840
.14597008.14621808
>>> para w em content.grid_slaves(row=3): print(w)
...
.14597008.14622128
.14597008.14622096
.14597008.14622064
.14597008.14622032
.14597008.14622000
>>> para w em content.grid_slaves(column=0): print(w)
...
.14597008.14622000
.14597008.14621808
>>> namelbl.grid_info()
{'rowspan': '1', 'coluna': '3', 'sticky': 'nw', 'ipady': '0', 'ipadx': '0', 'columnspan': '2', 'in': <tkinter.ttk.Frame object
at 0x00DEBB90>, 'pady': '0', 'padx': '5', 'row': '0'} >>> namelbl .grid_configure(sticky=(E,W)) >>>
namelbl.grid_info() {'rowspan': '1', 'column': '3', 'sticky': 'ew', 'ipady': '0', 'ipadx': '0', 'columnspan': '2', 'in': <objeto
tkinter.ttk.Frame em 0x00DEBB90>, 'pady': '0', 'padx': '5', 'linha': '0'}

```

Preenchimento Interno

Você viu como as opções de grade "padx" e "pady" adicionaram espaço extra ao redor de um widget. Há também um tipo de preenchimento menos usado chamado "preenchimento interno", que é controlado pelas opções de grade "ipadx" e "ipady".

A diferença pode ser sutil. Digamos que você tenha um quadro de 20x20 e especifique um preenchimento normal (externo) de 5 pixels em cada lado. O quadro solicitará um retângulo de 20x20 (seu tamanho natural) do gerenciador de geometria. Normalmente, é isso que será concedido, então obterá um retângulo de 20x20 para o quadro, cercado por uma borda de 5 pixels.

Com o preenchimento interno, o gerenciador de geometria adicionará efetivamente o preenchimento extra ao widget ao descobrir seu tamanho natural, como se o widget tivesse solicitado um retângulo de 30x30. Se a moldura estiver centralizada ou anexada a um único lado ou canto (usando "adesivo"), você terá uma moldura de 20x20 com espaço extra ao redor dela. Se, no entanto, o quadro for definido para esticar (ou seja, um valor "fixo" de "nós", "ns" ou "nwes"), ele preencherá o espaço extra, resultando em um quadro de 30x30, sem borda.

Esquecer e Remover

O método "esquecer" da grade, tomando como argumentos uma lista de um ou mais widgets escravos, pode ser

usado para remover escravos da rede da qual eles fazem parte. Isso não destrói o widget completamente, mas o tira da tela, como se não tivesse sido gradeado em primeiro lugar. Você pode gravá-lo novamente mais tarde, embora todas as opções de grade que você atribuiu originalmente tenham sido perdidas.

O método "remover" da grade funciona da mesma forma, exceto que as opções da grade serão lembradas.

Layouts aninhados À

medida que sua interface de usuário fica mais complicada, a grade que você está usando para organizar todos os seus widgets pode ficar cada vez mais complicada e mais refinada. Isso pode dificultar muito a mudança e a manutenção do seu programa.

Felizmente, você não precisa gerenciar toda a interface do usuário com uma única grade. Se você tiver uma área de sua interface de usuário bastante independente das outras, crie um novo quadro para conter essa área e coloque em grade os widgets que fazem parte dessa área dentro desse quadro. Se você tiver algum tipo de programa gráfico com várias paletas, barras de ferramentas e assim por diante, cada uma dessas áreas pode ser candidata a ser colocada em seu próprio quadro.

Em teoria, esses quadros, cada um com sua própria grade, podem ser aninhados arbitrariamente profundos, embora na prática isso geralmente não vá além de alguns níveis. Isso pode ser uma grande ajuda na modularização do seu programa. Se, por exemplo, você tiver uma paleta de ferramentas de desenho, poderá criar tudo em um procedimento separado, incluindo a criação de todos os widgets de componentes, agrupando-os em grade, configurando associações de eventos e assim por diante. Do ponto de vista de seu programa principal, tudo o que ele precisa ver é o widget de quadro único que contém tudo.

Nossos exemplos mostraram apenas uma sugestão disso, onde um quadro de conteúdo foi gradeado na janela principal e, em seguida, todos os outros widgets foram gradeados no quadro de conteúdo.

Mais widgets

Este capítulo apresenta vários outros widgets: caixa de listagem, barra de rolagem, texto, barra de progresso, escala e caixa de rotação. Alguns deles estão começando a ser um pouco mais poderosos do que os básicos que vimos antes. Aqui também veremos algumas instâncias de uso dos widgets Tk clássicos, em instâncias em que não há (ou não há necessidade de) uma contraparte temática.

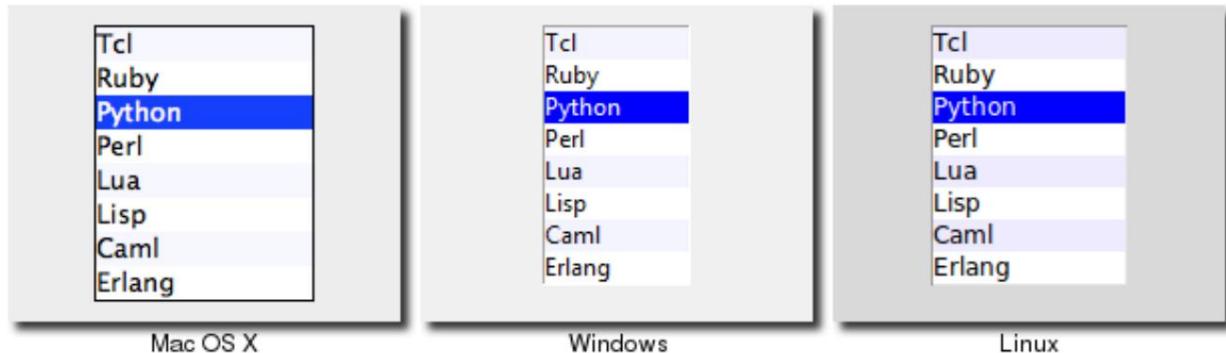
Caixa de listagem

- [Resumo de widgets](#) •
- [Manual de Referência](#)

Uma **caixa de listagem** exibe uma lista de itens de texto de uma linha, geralmente longa, e permite que o usuário navegue pela lista, selecionando um ou mais.

As caixas de listagem fazem parte dos widgets Tk clássicos; atualmente não há uma caixa de listagem no widget temático Tk definir.

O widget treeview do Tk (que é temático) também pode ser usado como uma caixa de listagem (uma árvore de um nível de profundidade), permitindo que você use ícones e estilos com a lista. Também é provável que um widget de lista de várias colunas (tabela) se transforme em Tk em algum momento, com base em uma das extensões disponíveis.



Widgets da caixa de listagem

Listboxes são criadas usando a função **Listbox** :

I = Listbox(pai, altura=10)

Preenchendo os itens da caixa de listagem

Há uma maneira fácil e uma maneira difícil de preencher e gerenciar todos os itens contidos na caixa de listagem.

Aqui está o caminho mais fácil. Cada listbox possui uma opção de configuração "listvariable", que permite vincular uma variável (que deve conter uma lista) ao listbox. Cada elemento desta lista é uma string que representa um item na caixa de listagem. Portanto, para adicionar, remover ou reorganizar itens na caixa de listagem, você pode simplesmente manipular essa variável como faria com qualquer outra lista. Da mesma forma, para descobrir qual item está na terceira linha da caixa de listagem, basta olhar para o terceiro elemento da variável de lista.

A razão pela qual existe uma maneira difícil é porque a opção "listvariable" foi introduzida apenas no Tk 8.3. Antes disso, você estava preso ao caminho mais difícil. Como usar a variável de lista permite que você use todas as operações de lista padrão, ela fornece uma API muito mais simples e certamente é uma atualização que vale a pena considerar se você tiver caixas de listagem fazendo as coisas da maneira antiga.

A maneira mais antiga e difícil de fazer as coisas é usar um conjunto de métodos que fazem parte do próprio widget de caixa de listagem que operam na lista (interna) de itens:

- O campo "*inserir idx item ?item... ?*" método é usado para adicionar um ou mais itens a lista; "*idx*" é um índice baseado em 0 que indica a posição do item antes do qual o(s) item(ns) deve(m) ser adicionado(s); especifique "*end*" para colocar os novos itens no final da lista.
- Use a opção "*excluir primeiro?último?*" método para excluir um ou mais itens da lista;
- "first" e "last" são índices de acordo com o método "insert".
- Use o "*obter primeiro?último?*" método para retornar o conteúdo de um único item no determinada posição, ou uma lista dos itens entre "*primeiro*" e "*último*".
- O método "tamanho" retorna o número de itens da lista.

Selecionando Itens

A primeira coisa que você precisa decidir é se é possível para o usuário selecionar apenas um único item por vez ou se vários itens podem ser selecionados simultaneamente. Isso é controlado pela opção "modo de seleção": o padrão é apenas poder selecionar um único item ("navegar"), enquanto um modo de seleção "estendido" permite ao usuário selecionar vários itens.

Os nomes "navegar" e "estendido", novamente por motivos de compatibilidade com versões anteriores, são realmente horríveis. Isso é agravado pelo fato de que existem dois outros modos, "único" e "múltiplo" que você **não deve usar** (eles usam um estilo de interação antigo que é inconsistente com a interface de usuário moderna e as convenções de plataforma).

Para saber qual item ou itens do listbox o usuário selecionou no momento, utilize o método "curselection", que retornará a lista de índices de todos os itens selecionados no momento; esta pode ser uma lista vazia e, para listas com um modo de seleção de "browse", nunca será maior que um item. Você também pode usar o método "seleção inclui *índice*" para verificar se o item com o índice fornecido está selecionado no momento.

Para alterar programaticamente a seleção, você pode usar a opção "seleção limpar *primeiro? último?*" para desmarcar um único item ou qualquer um dentro do intervalo de índices especificados. Para selecionar um item ou todos os itens em um intervalo, use a opção "seleção definir *primeiro? último?*" método.

Ambos não afetarão a seleção de nenhum item fora do intervalo especificado.

Se você alterar a seleção, também deverá certificar-se de que o item recém-selecionado esteja visível para o usuário (ou seja, não seja rolado para fora da visualização). Para fazer isso, use o método "ver *índice*".

Quando a seleção é alterada pelo usuário, um evento virtual "<ListboxSelect>" é gerado.

Você pode se vincular a isso para realizar qualquer ação necessária. Dependendo do seu aplicativo, você também pode querer vincular a um evento "Double-1" de clique duplo e usá-lo para invocar uma ação com o item atualmente selecionado.

Estilizando a lista Como

a maioria dos widgets Tk "clássicos", você tem imensa flexibilidade para modificar a aparência de uma caixa de listagem.

Conforme descrito no [manual de referência](#), você pode modificar a fonte em que os itens da caixa de listagem são exibidos, as cores de primeiro plano (texto) e de fundo dos itens em seu estado normal, quando selecionados, quando o widget está desativado e assim por diante. Há também um método "itemconfigure" que permite alterar as cores de primeiro plano e de fundo de itens individuais.

Como costuma acontecer, a moderação é útil. Geralmente, os valores padrão serão totalmente adequados e uma boa correspondência para as convenções da plataforma. No exemplo ao qual chegaremos momentaneamente, mostraremos como o uso restrito dessas opções pode ter bons resultados, neste caso exibindo linhas alternadas da caixa de listagem em cores ligeiramente diferentes.

Mantendo dados de itens extras

A "variável de lista" (ou a lista interna, se você estiver gerenciando as coisas da maneira antiga) contém o

strings que serão mostradas na caixa de listagem. Muitas vezes, porém, cada string em seu programa está associada a algum outro item de dados, e o que realmente interessa não é tanto a string exibida na caixa de listagem, mas o item de dados associado. Por exemplo, uma caixa de listagem contendo nomes pode ser o que é apresentado ao usuário, mas seu programa está realmente interessado no objeto do usuário (ou número de id) que está selecionado, não no nome específico.

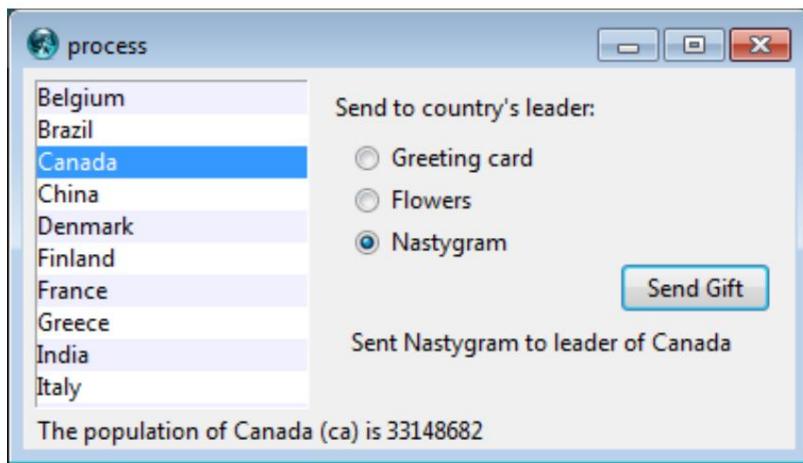
Como podemos associar este valor ao nome exibido? Infelizmente, o widget listbox em si não oferece nenhuma facilidade, então é algo que teremos que administrar separadamente. Existem algumas abordagens óbvias. Primeiro, se as strings exibidas forem únicas garantidas, você pode usar uma tabela de hash para mapear os nomes para o objeto associado. Assim, dado o nome, você pode obter facilmente o objeto associado. Isso provavelmente não funcionaria bem para fazer nomes, mas poderia funcionar para escolher países, por exemplo.

Uma segunda abordagem é manter uma segunda lista, paralela à lista de strings exibidas na caixa de listagem, que conterá os objetos associados. Assim, o primeiro item na lista de strings corresponde ao primeiro item na lista de objetos, o segundo ao segundo e assim por diante. Quaisquer alterações feitas em uma lista (inserir, excluir, reordenar) precisam ser feitas na outra. Você pode mapear facilmente do item da lista para o objeto subjacente, com base em sua posição na lista.

Exemplo

Aqui está um exemplo bobo mostrando essas várias técnicas de caixa de listagem. Teremos uma lista de países exibidos. Nos bastidores, temos um banco de dados (uma simples tabela hash) que contém a população de cada país, indexada pelo código de duas letras do país. Seremos capazes de selecionar apenas um único país por vez e, ao fazê-lo, uma barra de status exibirá a população do país.

Clicar duas vezes na lista ou pressionar a tecla Enter enviará um dos vários presentes para o chefe de estado do país selecionado (bem, na verdade não, mas use sua imaginação).



Exemplo de caixa de listagem do seletor de país

```
from tkinter import * from
tkinter import ttk root = Tk()
```

```
# Inicialize nossos "banhos de dados" de países:
# - a lista de códigos de países (um subconjunto de qualquer
maneira) # - uma lista paralela de nomes de países, na mesma ordem dos códigos de países
```

```

# - uma tabela hash que mapeia o código do país para a população< countrycodes =
('ar', 'au', 'be', 'br', 'ca', 'cn', 'dk', 'fi', 'fr', 'gr', 'in', 'it', 'jp', 'mx', 'nl', 'no', 'es', 'se', 'ch') countrynames = ('Argentina', 'Australia' ,
'Bélgica', 'Brasil', 'Canadá', 'China', 'Dinamarca', \
'Finlândia', 'França', 'Grécia', 'Índia', 'Itália', 'Japão', 'México',
'Holanda', 'Noruega', 'Espanha', \
'Suécia', 'Suíça') cnames =
StringVar(value=countrynames) populações = {'ar':41000000,
'au':21179211, 'be':10584534, 'br':185971537, \ 'ca':33148682, 'cn':1323128240, 'dk':5457415, 'fi':5302000, 'fr':64102140,
'gr':11147000, \ 'in':1131043000, 'it':59206382, 'jp':127718000, 'mx':106535000, 'nl':16402414, \ 'no':4738085,
'es':45116894, 'se':9174082, 'ch':7508700}

# Nomes dos presentes que podemos enviar=
{ 'cartão':'Cartão comemorativo', 'flores':'Flores', 'nastygram':'Nastygram'}}

# Variáveis de estado gift =
StringVar() sentmsg =
StringVar() statusmsg = StringVar()

# Chamado quando a seleção na caixa de listagem muda; descubra # qual país está selecionado no momento e, em seguida, procure o código do país # e, a partir disso, sua população. Atualize a mensagem de status # com a nova população. Além disso, limpe a mensagem sobre o # presente que está sendo enviado, para que ele não fique por perto depois que começarmos a fazer # outras coisas. def showPopulation(*args):

idxs = lbox.curselection() if len(idxs)==1:
    idx = int(idxs[0]) code = countrycodes[idx]
    name = countrynames[idx] popn =
    populations[code] statusmsg.set("A
    população de %s (%s) é %d" % (nome,
    código, popn)) sentmsg.set(")

# Chamado quando o usuário clica duas vezes em um item na caixa de listagem, pressiona # o botão "Enviar
    presente" ou pressiona a tecla Return. Caso o # item selecionado esteja fora de vista, verifique se ele está visível. # #
    Descubra qual país foi selecionado, qual presente foi selecionado com os # botões de opção, "enviar o presente" e fornecer feedback de que foi enviado. def sendGift(*args): idxs = lbox.curselection() if len(idxs)==1: idx = int(idxs[0]) lbox.see(idx)
    name = countrynames[idx]

# Envio de presente deixado como exercício para o leitor sentmsg.set("Enviou
    %s para líder de %s" % (gifts[gift.get()], nome))

# Crie e grade o quadro de conteúdo externo c = ttk.Frame(root,
padding=(5, 5, 12, 0)) c.grid(column=0, row=0, sticky=(N,W,E,S ))
root.grid_columnconfigure(0, peso=1) root.grid_rowconfigure(0,peso=1)

```

```

# Crie os diferentes widgets; observe as variáveis às quais muitos # deles estão
vinculados, bem como o retorno de chamada do botão.
# Observe que estamos usando StringVar() 'cnames', construído a partir de 'countrynames' lbox =
Listbox(c, listvariable=cnames, height=5) lbl = ttk.Label(c, text="Enviar para o líder do país:") g1 =
ttk.Radiobutton(c, text=presentes['cartão'], variável=presente, valor='cartão') g2 = ttk.Radiobutton(c,
texto=presentes['flores'], variável=presente, valor ='flores') g3 = ttk.Radiobutton(c, text=gifts['nastygram'],
variável=gift, value='nastygram') send = ttk.Button(c, text='Enviar presente', command=sendGift ,
default='active') sentlbl = ttk.Label(c, textvariable=sentmsg, âncora='center') status = ttk.Label(c,
textvariable=statusmsg, âncora=W)

# Grade todos os widgets
lbox.grid(column=0, row=0, rowspan=6, sticky=(N,S,E,W)) lbl.grid(column=1,
row=0, padx=10, pady =5) g1.grid(column=1, linha=1, sticky=W, padx=20)
g2.grid(column=1, linha=2, sticky=W, padx=20) g3.grid(column=1 , linha=3,
sticky=W, padx=20) send.grid(column=2, row=4, sticky=E)
sentlbl.grid(column=1, row=5, columnspan=2, sticky=N, pady =5, padx=5)
status.grid(column=0, row=6, columnspan=2, sticky=(W,E))
c.grid_columnconfigure(0, peso=1) c.grid_rowconfigure(5, peso=1 )

# Definir ligações de evento para quando a seleção na caixa de listagem mudar, # quando o
usuário clicar duas vezes na lista e quando pressionar a tecla Return lbox.bind('<<ListboxSelect>>',
showPopulation) lbox.bind('<Double -1>', sendGift) root.bind('<Return>', sendGift)

# Colorize linhas alternadas da caixa de listagem para i in
range(0,len(countrynames),2):
    lbox.itemconfigure(i, background='#f0f0ff')

# Defina o estado inicial da interface, incluindo selecionar o # presente padrão a ser enviado
e limpar as mensagens. Selecione o primeiro # país da lista; como o evento <<ListboxSelect>>
só é # gerado quando o usuário faz uma alteração, chamamos explicitamente showPopulation.
gift.set('card') sentmsg.set("") statusmsg.set("") lbox.selection_set(0) showPopulation()

root.mainloop()

```

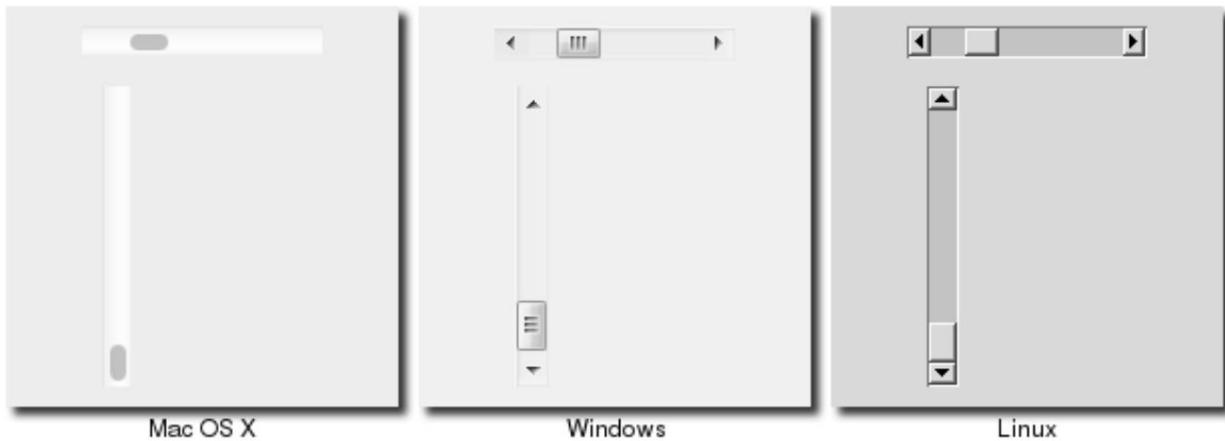
Uma coisa óbvia que faltou neste exemplo foi que, embora a lista de países pudesse ser bastante longa, apenas parte dela cabia na tela de uma só vez. Para mostrar os países mais abaixo na lista, você tinha que arrastar com o mouse ou usar a tecla de seta para baixo. Uma barra de rolagem teria sido legal. Vamos consertar isso.

Barra de rolagem

- [Resumo de widgets](#)

- [Manual de Referência](#)

Uma **barra de rolagem** ajuda o usuário a ver todas as partes de outro widget, cujo conteúdo normalmente é muito maior do que pode ser mostrado no espaço de tela disponível.



Widgets da barra de rolagem

As barras de rolagem são criadas usando o comando **ttk.Scrollbar** :

```
s = ttk.Scrollbar( pai, orient=VERTICAL, command=listbox.yview)
listbox.configure(yscrollcommand=s.set)
```

Ao contrário de alguns kits de ferramentas, as barras de rolagem não fazem parte de outro widget (por exemplo, uma caixa de listagem), mas são um widget separado. Em vez disso, as barras de rolagem se comunicam com o widget rolado chamando métodos no widget rolado; Acontece que o widget com rolagem também precisa chamar métodos na barra de rolagem.

Se você estiver usando uma distribuição recente do Linux, provavelmente notou que as barras de rolagem que você vê em muitos aplicativos mudaram para se parecer mais com o que você veria no Mac OS X. Essa nova aparência ainda não é suportada no Tk.

A opção de configuração "orientar" das barras de rolagem determina se será utilizada para rolar na direção "horizontal" ou "vertical". Em seguida, você precisa definir a opção de configuração "comando" para se comunicar com o widget rolado. Este precisa ser o método para chamar o widget de rolagem.

Cada widget que pode ser rolado verticalmente inclui um método chamado "yview" (aqueles que podem ser rolados horizontalmente têm um método chamado "xview"). Enquanto este método estiver presente, a barra de rolagem não precisa saber mais nada sobre o widget rolado. Quando a barra de rolagem for manipulada, ela adicionará alguns parâmetros à chamada do método, indicando como ela foi rolada, em que posição, etc.

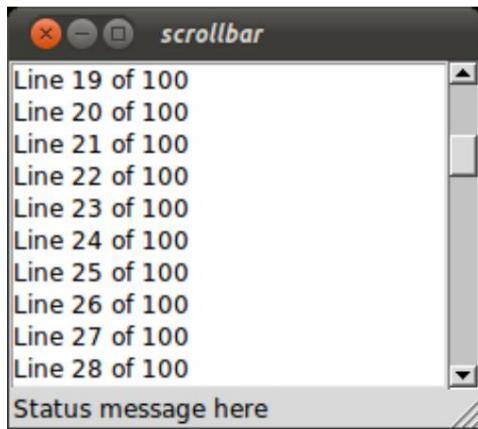
O widget rolado também precisa se comunicar de volta com a barra de rolagem, informando qual porcentagem de toda a área de conteúdo agora está visível. Além dos métodos yview e/ou xvview, todo widget rolável também possui uma opção de configuração "yscrollcommand" e/ou "xscrollcommand". Isso é usado para especificar uma chamada de método, que deve ser o método "set" da barra de rolagem. Mais uma vez, adicional

os parâmetros serão adicionados automaticamente à chamada do método.

Se por algum motivo você quiser mover a barra de rolagem para uma posição específica dentro do seu programa, você mesmo pode chamar o método "definir o *primeiro* por *último*". Passe dois valores entre 0 e 1 indicando a porcentagem inicial e final da área de conteúdo que está visível.

Exemplos de

caixas de listagem são um dos vários tipos de widgets que podem ser rolados. Aqui vamos construir uma interface de usuário muito simples, consistindo apenas em uma caixa de listagem rolável verticalmente que ocupa toda a janela, com apenas uma linha de status na parte inferior.



Rolar uma caixa de listagem

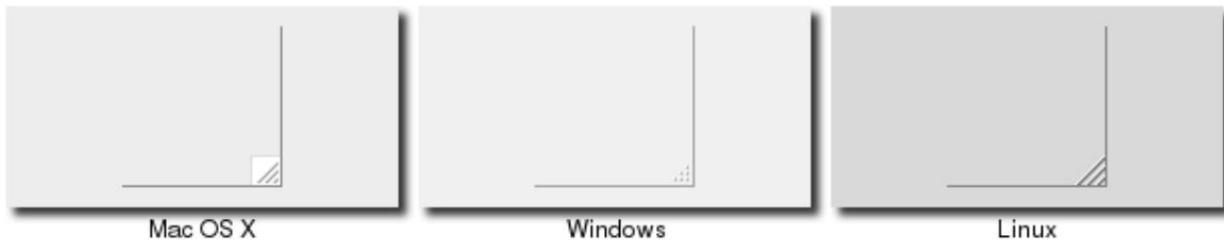
```
da importação tkinter *
da
importação ttk
```

```
root = Tk() l =
Listbox(root, height=5) l.grid(column=0,
row=0, sticky=(N,W,E,S)) s = ttk.Scrollbar(root,
orient=VERTICAL , command=l.yview) s.grid(column=1, row=0, sticky=(N,S))
l['yscrollcommand'] = s.set
ttk.Sizegrip().grid(column=1, row =1, sticky=(S,E))
root.grid_columnconfigure(0, peso=1) root.grid_rowconfigure(0, peso=1) for i
in range(1,101): l.insert('end', 'Line % d de 100' % i)
root.mainloop()
```

SizeGrip

- [Resumo de widgets •](#)
- [Manual de Referência](#)

Na verdade, introduzimos um novo widget nesse último exemplo, o **sizegrip**. Esta é a pequena caixa no canto inferior direito da janela que permite redimensioná-la.



Widgets SizeGrip

SizeGrips são criados usando a função `ttk.Sizegrip` :

```
ttk.Sizegrip(parent).grid(coluna=999, linha=999, sticky=(S,E))
```

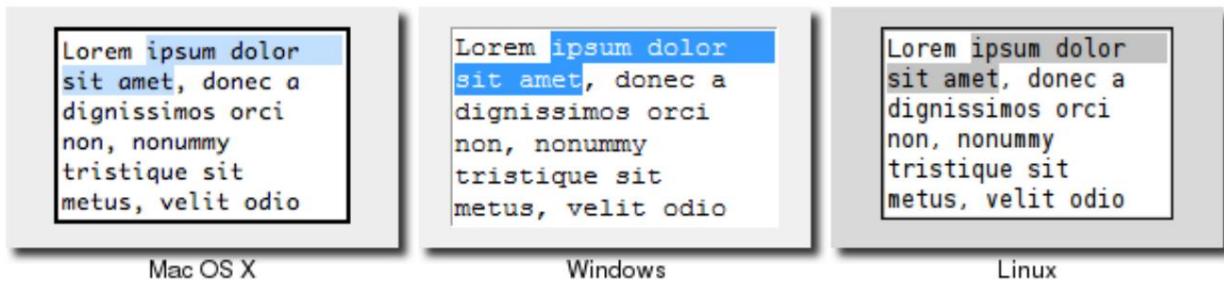
Embora você perceba que em algumas plataformas (por exemplo, Mac OS X), o Tk colocará automaticamente a alça de tamanho lá para você, não custa nada adicioná-la explicitamente você mesmo. Discutiremos como alterar o tamanho da janela, determinar se ela é redimensionável etc. em um capítulo posterior.

As convenções de plataforma tendem a evoluir mais rapidamente do que os kits de ferramentas GUI de software livre de longa duração. O Mac OS X 10.7 eliminou a alça de tamanho no canto, permitindo o redimensionamento de qualquer borda da janela, finalmente alcançando o resto do mundo. Portanto, é melhor verificar a versão do sistema operacional que você está executando antes de adicionar o sizegrip.

Texto

- [Resumo de widgets •](#)
- [Manual de Referência](#)

Um widget **de texto** fornece aos usuários uma área para que eles possam inserir várias linhas de texto. Os widgets de texto fazem parte dos widgets Tk clássicos, não dos widgets Tk temáticos.



Widgets de texto

O widget de texto do Tk é, juntamente com o widget de tela, um dos dois widgets superpoderosos que fornecem recursos incrivelmente profundos, mas facilmente programáveis. Os widgets de texto formaram a base para processadores de texto completos, esboços, navegadores da web e muito mais. Abordaremos algumas das coisas avançadas em um capítulo posterior, mas aqui mostraremos apenas o que você precisa para usar o widget de texto para capturar entradas de texto bastante simples e com várias linhas.

Widgets de texto são criados usando a função **Texto** :

`t = Texto(pai, largura=40, altura=10)`

As opções "largura" e "altura" especificam o tamanho de tela solicitado do widget de texto, em caracteres e linhas, respectivamente. O conteúdo do texto pode ser arbitrariamente grande. Você pode usar a opção de configuração "wrap" para controlar como a quebra de linha é tratada: os valores são "none" (sem quebra automática, o texto pode rolar horizontalmente), "char" (quebra em qualquer caractere) e "palavra" (a quebra só ocorrem em limites de palavras).

Um widget de texto pode ser desabilitado para que nenhuma edição ocorra; como o texto não é um widget temático, os métodos usuais "state" e "instate" não estão disponíveis. Em vez disso, use a opção de configuração "estado", definindo-a como "desativada" ou "normal".

A rolagem funciona da mesma forma que nas caixas de listagem. As opções de configuração "xscrollcommand" e "yscrollcommand" estão disponíveis para anexar o widget de texto a barras de rolagem horizontais e/ou verticais, e os métodos "xview" e "yview" estão disponíveis para serem chamados a partir das barras de rolagem. Para garantir que uma determinada linha esteja visível (ou seja, não rolada para fora da visualização), você pode usar o método "ver índice", onde o índice está no formato "*número da linha.número do caractere*", por exemplo, "5.0" para o primeiro (0 -baseado) caractere da linha 5 (baseado em 1).

Conteúdo

Os widgets de texto não possuem uma variável vinculada associada a eles, como, por exemplo, os widgets de entrada. Para recuperar o conteúdo de texto de todo o widget, chame o método "get 1.0 end"; o "1.0" é um índice no texto, e significa o primeiro caractere da primeira linha, e "end" é um atalho para o índice do último caractere, última linha. Outros índices podem ser passados para recuperar intervalos menores de texto, se necessário.

O texto pode ser adicionado ao widget usando o método "inserir *string de índice*"; novamente *index* está no formato "*line.char*" e marca o caractere antes do qual o texto é inserido; use "end" para adicionar texto ao final do widget. Você pode excluir um intervalo de texto usando o método "excluir *início e fim*", onde o *início* e o *fim* são índices de texto, conforme já descrito.

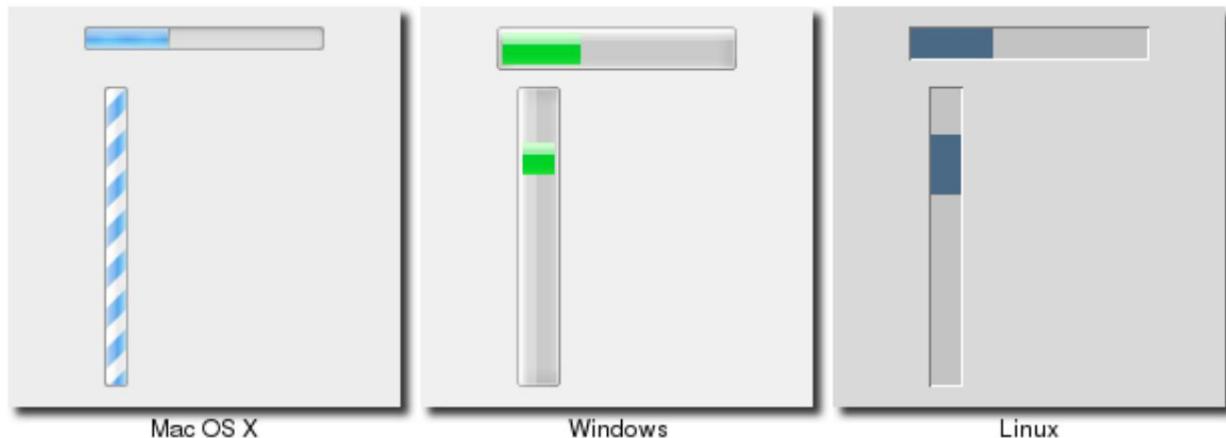
Veremos os muitos recursos avançados adicionais do widget de texto em um capítulo posterior.

Barra de progresso

- [Resumo de widgets](#)
- [Manual de Referência](#)

Um widget de **barra de progresso** fornece uma maneira de fornecer feedback ao usuário sobre o progresso de uma operação demorada. Isso pode ser feito como uma exibição de porcentagem concluída para operações em que isso pode ser estimado ou uma exibição que muda para indicar que a operação continua, mas sem um

estimativa de conclusão.



Widgets da barra de progresso

Widgets Progressbar são criados usando a classe **ttk.Progressbar** :

```
p = ttk.Progressbar(parent, orient=HORIZONTAL, length=200, mode='determinate')
```

A opção "orientar" pode ser "horizontal" ou "vertical". A opção "comprimento", que representa o eixo mais longo das barras de progresso horizontal ou vertical, é especificada em unidades de tela (por exemplo, pixels). A opção de configuração "modo" pode ser definida como "determinado", onde a barra de progresso indicará o progresso relativo em direção à conclusão, ou como "indeterminado", onde não é possível saber em que ponto da tarefa o programa está, mas ainda assim deseja fornecer feedback de que as coisas ainda estão funcionando.

Progresso determinado No

modo determinado, você pode fornecer feedback mais ou menos preciso ao usuário sobre o progresso de uma operação. Para fazer isso, você precisa primeiro informar à barra de progresso quantos "passos" a operação levará e, à medida que avança, informar à barra de progresso o andamento da operação.

Você pode fornecer o número total de etapas para a barra de progresso usando a opção de configuração "máximo"; este é um número de ponto flutuante cujo padrão é 100 (ou seja, cada passo é 1%). Para informar à barra de progresso quanto tempo você está na operação, você alterará repetidamente a opção de configuração "valor". Portanto, isso começaria em 0 e, em seguida, contaria para cima até o valor máximo que você definiu. Existem duas pequenas variações sobre isso. Primeiro, você pode apenas armazenar o valor atual da barra de progresso em uma variável vinculada a ela pela opção de configuração "variável" da barra de progresso; dessa forma, quando você alterar a variável, a barra de progresso será atualizada. A outra alternativa é chamar o "step ?amount?" da barra de progresso. método para incrementar o valor pelo "valor" fornecido (o padrão é 1,0).

Progresso Indeterminado

O modo indeterminado é para quando você não é capaz de saber facilmente (ou estimar) a que distância em um longo

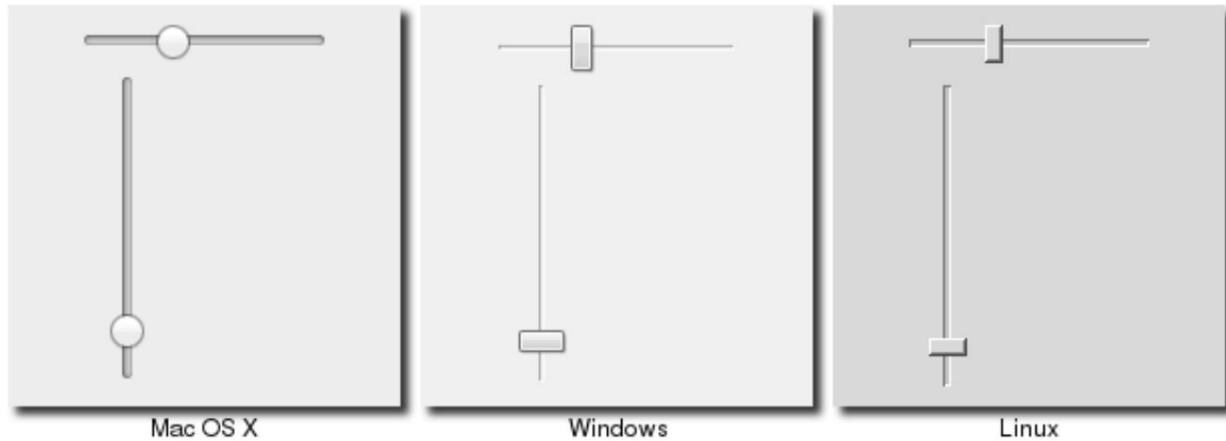
tarefa em execução que você realmente está, mas ainda deseja fornecer feedback ao usuário de que a operação ainda está em execução (ou que seu programa não travou). Em vez de fornecer valores específicos para indicar o progresso ao longo do caminho, no início da operação, você apenas chamará o método "start" da barra de progresso e, no final da operação, chamará seu método "stop". A barra de progresso levará cuidado do resto.

Escala

[• Resumo de widgets •](#)

[Manual de Referência](#)

Um widget **de escala** fornece uma maneira para os usuários escolherem um valor numérico por meio de manipulação direta.



Widgets de escala

Widgets de escala são criados usando a função **ttk.Scale** :

```
s = ttk.Scale(parent, orient=HORIZONTAL, length=200, from_=1.0, to=100.0)
```

Como 'from' é uma palavra-chave reservada, você precisará adicionar um sublinhado à direita ao usá-lo como uma opção de configuração.

De certa forma, os widgets de escala são como barras de progresso, exceto que são projetados para o usuário manipulá-los. Assim como as barras de progresso, elas devem receber uma orientação (horizontal ou vertical) com a opção de configuração "orientar" e um "comprimento" opcional. Você também deve definir o intervalo do número que a escala permite que os usuários escolham; para fazer isso, defina um número de ponto flutuante para cada uma das opções de configuração "de" e "para".

Existem várias maneiras diferentes de definir o valor atual da escala (que deve ser um valor de ponto flutuante entre os valores "de" e "para"). Você pode definir (ou ler, para obter o valor atual) a opção de configuração "valor" da balança. Você pode vincular a escala a uma variável usando a opção "variável". Ou você pode chamar o método de "definir valor" da escala para alterar o valor ou

o método "get" para ler o valor atual.

Existe uma opção de configuração "comando", que permite especificar um script para chamar sempre que a escala for alterada. Tk anexará automaticamente o valor atual da escala como um parâmetro cada vez que invocar este script (vimos uma coisa semelhante com parâmetros extras sendo adicionados aos retornos de chamada da barra de rolagem e aos widgets que eles rolam).

Assim como em outros widgets temáticos, você pode usar os métodos "estado desativado", "estado !desativado" e "instaurado desativado" se desejar impedir que o usuário modifique a escala.

Como o widget de escala não exibe os valores reais, você pode querer adicioná-los como rótulos.

Spinbox

[• Resumo de widgets •](#)

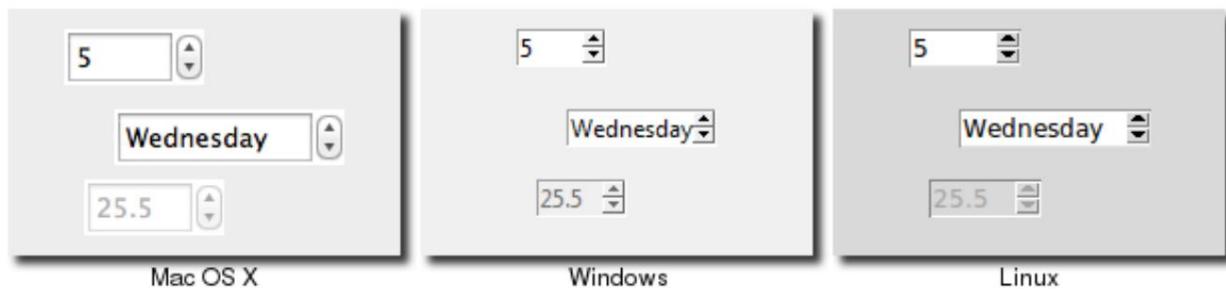
[Manual de Referência](#)

Um widget **spinbox** permite que os usuários escolham números (ou, na verdade, itens de uma lista arbitrária). Ele faz isso combinando um widget semelhante a uma entrada que mostra o valor atual com um par de pequenas setas para cima/para baixo que podem ser usadas para percorrer o intervalo de opções possíveis.

Spinboxes fazem parte dos widgets Tk clássicos. Não há atualmente uma caixa de rotação no conjunto de widgets temáticos Tk.

Essa última afirmação não é totalmente precisa. Um widget spinbox temático foi de fato adicionado no Tk 8.5.9.

Portanto, se seu código apenas assume 8.5 (como este tutorial faz), você não pode confiar nele. Pode valer a pena verificar se você está executando o 8.5.9 ou superior e, em caso afirmativo, use a versão temática. Mas, como em outros widgets temáticos, a API é um pouco diferente do widget clássico.



Widgets da caixa de rotação

Os widgets Spinbox são criados usando a função **Spinbox** :

```
spinval = StringVar()
Spinbox(parent, from_=1.0, to=100.0, textvariable=spinval)
```

Como os widgets de escala, os spinboxes são normalmente usados para permitir que o usuário escolha um número entre um determinado intervalo (especificado usando as opções de configuração "de" e "para"), embora por meio de uma interface de usuário muito diferente. Você também pode especificar um "incremento", que controla quanto o valor

muda toda vez que você clica no botão para cima ou para baixo.

Como uma caixa de listagem ou caixa de combinação, caixas de rotação também podem ser usadas para permitir que o usuário escolha um item de uma lista arbitrária de strings; estes podem ser especificados usando a opção de configuração "valores", que funciona da mesma forma que nas comboboxes. A especificação de uma lista de valores substituirá as configurações "de" e "para".

Você pode estar confuso sobre quando escolher uma escala, caixa de listagem, caixa de combinação, entrada ou caixa de rotação, já que geralmente vários deles podem ser usados para os mesmos tipos de dados. A resposta realmente depende do tipo de dados que você deseja que o usuário selecione, das convenções da interface do usuário da plataforma e da função que o valor desempenha na interface do usuário.

Por exemplo, tanto uma caixa de combinação quanto uma caixa de rotação têm o benefício de ocupar uma quantidade relativamente pequena de espaço, o que pode fazer sentido para uma configuração mais periférica, onde a escolha principal em uma interface de usuário pode justificar o espaço extra que uma caixa de listagem ocupa. Spinboxes não fazem muito sentido quando os itens não têm uma ordem natural e óbvia para eles. Você deve ter cuidado com caixas de combinação e caixas giratórias que contêm muitos itens, o que pode tornar mais demorado selecionar um item.

Tanto um intervalo numérico quanto arbitrário, há uma opção "wrap" que aceita um booleano e determina se o valor deve ser quebrado quando ultrapassar os valores inicial ou final. Você também pode especificar uma "largura" para a entrada que contém o valor atual da caixa de rotação.

Novamente, há opções de como definir ou obter o valor atual na caixa de rotação. Normalmente, você especificaria uma variável vinculada com a opção de configuração "textvariable"; como de costume, quaisquer alterações na variável são refletidas na caixa de rotação, enquanto quaisquer alterações na caixa de rotação são refletidas na variável vinculada. Além disso, os métodos "set value" e "get" permitem que você defina ou obtenha o valor diretamente.

Você pode organizar para ser chamado sempre que a caixa de rotação mudar usando a opção de configuração "comando".

O comando tem algumas substituições percentuais, %s = valor atual e %d = para cima ou para baixo.

Precisa descobrir a maneira certa de especificar isso em Ruby. Também precisa adicionar coisas sobre validação.

Como o spinbox não é um widget temático, os métodos "state" e "instate" não estão disponíveis. Em vez disso, você pode alterar seu estado usando a opção de configuração "estado". Isso pode ser "normal", "desativado" para evitar alterações.

Menus

Este capítulo descreve como lidar com barras de menus e menus pop-up no Tk. Para uma aplicação polida, essas são as áreas às quais você deve prestar atenção. Os menus precisam de cuidados especiais se você deseja que seu aplicativo se encaixe em outros aplicativos na plataforma de seus usuários.

Falando nisso, a maneira recomendada de descobrir em qual plataforma você está executando é:

```
root.tk.call('tk', 'sistema de janelas')           # retornará x11, win32 ou aqua
```

Pelo que sei, o Tkinter não fornece um equivalente direto a esta chamada. No entanto, como você pode ver no exemplo, é possível executar um comando Tk baseado em Tcl diretamente, usando a função ".tk.call()" disponível em qualquer widget Tkinter.

Isso provavelmente é mais útil do que examinar variáveis globais como tcl_platform ou RUBY_PLATFORM, e verificações mais antigas que usaram esses métodos devem ser examinadas. Embora antigamente houvesse uma correlação muito boa entre plataforma e sistema de janelas, isso é menos verdade hoje. Por exemplo, se sua plataforma for identificada no Unix, isso pode significar Linux sob X11, Mac OS X sob Aqua ou até mesmo Mac OS X sob X11.

Barras de menu

Nesta seção, veremos as barras de menu: como criá-las, o que há nelas, como são usadas e breve.

Projetar adequadamente uma barra de menu e seu conjunto de menus está além do escopo deste tutorial, mas alguns conselhos. Primeiro, se você se deparar com um grande número de menus, menus muito longos ou menus profundamente aninhados, talvez seja necessário repensar como sua interface de usuário está organizada. Em segundo lugar, muitas pessoas usam os menus para explorar o que o programa pode fazer, especialmente quando estão aprendendo, então tente garantir que os principais recursos sejam acessíveis pelos menus. Por fim, para cada plataforma que você deseja, familiarize-se com a forma como os aplicativos usam os menus e consulte as diretrizes de interface humana da plataforma para obter detalhes completos sobre design, terminologia, atalhos e muito mais. Esta é uma área que você provavelmente terá que personalizar para cada plataforma.

Você notará em algumas distribuições Linux recentes que muitos aplicativos mostram seus menus na parte superior da tela quando ativos, em vez de na própria janela. Tk ainda não suporta este estilo de menus.

Widgets de Menu e Hierarquia

- [Resumo de widgets](#)
- [Manual de Referência](#)

Menus são implementados como widgets no Tk, assim como botões e entradas. Cada widget de menu consiste em vários *itens* diferentes no menu. Itens são coisas como o comando "Abrir..." em um menu Arquivo, mas também separadores entre outros itens e itens que abrem seu próprio submenu (os chamados menus *em cascata*). Cada um desses itens de menu também possui atributos, como o texto a ser exibido para o item, um acelerador de teclado e um comando a ser invocado.

Os menus são organizados em uma hierarquia. A barra de menus é em si um widget de menu. Possui vários filhos (submenus) que consistem em itens como "Arquivo", "Editar" e assim por diante. Cada um deles, por sua vez, é um menu contendo diferentes itens, alguns dos quais podem conter submenus. Como seria de esperar de outras coisas que você já viu no Tk, sempre que você tiver um submenu, ele deve ser criado como um

filho de seu menu pai.

Antes que você comece

É importante colocar a seguinte linha em seu aplicativo em algum lugar antes de começar a criar menus.

```
root.option_add("*tearOff", FALSE)
```

Sem ele, cada um dos seus menus (no Windows e no X11) começará com o que parece ser uma linha tracejada e permite que você "corte" o menu para que apareça em sua própria janela. Você realmente não quer isso lá.

Este é um retrocesso para o X11 estilo Motif, no qual a aparência original do Tk foi baseada. A menos que seu aplicativo seja projetado para ser executado naquela caixa velha acumulando poeira no porão, você realmente não quer ver isso, pois não faz parte de nenhum estilo moderno de interface do usuário.

E todos estaremos ansiosos por uma versão do Tk em que essa compatibilidade com versões anteriores não seja preservada e o padrão seja não ter esses menus destacáveis.

Criando uma Barra de Menu

Em Tk, as barras de menu são associadas a janelas individuais; cada janela de nível superior pode ter no máximo uma barra de menu. No Windows e no X11, isso é visualmente óbvio, pois os menus fazem parte de cada janela, logo abaixo da barra de título na parte superior.

No entanto, no Mac OS X, há uma única barra de menu na parte superior da tela, compartilhada por cada janela. No que diz respeito ao seu programa Tk, cada janela ainda tem sua própria barra de menu; à medida que você alterna entre as janelas, o Tk cuidará automaticamente de garantir que a barra de menus correta seja exibida na parte superior da tela. Se você não especificar uma barra de menu para uma janela em particular, o Tk usará a barra de menu associada à janela raiz; você já deve ter notado que isso é criado automaticamente para você quando seu aplicativo Tk é iniciado.

Como no Mac OS X *todas* as janelas têm uma barra de menu, é importante certificar-se de definir uma, seja para cada janela ou uma barra de menu alternativa para a janela raiz. Caso contrário, você acabará com a barra de menu "embutida", que contém menus destinados apenas ao uso ao digitar comandos diretamente no interpretador.

Para realmente criar uma barra de menus para uma janela, primeiro criamos um widget de menu e, em seguida, usamos a opção de configuração "menu" da janela para anexar o widget de menu à janela.

```
win = Toplevel(root)
menubar = Menu(win)
win['menu'] = menubar
```

Observe que você *pode* usar a mesma barra de menu para mais de uma janela (ou seja, use uma barra de menu como o valor da opção de configuração "menu" para diferentes janelas de nível superior). Isso é particularmente útil no Windows e no X11, onde você pode querer que uma janela inclua um menu, mas não precisa necessariamente fazer malabarismos com diferentes menus em seu aplicativo. Mas lembre-se, se o conteúdo ou o estado da barra de menu

depende do que está acontecendo na janela ativa, você mesmo terá que lidar com isso.

Esta é uma história verdadeiramente antiga, mas as barras de menu costumavam ser criadas criando um widget de quadro contendo os itens do menu e empacotando-o na parte superior da janela como você faria com qualquer outro widget.

Espero que você não tenha nenhum código ou documentação que ainda faça isso.

Adicionando menus

Agora temos uma barra de menus, mas ela é bastante inútil sem alguns menus. Então, novamente, vamos querer criar um widget de menu para cada menu que irá na barra de menus (cada um filho da barra de menus) e, em seguida, adicioná-los todos à barra de menus.

```
menubar = Menu(pai)
menu_file = Menu(barra de menu)
menu_edit = Menu(barra de
menu) menubar.add_cascade(menu=menu_file, label='Arquivo')
menubar.add_cascade(menu=menu_edit, label='Editar')
```

Adicionando Itens de Menu

Agora que temos alguns menus em nossa barra de menus, provavelmente é um bom momento para adicionar alguns itens a cada menu. Lembre-se de que os itens de menu fazem parte do próprio menu, portanto, felizmente, não precisamos criar outro widget de menu para cada um.

```
menu_file.add_command(label='Novo', command=novoArquivo)
menu_file.add_command(label='Abrir...', command=arquivo aberto)
menu_file.add_command(label='Fechar', command=fecharArquivo)
```

No Mac OS X, as reticências ("...") são, na verdade, um caractere especial, com espaçamento mais estreito do que três pontos seguidos. Tk se encarrega de substituir esse personagem por você automaticamente.

Portanto, adicionar itens de menu a um menu é essencialmente o mesmo que adicionar um submenu, mas em vez de adicionar um item de menu do tipo "cascata", estamos adicionando um do tipo "comando".

Cada item de menu possui várias opções de configuração associadas, da mesma forma que os widgets, embora cada tipo de item de menu tenha um conjunto diferente de opções relevantes. Os itens de menu em cascata têm uma opção de "menu" usada para especificar o submenu, os itens de menu de comando têm uma opção de "comando" usada para especificar o comando a ser invocado quando o item é selecionado e ambos têm uma opção de "rótulo" para especificar o texto a ser exibido para o item.

Além de adicionar itens ao final dos menus, você também pode inseri-los no meio dos menus por meio da opção "inserir *índice* ?valor da opção...?" método; aqui "*índice*" é a posição (0..n-1) do item que você deseja inserir antes. Você também pode excluir um menu usando o método "excluir *índice*".

Tipos de itens de menu

Já vimos os itens de menu "comando", que são os itens de menu comuns que, quando selecionado invocará um comando.

Também vimos o uso de itens de menu em "cascata", usados para adicionar um menu a uma barra de menus. Não é de surpreender que, se você quiser adicionar um submenu a um menu existente, também use um item de menu em "cascata", exatamente da mesma maneira.

Um terceiro tipo de item de menu é o "separador", que produz a linha divisória que você costuma ver entre diferentes conjuntos de itens de menu.

```
menu_file.add_separator()
```

Finalmente, há também itens de menu "checkbutton" e "radiobutton", que se comportam de forma análoga aos widgets checkbutton e radiobutton. Esses itens de menu têm uma variável associada a eles e, dependendo do valor dessa variável, exibirão um indicador (ou seja, uma marca de seleção ou um botão de opção selecionado) ao lado do rótulo do item.

```
check = StringVar()
menu_file.add_checkbutton(label='Check', variable=check, onvalue=1, offvalue=0)
radio = StringVar()
menu_file.add_radiobutton(label='One', variable=radio, value=1)
menu_file.add_radiobutton(label='Dois', variable=radio, value=2)
```

Quando o usuário seleciona um item do botão de seleção que ainda não está marcado, ele definirá a variável associada com o valor em "onvalue", enquanto ao selecionar um item que já está marcado, ele será definido como o valor em "offvalue". A seleção de um item de botão de opção define a variável associada para o valor em "valor". Ambos os tipos de itens também reagem a mudanças na variável associada de outras partes do seu programa.

Assim como os itens de comando, os itens de menu botão de seleção e botão de opção aceitam uma opção de configuração de "comando", que será invocada quando o item de menu for selecionado; a variável associada e, portanto, o estado do item de menu, é atualizada antes que o retorno de chamada seja invocado.

Os itens de menu do botão de opção não fazem parte das diretrizes de interface humana do Windows ou do Mac OS X, portanto, nessas plataformas, o indicador ao lado do rótulo do item é uma marca de seleção, como seria para um item de botão de seleção. A semântica ainda funciona; é uma boa forma de selecionar entre vários itens, já que o visor mostrará um dos itens selecionados (marcado).

Teclas Aceleradoras

A opção "acelerador" é utilizada para indicar o acelerador de menu que deve ser associado a este menu. Na verdade, isso não cria o acelerador, mas apenas exibe o que está ao lado do item de menu. Você ainda precisa criar uma ligação para o acelerador por conta própria.

Lembre-se de que as ligações de eventos podem ser definidas em widgets individuais, em todos os widgets de um determinado tipo, na janela de nível superior que contém o widget no qual você está interessado ou no aplicativo como um todo. Como as barras de menu estão associadas a janelas individuais, normalmente as associações de evento que você criar estarão na janela de nível superior à qual o menu está associado.

Os aceleradores são muito específicos da plataforma, não apenas em termos de quais teclas são usadas para qual operação, mas também quais teclas modificadoras são usadas para aceleradores de menu (por exemplo, no Mac OS X, é o

tecla "Command", no Windows e X11 geralmente é a tecla "Control"). Exemplos de opções de acelerador válidas são "Command-N", "Shift+Ctrl+X" e "Command-Option-B". Os modificadores comumente usados incluem "Control", "Ctrl", "Option", "Opt", "Alt", "Shift", "Command", "Cmd" e "Meta").

No Mac OS X, esses modificadores serão mapeados automaticamente para os diferentes ícones de modificadores que aparecem nos menus.

Mais sobre opções de itens

Existem algumas opções mais comuns para itens de menu.

Sublinhado

Embora todas as plataformas suportem a passagem do teclado pela barra de menus por meio das teclas de seta, no Windows e no X11, você também pode usar outras teclas para pular para menus ou itens de menu específicos. As teclas queacionam esses saltos são indicadas por uma letra sublinhada no rótulo do item de menu. Se você deseja adicionar um deles a um item de menu, pode usar a opção de configuração "sublinhado" para o item. O valor desta opção deve ser o índice do caractere que você gostaria de sublinhar (de 0 ao comprimento da string - 1).

Imagens

Também é possível usar imagens em itens de menu, seja ao lado do rótulo do item de menu ou substituindo-o completamente. Para fazer isso, você pode usar as opções "image" e "compound", que funcionam exatamente como nos widgets de rótulos. O valor para "image" deve ser um objeto de imagem Tk, enquanto "compound" pode ter os valores "bottom", "center", "left", "right", "top" ou "none".

Estado

Também é possível desativar um menu, para que o usuário não possa selecioná-lo. Isso pode ser feito através da opção "estado", configurando-o para um valor de "desativado" ou um valor de "normal" para reativar o item.

Consultando e alterando opções de itens

Como quase tudo em Tk, você pode ver ou alterar o valor das opções de um item a qualquer momento.

Os itens são referidos por meio de um *índice*. Normalmente, este é um número (0..n-1) que indica a posição do item no menu, mas você também pode especificar o rótulo do item de menu (ou, na verdade, um padrão de "estilo glob" para corresponder ao item rótulo).

```
print( menu_file.entrycget(0, 'label') )
menu_file.entryconfigure('Close', state=DISABLED)
print( menu_file.entryconfigure(0) )
```

Menus da plataforma

Cada plataforma possui alguns menus que são tratados especialmente pelo Tk.

Mac OS X

Você provavelmente notou se esteve brincando com os exemplos anteriores no Mac OS X, que Tk fornece sua própria barra de menu padrão, incluindo um menu com o nome do programa que está sendo executado (neste caso, o shell de sua linguagem de programação, por exemplo ' Wish', 'Python', etc.), um menu Arquivo e um menu padrão Editar, Windows e Ajuda, todos abastecidos com vários itens de menu.

Você pode substituir esta barra de menus em seu próprio programa, mas para obter os resultados desejados, você precisará seguir algumas etapas específicas (em alguns casos, em uma ordem específica).

A partir do Tk 8.5.13, a manipulação de menus especiais no Mac mudou, resultado da mudança do código Tk subjacente da obsoleta Carbon API para Cocoa. Se você estiver vendo nomes de menu duplicados, itens ausentes, coisas que você não colocou lá, etc., revise esta seção cuidadosamente.

A primeira coisa a saber é que se você não especificar uma barra de menus para uma janela (ou sua janela pai, por exemplo, a janela raiz) você terminará com a barra de menus padrão que o Tk fornece, que a menos que você esteja apenas mexendo no seu próprio, quase certamente não é o que você deseja.

O menu do aplicativo

Se você fornecer uma barra de menu, *no momento em que a barra de menu for anexada à janela*, se não houver um menu especialmente denominado ".apple" (veja abaixo), Tk fornecerá um menu de aplicativo padrão, *nomeado após o binário sendo executado*. Ele conterá um item "Sobre Tcl & Tk", seguido pelos itens de menu padrão: preferências, submenu de serviços, ocultar/mostrar itens e sair. Novamente, você não quer isso.

Se você fornecer seu próprio menu ".apple", quando a barra de menus for anexada à janela, ela adicionará os itens padrão (preferências e seguintes) no final de todos os itens adicionados. Perfeito! (Os itens que você adicionar *depois que a barra de menus for anexada à janela* aparecerão após o item sair, o que, novamente, você não deseja.)

O menu de aplicativos, que é o de que estamos tratando aqui, é diferente do menu de maçã (aquele com o ícone da maçã, logo à esquerda do menu de aplicativos). Apesar de realmente nos referirmos ao menu do aplicativo, em Tk ele ainda é chamado de menu "maçã". Este é um resquício dos dias anteriores ao OS X, quando esses tipos de itens iam para o menu real da maçã e não havia um menu de aplicativo separado.

Em outras palavras, em seu programa, certifique-se de:

1. Crie uma barra de menus para cada janela ou a janela raiz. *Não anexe a barra de menu ao janela ainda!*
2. Adicione um menu à barra de menus chamado ".apple" que será usado como menu do aplicativo.
3. O título do menu terá automaticamente o mesmo nome do binário do aplicativo; se você quiser mudar isso, renomeie (ou faça uma cópia) o binário usado para executar seu script.
4. Adicione os itens que deseja que apareçam na parte superior do menu do aplicativo, ou seja, um "Sobre seu aplicativo"

item, seguido de um separador.

5. Depois de fazer tudo isso, você pode *anexar* a barra de menu à sua janela.

```
win = Toplevel(root)
menubar = Menu(win)
appmenu = Menu(menubar, name='apple')
menubar.add_cascade(menu=appmenu)
appmenu.add_command(label='About My Application')
appmenu.add_separator() win ['menu'] = barra de menus
```

Embora normalmente o Tkinter escolha um nome de caminho de widget para nós, aqui tivemos que fornecer explicitamente um ('apple') usando a opção 'name' ao criar o menu do aplicativo.

Manipulando o item de menu de preferências

Como você notou, o menu do aplicativo sempre inclui um item de menu "Preferências..."; isso é incluído automaticamente. Se seu aplicativo tiver uma caixa de diálogo de preferências, a seleção desse item de menu deve abri-la. Se seu aplicativo não tiver uma caixa de diálogo de preferências, esse item de menu deve ser desativado, o que ocorre por padrão.

Para conectar sua caixa de diálogo de preferências, você precisará definir um procedimento Tcl chamado "::tk::mac::ShowPreferences". Isso será chamado quando o item de menu Preferências for escolhido; se o procedimento não for definido, o item de menu será desabilitado.

```
def showMyPreferencesDialog():
    ...
root.createcommand('tk::mac::ShowPreferences', showMyPreferencesDialog)
```

Fornecendo um menu de ajuda

Assim como o menu do aplicativo, qualquer menu de ajuda adicionado à sua própria barra de menus é tratado especialmente no Mac OS X. Assim como o menu do aplicativo que precisava de um nome especial ('.apple'), o menu de ajuda deve receber o nome '.help'. Também como o menu do aplicativo, o menu de ajuda também deve ser adicionado *antes que a barra de menus seja anexada à janela*.

O menu de ajuda incluirá a caixa de pesquisa padrão do OS X para pesquisar ajuda, bem como um item denominado 'ajuda do seu aplicativo'. Assim como o nome do menu do aplicativo, o nome deste item vem do nome do binário que executa seu programa e não pode ser alterado. Semelhante a como as caixas de diálogo de preferências são tratadas, para responder a este item de ajuda, você precisa definir um procedimento Tcl chamado "::tk::mac::ShowHelp". Ao contrário das preferências, não definir este procedimento gerará um erro, não desativará o item de menu.

Se você não deseja incluir ajuda, simplesmente não adicione um menu de ajuda à barra de menus e nenhum será exibido.

Ao contrário do X11 e versões anteriores do Tk no Mac OS X, o menu Ajuda não será colocado automaticamente no final da barra de menus, portanto, certifique-se de que seja o último menu adicionado.

Você também pode adicionar outros itens ao menu de ajuda, que aparecerá após o item de ajuda do aplicativo.

```
helpmenu = Menu(barra de menu, name='ajuda')
menubar.add_cascade(menu=menu de ajuda, label='Ajuda')
root.createcommand('tk::mac::MostrarAjuda', ...)
```

Fornecendo um menu de janela

No Mac OS X, um menu 'Janela' é usado para conter itens como minimizar, ampliar, trazer tudo para frente, etc.

Ele também contém uma lista de janelas abertas no momento. Antes dessa lista, às vezes são fornecidos outros itens específicos do aplicativo.

Ao fornecer um menu chamado ".window", este menu de janela padrão será adicionado e o Tk irá automaticamente mantê-lo sincronizado com todas as suas janelas de nível superior, sem nenhum código extra de sua parte.

Você também pode adicionar quaisquer comandos específicos do aplicativo a este menu, que aparecerá antes da lista de suas janelas.

```
windowmenu = Menu(menubar, name='janela')
menubar.add_cascade(menu=windowmenu, label='Janela')
```

Outros manipuladores de menu

Você viu anteriormente como lidar com certos itens de menu exigia que você definisse procedimentos de retorno de chamada Tcl, em particular para exibir o diálogo de preferências ('tk::mac::ShowPreferences') e exibir ajuda ('tk::mac::ShowHelp').

Existem vários outros retornos de chamada que você pode definir, por exemplo, para interceptar o item de menu Sair para solicitar que as alterações sejam salvas ou para ser informado quando o aplicativo estiver oculto ou exibido. Aqui está a lista completa:

Chamado quando
o item de menu tk::mac::ShowPreferences...
é selecionado.
rendas

tk::mac::MostrarAjuda	Chamado para exibir a ajuda on- line principal do aplicativo.
tk::mac::Sair	Chamado quando o item de menu Sair é selecionado, quando o usuário está tentando desligar o sistema, etc.
tk::mac::OnHide	Chamado quando seu aplicativo foi oculto.
tk::mac::OnShow	Chamado quando seu aplicativo é mostrado após ser oculto. tk::mac::OpenAppli Chamado quando

cação seu aplicativo é
aberto pela primeira vez.
Chamado quando
o usuário "reabre"
tk::mac::ReopenApp em execução (por exemplo,
clica nele no Dock)

Chamado quando
o Finder deseja
que o aplicativo
abra um ou mais
documentos
tk::mac::OpenDocum (por exemplo, que fornece).

O procedimento recebe
uma lista de nomes de
caminhos de arquivos
a serem abertos.

tk::mac::PrintDocument Como no
OpenDocument,
mas os documentos
devem ser
impressos em vez
de abertos.

janelas

No Windows, cada janela tem um menu "Sistema" no canto superior esquerdo da moldura da janela, com um pequeno ícone para o seu aplicativo. Ele contém itens como "Fechar", "Minimizar", etc. No Tk, se você criar um menu de sistema, poderá adicionar novos itens que aparecerão abaixo dos itens padrão.

```
sysmenu = Menu(menubar, name='sistema')
menubar.add_cascade(menu=sysmenu)
```

Embora normalmente o Tkinter escolha um nome de caminho de widget para nós, aqui tivemos que fornecer explicitamente um com o nome 'sistema'; esta é a dica que Tk precisa para reconhecê-lo como o menu do sistema.

X11

No X11, se você criar um menu de ajuda, o Tk garantirá que seja sempre o último menu da barra de menus.

```
menu_help = Menu(menubar, name='ajuda')
menubar.add_cascade(menu=menu_ajuda, label='Ajuda')
```

O caminho do widget do menu deve ser fornecido explicitamente, neste caso com o nome "**ajuda**". Isso pode ser especificado para qualquer widget Tkinter usando a opção 'nome' ao criar o widget.

Menus contextuais

Menus contextuais (menus "pop-up") são normalmente chamados por um clique com o botão direito do mouse em um objeto no aplicativo. Um menu aparece no local do cursor do mouse e o usuário pode selecionar um dos itens do menu (ou clicar fora do menu para descartá-lo sem escolher nenhum item).

Para criar um menu contextual, você usará exatamente os mesmos comandos usados para criar menus na barra de menus. Normalmente, você criará um menu com vários itens de comando nele e, possivelmente, alguns itens de menu em cascata e seus menus associados.

Para ativar o menu, o usuário usará um clique de menu contextual, ao qual você terá que se vincular.

Isso, no entanto, pode significar coisas diferentes em diferentes plataformas. No Windows e X11, este é o botão direito do mouse sendo clicado (o terceiro botão do mouse). No Mac OS X, isso é um clique do botão esquerdo (ou apenas) com a tecla de controle pressionada ou um clique direito em um mouse com vários botões.

Ao contrário do Windows e do X11, o Mac OS X se refere a isso como o segundo botão do mouse, não o terceiro, então esse é o evento que você verá em seu programa.

A maioria dos programas anteriores que usavam menus pop-up presumiam que era apenas com o "botão 3" que eles precisavam se preocupar.

Além de capturar o evento de menu contextual correto, você também precisará capturar o local em que o mouse foi clicado. Acontece que você precisa fazer isso em relação à tela inteira (coordenadas globais) e não localmente à janela ou widget em que clicou (coordenadas locais). As substituições "%X" e "%Y" no sistema de vinculação de eventos do Tk irão capturá-las para você.

A última etapa é simplesmente dizer ao menu para aparecer no local específico. Aqui está um exemplo de todo o processo, usando um menu pop-up na janela principal do aplicativo.

```
from tkinter import * root =  
Tk() menu = Menu(root) for  
i in ('One', 'Two', 'Three'):  
menu.add_command(label=i)  
  
if (root.tk.call('tk', 'windowingsystem')=='aqua'):  
    root.bind('<2>', lambda e: menu.post(e.x_root, e.y_root)) root.bind('<Control-1>',  
    lambda e: menu.post(e.x_root, e.y_root)) else: root.bind('<3>', lambda e:  
menu.post(e.x_root, e.y_root))
```

Janelas e Diálogos

Tudo o que fizemos até agora foi em uma única janela. Neste capítulo, abordaremos como usar várias janelas, alterar vários atributos de janelas e usar algumas das janelas de caixa de diálogo padrão disponíveis no Tk.

Criando e Destruindo Janelas

Você já viu que todos os programas Tk começam com uma janela raiz de nível superior e, em seguida, os widgets são criados como filhos dessa janela raiz. A criação de novas janelas de nível superior funciona quase exatamente da mesma forma que a criação de novos widgets.

As janelas de nível superior são criadas usando a função **de nível superior** :

`t = nível superior(pai)`

Ao contrário dos widgets comuns, você não precisa "rede" um nível superior para que ele apareça na tela. Depois de criar um novo nível superior, você pode criar outros widgets que são filhos desse nível superior e colocá-los em grade dentro do nível superior. Em outras palavras, o novo nível superior se comporta exatamente como a janela raiz criada automaticamente.

Para destruir uma janela, você pode usar o método **destroy** em um widget:

`window.destroy()`

Observe que você pode usar destruir em qualquer widget, não apenas em uma janela de nível superior. Além disso, quando você destrói uma janela, todas as janelas (widgets) que são filhas dessa janela também são destruídas. Portanto, se você destruir a janela raiz (da qual todos os outros widgets são descendentes), isso normalmente encerrará seu aplicativo.

Mudando o Comportamento e os Estilos da Janela

Há muitas coisas sobre como as janelas se comportam e como elas se parecem que podem ser alteradas.

Título da Janela

Para examinar ou alterar o título da janela:

```
oldtitle = window.title()
window.title('Novo título')
```

Tamanho e Localização

Em Tk, a posição e o tamanho de uma janela na tela são conhecidos como sua *geometria*. Uma especificação de geometria completa se parece com isso:

largura x altura ± x ± a

Largura e altura (normalmente em pixels) são bastante auto-explicativas. O "x" (posição horizontal) é especificado com um sinal de mais ou menos, então "+25" significa que a borda esquerda da janela deve estar a 25 pixels da borda esquerda da tela, enquanto "-50" significa a borda direita da janela deve estar a 50 pixels da borda direita da tela. Da mesma forma, uma posição "y" (vertical) de "+10" significa que a borda superior da janela deve estar dez pixels abaixo da parte superior da tela, enquanto "-100" significa que a borda inferior da janela deve estar 100 pixels acima na parte inferior da tela.

Lembre-se de que a posição da geometria são as coordenadas reais na tela e não faça concessões para sistemas como o Mac OS X, que possuem uma barra de menu na parte superior ou uma área de encaixe na parte inferior. Portanto, especificar uma posição de "+0+0" na verdade colocaria a parte superior da janela sob a barra de menus do sistema. É uma boa ideia deixar uma margem saudável (pelo menos 30 pixels) de uma borda da tela.

Aqui está um exemplo de alteração de tamanho e posição, colocando a janela no canto superior direito da tela:

```
window.geometry('300x200-5+40')
```

Ordem de empilhamento

A ordem de empilhamento refere-se à ordem em que as janelas são "colocadas" na tela, de baixo para cima. Quando as posições de duas janelas se sobrepõem, a mais próxima do topo da ordem de empilhamento obscurecerá ou sobreporá a que está abaixo na ordem de empilhamento.

Você pode obter a ordem de empilhamento atual, uma lista do menor ao maior, por meio de:

```
root.tk.eval('wm stackorder '+str(janela))
```

Este método não parece ser exposto de forma limpa através da API do Tkinter.

Você também pode verificar se uma janela está acima ou abaixo de outra:

```
if (root.tk.eval('wm stackorder '+str(window)+'isabov
'+str(otherwindow))=='1') ... if (root.tk.eval('wm stackorder '+str (janela)+'
está abaixo de '+str(outrajanela))=='1') ...
```

Você também pode aumentar ou diminuir as janelas, bem no topo (inferior) da ordem de empilhamento ou logo acima (abaixo) de uma janela designada:

```
window.lift()
window.lift(otherwin)
window.lower()
window.lower(otherwin)
```

Tkinter usa o nome 'lift', já que 'raise' é uma palavra-chave reservada.

Quer saber por que você precisou passar por uma janela para obter a ordem de empilhamento atual? Acontece que a ordem de empilhamento, aumentar e diminuir, etc. funciona não apenas para janelas de nível superior, mas com quaisquer widgets "irmãos" (aqueles que têm o mesmo pai). Portanto, se você tiver vários widgets agrupados em grade, mas sobrepostos, poderá aumentá-los e diminuí-los um em relação ao outro. Por exemplo:

```
from tkinter import * from
tkinter import ttk root = Tk() little
= ttk.Label(root, text="Little")
large = ttk.Label(root, text='Label muito maior')
little.grid(column=0 ,linha=0) maior.grid(column=0,linha=0)
root.after(2000, lambda: little.lift()) root.mainloop()
```

O comando "after" agenda um script para ser executado em um determinado número de milissegundos no futuro, mas permite que o processamento normal do loop de eventos continue nesse meio tempo.

Comportamento de redimensionamento

Normalmente, as janelas de nível superior, tanto a janela raiz quanto outras que você criar, podem ser redimensionadas pelo usuário. No entanto, às vezes, em sua interface de usuário, você pode querer impedir que o usuário redimensione a janela. Você pode impedir que ela seja redimensionada, de fato, especificando independentemente se a largura da janela (primeiro parâmetro) pode ser alterada, bem como sua altura (segundo parâmetro). Portanto, para desativar todo o redimensionamento:

```
window.resizable(FALSE,FALSE)
```

Lembre-se de que, se você adicionou um widget `ttk::sizegrip` à janela, deve removê-lo se estiver tornando a janela não redimensionável.

Se o redimensionamento estiver ativado, você pode especificar um tamanho mínimo e/ou máximo ao qual gostaria que o tamanho da janela fosse limitado (novamente, os parâmetros são largura e altura):

```
window.minsize(200,100)
window.maxsize(500,500)
```

Iconificando e Retirando

Na maioria dos sistemas, você pode remover temporariamente a janela da tela ao transformá-la em ícone. Em Tk, o fato de uma janela ser iconificada ou não é referido como o *estado da janela*. Os estados possíveis para uma janela incluem "normal" e "íônico" (para uma janela iconificada), além de vários outros: "retirada", "ícone" ou "ampliada".

Você pode consultar ou definir o estado atual da janela e também existem os métodos "iconify" e "deiconify", que são atalhos para definir os estados "íônico" ou "normal", respectivamente.

```
thestate = window.state()
window.state('normal')
window.iconify() window.deiconify()
```

Diálogos padrão

As caixas de diálogo são um tipo de janela utilizada em aplicativos para obter algumas informações do usuário, informar que algum evento ocorreu, confirmar uma ação e muito mais. A aparência e o uso de caixas de diálogo geralmente são detalhados especificamente no guia de estilo de uma plataforma. O Tk vem com várias caixas de diálogo integradas para tarefas comuns e que ajudam você a se adequar às diretrizes de estilo específicas da plataforma.

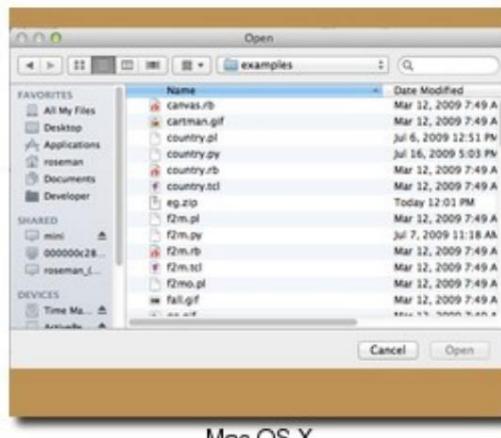
Seleção de arquivos e diretórios

Tk fornece vários diálogos para permitir que o usuário selecione arquivos ou diretórios. No Windows e no Mac, esses

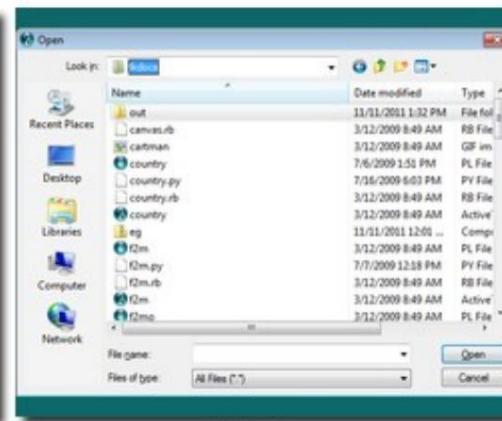
chame os diálogos subjacentes do sistema operacional diretamente. A variante "abrir" na caixa de diálogo é usada quando você deseja que o usuário selecione um arquivo existente (como em um comando de menu "Arquivo | Abrir..."), enquanto a variante "salvar" é usada para escolher um arquivo para salvar (normalmente usado pelo comando de menu "Arquivo | Salvar como...").

```
from tkinter import filedialog
filename = filedialog.askopenfilename()
filename = filedialog.asksaveasfilename()
dirname = filedialog.askdirectory()
```

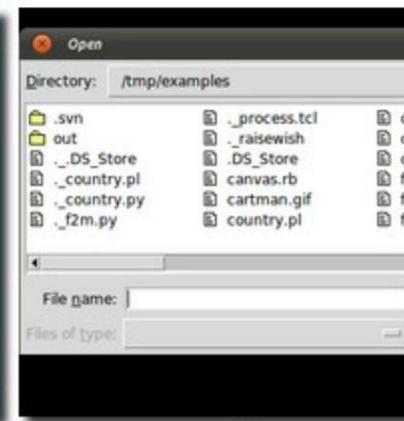
Todos esses comandos produzem diálogos *modais*, o que significa que os comandos (e, portanto, seu programa) não continuarão em execução até que o usuário envie o diálogo. Os comandos retornam o caminho completo do arquivo ou diretório que o usuário escolheu ou retornam uma string vazia se o usuário cancelar a caixa de diálogo.



Mac OS X

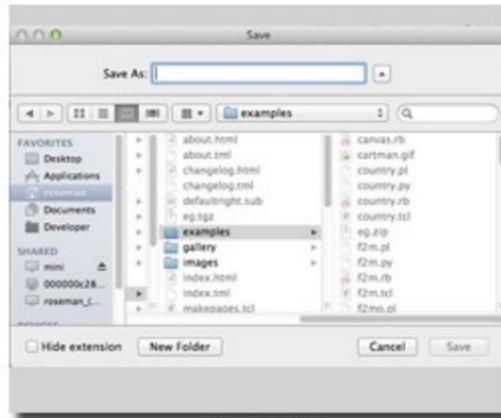


Windows

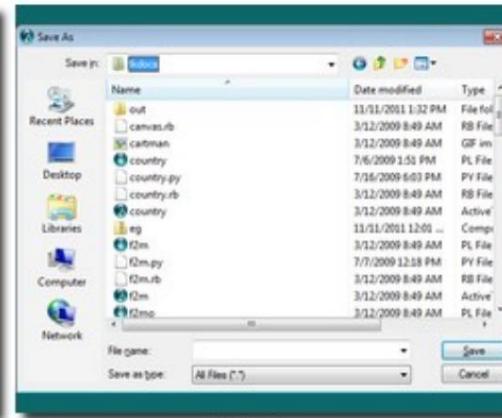


Linux

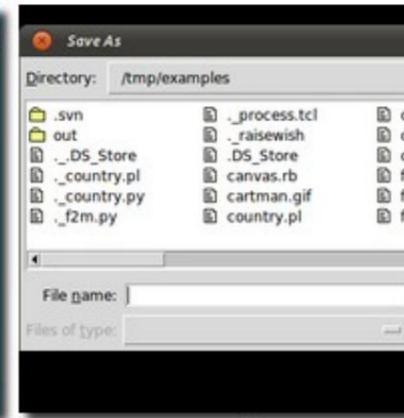
Diálogos de Abrir Ficheiro



Mac OS X

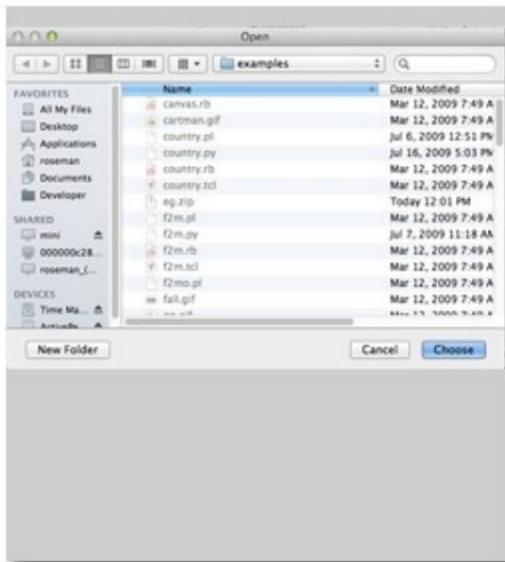


Windows

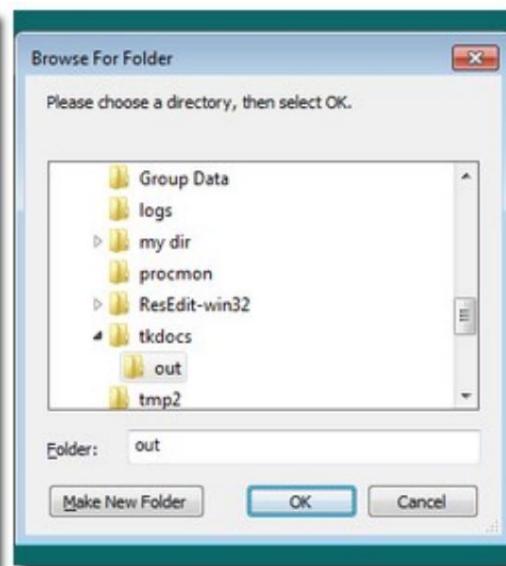


Linux

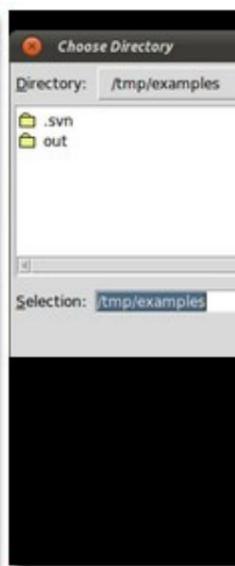
Diálogos de Salvamento de Arquivos



Mac OS X



Windows



Linux

Escolha Diálogos de Diretório

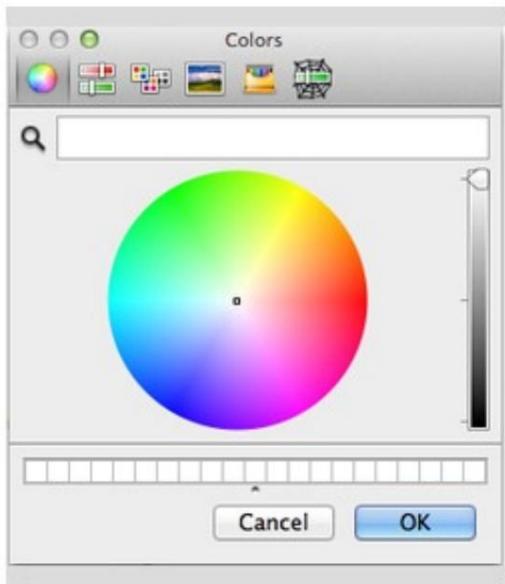
Há uma variedade de opções diferentes que podem ser passadas para essas caixas de diálogo, permitindo que você defina os tipos de arquivo permitidos, nome de arquivo padrão e muito mais. Eles são detalhados em [getOpenFile/getSaveFile](#) e [selectDirectory](#) páginas de manual de referência.

Selecionando Cores

Há também uma caixa de diálogo modal para permitir que o usuário selecione uma cor. Ele retornará um valor de cor, por exemplo, "#ff62b8". A caixa de diálogo aceita uma opção "initialcolor" opcional para especificar uma cor existente que o usuário presumivelmente está substituindo.

de `tkinter import colorchooser`

```
colorchooser.askcolor(initialcolor='#ff0000')
```



Mac OS X



Windows



Linux

Escolha as caixas de diálogo de cores

Uma coisa que falta no Tk 8.5 é uma caixa de diálogo do seletor de fontes. Na verdade, isso foi adicionado no Tk 8.6.

Diálogos de Alerta e Confirmação Muitos

aplicativos usam vários alertas modais simples ou diálogos para notificar o usuário sobre um evento, pedir-lhes para confirmar uma ação ou fazer outra escolha semelhante que é feita clicando em um botão. Tk fornece uma "caixa de mensagens" versátil que encapsula todos esses diferentes tipos de diálogos.

```
de tkinter import messagebox
messagebox.showinfo(message='Tenha um bom dia')
```



Caixas de Mensagens Simples

```
messagebox.askyesno(
    message='Tem certeza que deseja instalar o SuperVirus?' icon='question'
    title='Instalar')
```



Exemplos de Caixas de Mensagens

Como as caixas de diálogo anteriores que vimos, elas são modais e retornarão o resultado da ação do usuário ao chamador. O valor exato de retorno dependerá da opção "type" passada para o comando, conforme mostrado aqui:

Digite a opção	Possíveis valores de retorno
ok (padrão) okcancel	"OK"
	"ok" ou "cancelar"
sim não	"sim" ou "não" "sim",
simnão cancelar tentar	"não" ou "cancelar" "tentar novamente"
novamentecancelar	ou "cancelar"

abortretryignore "abort", "retry" ou "ignore"

Em vez de usar uma opção de 'tipo', o Tkinter usa um nome de método diferente para cada tipo de diálogo. Os métodos disponíveis são: askokcancel, askquestion, askretrycancel, askyesno, askyesnocancel, showerror, showinfo, showwarning.

A lista completa de opções possíveis é mostrada aqui:

`type` Como descrito acima.

`message` A mensagem principal exibida dentro do alerta.

`detalhe` Se necessário, uma mensagem secundária, geralmente exibida em uma fonte menor sob a mensagem principal. `title` Título da janela de diálogo. Não usado no Mac OS X. Ícone Ícone para mostrar: uma das "informações" (padrão), "erro", "pergunta" ou "aviso".

`padrão` Especifique qual botão (por exemplo, "ok" ou "cancelar" para uma caixa de diálogo do tipo "okcancelar") deve ser o padrão.

`pai` Especifique uma janela de seu aplicativo para a qual esta caixa de diálogo está sendo postada; isso pode fazer com que a caixa de diálogo apareça na parte superior ou, no Mac OS X, apareça como uma folha para a janela.

Esses novos diálogos de caixa de mensagem são uma substituição do antigo comando "`tk_dialog`", que não está em conformidade com as convenções de interface de usuário da plataforma atual.

Organizando Interfaces Complexas

Se você tiver uma interface de usuário grande, precisará encontrar maneiras de organizá-la de maneira que não sobrecarregue seus usuários com a complexidade. Existem várias abordagens diferentes para fazer isso; novamente, as diretrizes de interface humana gerais e específicas da plataforma são um bom recurso ao decidir.

Observe que quando falamos sobre complexidade neste capítulo, não é a complexidade técnica subjacente de como o programa é montado, mas como ele é apresentado ao usuário. Uma interface de usuário pode ser reunida a partir de muitos módulos diferentes, construída a partir de vários widgets de tela e quadros profundamente aninhados, mas isso não significa necessariamente que o usuário a considere complexa.

Várias janelas

Um dos benefícios de usar várias janelas em um aplicativo pode ser simplificar a interface do usuário, exigindo que o usuário se concentre apenas no conteúdo de uma janela por vez (exigir que eles se concentrem ou alternem entre várias janelas também pode ter o efeito oposto). Da mesma forma, mostrar apenas os widgets relevantes para a tarefa atual (via grade) pode ajudar a simplificar a interface do usuário.

espaço em branco

Se você precisa exibir um grande número de widgets na tela ao mesmo tempo, deve pensar em como organizá-los visualmente. Você viu como a grade pode ajudar facilitando o alinhamento dos widgets entre si. O espaço em branco é outra ajuda útil. Colocar widgets relacionados bem perto de

entre si (possível com um rótulo explicativo imediatamente acima) e separados de outros widgets menos relacionados por espaço em branco ajudam o usuário a organizar a interface do usuário em sua própria mente.

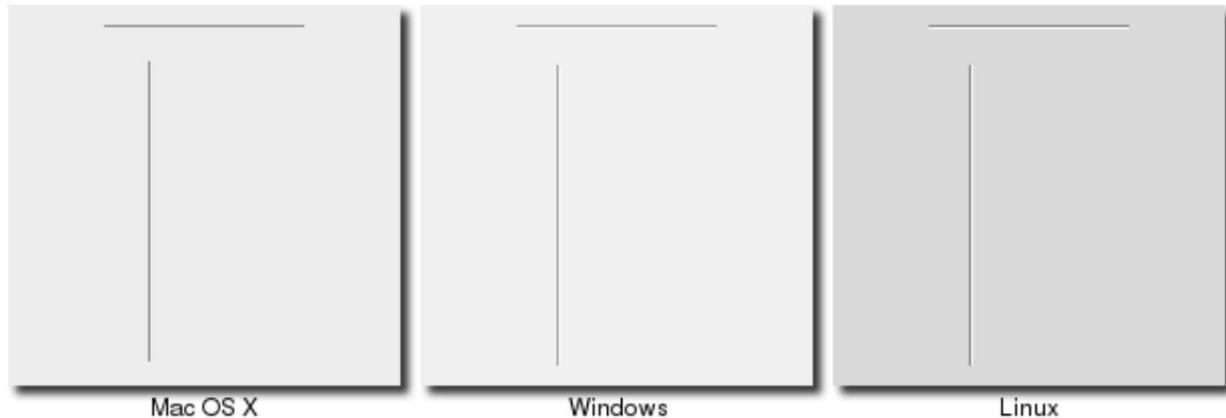
A quantidade de espaço em branco em torno de diferentes widgets, entre grupos de widgets, em torno de bordas e assim por diante é altamente específica da plataforma. Embora você possa fazer um trabalho adequado sem se preocupar com números exatos de pixels, se quiser uma interface de usuário altamente refinada, precisará ajustá-la para cada plataforma.

Separador

- [Resumo de widgets](#) •

[Manual de Referência](#)

Uma segunda abordagem para agrupar widgets em uma exibição é colocar uma régua fina horizontal ou vertical entre grupos de widgets; muitas vezes isso pode ser mais eficiente em termos de espaço do que usar espaço em branco, o que pode ser relevante para uma exibição apertada. Tk fornece um widget **separador** muito simples para essa finalidade.



Widgets Separadores

Os separadores são criados usando a função **ttk.Separator** :

```
s = ttk.Separator(parent, orient=HORIZONTAL)
```

A opção "orientar" pode ser especificada como "horizontal" ou "vertical".

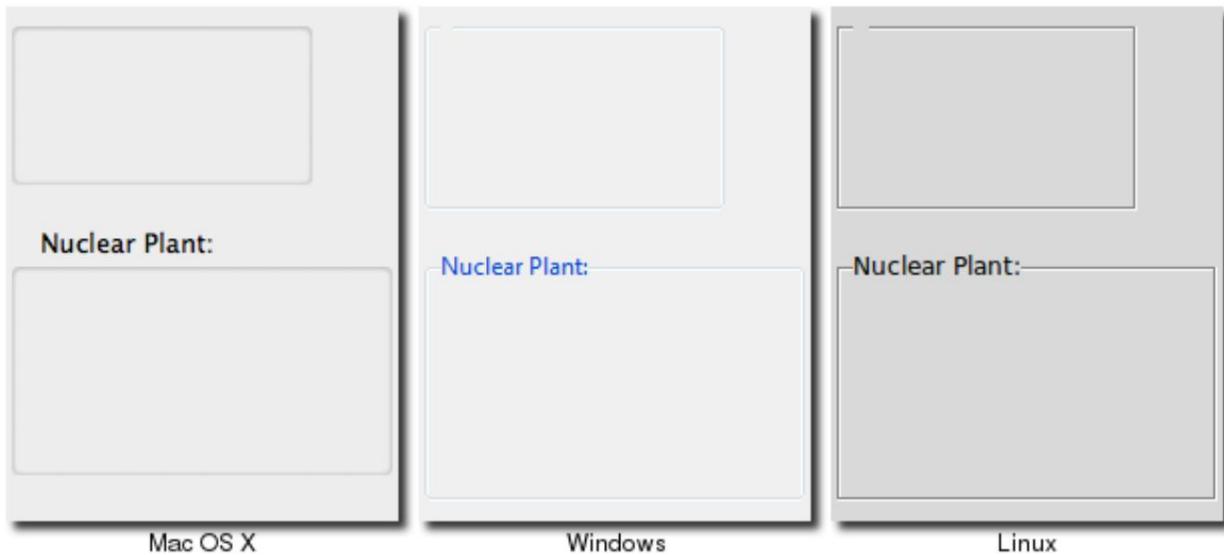
molduras de etiquetas

- [Resumo de widgets](#) •

[Manual de Referência](#)

Um widget de **frame de rótulo**, também conhecido como *caixa de grupo*, fornece outra maneira de agrupar componentes relacionados. Ele age como um `ttk::frame` normal, em que você normalmente o usará como um contêiner para outros widgets que você grade dentro dele. No entanto, ele é exibido de uma maneira que o diferencia visualmente do resto

da interface do usuário. Opcionalmente, você pode fornecer um rótulo de texto para ser exibido fora do quadro de rótulos.



Widgets de Labelframe

Labelframes são criados usando a função **ttk.Labelframe** :

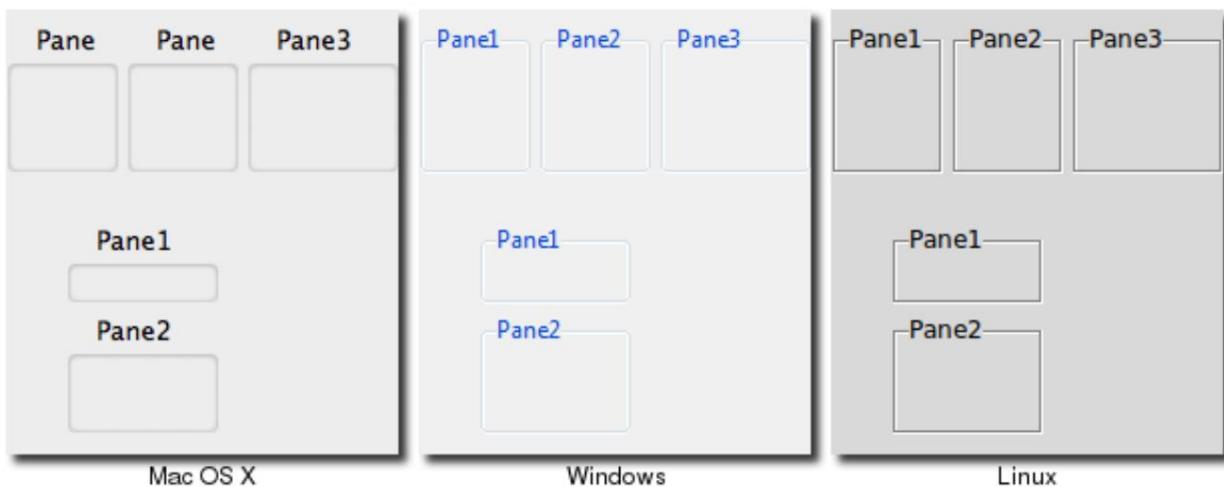
```
If = ttk.Labelframe(parent, text='Label')
```

Janelas em painel

[• Resumo de widgets •](#)

[Manual de Referência](#)

Um widget **de janela panorâmica** permite empilhar dois ou mais widgets redimensionáveis acima e abaixo um do outro (ou à esquerda e à direita). O usuário pode ajustar as alturas relativas (ou larguras) de cada painel arrastando uma faixa localizada entre eles. Normalmente, os widgets que você adiciona a uma janela com painel serão quadros contendo muitos outros widgets.



Widgets Panedwindow (mostrados aqui gerenciando vários quadros de rótulos)

Panedwindows são criados usando a função **ttk.Panedwindow** :

```
p = ttk.Panedwindow(parent, orient=VERTICAL) # primeiro painel, que receberia widgets em grade: f1 = ttk.Labelframe(p, text='Pane1', width=100, height=100) f2 = ttk.Labelframe(p, text='Pane2', width=100, height=100) # segundo painel p.add(f1) p.add(f2)
```

Uma janela com painel é "vertical" (seus painéis são empilhados verticalmente uns sobre os outros) ou "horizontal". É importante ressaltar que todos os painéis adicionados à janela em painel devem ser *filhos diretos* da própria janela em painel.

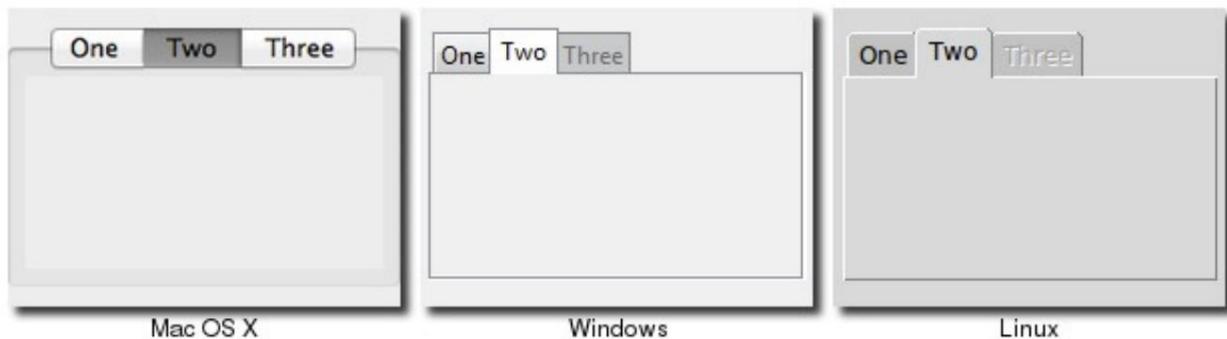
Chamar o método "add" adicionará um novo painel no final da lista de painéis. O método "inserir *subjanela de posição*" permite que você coloque o painel na posição dada na lista de painéis (0..n-1); se o painel já for gerenciado pela janela em painel, ele será movido para a nova posição. Você pode usar "esquecer a subjanela" para remover um painel da janela em painel; você também pode passar uma posição em vez de uma subjanela.

Outras opções permitem que você assine pesos relativos para cada painel de modo que, se a janela geral for redimensionada, determinados painéis terão mais espaço do que outros. Além disso, você pode ajustar a posição de cada faixa entre os itens na janela panorâmica. Consulte a [referência de comando](#) para obter detalhes.

Caderno

- [Resumo de widgets](#)
- [Manual de Referência](#)

Um widget **de notebook** usa a metáfora de um notebook com guias para permitir que o usuário alterne entre uma das várias *páginas*, usando uma *guia de índice*. Nesse caso, ao contrário das janelas em painel, estamos permitindo que o usuário veja apenas uma única página (semelhante a um painel) por vez.



Widgets de notebook

Os notebooks são criados usando a função **ttk.Notebook** :

```
n = ttk.Notebook(parent) f1 = ttk.Frame(n) # primeira página, que receberia widgets em grade f2 = ttk.Frame(n) # segunda página
```

```
n.add(f1, texto='Um')
n.add(f2, texto='Dois')
```

As operações em notebooks com guias são semelhantes àquelas em janelas em painel. Cada página é normalmente um quadro, novamente um filho direto (subjanela) do próprio notebook. Uma nova página e sua guia associada são adicionadas ao final da lista de guias com a opção "adicionar *subjanela* ?*valor da opção...?*" método. A opção de guia "texto" é usada para definir o rótulo na guia; também é útil a opção da guia "estado", que pode ter o valor "normal", "desativado" (não selecionável) ou "oculto".

Para inserir uma aba em algum lugar diferente do final da lista, você pode usar a função "inserir *subjanela* de *posição* ?*valor da opção...?*", e para remover uma determinada aba, use o método "esquecer", passando a posição (0..n-1) ou a subjanela da aba. Você pode recuperar a lista de todas as subjanelas contidas no notebook por meio do método "tabs".

Para recuperar a subjanela atualmente selecionada, chame o método "selecionar" e altere a guia selecionada passando a posição da guia ou a própria subjanela como parâmetro.

Para alterar uma opção de guia (como o rótulo de texto na guia ou seu estado), você pode usar o método "*tab(tabid, option=value*" (onde "tabid" é novamente a posição ou subjanela da guia); omita o "= value" para retornar o valor atual da opção.

Novamente, há uma variedade de opções e comandos usados com menos frequência detalhados na referência de comando .

Fontes, Cores, Imagens

Este capítulo descreve como o Tk lida com fontes, cores e imagens. Já abordamos tudo isso antes, mas aqui forneceremos um tratamento mais aprofundado.

Fontes

Vários widgets Tk, como rótulo, texto e tela, permitem que você especifique as fontes usadas para exibir o texto, geralmente por meio de uma opção de configuração de "fonte". Como acontece com muitas coisas no Tk, as fontes padrão geralmente são uma boa escolha, mas se você quiser fazer alterações, esta seção descreverá várias maneiras de fazer isso. As fontes são uma das várias áreas altamente específicas da plataforma, portanto, como você as especifica é importante.

A referência de comando de fonte fornece detalhes completos sobre a especificação de fontes, bem como outras operações relacionadas a fontes.

A maioria dos widgets temáticos que exibem texto não possui uma opção de configuração de "fonte", ao contrário dos widgets Tk clássicos. Em vez de modificar widgets individuais, a abordagem correta nos widgets temáticos é especificar as fontes usadas em um estilo e, em seguida, usar esse estilo para o widget individual. Isso é semelhante à diferença entre marcações orientadas para exibição codificadas, como tags de fonte dentro do HTML.

páginas, em vez de usar folhas de estilo CSS que mantêm todas as informações específicas de exibição em um só lugar.

Muitos programas Tk mais antigos codificavam muitas fontes, usando o formato de "estilo de tamanho de família" que veremos abaixo, nomes de fontes X11 ou a string de especificação de fonte X11 mais antiga e misteriosa. Em muitos casos, isso deixou esses aplicativos com uma aparência datada à medida que as plataformas evoluíam. Além disso, muitos programas especificavam fontes por widget, deixando as decisões de fonte espalhadas pelo programa.

As fontes nomeadas e o uso das fontes padrão fornecidas pelo Tk são uma solução muito melhor. Revisar e atualizar o uso de fontes é uma alteração fácil e importante a ser feita em qualquer aplicativo existente.

Fontes padrão

Particularmente para elementos de interface de usuário mais ou menos padrão, cada plataforma define fontes específicas que devem ser usadas. Tk encapsula muitos deles em um conjunto padrão de fontes que estão sempre disponíveis e, é claro, os widgets padrão usam essas fontes. Isso ajuda a abstrair as diferenças de plataforma. As fontes predefinidas são:

TkDefaultFont	O padrão para todos os itens GUI não especificados de outra forma.
TkTextFont	Usado para widgets de entrada, caixas de listagem, etc.
TkFixedFont	Uma fonte padrão de largura fixa.
TkMenuFont	A fonte usada para itens de menu.
TkHeadingFont	A fonte normalmente usada para cabeçalhos de coluna em listas e tabelas.
TkCaptionFont	Uma fonte para janelas e barras de legenda de diálogo.
TkSmallCaptionFont	Uma fonte de legenda menor para subjanelas ou caixas de diálogo de ferramentas
TkIconFont	Uma fonte para legendas de ícones.
TkTooltipFont	Uma fonte para dicas de ferramentas.

Fontes específicas da plataforma

Várias fontes predefinidas adicionais estão disponíveis, mas o conjunto preciso depende da plataforma.

Obviamente, se você estiver usando seu aplicativo portátil em diferentes plataformas, você precisará garantir que as fontes apropriadas sejam definidas individualmente para cada plataforma.

No X11, qualquer nome de fonte X11 válido (consulte, por exemplo, o comando "xlsfonts") pode ser usado. Lembre-se, porém, que não há garantia de que uma fonte específica tenha sido instalada em uma máquina específica.

No Windows, os seguintes nomes de fonte, que mapeiam para as fontes que podem ser definidas no "Display" Painel de controle, estão disponíveis: system, ansi, device, systemfixed, ansifixed, oemfixed.

No Mac OS X, as seguintes fontes estão disponíveis (consulte o Apple HIG para obter detalhes):

systemSystemFont, systemSmallSystemFont, systemApplicationFont, systemViewsFont, systemMenuItemFont, systemMenuItemCmdKeyFont, systemPushButtonFont, systemAlertHeaderFont, systemMiniSystemFont, systemDetailEmphasizedSystemFont, systemEmphasizedSystemFont, systemSmallEmphasizedSystemFont, systemLabelFont,

systemMenuTitleFont, systemMenuItemMarkFont, systemWindowTitleFont, systemUtilityWindowTitleFont, systemToolbarFont, systemDetailSystemFont.

Se você quiser examinar qual fonte real está sendo atribuída a qualquer uma das fontes padrão ou padrão (ou qualquer fonte), você pode usar a chamada "font actual".

Fontes Nomeadas

Você também pode criar suas próprias fontes, que podem ser usadas exatamente como as predefinidas. Para fazer isso, você precisará escolher um nome para se referir à fonte e, em seguida, especificar vários atributos de fonte que definem a aparência da fonte. Normalmente, você usaria diferentes atributos de fonte em diferentes plataformas; dessa forma, você pode usar a fonte em seu programa sem se preocupar com os detalhes, exceto no local em que a fonte é realmente definida.

Aqui está um exemplo:

```
de tkinter import font
appHighlightFont = font.Font(family='Helvetica', size=12, weight='bold') ttk.Label(root, text='Atenção!', font=appHighlightFont).grid()
```

A "família" especifica o nome da fonte; os nomes Courier, Times e Helvetica são garantidos para serem suportados (e mapeados para uma fonte monoespaçada, serif ou sans-serif apropriada), mas outras fontes instaladas no sistema podem ser usadas (novamente, tenha cuidado para garantir que a fonte exista, ou o sistema fornecerá uma fonte diferente, que pode não ser necessariamente uma boa correspondência). Você pode obter os nomes de todas as fontes disponíveis com:

`fonte.famílias()`

A opção "tamanho" especifica o tamanho da fonte, em pontos. A opção "peso" pode ser em negrito ou normal. Você pode especificar uma "inclinação" de romano (normal) ou itálico. Finalmente, as opções booleanas "sublinhado" e "sobreposto" estão disponíveis.

As configurações atuais dessas opções podem ser examinadas ou alteradas usando os mesmos mecanismos que você usaria para alterar as opções de configuração de um widget (por exemplo, configurar).

Descrições de fonte

Outra forma de especificar fontes é por meio de uma lista de atributos, começando com o nome da fonte e, opcionalmente, incluindo um tamanho e, opcionalmente, uma ou mais opções de estilo. Alguns exemplos disso são "Helvetica", "Helvetica 12", "Helvetica 12 negrito" e "Helvetica 12 negrito itálico". Essas descrições de fonte são usadas como o valor da opção de configuração "fonte", em vez de uma fonte predefinida ou nomeada.

Em geral, é aconselhável alternar de descrições de fonte para fontes nomeadas, novamente para isolar as diferenças de fonte em um local do programa.

cores

Tal como acontece com as fontes, existem várias maneiras de especificar as cores. Detalhes completos podem ser encontrados na [referência de comando de cores](#).

Em geral, o sistema fornecerá as cores certas para a maioria das coisas. Assim como as fontes, tanto o Mac quanto o Windows especificam um grande número de nomes de cores específicos do sistema (consulte a referência), cuja cor real pode depender das configurações do sistema (por exemplo, cores de realce de texto, planos de fundo padrão).

Você também pode especificar cores via RGB, como em HTML, por exemplo, "#3FF" ou "#FF016A". Finalmente, Tk reconhece o conjunto de nomes de cores definido por X11; normalmente estes não são usados, exceto os muito comuns como "vermelho", "preto", etc.

Para widgets Tk temáticos, as cores geralmente são usadas na definição de estilos que são aplicados aos widgets, em vez de aplicar a cor a um widget diretamente.

Provavelmente nem é preciso dizer que a moderação no uso de cores é normalmente justificada.

Imagens

Já vimos o básico de como usar imagens, exibindo-as em rótulos ou botões, por exemplo.

Criamos um objeto de imagem, geralmente a partir de um arquivo em disco.

```
imgobj = PhotoImage(file='myimage.gif') label['image']  
= imgobj
```

Fora da caixa, o Tk inclui suporte para imagens GIF e PPM/PNM. No entanto, existe uma biblioteca de extensão Tk chamada "Img" que adiciona suporte para muitos outros: BMP, XBM, XPM, PNG, JPEG, TIFF, etc. (por exemplo, ActiveTcl).

Em vez de usar a extensão Img do Tk, o Tkinter usa uma biblioteca de imagens feita para Python chamada 'PIL' (Biblioteca de imagens Python). Mais especificamente, usaremos um fork mais atualizado do PIL chamado 'pillow'. Como ele não vem junto com o Python, normalmente você precisará instalá-lo. Você deve ser capaz de fazer isso via, por exemplo, 'pip install Pillow'.

```
from PIL import ImageTk, Image myimg  
= ImageTk.PhotoImage(Image.open('myimage.png'))
```

A chamada 'ImageTk.PhotoImage' fornece um substituto para a PhotoImage de Tk, mas oferece suporte a uma gama mais ampla de tipos de imagem.

As imagens do Tk são bastante poderosas e sofisticadas e fornecem uma ampla variedade de maneiras de inspecionar e modificar imagens. Você pode descobrir mais na [referência de comando de imagem](#) e a [referência do comando photo](#).

Os tipos de imagens multicoloridas que vimos aqui são referidos em Tk como imagens *fotográficas*. Tk também fornece um segundo tipo de imagens, imagens *bitmap* de dois bits, que foram amplamente utilizadas nos anos 90 quando

a maioria das estações de trabalho Unix usava monitores bastante grandes (em comparação com os PCs), mas eram apenas preto e branco. Escusado será dizer que a cor é de rigueur nos dias de hoje, portanto, é altamente recomendável atualizar para imagens coloridas para ícones e assim por diante.

Problemas com PIL/Pillow no Mac OS X?

Se você estiver executando no Mac OS X, ocorreu uma falha ao usar o PIL/Pillow? Talvez você esteja vendo mensagens de erro em seu Terminal dizendo coisas como:

A classe TKApplication é implementada em /Library/Frameworks/Tk.framework/Versions/8.5/Tk e /System/Library/Frameworks/Tk.framework/Versions/8.5/Tk.
Um dos dois será usado. Qual deles é indefinido.

Isso pode ocorrer quando você baixa um binário de uma extensão que está vinculada a versões diferentes de Tcl e Tk daquelas que você está executando. Nesse caso, você está executando o ActiveTcl em /Library/Frameworks e a extensão está tentando vincular-se à fornecida pelo Mac OS X em /System/Library/Frameworks.

Esta é uma das dificuldades com extensões binárias. O snippet abaixo mostra como podemos corrigir esse problema. É específico para a versão da extensão Pillow que instalamos, mas esperamos que possa orientá-lo em termos de localização e correção dos binários de qualquer extensão que esteja causando problemas para você. As etapas são aproximadamente: encontre o binário, verifique se ele está vinculado à versão errada (via otool) e altere o que está vinculado, usando um programa chamado install_name_tool.

```
% cd /Library/Frameworks/Python.framework/Versions/3.4 % find . -name
'_imagingtk.so' ./lib/python3.4/site-packages/PIL/_imagingtk.so % cd lib/
python3.4/site-packages/PIL % otool -L _imagingtk.so _imagingtk.so:

/Sistema/Biblioteca/Frameworks/Tcl.framework/Versões/8.5/Tcl (...)

/System/Library/Frameworks/Tk.framework/Versions/8.5/Tk (...) /usr/lib/libSystem.B.dylib
(...)

% install_name_tool -change /
System/Library/Frameworks/Tcl.framework/Versions/8.5/Tcl /Library/
Frameworks/Tcl.framework/Versions/8.5/Tcl _imagingtk.so
% install_name_tool -change /
System/Library/Frameworks/Tk.framework/Versions/8.5/Tk /Library/
Frameworks/Tk.framework/Versions/8.5/Tk _imagingtk.so % otool -L _imagingtk.so
_imagingtk.so:

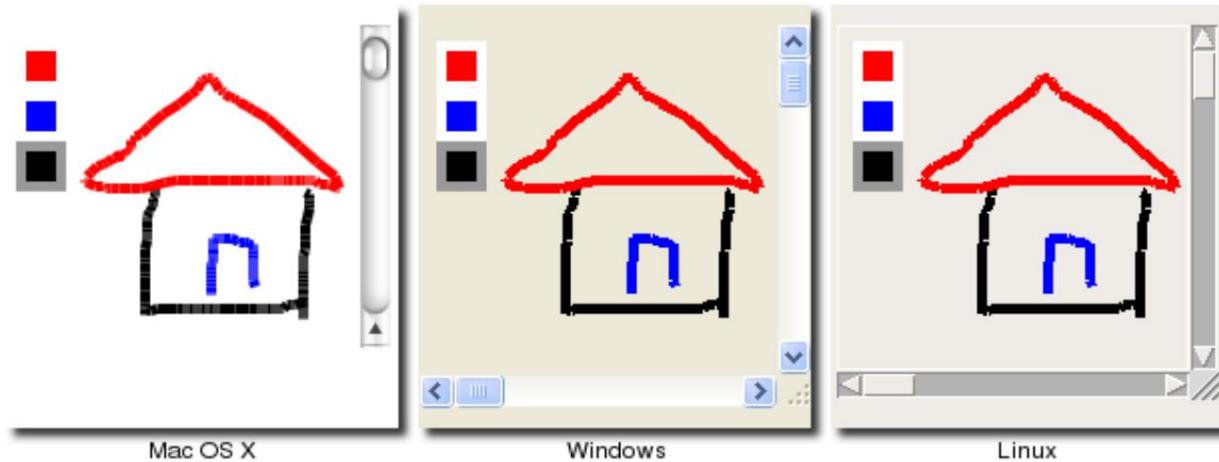
/Biblioteca/Frameworks/Tcl.framework/Versions/8.5/Tcl (...)

/Library/Frameworks/Tk.framework/Versions/8.5/Tk (...) /usr/lib/libSystem.B.dylib
(...)
```

Tela

- [Resumo de widgets](#) •
- [Manual de Referência](#)

Um widget **de tela** gerencia uma coleção 2D de objetos gráficos — linhas, círculos, imagens, outros widgets e muito mais. A tela do Tk é um widget incrivelmente poderoso e flexível e realmente um dos destaques do Tk. É adequado para uma ampla gama de usos, incluindo desenho ou diagramação, ferramentas CAD, exibição ou monitoramento de simulações ou equipamentos reais e para a construção de widgets mais complexos a partir dos mais simples. Os widgets de tela fazem parte dos widgets Tk clássicos, não dos widgets Tk temáticos.



Tela de widgets

Os widgets do Canvas são criados usando a função **Canvas** :

```
tela = tela(pai)
```

Como os widgets de tela têm uma grande quantidade de recursos, não poderemos cobrir tudo aqui.

O que faremos é pegar um exemplo bastante simples (uma ferramenta de esboço à mão livre) e adicionar novas peças a ele, cada uma mostrando outro novo recurso de widgets de tela. Perto do final do capítulo, abordaremos alguns dos outros principais recursos não ilustrados no exemplo.

Criando itens

Quando você cria um novo widget de tela, ele será essencialmente um grande retângulo sem nada; verdadeiramente uma tela em branco em outras palavras. Para fazer algo útil com ele, você precisará adicionar *itens* a ele. Como mencionado, há uma grande variedade de tipos diferentes de itens que você pode adicionar. Aqui, veremos como adicionar um item de linha simples à tela.

Para criar uma linha, a única informação que você precisa especificar é onde a linha deve estar. Isso é feito usando as coordenadas do ponto inicial e final, expressas como uma lista no formato $x0\ y0\ x1\ y1$. A origem $(0,0)$ está no canto superior esquerdo da tela, com o valor de x aumentando à medida que você move para a direita e o valor de y aumentando à medida que você move para baixo. Portanto, para criar uma linha de $(10,10)$ a $(200,50)$, usariammos este código:

```
canvas.create_line(10, 10, 200, 50)
```

O método "create_line" retornará um id de item (um número inteiro) que pode ser usado para se referir exclusivamente

a este item; cada item criado terá seu próprio id. Embora muitas vezes não precisemos nos referir ao item mais tarde e, portanto, ignoraremos o id retornado, veremos como ele pode ser usado em breve.

Vamos começar nosso exemplo simples de bloco de desenho. Por enquanto, queremos poder desenhar à mão livre na tela arrastando o mouse sobre ela. Criaremos um widget de tela e, em seguida, anexaremos associações de evento a ele para capturar cliques e arrastos do mouse. Quando clicarmos com o mouse pela primeira vez, lembraremos desse local como nossa posição "inicial". Toda vez que o mouse for movido com o botão do mouse ainda pressionado, criaremos um item de linha que vai desta posição "inicial" até a posição atual do mouse. A posição atual será então a posição "inicial" para o próximo segmento de linha.

```
da importação tkinter * da
importação ttk
```

```
último x, último = 0, 0
```

```
def xy(evento):
    global lastx, lasty
    lastx = evento.x, evento.y

def addLine(evento):
    global lastx, lasty
    canvas.create_line((lastx, lasty, evento.x, evento.y)) lastx, lasty = evento.x,
    evento.y

root = Tk()
root.columnconfigure(0, peso=1)
root.rowconfigure(0, peso=1)

canvas = Canvas(root)
canvas.grid(column=0, row=0, sticky=(N, W, E, S))
canvas.bind("<Button-1>", xy) canvas.bind("< B1-Motion>", addLine)

root.mainloop()
```

Experimente - arraste o mouse pela tela para criar sua obra-prima.

Atributos do Item

Ao criar itens, você também pode especificar um ou mais *atributos* para o item, que afetarão como ele é exibido. Por exemplo, aqui especificaremos que a linha deve ser vermelha e ter três pixels de largura.

```
canvas.create_line(10, 10, 200, 50, fill='red', width=3)
```

O conjunto exato de atributos varia de acordo com o tipo de item.

Assim como com os widgets Tk, também é possível alterar os atributos dos itens da tela depois de criá-los.

```
id = canvas.create_line(0, 0, 10, 10, fill='vermelho')
...
canvas.itemconfigure(id, fill='blue', width=2)
```

Ligações

Já vimos que o widget canvas como um todo, como qualquer outro widget Tk, pode capturar eventos usando o comando "bind".

Você também pode anexar vinculações a itens individuais na tela (ou grupos deles, como veremos na próxima seção usando tags). Portanto, se você quiser saber se um determinado item foi clicado ou não, não precisa observar os eventos de clique do mouse para a tela como um todo e, em seguida, descobrir se esse clique aconteceu em seu item. A Tk cuidará de tudo isso para você.

Para capturar esses eventos, você usa um comando de vinculação integrado à tela. Ele funciona exatamente como o comando normal de ligação, recebendo um padrão de evento e um retorno de chamada. A única diferença é que você especifica o item de tela ao qual esta vinculação se aplica.

```
canvas.tag_bind(id, '<1>', ...)
```

Observe a diferença entre o método "tag_bind" específico do item e o "bind" de nível de widget método.

Vamos adicionar algum código ao nosso exemplo de bloco de desenho para permitir a alteração da cor do desenho.

Primeiro criaremos alguns itens retangulares diferentes, cada um preenchido com uma cor diferente. Criar itens retangulares é como criar itens de linha, onde você especificará as coordenadas de dois cantos diagonalmente opostos.

Em seguida, anexaremos uma ligação a cada um deles para que, quando forem clicados, definam uma variável global para a cor a ser usada. Nossa vinculação de movimento do mouse examinará essa variável ao criar os segmentos de linha.

```
cor = "preto" def
setColor(nova cor):
    cor global cor =
    nova cor

def addLine(evento):
    global lastx, lasty
    canvas.create_line((lastx, lasty, evento.x, evento.y), fill=color) lastx, lasty = evento.x, evento.y

id = canvas.create_rectangle((10, 10, 30, 30), fill="red") canvas.tag_bind(id,
"<Button-1>", lambda x: setColor("red")) id = canvas.create_rectangle((10, 35, 30,
55), fill="blue") canvas.tag_bind(id, "<Button-1>", lambda x: setColor("blue")) id =
canvas.create_rectangle((10, 60, 30, 80), fill="black") canvas.tag_bind(id,
"<Button-1>", lambda x: setColor("black"))
```

Marcas

Vimos que cada item de tela tem um número de identificação exclusivo, mas há outra maneira muito útil e poderosa de se referir a itens em uma tela, que é usando *marcas*.

Uma tag é apenas um identificador da sua criação, algo significativo para o seu programa. Você pode anexar tags a itens de tela; cada item pode ter qualquer número de tags. Ao contrário dos números de identificação do item, que são exclusivos para cada item, muitos itens podem ter a mesma tag.

O que você pode fazer com tags? Vimos que você pode usar o id do item para modificar um item da tela (e veremos em breve que há outras coisas que você pode fazer com os itens, como movê-los, excluí-los etc.). Sempre que puder usar um ID de item, você pode usar uma tag. Por exemplo, você pode alterar a cor de todos os itens com uma tag específica.

As tags são uma boa maneira de identificar certos tipos de itens em sua tela (itens que fazem parte de uma linha desenhada, itens que fazem parte da paleta, etc.). Você pode usar tags para correlacionar itens de tela a objetos específicos em seu aplicativo (por exemplo, marque todos os itens de tela que fazem parte do robô com id #37 com a tag "robot37"). Com tags, você não precisa controlar os ids dos itens da tela para se referir a grupos de itens posteriormente; tags permitem que Tk faça isso por você.

Você pode atribuir tags ao criar um item usando a opção de configuração de item "tags". Você pode adicionar tags posteriormente com o método "addtag" ou removê-los com o método "dtags". Você pode obter a lista de tags para um item com o método "gettags" ou retornar uma lista de números de id de item com a tag fornecida com o comando "find".

Por exemplo:

```
>>> c = Canvas(raiz) >>>
c.create_line(10, 10, 20, 20, tags=('primeira linha', 'desenho')) 1

>>> c.create_rectangle(30, 30, 40, 40, tags='('desenho')') 2 >>>
c.addtag('retângulo', 'com etiqueta', 2) >>> c.addtag('polígono', 'withtag',
'retângulo') >>> c.gettags(2) ('desenho', 'retângulo', 'polígono') >>> c.dtag(2,
'polígono') >>> c.gettags(2) ('desenho', 'retângulo') >>>
c.find_withtag('desenho') (1, 2)
```

Como você pode ver, coisas como "withtag" levarão um item individual ou uma tag; neste último caso, eles se aplicarão a todos os itens com essa tag (que pode ser nenhuma). O "addtag" e o "find" têm muitas outras opções, permitindo que você especifique itens próximos a um ponto, sobrepondo uma área específica e muito mais.

Vamos usar tags primeiro para colocar uma borda em torno de qualquer item em nossa paleta de cores que esteja selecionado no momento.

```
def setColor(nova cor):
    cor global color
    = newcolor
    canvas.dtag('all', 'paletteSelected')
    canvas.itemconfigure('palette', outline='white')
    canvas.addtag('paletteSelected', 'withtag', 'palette% % color')
    canvas.itemconfigure('paletteSelected', sketch='#999999')

id = canvas.create_rectangle((10, 10, 30, 30), fill="vermelho", tags='('paleta',
```

```
'palettered')) id =
canvas.create_rectangle((10, 35, 30, 55), fill="blue", tags=('palette', 'paletteblue')) id =
canvas.create_rectangle((10, 60, 30, 80), fill="black", tags=('palette', 'paletteblack', 'paletteSelected'))

setColor('black')
canvas.itemconfigure('palette', width=5)
```

Vamos também usar tags para tornar o traço atual que estamos desenhando mais visível; quando soltarmos o mouse, voltaremos ao normal.

```
def addLine(evento):
    global lastx, lasty
    canvas.create_line((lastx, lasty, evento.x, evento.y), fill=color, width=5, tags='currentline') lastx, lasty =
    evento.x, evento.y

def doneStroke(evento):
    canvas.itemconfigure('currentline', width=1)

canvas.bind("<B1-ButtonRelease>", doneStroke)
```

Modificando Itens

Você viu como pode modificar as opções de configuração de um item — sua cor, largura e assim por diante.

Há uma série de outras coisas que você pode fazer itens.

Para excluir itens, use o método "excluir". Para alterar o tamanho e a posição de um item, você pode usar o método "coords"; isso permite que você forneça novas coordenadas para o item, especificadas da mesma forma que quando você criou o item pela primeira vez. Chamar esse método sem um novo conjunto de coordenadas retornará as coordenadas atuais do item. Para mover um ou mais itens em uma determinada quantidade horizontal ou vertical de sua localização atual, você pode usar o método "mover".

Todos os itens são ordenados de cima para baixo no que é chamado de ordem de empilhamento. Se um item posterior na ordem de empilhamento sobrepor as coordenadas de um item abaixo dele, o item superior será desenhado sobre o item inferior. Os métodos "aumentar" e "diminuir" permitem ajustar a posição de um item na ordem de empilhamento.

Existem várias outras operações descritas na página do manual de referência, tanto para modificar itens quanto para recuperar informações adicionais sobre eles.

Rolagem

Em muitos aplicativos, você desejará que a tela seja maior do que aparece na tela. Você pode anexar barras de rolagem horizontais e verticais à tela da maneira usual, por meio do "xview" e

métodos "yview".

No que diz respeito ao tamanho da tela, você pode especificar o tamanho que gostaria que ela tivesse na tela, bem como qual é o tamanho total da tela, o que exigiria rolagem para ver. As opções de configuração de "largura" e "altura" para o widget de tela solicitarão a quantidade de espaço fornecida do gerenciador de geometria. A opção de configuração "scrollregion" (por exemplo, "0 0 1000 1000") informa a Tk o tamanho da superfície da tela.

Você deve ser capaz de modificar o programa sketchpad para adicionar rolagem, considerando o que você já sabe. De uma chance.

Depois de fazer isso, role a tela um pouco para baixo e tente desenhar. Você verá que a linha que está desenhando aparece *acima* de onde o mouse está apontando! Surpreso?

O que está acontecendo é que o comando global "bind" não sabe que a tela está rolada (ele não conhece os detalhes de nenhum widget específico). Portanto, se você rolou a tela para baixo em 50 pixels e clicar no canto superior esquerdo, o bind informará que você clicou em (0,0). Mas sabemos que por causa da rolagem, essa posição deveria ser realmente (0,50).

Os métodos "canvasx" e "canvasy" traduzirão a posição na tela (que o bind está relatando) no ponto real na tela, levando em consideração a rolagem. Se você os estiver adicionando diretamente às associações de evento (em oposição aos procedimentos chamados das associações de evento), tenha cuidado com aspas e substituições, para garantir que as conversões sejam feitas quando o evento for acionado.

Aqui, então, está o nosso exemplo completo. Provavelmente não queremos que a paleta seja rolada quando a tela for rolada, mas deixaremos isso para outro dia.

```
from tkinter import * from
tkinter import ttk root = Tk()
```

```
h = ttk.Scrollbar(root, orient=HORIZONTAL) v =
ttk.Scrollbar(root, orient=VERTICAL) canvas = Canvas(root,
scrollregion=(0, 0, 1000, 1000), yscrollcommand=v.set, xscrollcommand=h.set) h['command'] =
canvas.xview v['command'] = canvas.yview ttk.Sizegrip(root).grid(column=1, linha=1, sticky=(S,E))
```

```
canvas.grid(column=0, row=0, sticky=(N,W,E,S)) h.grid(column=0,
row=1, sticky=(W,E)) v.grid(column= 1, linha=0, sticky=(N,S))
root.grid_columnconfigure(0, peso=1) root.grid_rowconfigure(0,
peso=1)
```

último x, último = 0, 0

```
def xy(evento):
    global lastx, lasty
    lastx = canvas.canvasx(event.x), canvas.canvasy(event.y)

def setColor(nova cor):
    cor global cor =
    nova cor
```

```

canvas.dtag('all', 'paletteSelected')
canvas.itemconfigure('palette', outline='white')
canvas.addtag('paletteSelected', 'withtag', 'palette%s' % color)
canvas.itemconfigure( 'paletteSelected', contorno=#999999')

def addLine(evento):
    global lastx, lasty x, y =
    canvas.canvasx(event.x), canvas.canvasy(event.y) canvas.create_line((lastx,
    lasty, x, y), fill=color, width =5, tags='currentline') lastx, lasty = x, y

def doneStroke(evento):
    canvas.itemconfigure('currentline', width=1)

canvas.bind("<Button-1>", xy)
canvas.bind("<B1-Motion>", addLine)
canvas.bind("<B1-ButtonRelease>", doneStroke)

id = canvas.create_rectangle((10, 10, 30, 30), fill="red", tags=('palette', 'palettered')) canvas.tag_bind(id,
    "<Button-1>", lambda x : setColor("vermelho")) id = canvas.create_rectangle((10, 35, 30, 55), fill="azul",
    tags=('paleta', 'paletaazul')) canvas.tag_bind(id, "< Button-1>", lambda x: setColor("blue")) id =
    canvas.create_rectangle((10, 60, 30, 80), fill="black", tags=('palette', 'paletteblack', ' paletteSelected'))
    canvas.tag_bind(id, "<Button-1>", lambda x: setColor("black"))

setColor('black')
canvas.itemconfigure('palette', width=5) root.mainloop()

```

Outros tipos de itens

Além de linhas e retângulos, há vários tipos diferentes de itens que os widgets de tela suportam. Lembre-se que cada um possui seu próprio conjunto de opções de configuração de itens, detalhado no manual de referência.

Itens do tipo "linha" podem ser um pouco mais sofisticados do que vimos. Na verdade, um item de linha pode ser uma série de segmentos de linha, não apenas um; em nosso exemplo, poderíamos ter escolhido usar um único item de linha para cada curso completo. A linha também pode ser desenhada diretamente ponto a ponto ou suavizada em uma linha curva.

Itens do tipo "retângulo" que vimos. Itens do tipo "oval" funcionam da mesma forma, mas desenham como um oval. Itens do tipo "arco" permitem que você desenhe apenas um pedaço de um oval. Itens do tipo "polígono" permitem desenhar um polígono fechado com qualquer número de lados.

As imagens podem ser adicionadas aos widgets de tela, usando itens do tipo "bitmap" (para preto e branco) ou do tipo "imagem" (para cores).

Você pode adicionar texto a uma tela usando itens do tipo "texto". Você tem controle total da fonte,

tamanho, cor e muito mais, bem como o texto real que é exibido.

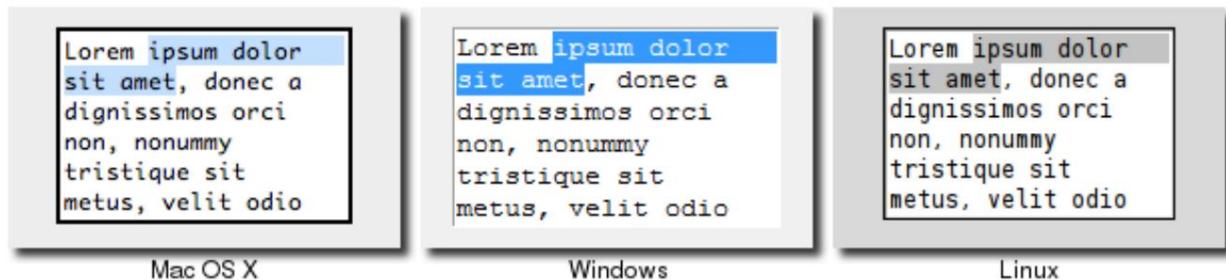
Talvez o mais interessante seja que você pode incorporar outros widgets (que incluiriam um quadro que contém outros widgets) em uma tela usando um item do tipo "janela". Quando fazemos isso, a tela atua como um gerenciador de geometria para esses outros widgets. Esse recurso gera todos os tipos de possibilidades para sua aplicação.

Há muito mais em widgets de tela do que descrevemos aqui; certifique-se de consultar o manual de referência, bem como a ampla gama de programas de exemplo incluídos na distribuição Tk.

Texto

- [Resumo de widgets](#)
- [Manual de Referência](#)

Um widget **de texto** gerencia uma área de texto com várias linhas. Como o widget de tela, o widget de texto do Tk é uma ferramenta extremamente flexível e poderosa que pode ser usada para uma ampla variedade de tarefas. Alguns exemplos de uso de widgets de texto incluem qualquer coisa, desde fornecer uma área de texto simples de várias linhas como parte de um formulário, até um editor de código estilizado, um esboço e um navegador da web. Os widgets de texto fazem parte dos widgets Tk clássicos, não dos widgets Tk temáticos.



Widgets de texto

Widgets de texto são criados usando a função **Texto** :

```
texto = Texto(pai, largura=40, altura=10)
```

Embora tenhamos apresentado brevemente os widgets de texto em um capítulo anterior, aqui entraremos em muito mais detalhes, para lhe dar uma noção do nível de sofisticação que eles permitem. Ainda assim, para qualquer trabalho significativo com o widget de texto, o [manual de referência](#) é muito bem organizado e útil.

O básico

Se você deseja apenas usar o widget de texto para obter um texto simples de várias linhas do usuário como parte de um formulário, há apenas algumas coisas com as quais você precisa se preocupar: criar e dimensionar o widget (verifique), fornecer um valor inicial para o texto no formulário e recuperar o texto no widget depois que o usuário enviar o formulário.

Fornecimento de conteúdo inicial

Quando você o cria pela primeira vez, o widget de texto não tem nada nele, portanto, se você deseja fornecer uma parte inicial do texto, terá que adicioná-lo você mesmo. Ao contrário, por exemplo, do widget de entrada, os widgets de texto não suportam uma opção de configuração "textvariable"; como veremos em breve, os widgets de texto podem conter muito mais do que apenas texto simples, portanto, uma variável simples não é suficiente para armazenar tudo.

Em vez disso, para definir o texto inicial do widget, você usará o método "inserir" do widget:

```
text.insert('1.0', 'aqui está o meu texto para inserir')
```

O "1.0" aqui representa onde inserir o texto e pode ser lido como "linha 1, caractere 0". Isso se refere ao primeiro caractere da primeira linha; para convenções históricas relacionadas a como os programadores normalmente se referem a linhas e caracteres, os números das linhas são baseados em 1 e os números dos caracteres são baseados em 0.

O texto a inserir é apenas uma string. Como o widget pode conter texto de várias linhas, a string fornecida também pode ser de várias linhas. Para fazer isso, simplesmente incorpore os caracteres "\n" (nova linha) em sua string nos locais apropriados.

Rolagem

Barras de rolagem , horizontais e verticais, podem ser anexadas ao widget de texto. Isso funciona exatamente da mesma forma que usar barras de rolagem em qualquer outro widget, como uma caixa de listagem ou tela.

Você também pode pedir ao widget para garantir que uma determinada parte do texto esteja visível. Por exemplo, se você adicionou mais texto ao widget do que caberá na tela (para que ele role), mas deseja garantir que a parte superior do texto, e não a parte inferior, esteja visível, chame o método "ver", passando o método posição do texto (por exemplo, "1.0").

Controle de embalagem

E se algumas linhas de texto no widget forem muito longas, maiores que a largura do widget? Por padrão, o texto irá apenas passar para a próxima linha, mas se e como isso pode ser controlado pela opção de configuração "wrap". O valor padrão é "char", o que significa contornar o caractere no final da linha; outras opções são "palavra" para causar quebra automática, mas apenas em quebras de palavras (por exemplo, espaços) e "nenhuma" significando que não há quebra automática. No último caso, parte do texto não ficará visível, a menos que você anexe uma barra de rolagem horizontal ao widget.

Desativando o widget

Alguns formulários desativarão temporariamente a edição em determinados widgets, a menos que certas condições sejam atendidas (por exemplo, algumas outras opções são definidas para um determinado valor). Para evitar que o usuário faça alterações em um widget de texto, defina a opção de configuração "estado" como "desativada"; reactive a edição definindo esta opção de volta para "normal".

Recuperando o Texto

Por fim, depois que o usuário fizer as alterações e enviar o formulário, seu programa desejará

recupere o conteúdo do widget, o que é feito com o método "get":

```
thetext = text.get('1.0', 'end')
```

Modificando o texto no código

Embora o usuário possa modificar o texto no widget de texto interativamente, seu programa também pode fazer alterações. A adição de texto é feita com o método "insert", que usamos acima para fornecer um valor inicial para o widget de texto.

Posições e índices de texto

Quando especificamos uma posição de "1.0" (primeira linha, primeiro caractere), este foi um exemplo de *índice*. Ele informa ao método insert onde inserir o novo texto (logo antes da primeira linha, primeiro caractere, ou seja, bem no início do widget). Há uma variedade de maneiras de especificar esses índices. Você também viu outro: o "end" (do exemplo "get") significa apenas após o final do texto ("logo após" porque as inserções de texto vão logo antes do índice fornecido, então inserir no "end" irá adicione texto ao final do widget). Observe que Tk *sempre* adicionará uma nova linha no final do widget de texto.

Aqui estão alguns exemplos adicionais de índices e o que eles significam:

3.fim A nova linha no final da linha 3.

1.0 + 3 caracteres Três caracteres após o início da linha 1. 2.end -1

caracteres O último caractere antes da nova linha na linha 2. end -1 caracteres A

nova linha que Tk sempre adiciona no final do texto. end -2 caracteres O último caractere real do texto. end -1 linhas O início da última linha real do texto. 2.2 + 2 linhas O terceiro caractere (índice 2) da quarta linha do texto. 2.5 wordstart O primeiro caractere da linha 2.

2.5 lineend

A posição da nova linha no final da linha 2. 2.5 wordstart

O primeiro caractere da palavra que contém o caractere no índice 2.5.

2,5 final de palavra O primeiro caractere logo após o último caractere da palavra que contém o índice 2.5.

Algumas coisas adicionais a ter em mente:

- O termo "chars" pode ser abreviado como "c" e "lines" como "l". • Você pode omitir os espaços entre os termos, por exemplo, "1.0+3c". • Se você especificar um índice após o final do widget (por exemplo, "end + 100c"), ele será interpretado como o fim.
- A adição de caracteres passará para as próximas linhas conforme necessário; por exemplo, "1,0 + 10 caracteres" em uma linha com apenas cinco caracteres ficarão na segunda linha. • Ao usar índices contendo várias palavras, certifique-se de citá-los adequadamente para que Tk vê o índice inteiro como um único argumento.

- Ao mover para cima ou para baixo um certo número de linhas, isso é interpretado como linhas *lógicas*, onde cada linha é terminada apenas pelo "\n". Com linhas longas e quebra habilitadas, isso pode representar várias linhas na tela. Se você quiser mover para cima ou para baixo uma única linha no visor, você pode especificar isso como, por exemplo, "1,0 + 2 linhas de exibição".
- Para determinar a posição canônica real de um índice, utilize o método "index", passando a ele a expressão de índice, e ele retornará o índice correspondente na forma "*line.char*".
- Você pode comparar dois índices usando o método "compare", que permite verificar a igualdade, se um índice está depois do outro no texto, etc.

Excluindo texto

Enquanto o método "inserir" adiciona novo texto em qualquer lugar do widget, o método "excluir" o remove. Você pode especificar um único caractere a ser excluído (por índice) ou um intervalo de caracteres especificado pelo índice inicial e final. No último caso, os caracteres de (e incluindo) o índice inicial até *pouco antes* do índice final serão excluídos (portanto, o caractere no índice final não é excluído). Portanto, isso removeria uma única linha de texto (incluindo a nova linha à direita) do início do texto:

```
text.delete('1.0', '2.0')
```

Há também um método "substituir", usando um índice inicial, um índice final e uma string como parâmetros. Ele faz o mesmo que uma exclusão, seguida de uma inserção no mesmo local.

Exemplo: Janela de registro

Aqui está um pequeno exemplo ilustrando como usar um widget de texto como uma janela de registro de 80x24 para seu aplicativo. O usuário não edita o widget de texto; em vez disso, seu programa gravará mensagens de log nele. Você gostaria de manter o conteúdo em não mais de 24 linhas (portanto, sem rolagem), portanto, ao adicionar novas mensagens no final, você terá que remover as antigas do topo se já houver 24 linhas.

```
da importação tkinter * da
importação ttk

root = Tk() log
= Text(root, state='disabled', width=80, height=24, wrap='none') log.grid()

def writeToLog(msg):
    numlines = log.index('end - 1 line').split('.')[0] log['state'] = 'normal' if
    numlines==24: log.delete(1.0 , 2.0) if log.index('end-1c')!=1.0':
        log.insert('end', '\n') log.insert('end', msg) log['state'] = 'desabilitado'
```

Observe que, como o widget foi desativado, tivemos que reativá-lo para fazer qualquer alteração, mesmo de nosso programa.

Formatação com Tags

Até agora, lidamos apenas com texto simples. Agora é hora de ver como adicionar formatação especial, como negrito, itálico, tachado, cores de fundo, tamanhos de fonte e muito mais. O widget de texto do Tk os implementa usando um recurso chamado *tags*.

Tags são objetos associados ao widget de texto. Cada tag é referida por meio de um nome escolhido pelo programador. Cada tag pode ter várias opções de configuração diferentes; são coisas como fontes, cores, etc. que serão usadas para formatar o texto. Embora as tags sejam objetos com estado, elas não precisam ser criadas explicitamente; eles serão criados automaticamente na primeira vez que o nome da tag for usado.

Adicionando tags ao texto

As tags podem ser associadas a um ou mais intervalos de texto no widget. Como antes, eles são especificados por meio de índices; um único índice para representar um único caractere e um índice inicial e final para representar o intervalo do caractere inicial até um pouco antes do caractere final. Tags podem ser adicionadas a intervalos de texto usando o método "tag add", por exemplo

```
text.tag_add('highlightline', '5.0', '6.0')
```

As tags também podem ser fornecidas ao inserir texto adicionando um parâmetro opcional ao método "inserir", que contém uma lista de uma ou mais tags para adicionar ao texto que você está inserindo, por exemplo

```
text.insert('fim', 'hovo material a ser inserido', ('linha de destaque', 'recente', 'aviso'))
```

Conforme o texto do widget é modificado, seja pelo usuário ou pelo seu programa, as tags vão se ajustando automaticamente. Por exemplo, se você marcou o texto "the quick brown fox" com a tag "nounphrase" e substituiu a palavra "quick" por "speedy", a tag ainda se aplicaria à frase inteira.

Aplicando formatação a tags

A formatação é aplicada às tags por meio de opções de configuração; eles funcionam de maneira semelhante às opções de configuração de todo o widget. Como um exemplo:

```
text.tag_configure('highlightline', background='amarelo', font='helvetica 14 negrito', relevo='levantado')
```

As opções de configuração atualmente disponíveis para tags são: "background", "bgstipple", "borderwidth", "elide", "fgstipple", "font", "foreground", "justify", "lmargin1", "lmargin2", "offset", "overstrike", "relief", "rmargin", "spacing1", "spacing2", "spacing3", "tabs", "tabstyle", "sublinhado" e "wrap". Verifique o manual de referência para obter descrições detalhadas sobre eles. O método "tag cget" permite consultar as opções de configuração de um tag.

Como várias tags podem ser aplicadas ao mesmo intervalo de texto, existe a possibilidade de conflito (por exemplo, duas tags especificando fontes diferentes). Uma ordem de prioridade é usada para resolvê-los; as tags criadas mais recentemente têm a prioridade mais alta, mas as prioridades podem ser reorganizadas usando os métodos "aumento de tag" e "diminuição de tag".

Mais manipulações de tags

Para excluir uma tag completamente, você pode usar o método "tag delete". Isso também remove quaisquer referências à tag no texto. Você também pode remover uma tag de um intervalo de texto usando o método "tag remove"; mesmo que isso não deixe intervalos de texto com essa tag, o próprio objeto da tag ainda existirá.

O método "tag ranges" retornará uma lista de intervalos no texto ao qual a tag foi aplicada.

Existem também os métodos "tag nexrange" e "tag prevrange" para pesquisar para frente ou para trás o primeiro intervalo a partir de uma determinada posição.

O método "nomes de tags", chamado sem parâmetros adicionais, retornará uma lista de todas as tags atualmente definidas no widget de texto (incluindo aquelas que não podem ser usadas atualmente). Se você passar um índice para o método, ele retornará a lista de tags aplicadas apenas ao caractere no índice.

Finalmente, você pode usar o primeiro e o último caractere do texto com uma determinada tag como índices, da mesma forma que você pode usar "end" ou "2.5". Para isso, basta especificar "*tagname.first*" ou "*tagname.last*".

Diferenças entre tags na tela e widgets de texto

Embora os widgets de tela e texto suportem "tags" que podem ser usadas para aplicar a vários objetos, estilizá-los e assim por diante, essas tags não são a mesma coisa. Existem diferenças importantes a serem observadas.

Em widgets de tela, itens de tela individuais possuem opções de configuração que controlam sua aparência.

Quando você se refere a uma tag em uma tela, o significado disso é idêntico a "todos os itens da tela que atualmente possuem essa tag". A tag em si não existe como um objeto separado. Portanto, no trecho a seguir, o último retângulo adicionado *não* será colorido em vermelho.

```
canvas.itemconfigure('important', fill='red')
canvas.create_rectangle(10, 10, 40, 40, tags=('important'))
```

Em contraste, em widgets de texto, não são os caracteres individuais que retêm as informações de estado sobre a aparência, mas as tags, que são objetos por si só. Portanto, neste snippet, o texto recém-adicionado *será* colorido em vermelho.

```
text.insert('fim', 'primeiro texto', ('importante'))
text.tag_configure('importante', foreground='vermelho')
text.insert('fim', 'segundo texto', ('importante'))
```

Eventos e Ligações

Uma coisa bem legal é que você pode definir ligações de eventos em tags. Isso permite que você faça coisas como reconhecer facilmente os cliques do mouse apenas em determinados intervalos de texto e abrir um menu ou caixa de diálogo em resposta. Tags diferentes podem ter associações diferentes, portanto, você evita o incômodo de resolver questões como "o que significa um clique neste local?". As vinculações nas tags são implementadas usando o método "tag bind":

```
text.tag_bind('importante', '<1>', popupImportanteMenu)
```

A vinculação de todo o widget a eventos funciona da mesma forma que para todos os outros widgets. Além dos eventos normais de baixo nível, também serão gerados dois eventos virtuais: <Modified> sempre que for feita uma alteração no conteúdo do widget, e <Selection> sempre que houver uma alteração feita no texto selecionado.

Selecionando Texto

Seu programa pode querer saber se um intervalo de texto foi selecionado pelo usuário e, em caso afirmativo, qual é esse intervalo. Por exemplo, você pode ter um botão na barra de ferramentas para colocar em negrito o texto selecionado em um editor. Embora você possa saber quando a seleção foi alterada (por exemplo, para atualizar se o botão em negrito está ou não ativo) por meio do evento virtual <Selection>, isso não informa o que foi selecionado.

O widget de texto mantém automaticamente uma tag chamada "sel", que se refere ao texto selecionado.

Sempre que a seleção mudar, a tag "sel" será atualizada. Assim, você pode encontrar o intervalo de texto selecionado usando o método "tag ranges", passando "sel" como o tag para relatar.

Da mesma forma, você pode alterar a seleção usando "tag add" para definir uma nova seleção ou "tag remove" para remover a seleção. Você não pode excluir a tag "sel", é claro.

Embora as ligações padrão do widget evitem que isso aconteça, "sel" é como qualquer outra tag, pois pode suportar vários intervalos, ou seja, seleções separadas. Para evitar que isso aconteça ao alterar a seleção do seu código, certifique-se de remover qualquer seleção antiga antes de adicionar uma nova.

O widget de texto gerencia o conceito do cursor de inserção (onde o texto recém-digitado aparecerá) separado da seleção. Ele faz isso usando um novo conceito chamado *marca*.

Marcas

Marcas são usadas para indicar um lugar específico no texto. A esse respeito, eles são como índices, exceto que, à medida que o texto é modificado, a marca se ajustará para estar no mesmo local relativo. Dessa forma, eles se assemelham a tags, mas referem-se a uma única posição em vez de um intervalo de texto. Na verdade, as marcas não se referem a uma posição ocupada por um caractere no texto, mas especificam uma posição *entre* dois caracteres.

Tk mantém automaticamente duas marcas diferentes. O primeiro é denominado "insert", e é o presente

localização do cursor de inserção. À medida que o cursor é movido (via mouse ou teclado), a marca se move com ele. A segunda marca é denominada "atual" e reflete a posição do caractere abaixo da posição atual do mouse.

Para criar suas próprias marcas, use o método "conjunto de marcas" do widget, passando o nome da marca e um índice (a marca é posicionada logo antes do caractere no índice fornecido). Isso também é usado para mover uma marca existente para uma posição diferente. As marcas podem ser removidas usando o método "marca desmarcada", passando o nome da marca. Se você excluir um intervalo de texto contendo uma marca, isso também removerá a marca.

O nome de uma marca também pode ser usado como um índice (da mesma forma que "1.0" ou "end-1c" são índices).

Você pode encontrar a próxima marca (ou anterior) de um determinado índice no texto usando os métodos "marcar próximo" ou "marcar anterior". O método "marcar nomes" retornará uma lista dos nomes de todas as marcas.

As marcas também têm uma *gravidade*, que pode ser modificada com o método "gravidade da marca", que afeta o que acontece quando o texto é inserido na marca. Suponha que temos o texto "ac", com uma marca no meio que simbolizaremos com um tubo, ou seja, "a|c". Se a gravidade dessa marca for "certa" (o padrão), a marca será anexada ao "c". Se o novo texto "b" for inserido na marca, a marca ficará presa ao "c", e assim o novo texto será inserido antes da marca, ou seja, "ab|c". Se a gravidade for "esquerda", a marca será anexada ao "a" e, portanto, um novo texto será inserido após a marca, ou seja, "a|bc".

Imagens e Widgets

Como os widgets de tela, os widgets de texto podem conter não apenas texto, mas também imagens e quaisquer outros widgets Tk (incluindo um quadro em si contendo muitos outros widgets). Em alguns sentidos, isso permite que o widget de texto funcione como um gerenciador de geometria por conta própria. A capacidade de adicionar imagens e widgets dentro do texto abre um mundo de possibilidades para o seu programa.

As imagens são adicionadas a um widget de texto em um determinado índice, com a imagem normalmente especificada como uma imagem Tk existente. Existem também outras opções que permitem ajustar o preenchimento e assim por diante.

```
flores = PhotoImage(file='flores.gif')
text.image_create('sel.first', image=flores)
```

Os widgets são adicionados a um widget de texto praticamente da mesma forma que as imagens. O widget que você está adicionando deve ser um descendente do widget de texto na hierarquia geral da janela.

```
b = ttk.Button(text, text='Push Me')
text.window_create('1.0', window=b)
```

Ainda mais

Há muito mais coisas que o widget de texto pode fazer; aqui vamos mencionar brevemente apenas mais alguns deles. Para obter detalhes sobre como usar qualquer um deles, consulte o manual de referência.

Procurar

O widget de texto inclui um poderoso método de "pesquisa" que permite localizar um trecho de texto dentro do widget; isso é útil para uma caixa de diálogo "Localizar", como um exemplo óbvio. Você pode pesquisar para trás ou para frente a partir de uma determinada posição ou dentro de um determinado intervalo, especificar sua pesquisa usando texto exato, sem distinção entre maiúsculas e minúsculas ou usando expressões regulares, localizar uma ou todas as ocorrências do seu termo de pesquisa e muito mais.

Modificações, Desfazer e Refazer

O widget de texto acompanha se foram ou não feitas alterações no texto (útil para saber se você precisa salvá-lo, por exemplo), que você pode consultar (ou alterar) usando o método "editar modificado". Há também um mecanismo completo de desfazer/refazer em vários níveis, gerenciado automaticamente pelo widget quando você define sua opção de configuração "desfazer" como verdadeira. Chamar "editar desfazer" ou "editar refazer" modificará o texto atual usando as informações armazenadas na pilha de desfazer/refazer.

Elimine o texto

Na verdade, você pode incluir texto no widget que não é exibido; isso é conhecido como texto "eliminado" e é disponibilizado usando a opção de configuração "eliminar" para tags. Você pode usar isso para implementar, por exemplo, um esboço, um editor de código "dobrável" ou apenas para enterrar alguns metadados extras misturados ao seu texto. Ao especificar o posicionamento com texto elidido, você deve ser um pouco mais cuidadoso e, portanto, os comandos que lidam com posições têm opções extras para incluir ou ignorar o texto elidido.

Introspecção

Como a maioria dos widgets Tk, o widget de texto faz de tudo para expor informações sobre seu estado interno; vimos muito disso em termos do método "get", opções de configuração de widget, "nomes" e "cget" para tags e marcas e assim por diante. Há ainda mais informações disponíveis que você pode usar para uma ampla variedade de tarefas. Confira os métodos "debug", "dlineinfo", "bbox", "count" e "dump" no manual de referência.

Peering O

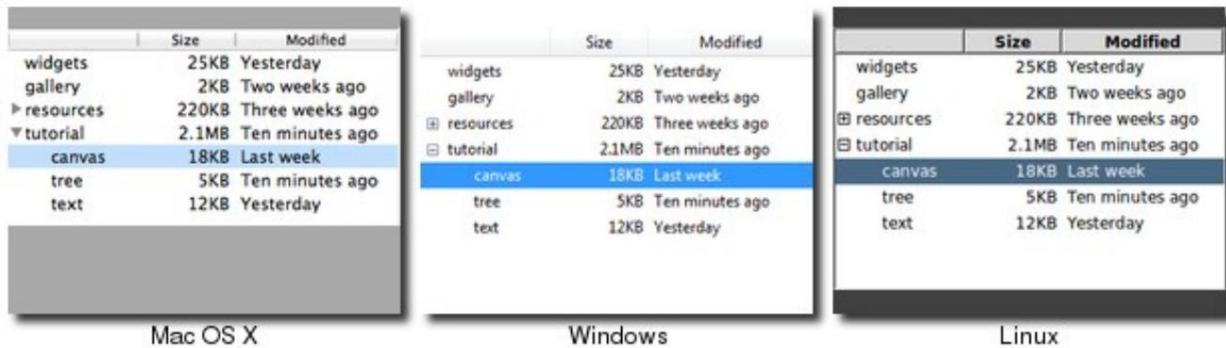
widget de texto Tk permite que os mesmos dados de texto subjacentes (contendo todo o texto, marcas, tags, imagens e assim por diante) sejam compartilhados entre dois ou mais widgets de texto diferentes. Isso é conhecido como *peering* e é controlado por meio do método "peer".

Árvore

- [Resumo de widgets •](#)

[Manual de Referência](#)

Um widget de **exibição em árvore** pode exibir e permitir a navegação por uma hierarquia de itens e pode mostrar um ou mais atributos de cada item como colunas à direita da árvore. Ele permite que você crie interfaces de usuário semelhantes à exibição em árvore encontrada em gerenciadores de arquivos como o OS X Finder ou o Windows Explorer. Como na maioria dos widgets Tk, há muita flexibilidade para fazê-lo se comportar conforme necessário para uma ampla gama de situações.



Widgets de visualização em árvore

Widgets Treeview são criados usando a função **ttk.Treeview** :

árvore = `ttk.Treeview(pai)`

Barras de rolagem horizontais e verticais podem ser adicionadas da maneira usual, se desejado.

Adicionando itens à árvore

Para fazer algo útil com a exibição em árvore, você precisará adicionar um ou mais *itens* a ela. Cada item representa um único nó na árvore, seja um nó folha ou um nó interno. Os itens são referidos por um id único; isso pode ser atribuído pelo programador quando o item é criado pela primeira vez ou o widget pode escolher automaticamente um id para o item.

Os itens são criados inserindo-os na árvore, usando o método "inserir" do treeview. Para inserir um item, você precisa saber onde inseri-lo na árvore, o que significa especificar o item pai, bem como em qual posição na lista de filhos do pai o novo item deve ser inserido.

O widget treeview cria automaticamente um nó raiz (que não é exibido), tendo o id de "{}" (ou seja, a string vazia), que pode servir como o pai do primeiro nível de itens adicionados.

As posições dentro da lista de filhos do pai são especificadas pelo índice (0 sendo o primeiro, com o índice especial "end" significando a inserção depois de todos os filhos existentes).

Normalmente, você também especificará o nome de cada item, que é exibido na árvore. Existem outras opções para adicionar uma imagem ao lado do nome, especificar se o nó está aberto ou fechado e assim por diante.

```
# Inserido na raiz, o programa escolhe o id: tree.insert("",  
'end', 'widgets', text='Widget Tour')  
  
# Mesma coisa, mas inserido como primeiro filho:  
tree.insert("", 0, 'gallery', text='Applications')  
  
# Treeview escolhe o id: id =  
tree.insert("", 'end', text='Tutorial')  
  
# Inserido abaixo de um nó existente:  
tree.insert('widgets', 'end', text='Canvas') tree.insert(id, 'end',  
text='Tree')
```

Inserir o item retorna o id do item recém-criado.

Reorganizando itens

Um nó (e seus descendentes, se houver) pode ser movido para um local diferente na árvore; a única restrição é que um nó não pode ser movido para baixo de um de seus descendentes. O local de destino é especificado via pai e índice na lista de filhos, como foi feito com "inserir".

```
tree.move('widgets', 'gallery', 'end') # mover widgets sob a galeria
```

Os itens podem ser *desconectados* da árvore, o que remove o item e seus descendentes da hierarquia, mas não destrói os itens, permitindo que você os reinsira posteriormente com "mover".

```
tree.detach('widgets')
```

Os itens também podem ser *excluídos*, o que elimina completamente o item e seus descendentes.

```
tree.delete('widgets')
```

Se você quiser navegar na hierarquia, existem métodos que permitem encontrar o pai de um item ("pai"), encontrar os irmãos próximos ou anteriores de um item ("próximo" e "anterior") e retornar a lista de filhos de um item ("filhos").

Você pode controlar se o item está aberto ou não e mostra seus filhos modificando a opção de configuração do item "aberto".

```
tree.item('widgets', open=TRUE) isopen =  
tree.item('widgets', 'open')
```

Exibição de informações para cada item

A exibição em árvore também pode exibir uma ou mais informações adicionais sobre cada item, que são mostradas como colunas à direita da exibição da árvore principal.

Novamente, cada coluna é referenciada por um nome simbólico que você atribui. Você pode especificar a lista de

colunas usando a opção de configuração "colunas" do widget treeview, seja ao criar o widget pela primeira vez ou posteriormente.

```
árvore = ttk.Treeview(raiz, colunas=('tamanho', 'modificado')) árvore['colunas'] = ('tamanho', 'modificado', 'proprietário')
```

Você pode especificar a largura da coluna, como a exibição das informações do item na coluna é alinhada e muito mais. Você também pode fornecer informações sobre o cabeçalho da coluna, como o texto a ser exibido, uma imagem opcional, alinhamento e um script a ser invocado quando o item for clicado (por exemplo, para classificar a árvore).

```
tree.column('size', width=100, âncora='center') tree.heading('size', text='Size')
```

Os valores a serem colocados em cada coluna para cada item podem ser especificados individualmente ou fornecendo uma lista de valores para o item. Neste último caso, isso é feito usando a opção de configuração do item "valores" (e portanto pode ser usada na primeira inserção do item ou posteriormente) que leva uma lista dos valores; a ordem da lista deve ser a mesma da opção de configuração do widget "colunas".

```
tree.set('widgets', 'size', '12KB') size = tree.set('widgets', 'size') tree.insert("", 'end', text='Listbox', values=( 'Marca de ontem de 15 KB'))
```

Aparência de itens e eventos

Assim como os widgets de texto e tela, o widget de visualização em árvore usa *tags* para ajudá-lo a modificar a aparência dos itens na árvore. Você pode atribuir uma lista de tags para cada item usando a opção de configuração de item "tags" (novamente, ao criar o item ou posteriormente).

As opções de configuração de tags podem ser especificadas, que serão aplicadas a todos os itens que possuem essa tag. As opções de tags válidas incluem "foreground" (cor do texto), "background", "font" e "image" (não usado se o item especificar sua própria imagem).

Você também pode criar associações de eventos em tags, que permitem capturar cliques do mouse, eventos de teclado etc.

```
tree.insert("", 'end', text='button', tags=('ttk', 'simples')) tree.tag_configure('ttk', background='yellow') tree.tag_bind('ttk', '<1>', itemClicked) # o item clicado pode ser encontrado via tree.focus()
```

A treeview irá gerar os eventos virtuais "<TreeviewSelect>", "<TreeviewOpen>" e "<TreeviewClose>" que permitem monitorar as alterações feitas no widget pelo usuário. Você pode usar o método de "seleção" para determinar a seleção atual (a seleção também pode ser alterada em seu programa).

Personalizando a exibição

Há muitos aspectos de como o widget de exibição em árvore é exibido que você pode personalizar. Alguns deles já vimos, como o texto dos itens, fontes e cores, nomes dos cabeçalhos das colunas e muito mais. Aqui estão alguns adicionais.

- Especifique o número desejado de linhas a serem exibidas usando a configuração do widget "altura" opção.
- Controle a largura de cada coluna usando as opções "width" ou "minwidth" da coluna.
A coluna que contém a árvore pode ser acessada com o nome simbólico "#0". A largura geral solicitada para o widget é baseada na soma das larguras das colunas.
- Escolha quais colunas exibir e a ordem para exibi-las usando o
opção de configuração do widget "displaycolumns".
- Você pode, opcionalmente, ocultar um ou ambos os títulos das colunas ou a própria árvore (deixando apenas as colunas) usando a opção de configuração do widget "mostrar" (o padrão é "títulos da árvore" para mostrar ambos).
- Você pode especificar se um único item ou vários itens podem ser selecionados pelo usuário por meio da opção de configuração do widget "selectmode", passando "browse" (item único), "extended" (vários itens, o padrão) ou "nenhum" .

Estilos e Temas

O aspecto "temático" dos novos widgets Ttk é um dos aspectos mais poderosos e empolgantes do novo conjunto de widgets. No entanto, como ele faz as coisas de maneira bem diferente de como Tk funcionou tradicionalmente e porque, ao tentar ser flexível, faz muitas coisas , certamente é o que mais confunde as pessoas.

Definições

Primeiro, definiremos alguns conceitos e termos nos quais os temas e estilos do Ttk se baseiam.

Classe de widget

Uma classe de widget é usada por Tk para identificar o tipo de um widget específico; essencialmente, seja um botão, um rótulo, uma tela, etc. No Tk clássico, todos os botões tinham a mesma classe ("Button"), todos os rótulos tinham a mesma classe ("Label"), etc.

Você pode usar essa classe de widget para introspecção e para alterar as opções globalmente por meio do banco de dados de opções. Isso pode permitir que você diga, por exemplo, que todos os botões, por padrão, têm um fundo vermelho.

Havia *alguns* widgets Tk clássicos, incluindo widgets de quadro e de nível superior, que permitiam alterar a classe de widget de um widget específico quando o widget foi criado pela primeira vez, passando a ele um

opção de configuração "classe". Portanto, embora normalmente os quadros tenham uma classe de widget "Frame", você pode especificar que um determinado widget de quadro tenha uma classe de widget "SpecialFrame".

Por causa disso, você pode usar o banco de dados de opções para definir aparências diferentes para *diferentes tipos* de widgets de quadro (não apenas todos os widgets de quadro ou widgets de quadro localizados em um local específico na hierarquia).

O que o Ttk faz é pegar essa ideia simples e dar a ela reforços de foguetes.

Estado do widget

Um *estado do widget* permite que um único widget tenha mais de uma aparência ou comportamento, dependendo de coisas como a posição do mouse, diferentes opções de estado definidas pelo aplicativo e assim por diante. No Tk clássico, vários widgets tinham opções de configuração de "estado" que permitiam defini-los como "normal" ou "desativado"; um estado "desativado" para um botão, por exemplo, desenharia seu rótulo em cinza. Alguns usaram um estado adicional, "ativo", novamente representando um comportamento diferente.

O próprio widget, ou mais tipicamente, as ligações de classe do widget, controlava como a aparência do widget mudava em diferentes estados, geralmente consultando as opções de configuração do widget como "foreground", "activeforeground" e "disabledforeground".

Ttk novamente estende e generaliza essa ideia básica do estado do widget, de duas maneiras importantes. Primeiro, em vez de serem específicos do widget, todos os widgets Ttk têm opções de estado e, na verdade, todos têm exatamente as mesmas opções de estado, acessadas pelos comandos de widget "state" e "instate".

Um estado de widget Ttk é, na verdade, um conjunto de sinalizadores de estado independentes, contendo zero ou mais dos seguintes sinalizadores: "ativo", "desativado", "foco", "pressionado", "selecionado", "plano de fundo", "somente leitura", "alternativo" ou "inválido" (consulte o [widget](#) página no manual de referência para os significados exatos).

Observe que, embora todos esses sinalizadores de estado estejam disponíveis para cada widget, eles não podem ser usados por cada widget. Por exemplo, é provável que um widget de rótulo ignore um sinalizador de estado "inválido" e, portanto, nenhuma aparência especial seria associada a esse sinalizador.

A segunda grande mudança que o Ttk faz é que ele toma a decisão do que mudar quando o estado é ajustado fora do controle do widget. Ou seja, um autor de widget não codificará mais a lógica para o efeito de "quando o estado estiver desabilitado, consulte a opção de configuração de primeiro plano desabilitado e use-a para a cor de primeiro plano". Com essa lógica codificada, não apenas tornou os widgets de codificação mais longos (e mais repetitivos), mas também restringiu como um widget poderia ser alterado com base em seu estado.

Ou seja, se o autor do widget não tivesse codificado na lógica para alterar a fonte quando o estado mudasse, você, como usuário do widget, estaria sem sorte.

Em vez de codificar essas decisões em cada widget, o Ttk move as decisões para um local separado: estilos. Isso significa que o autor do widget não precisa fornecer código para todas as opções de aparência possíveis, o que não apenas simplifica o widget, mas paradoxalmente garante que uma gama mais ampla de aparências possa ser definida , incluindo aquelas que o autor do widget pode não ter previsto.

Estilo

Isso nos leva então a definir um estilo de widget. Muito simplesmente, um *estilo* descreve a aparência (ou aparências) de uma classe de widget Ttk. Todos os widgets criados com essa classe de widget terão a(s) mesma(s) aparência(s). Embora cada widget temático tenha uma classe padrão (por exemplo, "TButton" para widgets "ttk::button"), você pode, ao contrário do Tk clássico, atribuir uma classe de widget diferente para qualquer widget temático que você criar. Isso é feito usando a opção de configuração "estilo" do Ttk, compatível com todos os widgets temáticos.

Portanto, um estilo define a aparência normal de widgets de uma determinada classe de widget, mas também pode definir variações dessa aparência que dependem do sinalizador de estado atual. Assim, por exemplo, um estilo pode especificar que, quando o sinalizador de estado "pressionado" é definido, a aparência deve mudar de uma maneira específica. Por isso, um estilo pode descrever uma ou mais formas de exibição do widget, dependendo do estado.

O restante deste capítulo irá aprofundar muito mais detalhadamente o que um estilo realmente é, mas pelo menos agora você sabe a responsabilidade que ele tem.

Temas

Você pode pensar em um *tema* como uma coleção de estilos. Embora cada estilo seja específico do widget (um para botões, outro para entradas, etc.), um tema reunirá muitos estilos. Normalmente, um tema definirá um estilo para cada tipo de widget, mas cada um desses estilos será projetado para que eles se "encaixem" visualmente uns com os outros - embora talvez infelizmente Ttk não restrinja tecnicamente um design ou julgamento ruim!

Usar um determinado tema para um aplicativo é realmente dizer que você gostaria de ter um conjunto de estilos definidos para que, por padrão, todos os diferentes tipos de widgets tenham uma aparência comum e se encaixem bem uns com os outros.

Usando estilos e temas

Agora sabemos o que os estilos e temas devem fazer, mas como exatamente os usamos? Para fazer isso, precisamos saber como nos referir a estilos e temas e como aplicá-los a um widget ou interface de usuário.

Nomes de estilo

Todo estilo tem um nome. Se for modificar um estilo, criar um novo ou usar um estilo para um widget, você precisa saber seu nome.

Como você sabe quais são os nomes dos estilos? Se você tiver um widget específico e quiser saber qual estilo ele está usando no momento, verifique primeiro o valor de sua opção de configuração "estilo". Se estiver vazio, significa que o widget está usando o estilo *padrão* para o widget. Você pode recuperá-lo por meio da classe do widget. Por exemplo:

```
>>> b = ttk.Button()
```

```
>>> b['estilo']
>>> b.winfo_class()
'TButton'
```

Portanto, neste caso, o estilo que está sendo usado é o "TButton". Os estilos padrão para outros widgets temáticos são nomeados de forma semelhante, por exemplo, "TEntry", " TLabel", "TSizeGrip", etc. É sempre bom verificar os detalhes; por exemplo, a classe do widget treeview é "Treeview", não "TTreeview".

Além dos estilos padrão, os estilos podem ter praticamente qualquer nome. Você pode criar seu próprio estilo (ou usar um tema que tenha um estilo) chamado "FunButton", "NuclearReactorButton" ou até mesmo "GuessWhatIAm" (não é uma escolha inteligente). Com mais frequência, você encontrará nomes como "Fun.TButton" ou "NuclearReactor.TButton", que sugerem variações de um estilo básico; como você verá, isso é algo que o Ttk suporta para criar e modificar estilos.

A capacidade de recuperar uma lista de todos os estilos atualmente disponíveis não é suportada no momento.

Usando um estilo

Usar um estilo significa aplicar esse estilo a um widget individual. Se você souber o nome do estilo que deseja usar e em qual widget aplicá-lo, é fácil. A definição do estilo pode ser feita no momento da criação:

```
b = ttk.Button(parent, text='Hello', style='Fun.TButton')
```

Além disso, você pode alterar o estilo de um widget a qualquer momento depois de criá-lo com a opção de configuração "estilo":

```
b['estilo'] = 'NuclearReactor.TButton'
```

Usando temas

Enquanto os estilos controlam a aparência de widgets individuais, os temas controlam a aparência de toda a interface do usuário. A capacidade de alternar entre temas é um dos recursos importantes dos widgets temáticos.

Assim como os estilos, os temas são identificados por um nome. Você pode obter os nomes de todos os temas disponíveis:

```
>>> s = ttk.Style() >>>
s.theme_names() ('aqua',
'step', 'clam', 'alt', 'default', 'classic')
```

Apenas um tema pode estar ativo por vez. Para obter o nome do tema atualmente em uso, você pode usar o seguinte:

Essa API, originalmente destinada ao Tk 8.6, foi portada novamente para o Tk 8.5.9. Se você estiver usando uma versão anterior do Tk, obter essas informações é um pouco mais complicado.

```
>>> s.theme_use()
'aqua'
```

A mudança para um novo tema pode ser feita com:

```
s.theme_use('nome do tema')
```

O que isso realmente faz? Obviamente, ele define o tema atual para o tema indicado. Fazer isso, portanto, substitui todos os estilos atualmente disponíveis pelo conjunto de estilos definido pelo tema. Por fim, atualiza todos os widgets, para que tenham a aparência descrita pelo novo tema.

O que há dentro de um estilo?

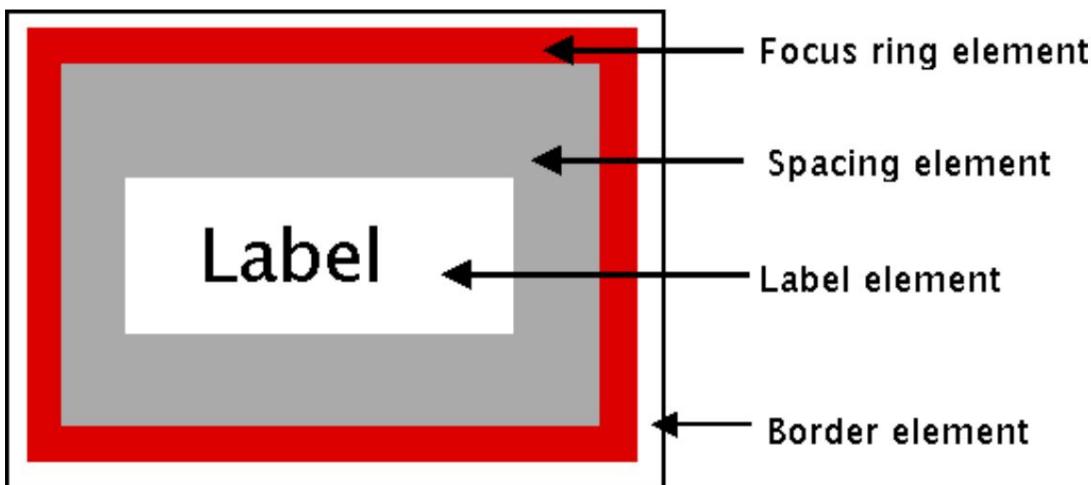
Se tudo o que você quer fazer é *usar* um estilo, agora você sabe tudo o que precisa. Se, no entanto, você deseja criar seus próprios estilos ou modificar um já existente, agora fica "interessante".

elementos

Embora cada estilo represente um único widget, cada widget é normalmente composto de peças menores, chamadas *de elementos*. É trabalho do autor do estilo construir o widget inteiro a partir desses elementos menores. O que são esses elementos depende do widget.

Aqui está um exemplo de um botão. Pode ter uma borda bem do lado de fora, que é um elemento.

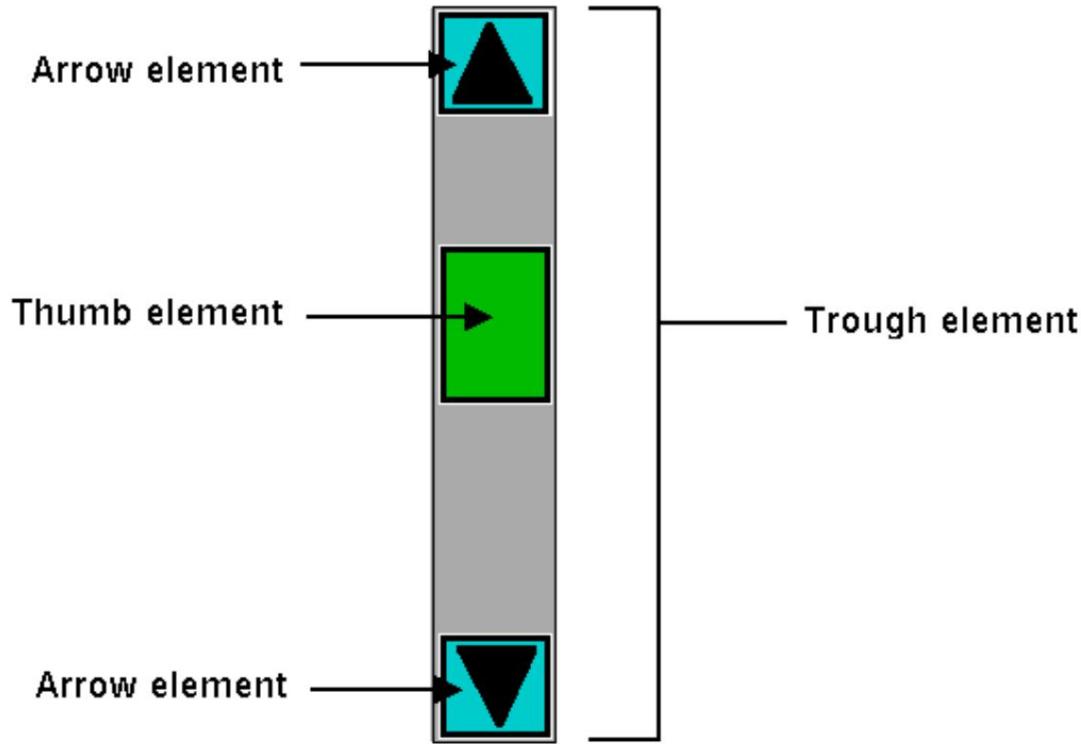
Dentro dele pode haver um anel de foco, que normalmente é apenas a cor de fundo, mas pode ser realçado quando o usuário pressiona o botão. Então esse é um segundo elemento. Pode haver algum espaçamento entre o anel de foco e o rótulo do botão. Então esse espaçamento seria um terceiro elemento. Por fim, há o rótulo do próprio botão, um quarto elemento.



Possíveis elementos de um botão.

Por que o autor do estilo pode ter dividido dessa maneira? Se você tiver uma parte do widget que pode estar em um local diferente de outro, ou pode ter uma cor diferente da outra, pode ser um bom candidato para um elemento. Observe que este é apenas um exemplo de como um botão pode ser construído a partir de elementos. Diferentes estilos e temas podem (e fazem) isso de maneiras diferentes.

Aqui está um segundo exemplo de uma barra de rolagem vertical, contendo um elemento "vale" contendo o restante, que inclui os elementos de seta para cima e para baixo em cada extremidade e um elemento "thumb" no meio.



Possíveis elementos de uma barra de rolagem.

Layout

Além da escolha de quais elementos fazem parte de um widget, um estilo também define como esses elementos são dispostos dentro do widget, ou seja, seu layout. No exemplo do botão, tínhamos um elemento de rótulo dentro de um elemento de espaçamento, dentro de um elemento de anel de foco, dentro de um elemento de borda. Então o layout lógico é assim:

```
borda
  { foco
    { espaçoamento { rótulo
      }
    }
  }
```

Podemos perguntar a Ttk como é o layout do estilo TButton assim:

```
>>> s.layout('TButton')
[("Button.border", {"children": [("Button.focus", {"children": [("Button.spacing", {"children": [("Button.label", {"sticky": "nswe"}), {"sticky": "nswe"})], "sticky": "nswe"})], "sticky": "nswe", "border": "1"}])]
```

Se limparmos e formatarmos um pouco, obteremos algo com esta estrutura:

```
Button.border -sticky nswe -border 1 -children {
  Button.focus -sticky nswe -children {
    Button.spacing -sticky nswe -children { Button.label
      -sticky nswe
    }
  }
}
```

```
}
```

Isso começa a fazer sentido; temos quatro elementos, denominados "Button.border", "Button.focus", "Button.spacing" e "Button.label". Cada um deles tem diferentes opções de elementos, como "-children", "-sticky" e "-border" que aqui especificam layout ou tamanhos. Sem entrar em muitos detalhes neste ponto, podemos ver claramente o layout aninhado, baseado nos atributos "-children" e "-sticky"; O Ttk usa uma versão simplificada do gerenciador de geometria "pack" do Tk para especificar o layout do elemento.

Opções de elemento

Cada um desses diferentes elementos pode ter várias opções diferentes. Por exemplo, a miniatura da barra de rolagem pode ter uma opção para definir sua cor de fundo ou outra para fornecer a largura de uma borda, se houver. Eles podem ser personalizados para ajustar a aparência dos elementos no widget.

Quais opções estão disponíveis para cada elemento? Aqui está um exemplo de verificação de quais opções estão disponíveis para o rótulo dentro do botão (que sabemos pelo comando "layout" é identificado como "Button.label"):

```
>>> s.element_options('Button.label') ('-compound',  
'-space', '-text', '-font', '-foreground', '-underline', '-width', '-anchor', '-justify', '-wraplength', '-embossed', '-image',  
'-stipple', '-background')
```

Presumivelmente, essas opções não devem ter o traço inicial.

Nas próximas seções, veremos a maneira não totalmente direta de trabalhar com opções de elemento.

Alterando opções de estilo

Nesta seção, veremos como você pode alterar a aparência do estilo modificando as opções de estilo.

Você pode fazer isso modificando um estilo existente ou, mais comumente, criando um novo estilo.

Modificando uma opção de estilo

A modificação de uma opção de configuração para um estilo existente é feita de maneira semelhante à modificação de qualquer outra opção de configuração, especificando o estilo, o nome da opção e o novo valor:

```
s.configure('TButton', font='helvetica 24')
```

Você aprenderá mais sobre quais são as opções válidas em breve.

Se você precisar recuperar o valor atual de uma opção, isso pode ser feito com o método "lookup".

```
>>> s.lookup('TButton', 'fonte') 'helvetica  
24'
```

Criando um novo estilo derivado

Se você modificar um estilo existente, como "TButton", essa modificação será aplicada a todos os widgets que usam esse estilo (por padrão, todos os botões). Isso pode muito bem ser o que você quer fazer.

Mais frequentemente, você está interessado em criar um novo estilo que seja semelhante a um existente, mas que varie em um determinado aspecto. Por exemplo, você gostaria que a maioria dos botões em seu aplicativo mantivessem sua aparência normal, mas criar alguns botões especiais de "emergência", que seriam realçados de uma maneira diferente. Nesse caso, criar um novo estilo (por exemplo, "Emergency.TButton") derivado do estilo base ("TButton") seria a coisa apropriada a fazer.

Ao acrescentar outro nome ("Emergência") seguido de um ponto em um estilo existente, você está criando implicitamente um novo estilo derivado do existente. Portanto, neste exemplo, nosso novo estilo terá exatamente as mesmas opções de um botão normal, exceto pelas diferenças indicadas:

```
s.configure('Emergency.TButton', font='helvetica 24', foreground='red',
padding=10)
```

Opções de estilo específicas do estado

Além das opções de configuração normais para o estilo, o autor do widget pode ter especificado opções diferentes para usar quando o widget estiver em um determinado estado de widget. Por exemplo, quando um botão está desabilitado, você gostaria que a cor do rótulo do botão ficasse esmaecida.

Para fazer isso, você pode especificar um "mapa", que permite especificar variações para uma ou mais opções de configuração de um estilo. Para cada opção de configuração, você pode especificar uma lista de estados do widget, juntamente com o valor específico que a opção deve ser atribuída quando o widget estiver nesse estado.

Lembre-se de que o estado é composto por um ou mais sinalizadores de estado (ou sua negação), conforme definido pelo método "state" do widget ou consultado por meio do método "instate".

O exemplo a seguir fornece as seguintes variações da aparência "normal" de um botão:

- quando o widget está no estado desativado, a cor de fundo deve ser definida como "#d9d9d9" • quando o widget está no estado ativo (passe o mouse sobre ele), a cor de fundo deve ser definida como "#ececce"
- quando o widget está desativado, a cor de primeiro plano deve ser definida como "#a3a3a3" (isso é um acréscimo à mudança de cor de fundo que já observamos) • quando o widget está no estado em que o botão é pressionado e o widget não está desativado, o alívio deve ser definido como "afundado"

```
s.map('TButton',
background=[('disabled','#d9d9d9'), ('active','#ececce'),
foreground=[('disabled','#a3a3a3')], relevo=[ ('pressionado', '!desabilitado',
'afundado')])
```

Lembre-se de que, no passado, com os widgets Tk clássicos, exatamente o que mudava quando o widget estava em cada estado era determinado exclusivamente pelo autor do widget. Com widgets temáticos, é o próprio estilo que determina quais mudanças, o que pode incluir coisas que o autor original do widget

nunca tinha previsto.

Como os estados do widget podem conter vários sinalizadores, é possível que mais de um estado corresponda a uma opção (por exemplo, "pressionado" e "pressionado |desativado" corresponderão se o sinalizador de estado "pressionado" do widget estiver definido). A lista de estados é avaliada na ordem fornecida no comando map, com o primeiro estado na lista correspondente sendo usado.

Parece difícil para você?

Então você já sabe que os estilos são formados por elementos, que possuem diversas opções, e são compostos juntos em um determinado layout. Você pode alterar várias opções de estilos para fazer com que todos os widgets que usam o estilo apareçam de maneira diferente. Quaisquer widgets que usam esse estilo assumem a aparência que o estilo define. Os temas reúnem um conjunto completo de estilos relacionados, facilitando a alteração da aparência de toda a interface do usuário.

Então, o que torna estilos e temas tão difíceis na prática? Três coisas. Primeiro:

Você só pode modificar as opções de um estilo, não as opções do elemento (exceto às vezes).

Falamos anteriormente sobre como descobrir quais elementos foram usados no estilo examinando o layout do estilo e também como descobrir quais opções estavam disponíveis para cada elemento. Mas quando íamos fazer alterações em um estilo, parecia que estávamos configurando uma opção para o estilo, sem especificar um elemento individual. O que está acontecendo?

Novamente, usando nosso exemplo de botão, tínhamos um elemento "Button.label", que entre outras coisas tinha uma opção de configuração de "fonte". O que acontece é que, quando o elemento "Button.label" é desenhado, ele examina a opção de configuração "fonte" definida no *estilo* para determinar em qual fonte se desenhar.

Para entender o porquê, você precisa saber que quando um estilo inclui um elemento como parte dele, esse elemento não mantém nenhum armazenamento (específico do elemento). Em particular, ele não armazena nenhuma opção de configuração. Quando ele precisa recuperar opções, ele o faz por meio do estilo recipiente, que é passado para o elemento. Elementos individuais, portanto, são objetos "flyweight" na linguagem padrão GoF.

Da mesma forma, quaisquer outros elementos procurarão suas opções de configuração nas opções definidas no estilo. E se dois elementos usarem a mesma opção de configuração (como uma cor de fundo)? Como existe apenas uma opção de configuração de fundo, armazenada no estilo, isso significa que ambos os elementos usarão a mesma cor de fundo. Você não pode ter um elemento usando uma cor de fundo e o outro usando uma cor de fundo diferente.

Exceto quando você pode. Existem algumas coisas desagradáveis, específicas do widget, chamadas de "sublayouts" na implementação atual, que às vezes permitem que você modifique apenas um único elemento, por meio da configuração de uma opção como "TButton.Label" (em vez de apenas "TButton", o nome do estilo). Os casos em que você pode fazer isso estão documentados? Existe alguma maneira de introspecção para determinar quando você pode fazer isso? Não para ambos. Esta é uma área da API temática do widget que eu definitivamente espero evoluir com o tempo.

A segunda dificuldade também está relacionada à modificação das opções de estilo:

As opções disponíveis não necessariamente têm efeito e não é um erro modificar uma opção falsa.

Às vezes, você tentará alterar uma opção que deveria existir de acordo com as opções do elemento, mas não terá efeito. Por exemplo, você não pode modificar a cor de fundo de um botão no tema "aqua" usado pelo Mac OS X. Embora existam razões válidas para esses casos, no momento não é fácil descobri-los, o que pode tornar a experiência frustrante às vezes.

Talvez mais frustrante quando você está experimentando é que especificar um nome de estilo ou nome de opção "incorrecto" não gera um erro. Ao fazer uma "configuração" ou "pesquisa", você pode especificar qualquer nome para um estilo e especificar qualquer nome para uma opção. Portanto, se você está entediado com as opções de "plano de fundo" e "fonte", sinta-se à vontade para configurar uma opção "fazer o que significa". Pode não fazer nada, mas não é um erro. Novamente, pode ser difícil saber o que você deve modificar e o que não deve.

Essa é uma das desvantagens de ter um sistema muito leve e dinâmico. Você pode criar novos estilos apenas fornecendo seus nomes ao configurar as opções de estilo; isso significa que você não precisa criar explicitamente um objeto de estilo. Ao mesmo tempo, isso se abre para erros. Também não é possível descobrir quais estilos existem ou são usados atualmente. E como as opções de estilo são apenas um front-end para as opções de elemento, e os elementos em um estilo podem mudar a qualquer momento, não é necessariamente óbvio que as opções devam ser restritas àquelas referidas apenas pelos elementos atuais, que podem não ser todos introspectivo.

Finalmente, aqui está a última coisa que torna estilos e temas tão difíceis:

Os elementos disponíveis, os nomes desses elementos, quais opções estão disponíveis ou têm efeito para cada um desses elementos e quais são usados para um determinado widget podem ser diferentes em cada tema.

Então? Lembre-se, entre outras coisas, de que o tema padrão para cada plataforma (Windows, Mac OS X e Linux) é diferente (o que é bom). Algumas implicações disso:

1. Se você quiser definir um novo tipo de widget (ou mais provavelmente uma variação de um widget existente) para seu aplicativo, precisará fazê-lo separadamente e de forma diferente para cada tema que seu aplicativo usa (portanto, pelo menos três para um aplicativo de plataforma cruzada).
2. Os elementos e opções disponíveis serão diferentes para cada tema/plataforma, o que significa que você pode ter que criar uma abordagem de personalização bem diferente para cada tema/plataforma.
3. Os elementos, nomes e opções de elementos disponíveis em cada tema normalmente não são documentados (além da leitura dos próprios arquivos de definição do tema), mas geralmente são identificados por meio da introspecção do tema (que veremos em breve). Como nem todos os temas estão disponíveis em todas as plataformas (por exemplo, "aqua" só será executado no Mac OS X), você precisará de acesso imediato a todas as plataformas e temas nos quais precisa executar.

Como exemplo, veja como fica o layout do estilo "TButton" no tema usado por padrão em três plataformas diferentes, bem como as opções anunciadas para cada elemento (nem todas

que têm um efeito):

`{Button.button -sticky nswe -children {Button.padding -sticky nswe -children Mac OS X
{Button.label -sticky nswe}},`

	Button.button -	
	Button.padding padding, relevo, shiftrelief	
	Button.label composto, espaço, texto, fonte, primeiro plano, sublinhado, largura, âncora, justificar, envoltório, em relevo, imagem, pontilhado, plano de fundo	
janelas	<code>Button.button -sticky nswe -children {Button.focus -sticky nswe -children {Button.padding -sticky nswe -children {Button.label -sticky nswe}}},</code>	
	Button.button -	
	Button.focus -	
	Button.padding padding, relevo, shiftrelief	
	Button.label composto, espaço, texto, fonte, primeiro plano, sublinhado, largura, âncora, justificar, envoltório, alto relevo, imagem, pontilhado, plano de fundo	
Linux	<code>Button.border -sticky nswe -border 1 -children {Button.focus -sticky nswe -children {Button.padding -sticky nswe -children Button.label -sticky nswe}}},</code>	
	Button.border background, borderwidth, relevo Button.focus cor de foco, espessura de borda, Button.padding padding,	
	Button.label composto, espaço, texto, fonte, primeiro plano, sublinhado, largura, âncora, justificar, envoltório, alto relevo, imagem, pontilhado, plano de fundo	

O ponto principal é que no clássico Tk, onde você tinha a capacidade de modificar qualquer um de um grande conjunto de atributos para um widget individual, você seria capaz de fazer algo em uma plataforma e meio que funcionaria (mas provavelmente precisaria de ajustes) em outros. No Tk temático, a opção fácil simplesmente não existe e você é praticamente forçado a fazê-lo da maneira certa se quiser que seu aplicativo funcione com vários temas/plataformas. É mais trabalho na frente.

Avançado: mais sobre elementos

Embora isso seja tudo o que abordaremos sobre estilos e temas neste tutorial, para usuários curiosos e aqueles que desejam se aprofundar na criação de novos temas, podemos fornecer mais alguns detalhes interessantes sobre os elementos.

Como os elementos são os blocos de construção de estilos e temas, surge a questão de "de onde vêm os elementos?" Em termos práticos, podemos dizer que os elementos são normalmente criados em código C e estão em conformidade com uma API específica que o mecanismo de criação de temas entende.

No nível mais baixo, os elementos vêm de algo chamado *fábrica de elementos*. Atualmente, existe um "padrão", que a maioria dos temas usa, e usa rotinas de desenho Tk para criar elementos. Um segundo permite que você crie elementos a partir de imagens e é realmente acessível no nível do script usando o método "ttk::style element create" (de Tcl). Por fim, há um terceiro mecanismo específico do Windows que usa a API da plataforma "Visual Styles" subjacente.

Se um tema usar elementos criados por meio de widgets nativos de uma plataforma, as chamadas para usar esses widgets nativos normalmente aparecerão no código de especificação do elemento desse tema. Obviamente, os temas cujos elementos dependem de widgets nativos ou chamadas de API só podem ser executados nas plataformas que os suportam.

Os temas pegarão um conjunto de elementos e os usarão para montar os estilos que são realmente usados pelos widgets. E dado que toda a ideia de temas é para que vários estilos possam compartilhar a mesma aparência, não é surpreendente que estilos diferentes compartilhem os mesmos elementos.

Portanto, embora o estilo "TButton" inclua um elemento "Button.padding" e o estilo "TEntry" inclua um elemento "Entry.padding", abaixo desses elementos de preenchimento provavelmente há um e o mesmo. Eles podem aparecer de forma diferente, mas isso é devido a diferentes opções de configuração, que, como lembramos, são armazenadas no estilo que usa o elemento.

Provavelmente também não é surpreendente descobrir que um tema pode fornecer um conjunto de opções comuns que são usadas como padrão para cada estilo, se o estilo não as especificar de outra forma. Isso significa que, se praticamente tudo em um tema inteiro tiver um fundo verde, o tema não precisa dizer isso explicitamente para cada estilo. Isso usa um estilo de raiz chamado ".; afinal, se "Fun.TButton" pode herdar de "TButton", por que "TButton" não pode herdar de ".?"

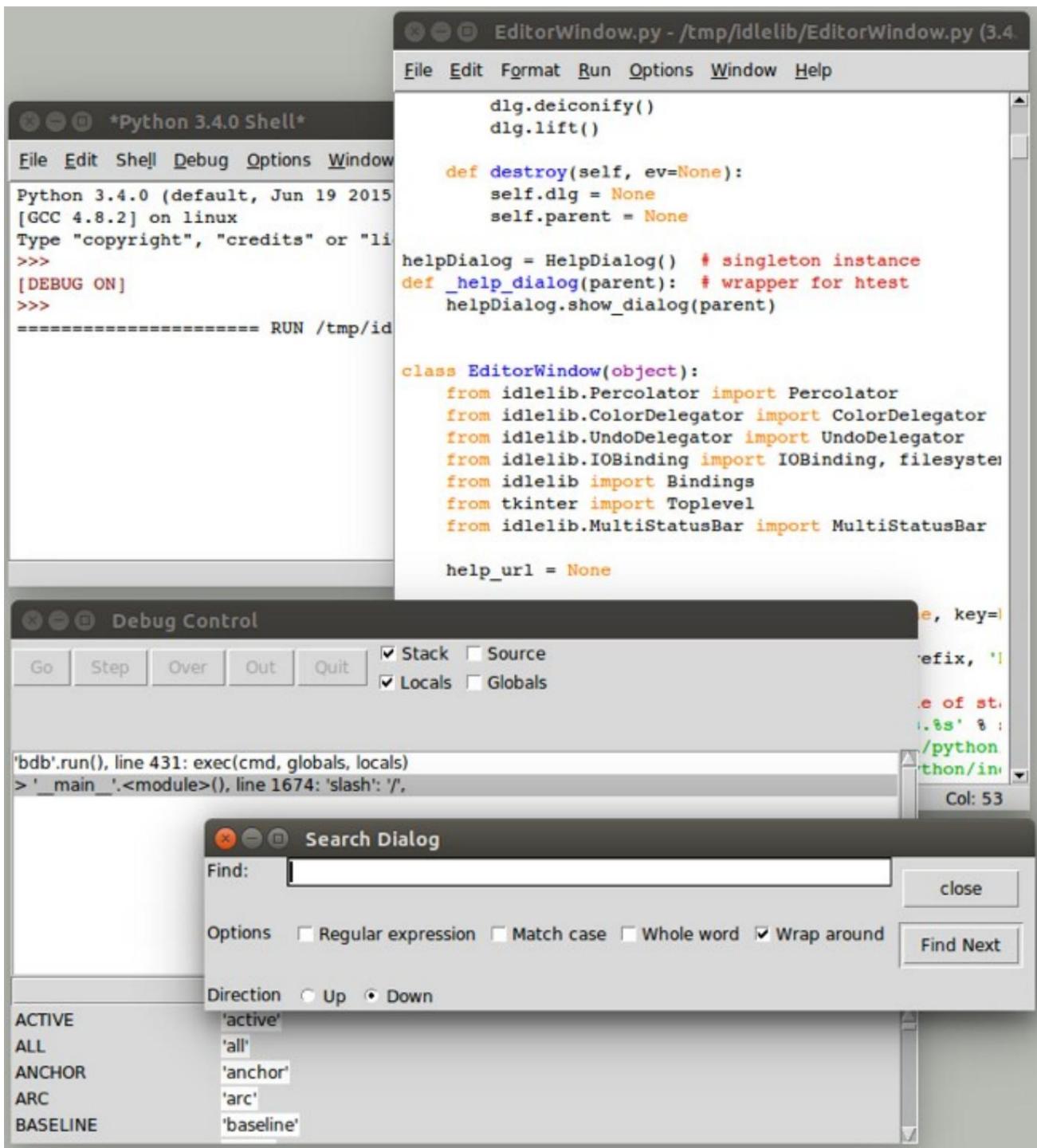
Finalmente, vale a pena dar uma olhada em como os temas existentes são definidos, tanto no nível do código C na biblioteca C do Tk, quanto também através dos scripts Tk encontrados no diretório "library/ttk" do Tk. Pesquise por "Ttk_RegisterElementSpec" na biblioteca C do Tk para ver como os elementos são especificados.

Estudo de caso: modernização do IDLE

Este capítulo apresenta um estudo de caso de modernização da aparência de um aplicativo substancial baseado em Tk.

Este capítulo é atualmente (novembro de 2015) um trabalho em andamento. O código aqui está lentamente encontrando seu caminho para a árvore de origem do Python, mas a maioria ainda não está lá. Você pode encontrar instantâneos temporários no [GitHub](#).

PARADO (Integrated DeveLopment Environment) é o ambiente de desenvolvimento padrão do Python que acompanha cada versão do Python. Ele consiste em um shell Python interativo, editores com realce de sintaxe, um depurador, etc. Sua interface de usuário é escrita em Tkinter.



Visão geral da interface de usuário IDLE (no Linux)

O IDLE nunca teve a intenção de ser um substituto para ambientes de desenvolvimento mais completos.

Por ser relativamente simples e integrado, e por ser empacotado com Python, é popular para quem está aprendendo (e ensinando) a linguagem.

Originalmente criado por Python BDFL Guido van Rossum em 1998, o IDLE foi adicionado gradualmente ao longo dos anos por vários outros desenvolvedores. Mas com esforço de desenvolvimento limitado gasto nele, ele estava mostrando sua idade, especialmente em plataformas (por exemplo, Mac OS X) usadas com pouca frequência por aqueles que aprimoram o IDLE.

Uma [comparação Python Central de IDEs](#) descreveu o IDLE desta maneira:

Todos esses recursos estão de fato presentes, mas não constituem realmente um IDE. Na verdade, embora o IDLE ofereça alguns dos recursos que você espera de um IDE, ele o faz sem ser um editor de texto satisfatório. A interface é cheia de bugs e não leva em consideração como o Python funciona, especialmente no shell interativo, o preenchimento automático é inútil fora da biblioteca padrão e a funcionalidade de edição é tão limitada que nenhum programador Python sério - diabos, nenhum digitador sério - poderia usá-lo em tempo integral.

Se você usar um IDE, não deve ser IDLE.

Com sua interface de usuário antiquada e cheia de bugs, o [IDLE corria alguns riscos](#) de ser completamente removido da distribuição do Python. No entanto, devido à relativa simplicidade e ao fato de ser integrado, havia várias pessoas, principalmente aquelas que ensinavam Python, ansiosas para ver o IDLE avançar.

O IDLE era obviamente um ótimo candidato para ser modernizado, usando recursos Tk mais recentes, como os widgets temáticos, para ajudar a estimular algum redesenho. Mas era mais do que apenas trocar widgets. Muitas melhorias poderiam ser feitas apenas mudando a forma como os widgets "clássicos" estavam sendo usados, para melhor refletir uma estética de design mais moderna.

Como o IDLE, que é um aplicativo, fazia parte da biblioteca padrão do Python (ou seja, 99% projetada para ser usada por outro código), havia muitas políticas e procedimentos não apropriados para um aplicativo que dificultavam as alterações. Removendo alguns desses obstáculos (ver [PEP 434](#)) foi uma etapa significativa [necessária para os tipos de mudanças](#) que estão sendo discutidas aqui.

Objetivos do Projeto

Como você pode imaginar, a modernização de um grande aplicativo como este, baseado no "clássico" Tkinter, não foi totalmente simples. Neste capítulo, examinarei algumas das alterações feitas na interface do usuário e por quê.

Todos os envolvidos queriam ver o IDLE muito melhor do que era, embora ninguém tivesse a ilusão de que ele se tornaria um exemplo impressionante de design de ponta. Mas algo que se encaixasse mais, para que as pessoas pudessem aprender sobre Python sem se distrair com a falta de jeito de suas ferramentas, parecia factível.

O objetivo também não era tentar competir com os IDEs Python mais usados por programadores profissionais, como [PyCharm](#), [WingIDE](#), [PyDev](#), [Komodo](#), etc. [Embora alguns desenvolvedores](#) de Python mais experientes usassem o IDLE, esporadicamente ou regularmente, ele precisava atrair principalmente os novatos na linguagem e, geralmente, os novos na programação.

Não faltaram tentativas anteriores de avançar radicalmente o IDLE, resultando em vários *forks* diferentes que apresentavam todos os tipos de melhorias. Muitos deles usavam outros kits de ferramentas GUI ou módulos que não faziam parte da biblioteca padrão. Era importante ficar com o Tkinter e a biblioteca padrão, porque queríamos garantir que todas as melhorias pudessem, com o tempo, chegar à versão *oficial* que acompanha o Python, em vez de se tornar apenas mais um fork.

Além disso, a esperança era tornar o IDLE mais fácil de contribuir. Isso sugeriu reduzir um volume substancial de código, remover algumas redundâncias e inconsistências, limpar algumas das partes mais complexas, simplificar as interações entre os componentes do sistema etc. em.

Embora não menos importante, vou adiar amplamente a discussão das questões substanciais em torno da arquitetura de software, compatibilidade com versões anteriores (incluindo para sistemas rodando Tk 8.4, pré-ttk, que receberam apenas algumas melhorias seletivas), etc.

Ao longo deste capítulo, você encontrará links para problemas individuais no sistema de rastreamento de problemas do Python, que geralmente fornecem informações adicionais sobre os processos de pensamento de várias pessoas sobre mudanças.

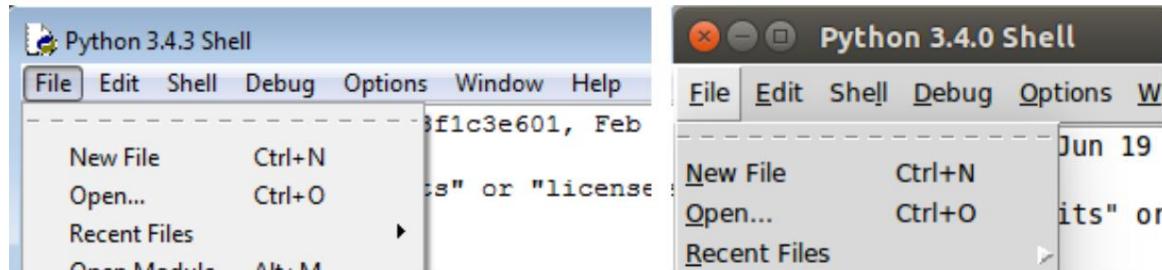
Como você verá, este capítulo destaca muitas das deficiências da interface de usuário do IDLE. Isso é feito principalmente para dar ênfase, porque muitos dos problemas mostrados aqui são comuns a muitos aplicativos e interfaces de usuário.

Tenho o maior respeito pelas pessoas que doaram seu tempo e recursos limitados para este projeto de código aberto e tiveram que fazer concessões conscientemente entre tempo, recursos e interface do usuário, tudo dentro do contexto de uma base de código decididamente não trivial.

Como o material deste capítulo se refere a um aplicativo Python específico, todos os exemplos serão dados exclusivamente em Python e usando o Tkinter. Então, se você está se perguntando por que está vendo código Python aqui onde não viu em todos os outros lugares, é por isso.

Menus

A primeira mudança que foi feita foi remover os menus destacáveis arcaicos [Problema#13884]. A versão Mac OS X do Tk nem mesmo os suporta, mas eles ainda estavam lá no Windows e no X11.



Menus destacáveis no Windows e Linux

A mudança aqui foi adicionar uma opção "tearoff=0" aos poucos lugares no código onde esses menus foram criados.

Pelo menos isso foi fácil.

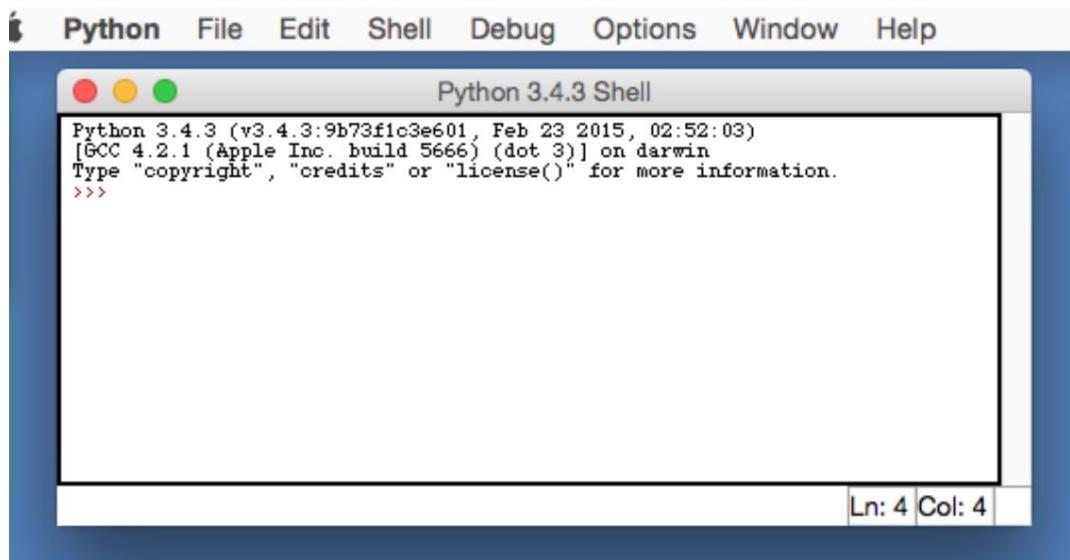
Também havia vários bugs em que os itens do menu não eram desabilitados corretamente quando o recurso não estava disponível, levando a itens de menu que não faziam nada (confusos para os alunos) ou a caixas de diálogo de erro que diziam pouco mais do que "você pode" fazer isso".

Janela principal

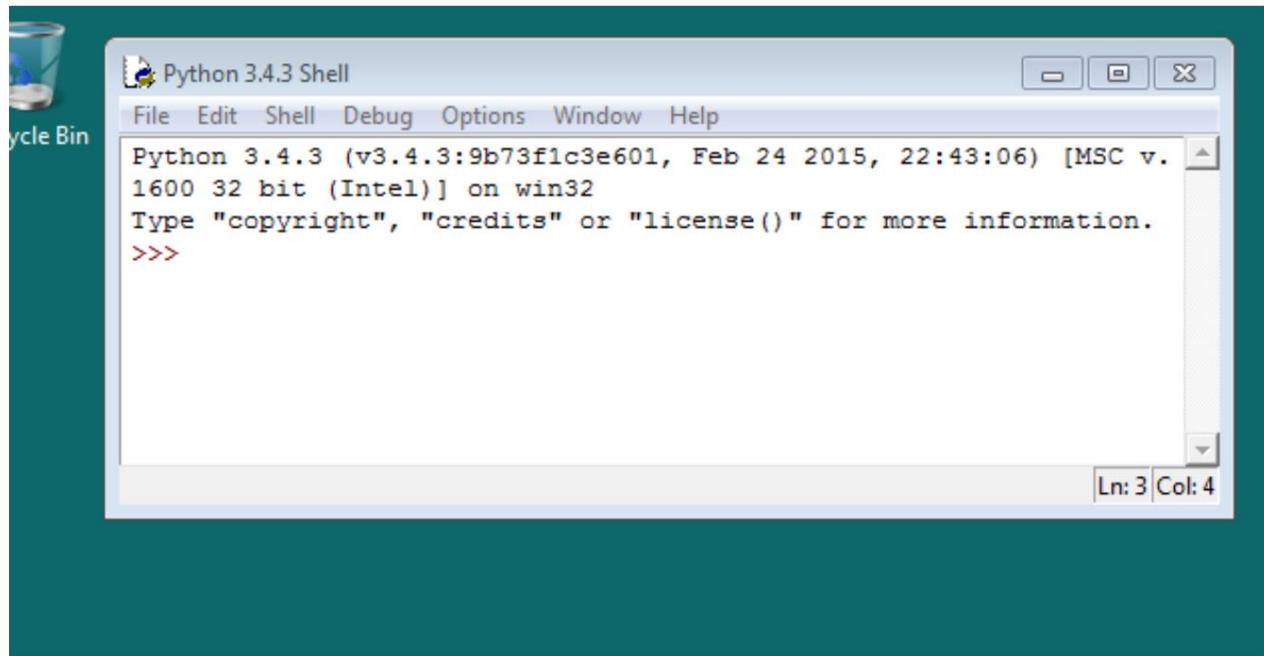
Uma coisa boa sobre o IDLE é que, girando em torno de um editor e um shell, a maior parte dele é realmente um widget de texto Tk, e havia muito pouco em sua interface de usuário que precisava ser alterado. Mas mesmo na janela principal, que é basicamente apenas um widget de texto, havia melhorias a serem feitas.

As imagens a seguir mostram a versão original da janela de shell do IDLE, pré-modernização, assim como alguém veria quando lançou o programa pela primeira vez.

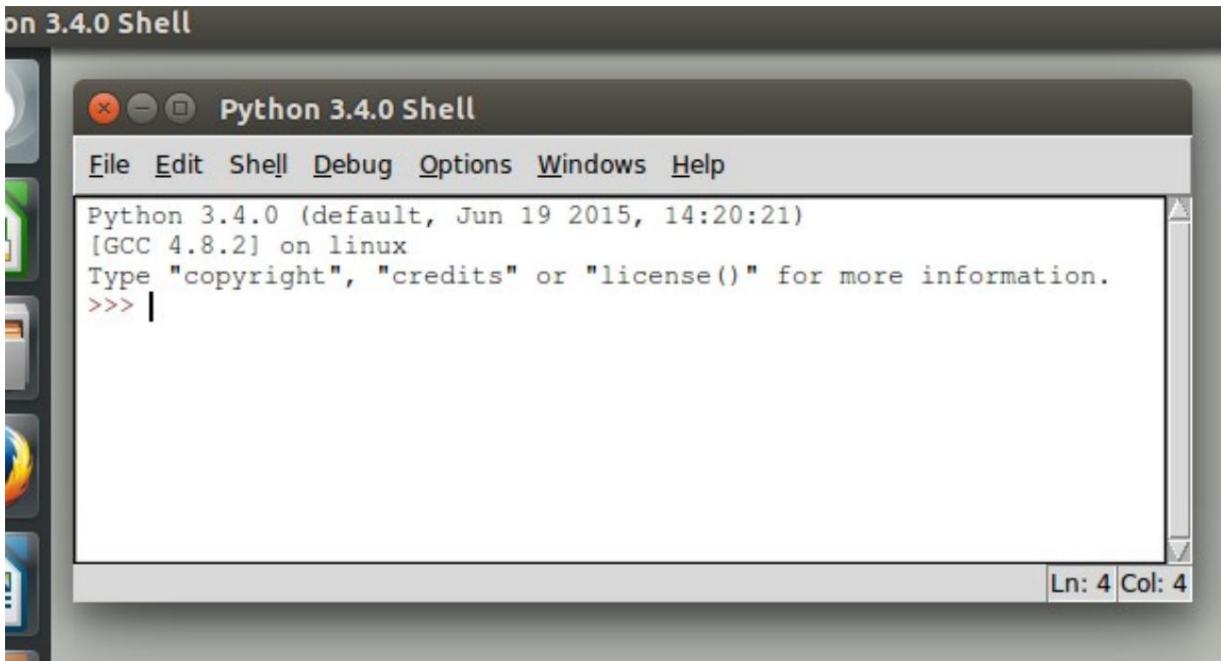
Olhando para eles, que melhorias você faria?



Janela principal IDLE no Mac OS X



Janela principal do IDLE no Windows



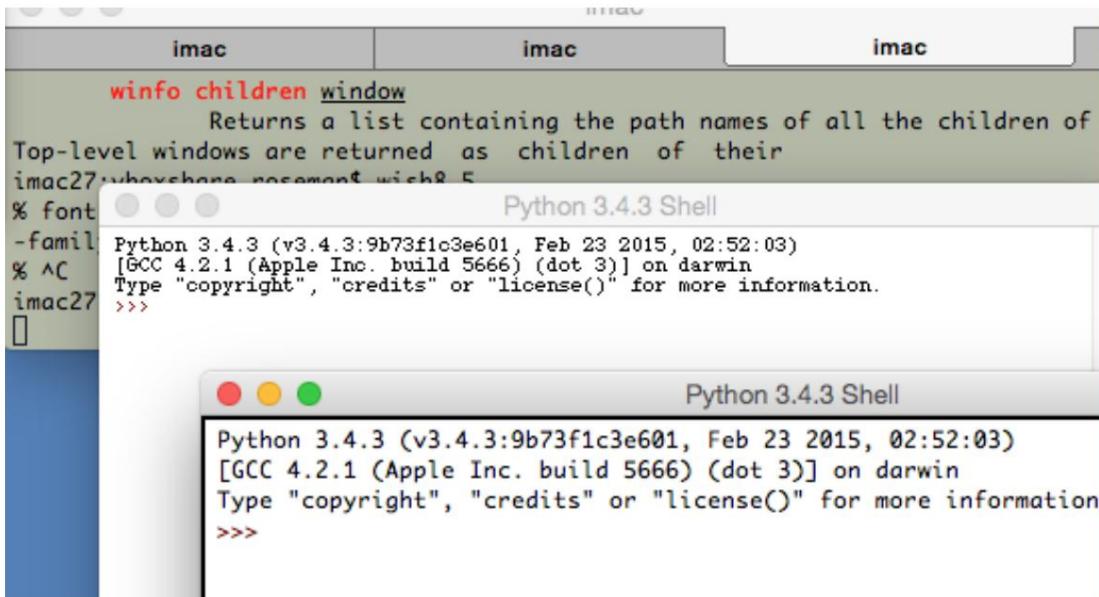
Janela principal do IDLE no Linux

Fonte padrão

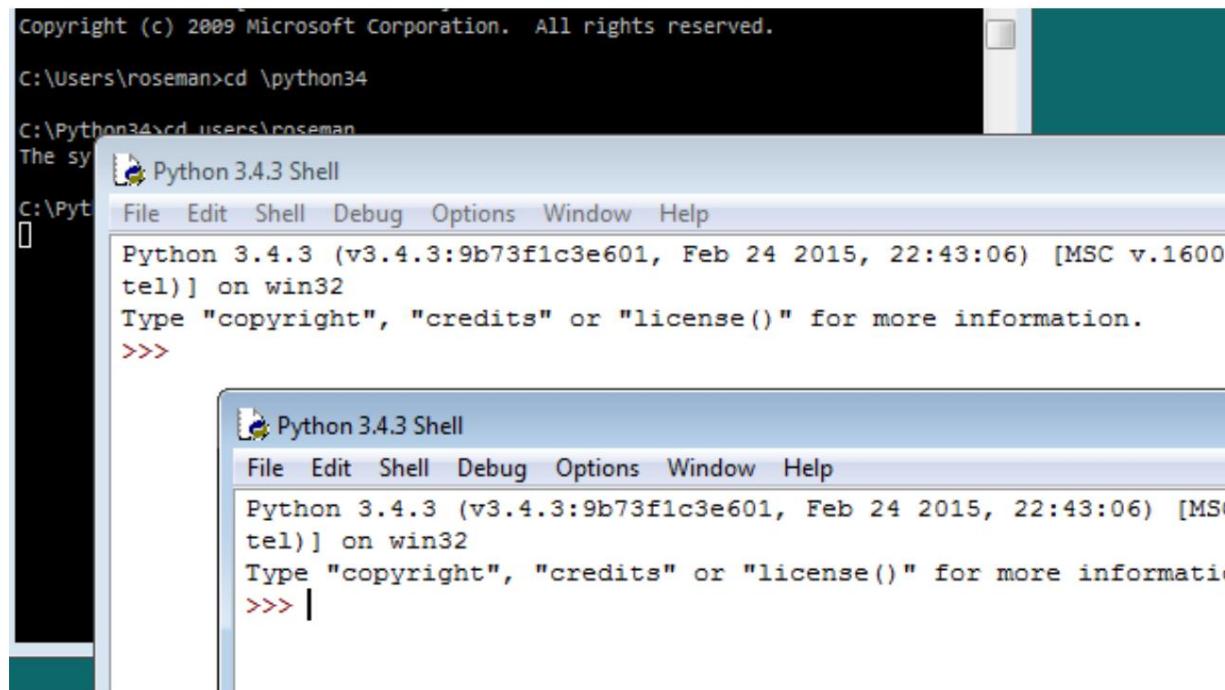
A primeira mudança que foi feita teve a ver com a fonte padrão. O IDLE codificou o Courier de 10 pontos e o usou em todas as plataformas. Na verdade, isso não parecia tão ruim no Windows, era bom no Linux, mas parecia terrível no OS X [Problema nº 24745]. Claro que isso pode ser alterado por meio de uma caixa de diálogo de preferências, mas os padrões certamente não deixaram uma boa impressão.

Embora uma opção seja simplesmente escrever o código para escolher uma boa fonte, dependendo da plataforma em que estamos executando, o padrão foi alterado para usar o Tk's integrado "TkFixedFont", que fornece um padrão melhor em cada plataforma.

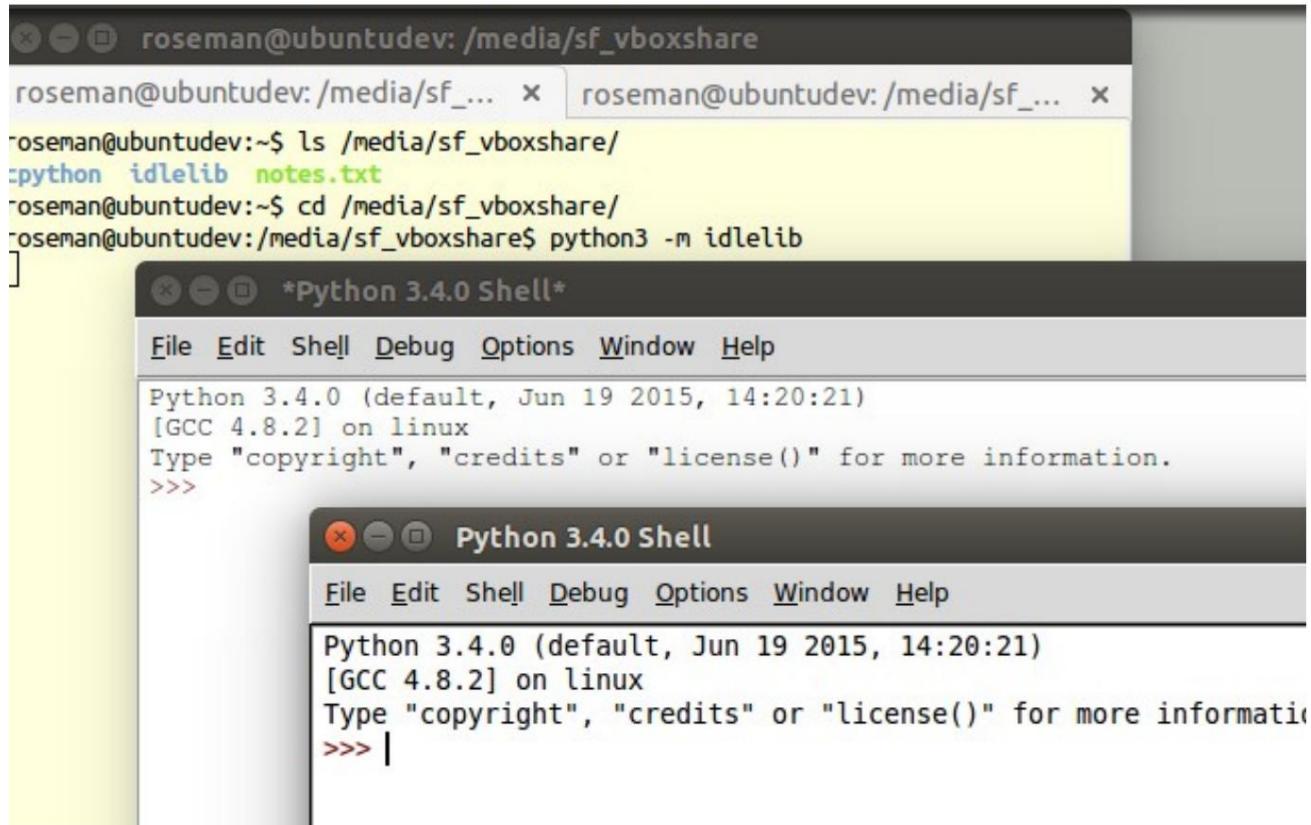
Você pode ver as diferenças nas capturas de tela abaixo. Observe como as novas fontes parecem corresponder melhor com as janelas do terminal do sistema que são mostradas.



Janela principal IDLE usando TkFixedFont (Mac OS X)



Janela principal IDLE usando TkFixedFont (Windows)



Janela principal IDLE usando TkFixedFont (Linux)

Falando em caixas de diálogo de preferências, se você quiser alterar a fonte, geralmente ajuda saber qual é a fonte (usando "Font.actual"). Aqui está o novo código da caixa de diálogo de preferências do IDLE que descobre isso:

```

if (família == 'TkFixedFont'):
    f = Font(name = 'TkFixedFont', exist = True, root = root) actualFont =
        Font.actual(f) family = actualFont['family'] size = actualFont['size'] if size < 0:
            size = 10 # se a fonte estiver em pixels, ignore o tamanho real
            negrito = 1 se fonte atual['peso']=='negrito' senão 0

```

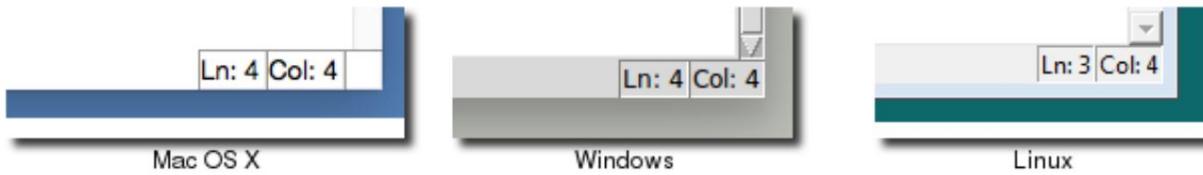
Para o registro, as fontes que Tk escolheu para TkFixedFont são Monaco 11 no Mac OS X, Courier New 10 no Windows e DejaVu Sans Mono 10 no Linux (Ubuntu 14 para ser específico).

Ao redor do widget de texto

Havia algumas outras coisas cosméticas que simplesmente não estavam bem na borda da janela principal [\[Problema#24750\]](#). Veja as capturas de tela anteriores da janela principal do IDLE.

Observe que há uma borda ao redor do widget de texto. É mais perceptível no OS X, onde é um preto escuro, um pouco menos no Linux e quase imperceptível no Windows. Este é o resultado do atributo "highlightthickness" de Tk, que está presente quando o widget de texto tem o foco.

Se o widget de texto não tiver o foco, como quando a janela fica inativa, o destaque vai embora:



Barra de status na janela inativa

Observe como na captura de tela do Mac OS X, sem o realce, a barra de status na parte inferior da janela se mistura ao widget de texto. Não é bom.

Como você verá em outros aplicativos, a borda ao redor do widget de texto não é mais uma convenção comum. Então, vamos começar removendo isso, que é tão fácil quanto adicionar "highlightthickness=0" quando criamos o widget de texto.

Isso ainda nos deixa com o problema da mistura da barra de status no editor. Alteramos a barra de status para um widget "ttk.Frame", que possui sombreado de fundo em todas as plataformas. Também colocamos um widget "ttk.Separator" logo acima da barra de status, para nos dar uma separação limpa.

Cada um dos indicadores de linha e coluna eram rótulos, previamente criados com:
label = Label(self, bd=1, relevo=SUNKEN, âncora=W)

Isso foi substituído por um "ttk.Label", para garantir que correspondia ao quadro. Também eliminamos o visual "3D" afundado dos anos 90.

label = ttk.Label(self)

Há uma última coisa, que é o espaço vazio no canto inferior direito da janela do Mac OS X.

Isso costumava ser ocupado pelo pequeno widget que permitia redimensionar a janela. O Mac OS X 10.7 adicionou a capacidade de redimensionar janelas arrastando qualquer lado (como qualquer outra plataforma tinha) e eliminou o widget de redimensionamento no canto.

Tk fornece o widget "ttk::Sizegrip" para colocar no canto se você precisar, mas infelizmente não ajuda quando se trata de precisar ou não. Então, isso é com você. IDLE agora usa este código:

```
def need_sizegrip():
```

As versões mais antigas do OS X, em particular, requerem um widget ttk::sizegrip no canto inferior direito da janela. Isso não é mais o caso em versões mais recentes.

```
"""
```

```
if sys.platform == 'darwin': =  
    major, minor = platform.split('')[1].split('.')[1]  
    if( int(major)  
== 10 e int(minor ) < 7):
```

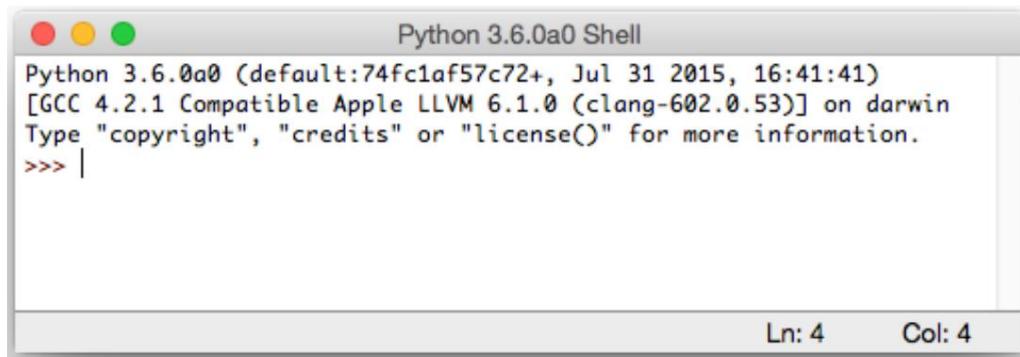
```
    retornar Verdadeiro  
retorna falso
```

```
...
```

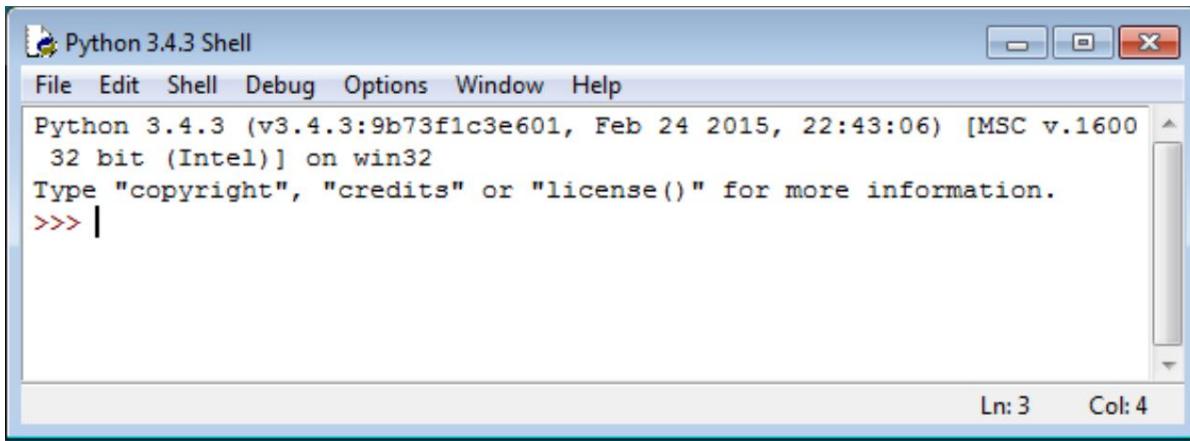
```
if need_sizegrip(): sz =  
    ttk.Sizegrip(...)
```

Por último, mas não menos importante, podemos substituir a barra de rolagem Tk original pelo novo widget "ttk.Scrollbar".

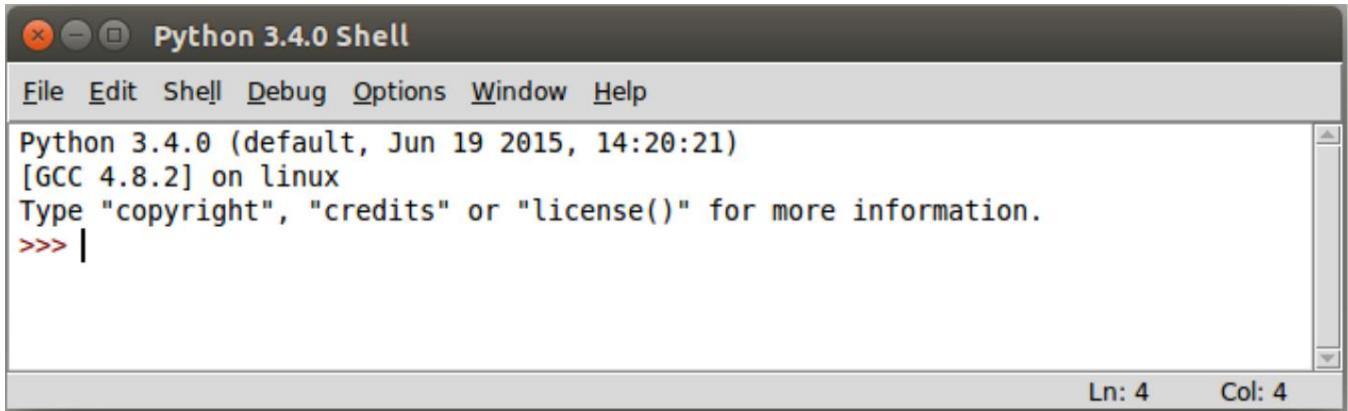
As alterações resultantes na janela principal são mostradas abaixo. Apesar de serem mudanças bem pequenas, muitas vezes sutis, elas percorrem um longo caminho em direção à janela principal do IDLE, parecendo muito mais limpas, mais modernas e "simplesmente adequadas" em todas as plataformas.



Janela principal do IDLE, com melhorias (Mac OS X)



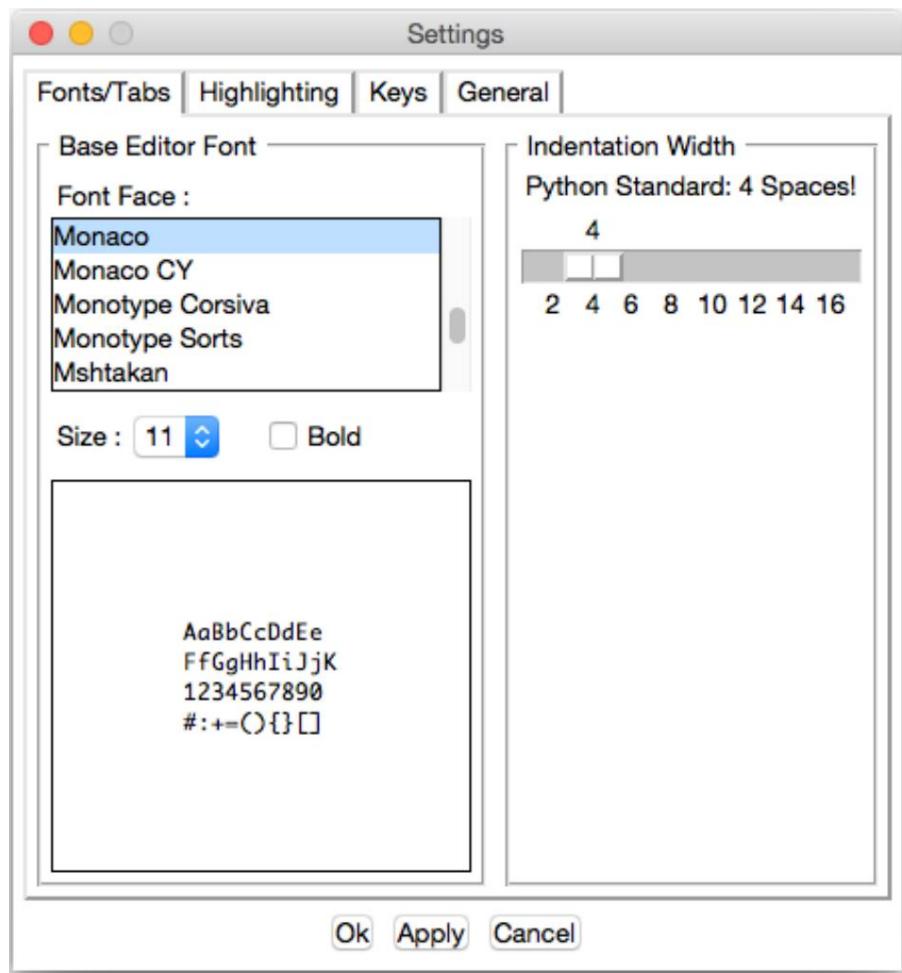
Janela principal do IDLE, com melhorias (Windows)



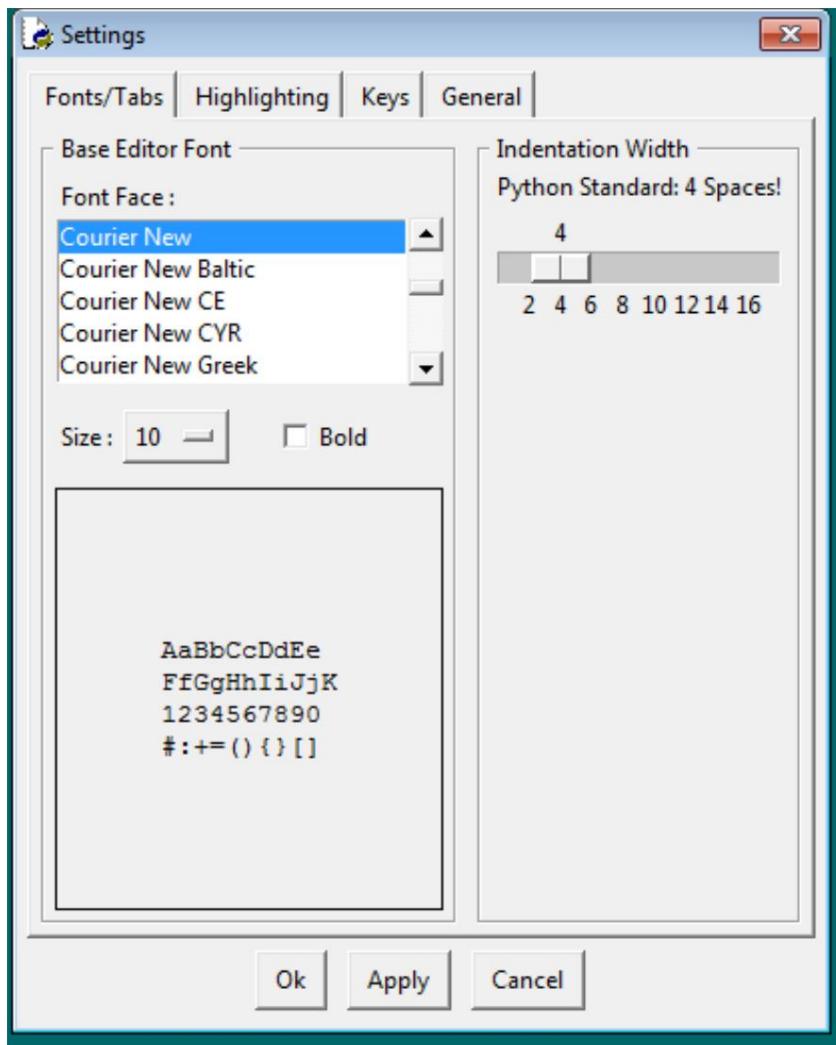
Janela principal do IDLE, com melhorias (Linux)

Diálogo de Preferências

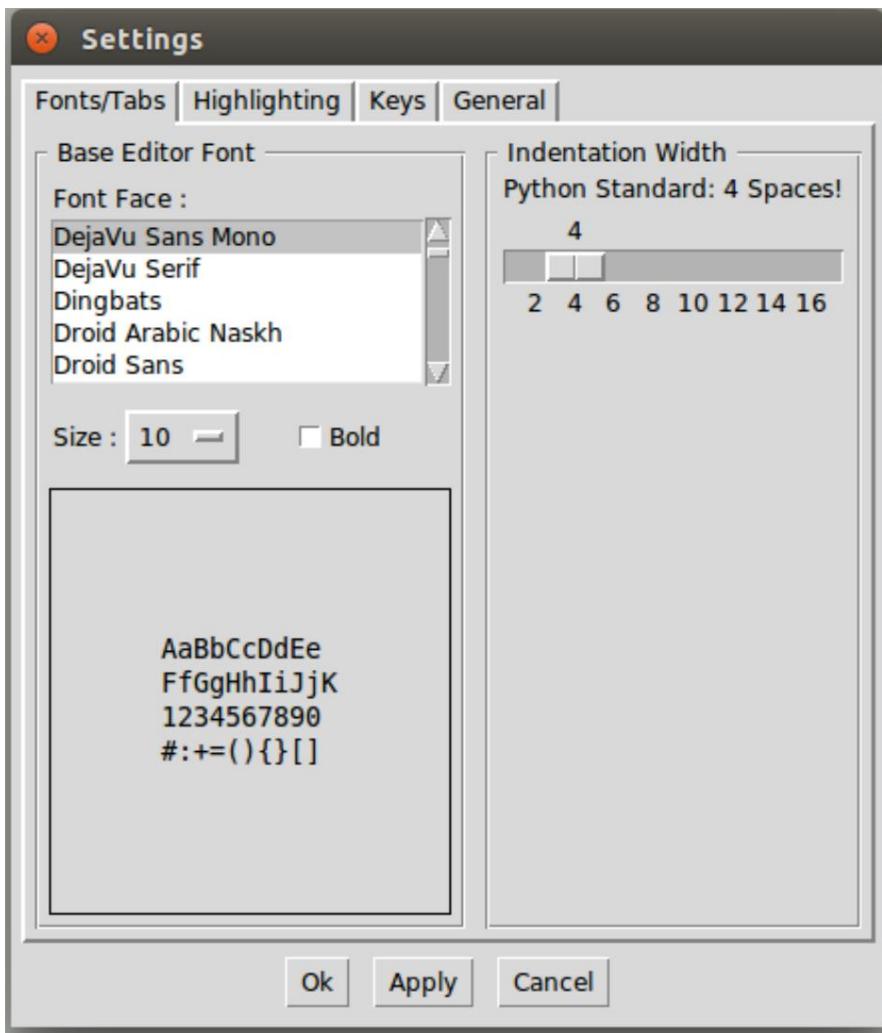
Uma peça visível que precisava muito de melhorias era a caixa de diálogo Preferências. Novamente, aqui estão as capturas de tela nas três plataformas:



Janela de preferências do IDLE (Mac OS X)



Janela de preferências do IDLE (Windows)



Janela de preferências do IDLE (Linux)

As outras guias permitem que você modifique cores individuais para destaque de sintaxe, pressionamentos de tecla atribuídos a operações específicas e algumas outras coisas diversas.

Embora houvesse algum debate sobre a necessidade desse nível de configuração no que era principalmente um ambiente de aprendizado [[Problema nº 24810](#)], parecia razoável pelo menos fazer o que estava lá parecer e funcionar melhor antes de considerar qualquer cirurgia mais radical.

Entre outras coisas, a caixa de diálogo Preferências foi alterada de modal (que, curiosamente, não funcionava bem no OS X, permitindo a criação de várias cópias) para sem modo [[Problema nº 24760](#)], embora eu não vá mais longe nisso neste ponto.

Guias

O primeiro problema a ser resolvido foram as guias usadas para alternar entre os quatro painéis de preferências diferentes. O original usava um "megawidget" customizado, já que o clássico Tk não tem seu próprio widget. Embora os do Windows e do Linux não pareçam tão ruins, nas versões recentes do OS X há um widget de guia embutido que parece bem diferente.

Na verdade, é mais comum em aplicativos do Mac OS X agora usar algo semelhante a uma barra de ferramentas (fileira de ícones com rótulos na parte superior ou lateral) para alternar entre os painéis de preferências, embora alguns

os programas ainda usam guias. As guias são muito comuns no Windows e no Linux.

O código foi modificado para usar o widget "ttk.Notebook", que não apenas fica melhor em cada plataforma, mas nos permite descartar muito código para gerenciar guias por conta própria.

Atualizando widgets

O próximo passo óbvio foi atualizar os widgets "clássicos" para suas contrapartes temáticas. Nessa tela, isso incluía botões, rótulos, molduras, caixa de seleção, barra de rolagem, etc. Havia alguns outros em alguns dos outros painéis. Geralmente, esse era um processo direto, geralmente envolvendo a remoção de opções de widget que não eram mais necessárias ou suportadas pelos widgets temáticos.

Às vezes, escolher um widget diferente fazia mais sentido. Nesta tela o menu de opções usado para tamanho de fonte foi melhor substituído por um combobox. Da mesma forma, o widget de escala não é comumente visto nas interfaces de usuário de hoje e foi substituído pelo widget de caixa de rotação mais familiar (e compacto).

Havia também várias formas não padronizadas de usar certos widgets ou especificar certos tipos de dados. Estes foram geralmente modificados para usar paradigmas mais familiares. Houve uma série de questões gerais discutidas relacionadas ao design dessas caixas de diálogo, por exemplo, [\[Issue#24776\]](#), [\[Problema nº 24782\]](#).

Layout

Embora esta caixa de diálogo seja um exemplo ruim, apenas devido ao desequilíbrio de espaço entre as metades esquerda e direita, muito tempo foi gasto em geral observando o espaçamento e o alinhamento dos widgets nas caixas de diálogo.

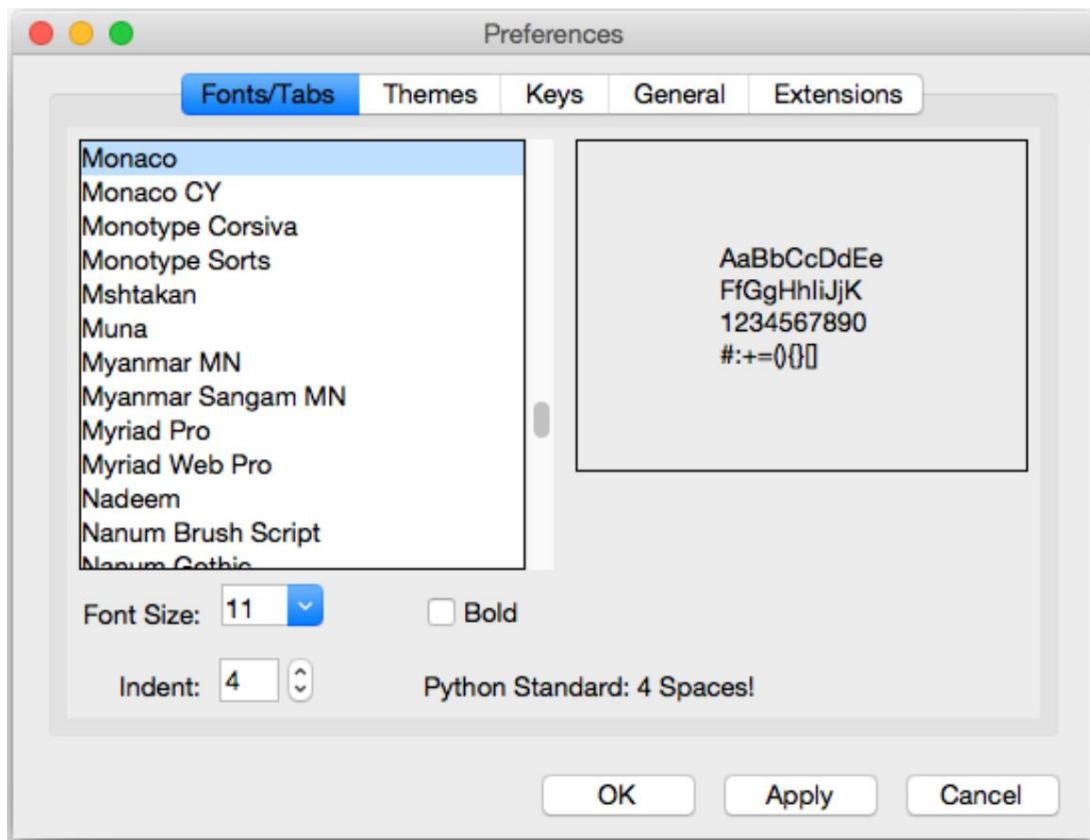
A abordagem geral foi encontrar exemplos semelhantes em outros aplicativos e usá-los como um guia.

Onde estão localizados os botões? Como vários campos de uma caixa de diálogo são organizados? Onde estão os rótulos em relação ao widget que estão rotulando? Eles estão alinhados à esquerda ou à direita, em letras maiúsculas, eles têm dois pontos à direita? Esses são os tipos de perguntas para se pensar.

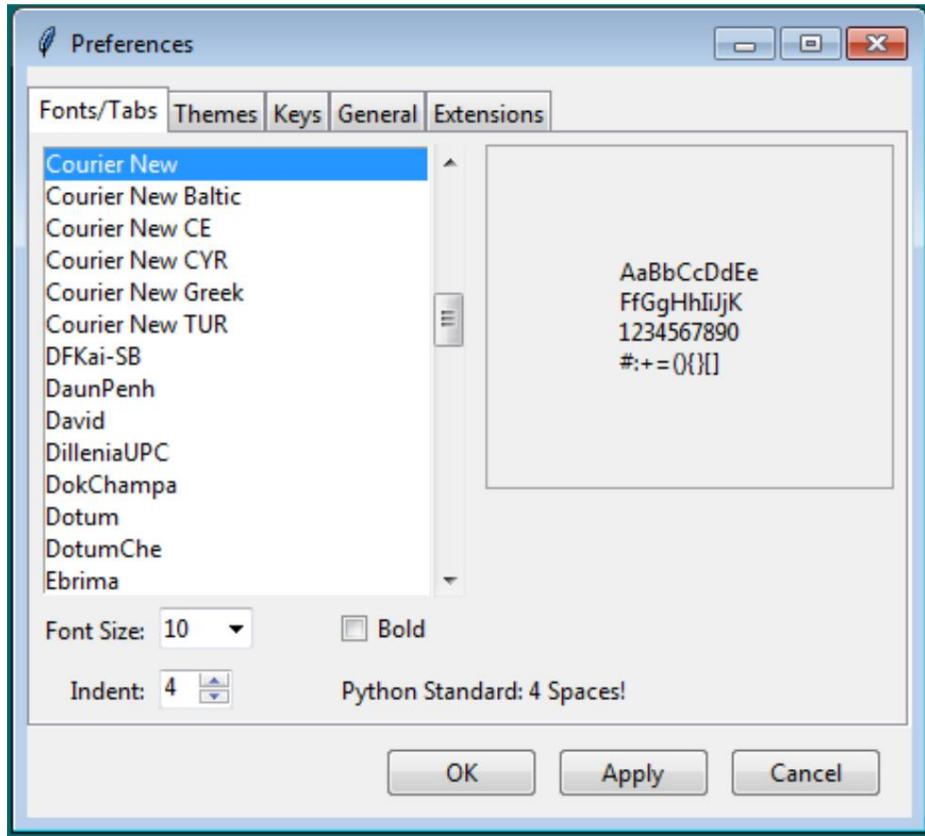
Um excelente ponto de partida é a conversão do antigo gerenciador de geometria 'pack' para 'grid'. Devido à maneira como funciona, os layouts baseados em pacotes tendem a ter alinhamento e espaçamento estranhos e inconsistentes, especialmente se tiverem sido modificados ao longo do tempo. O uso de grade aumentará a capacidade de manutenção, tanto porque usa um modelo mental mais familiar quanto porque não depende da ordem em que os widgets são inseridos.

É provavelmente impossível criar um layout que pareça fantástico em todas as plataformas, mas muitas vezes você pode criar um (possivelmente com alguns ajustes específicos da plataforma no código) que pareça decente.

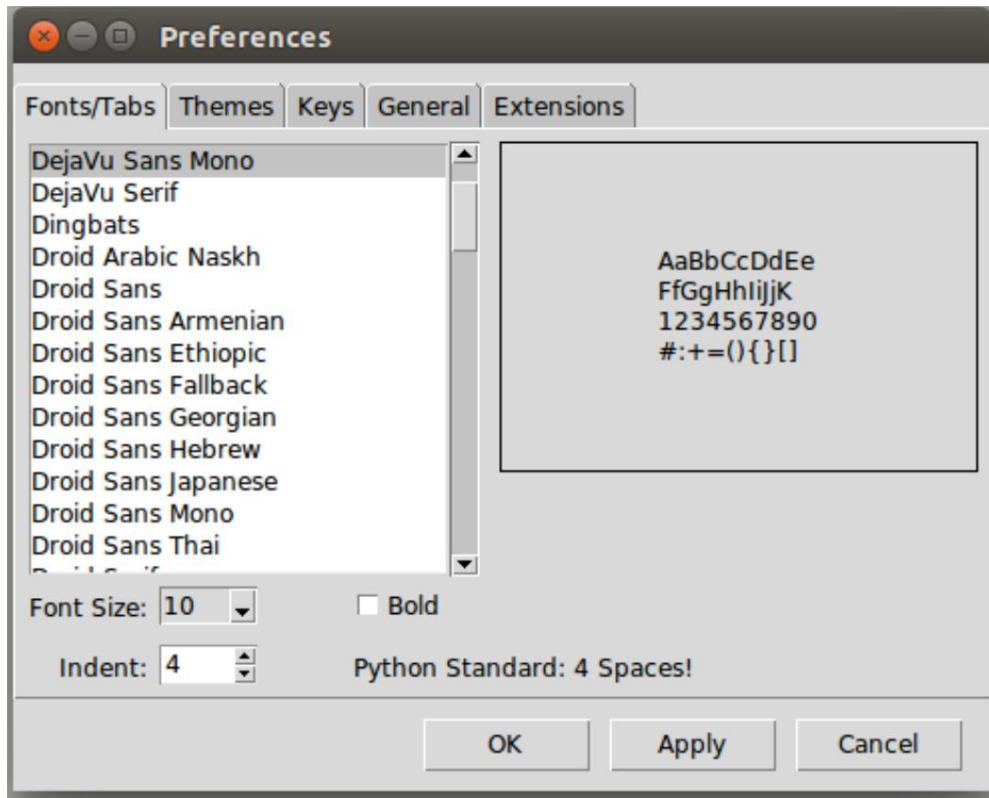
Uma versão revisada do diálogo, incorporando muitas das técnicas aqui, é mostrada abaixo.



Preferências IDLE, versão revisada (Mac OS X)



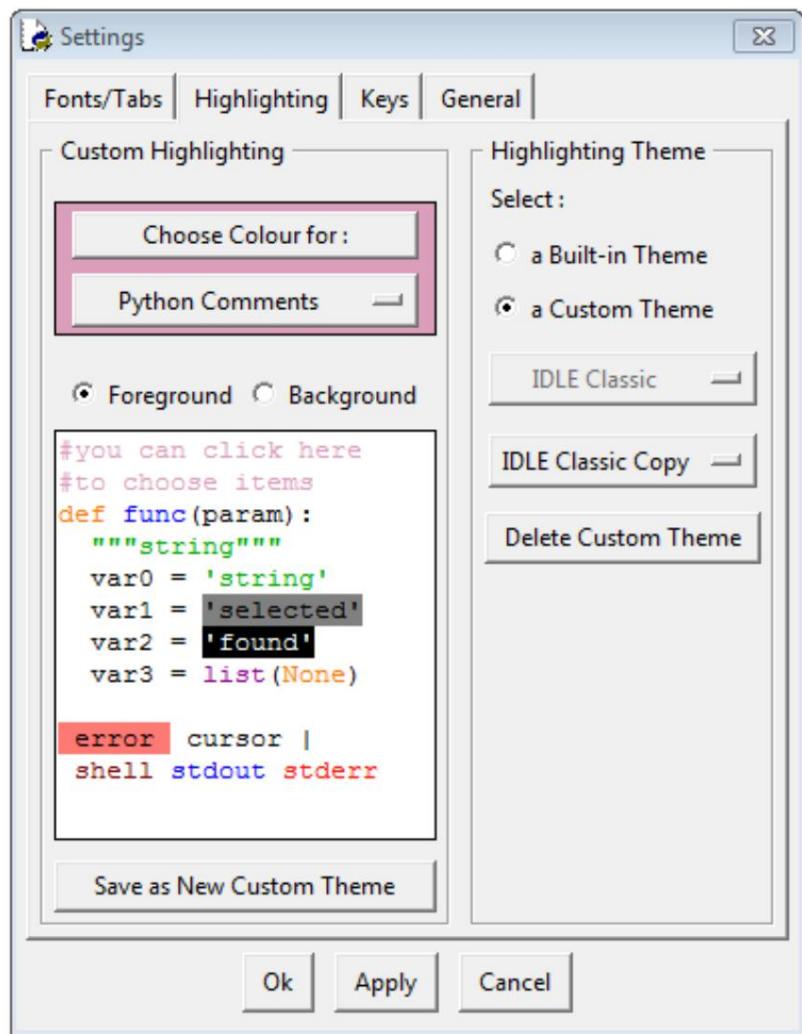
Preferências IDLE, versão revisada (Windows)



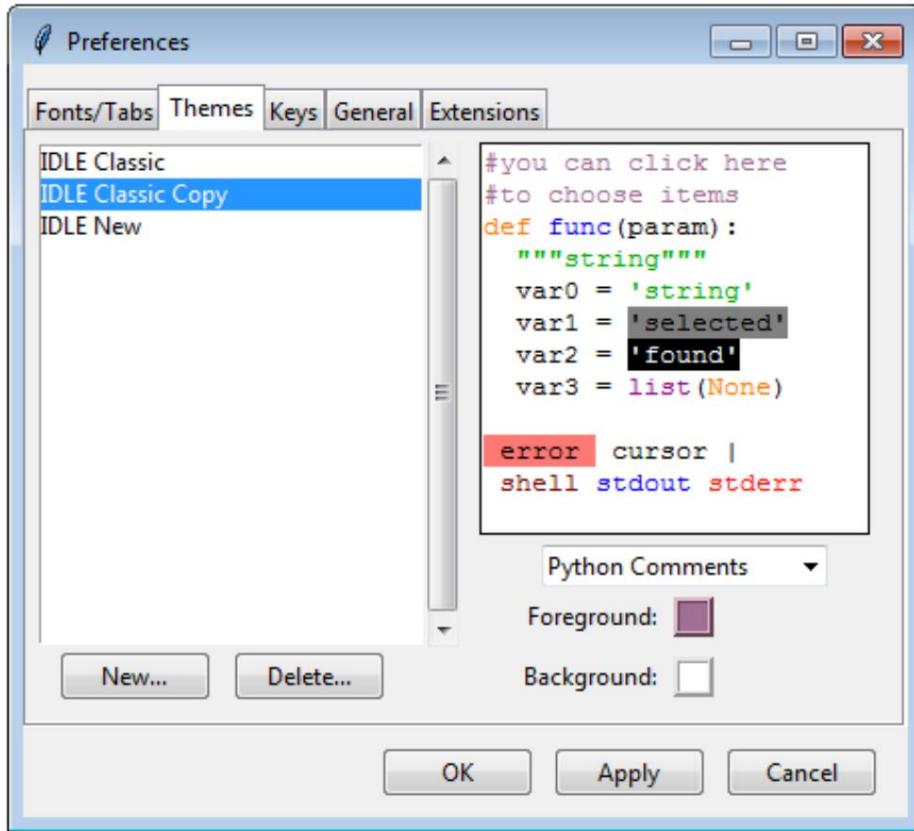
Preferências IDLE, versão revisada (Linux)

Outro exemplo

A captura de tela abaixo mostra um antes e depois do painel Preferências IDLE que controla a coloração da sintaxe [\[Problema nº 24781\]](#).



Painel de temas IDLE, antes (Windows)



Painel de temas IDLE, depois (Windows)

Novamente, substituir widgets e usar convenções mais familiares é uma parte disso. Acho que as maiores mudanças têm a ver com pensar nas coisas da perspectiva do usuário. Particularmente como uma ferramenta iniciante, se você estiver aqui, provavelmente é para mudar de tema, não ajustar cores, de modo que seja mais dominante na nova versão. Ele também acaba com uma distinção arbitrária entre temas "embutidos" e "personalizados".

Eu acho que a nova versão é uma grande melhoria, embora até o momento em que escrevo eu ainda não tenha convencido algumas pessoas disso. Sendo de código aberto, veremos o que acontece!

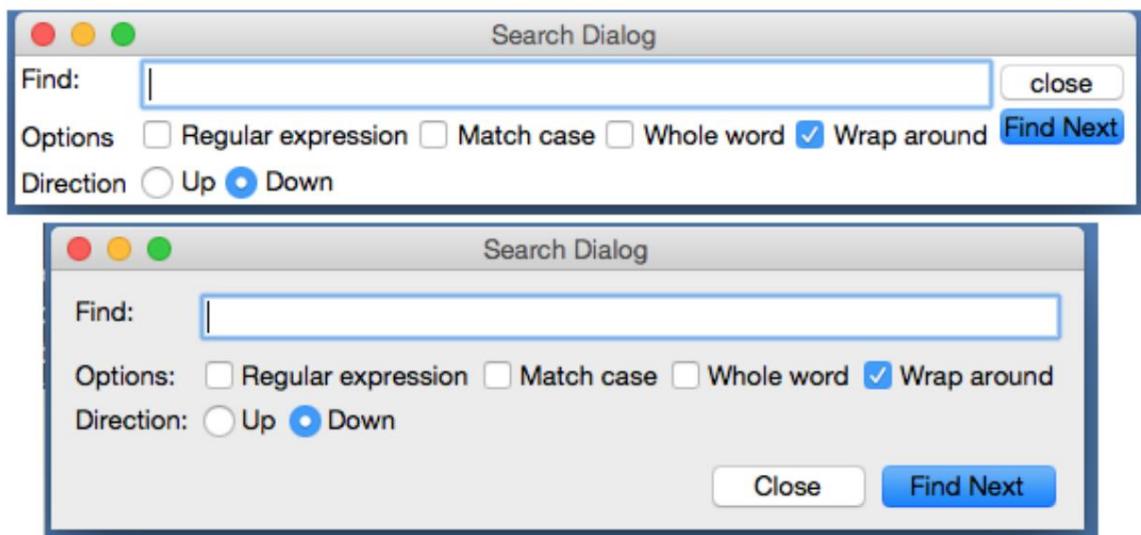
Outros Diálogos

Existem várias outras caixas de diálogo no IDLE; consideraremos mais alguns exemplos aqui.

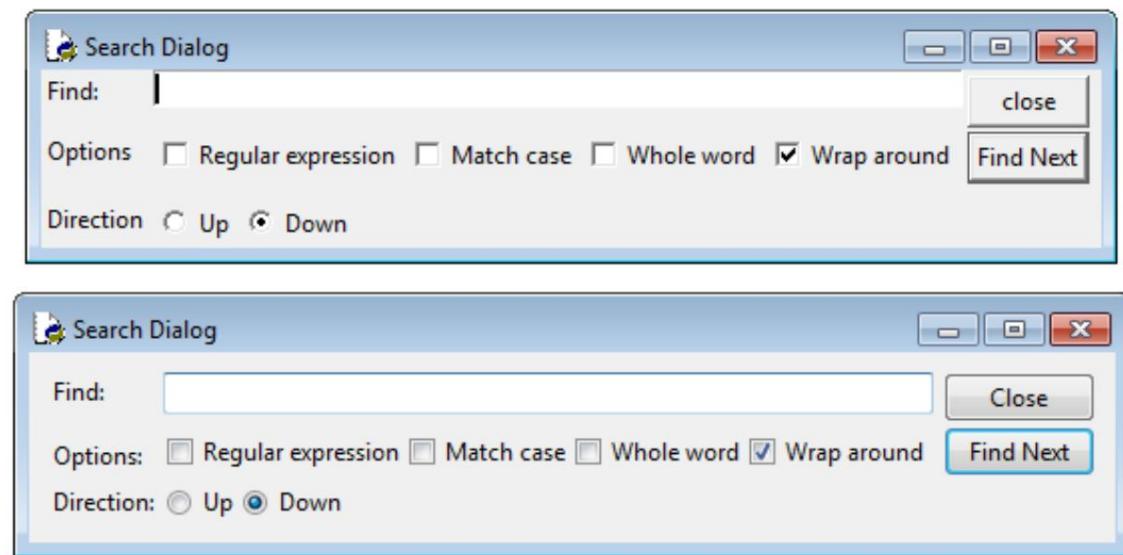
Caixa de Diálogo Localizar

Planejando fazer mais algumas alterações neles, provavelmente combinando Localizar e Substituir em uma única caixa de diálogo.

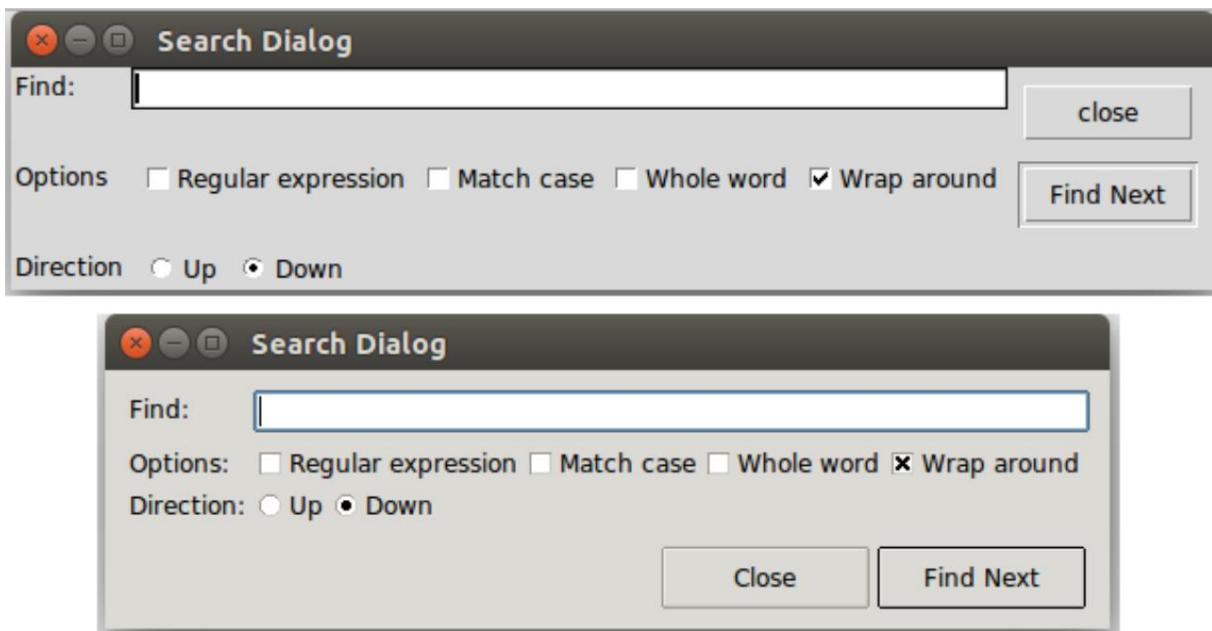
Uma das caixas de diálogo de localização/pesquisa é mostrada abaixo [Problema#23218], novamente com antes e depois nas três plataformas.



Caixa de diálogo IDLE Find, antes e depois (Mac OS X)



Diálogo IDLE Find, antes e depois (Windows)



Diálogo IDLE Find, antes e depois (Linux)

A primeira etapa foi atualizar os widgets para usar os widgets temáticos equivalentes. O efeito disso é mais óbvio com os botões (ser consistente com a capitalização também não prejudica).

Falando em botões, observe que na nova versão, os botões estão à direita no Windows (onde estavam antes), mas na parte inferior para Mac OS X e Linux. Ao examinar vários aplicativos diferentes que nossos usuários-alvo provavelmente encontrariam em cada plataforma, descobrimos que esses locais eram comuns. Com apenas algumas linhas de código extra necessárias para criar um caso especial para essa diferença de layout, fazia sentido lidar com as coisas dessa maneira.

Outros aspectos do layout também foram melhorados. Olhando para as caixas de diálogo originais (especialmente como os botões não se alinham com outros widgets), pensei originalmente que foi criado com pack e esperava convertê-lo em grade. Curiosamente, já estava usando grid. Por que então os widgets desalinhados, uma marca registrada dos layouts baseados em pacotes?

Isso acabou sendo resultado do uso de muitos *quadros aninhados*. Por exemplo, os botões foram colocados em seu próprio quadro e, em seguida, colocados no restante da janela. Por causa disso, os botões individuais não podiam ser alinhados com widgets no restante da interface do usuário.

Os quadros aninhados eram necessários ao usar o pacote, mas geralmente com grade é melhor evitá-los, exceto em circunstâncias excepcionais. Isso permite que você faça com que diferentes partes da interface do usuário se alinhem (às custas de lidar com muitos ajustes de columnpan/rowspan).

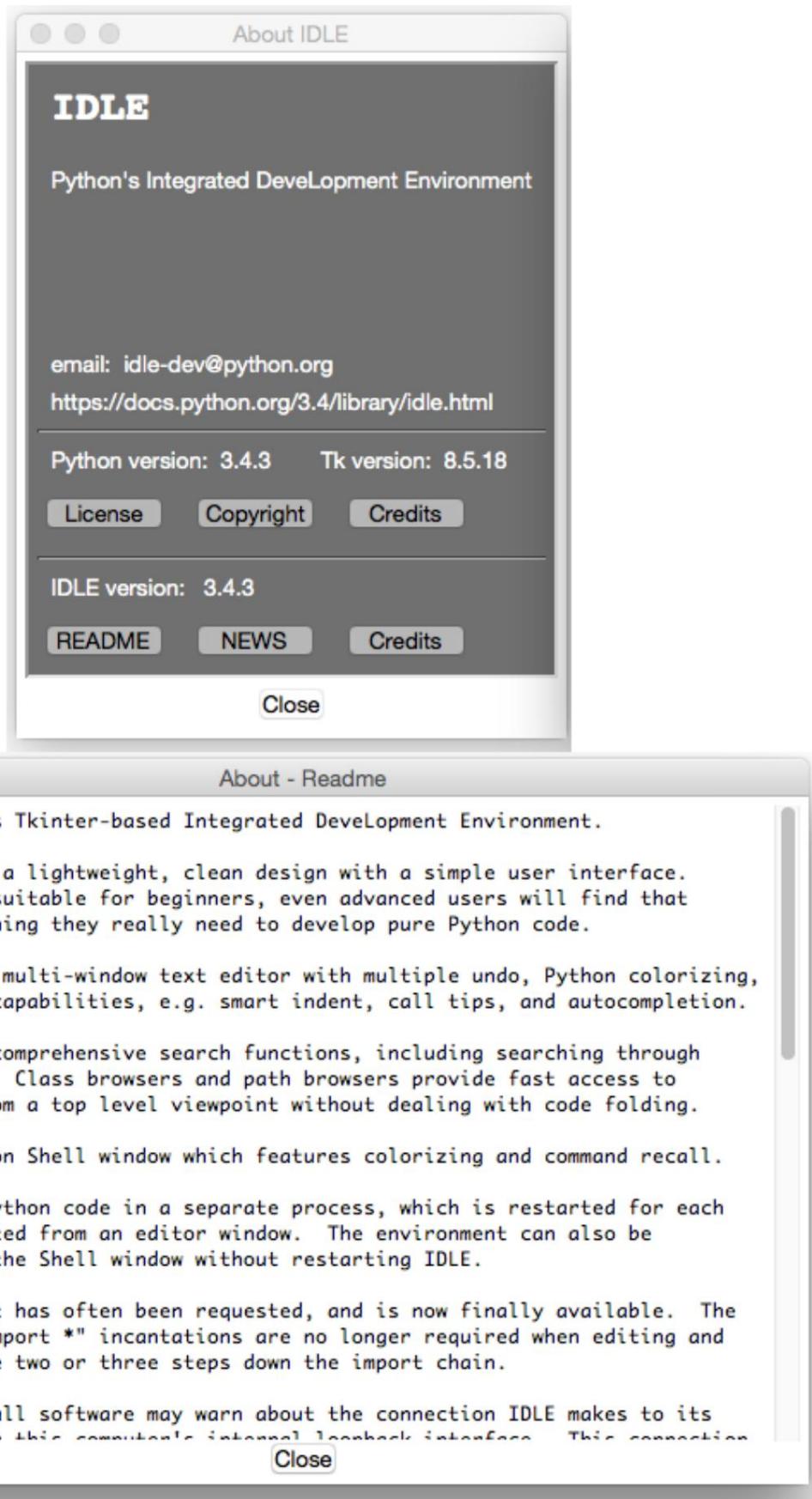
Uma reorganização mais substancial provavelmente teria removido os quadros aninhados, mas particularmente com o ajuste dos botões, esse layout relativamente simples poderia funcionar com apenas alguns ajustes.

Você notará que muitas interfaces de usuário Tkinter mais antigas têm o problema de seu conteúdo correr para a borda das janelas, o que geralmente não parece certo. Adquiri o hábito de colocar um `ttk.Frame` diretamente dentro de cada nível superior, com algum preenchimento adicional e, em seguida, colocar o restante da interface do usuário dentro dele.

Como as Preferências, as caixas de diálogo Localizar também eram modais, o que significava que você tinha que ignorá-las antes de fazer qualquer edição em seu arquivo, embora pelo menos elas se lembressem das configurações anteriores quando você as reabrisse. Estes foram eventualmente alterados para serem janelas sem janela modais regulares.

Sobre a caixa de diálogo

Falando em caixas de diálogo modais, a caixa Sobre também era originalmente restrita. Além disso, obter mais informações (por exemplo, o arquivo IDLE README), resultou no lançamento de *outro* diálogo modal, que precisava ser descartado para voltar ao primeiro diálogo modal, que precisava ... etc.



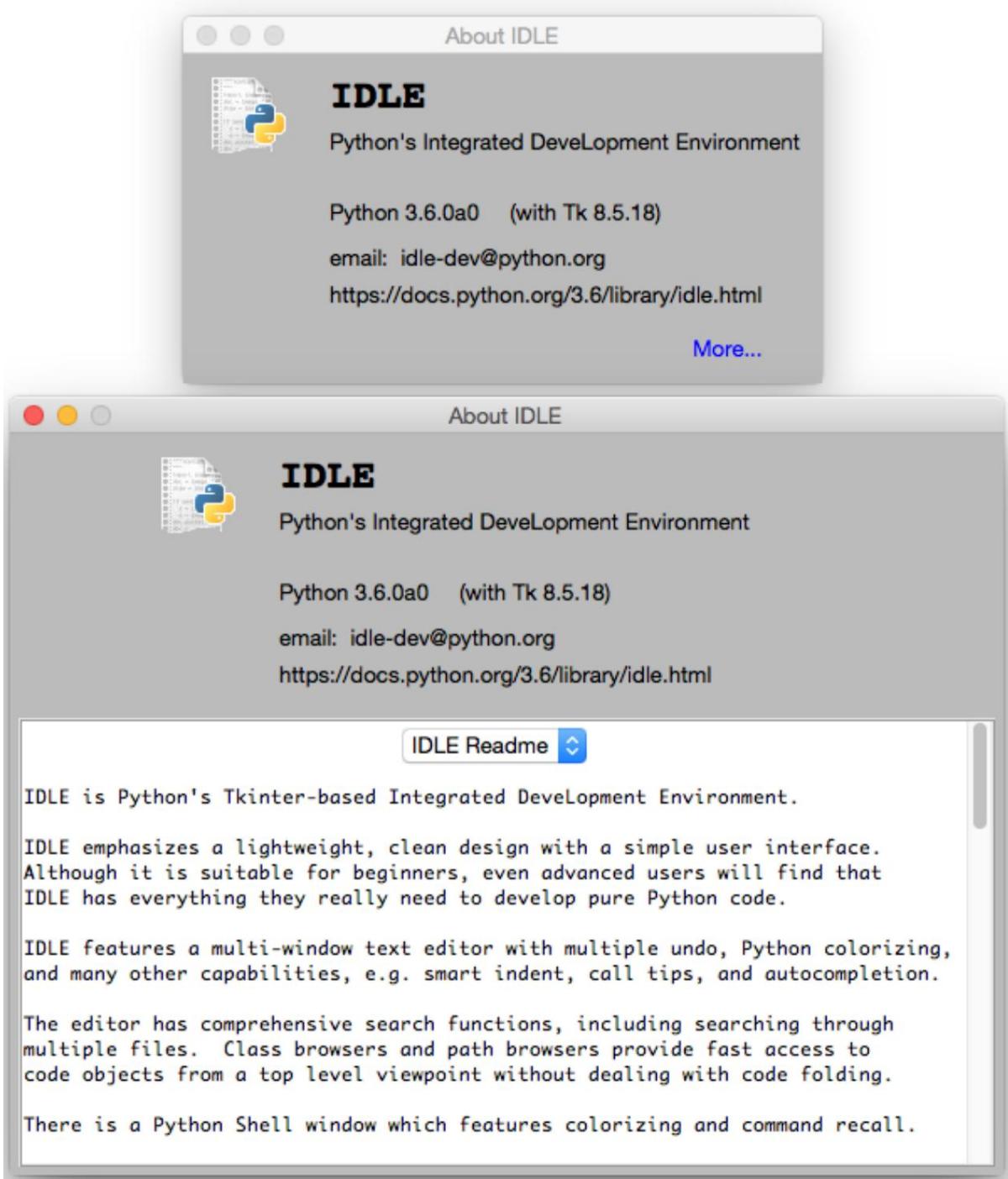
Diálogo Sobre do IDLE duplamente modal (Mac OS X)

Tornar o diálogo não modal era a primeira prioridade. Em segundo lugar, os diálogos aninhados foram eliminados,

usando uma técnica de divulgação progressiva. A caixa de diálogo quando iniciada é bastante esparsa, mas contém um "link" 'Mais ...', que quando clicado expande a janela para mostrar um dos arquivos de documentação, com a opção de alternar para qualquer um dos outros.

O "link" 'Mais...' desempenha efetivamente o papel de um botão, mas aproveita a familiaridade de todos com os navegadores da Web para fornecer uma alternativa visualmente mais simples. No que diz respeito à implementação, usamos um widget de rótulo (clássico, não temático), colorido de azul, e anexamos um vínculo de clique do mouse a ele.

Uma coisa que você pode fazer para ajudar a transmitir que o link é clicável é alterar o cursor quando o mouse estiver sobre ele (através da opção "cursor" encontrada em muitos widgets). No OS X, escolha o cursor "pointinghand" específico da plataforma, no Windows e no Linux, escolha o cursor "hand2", que na verdade é mapeado para algo mais apropriado nessas plataformas.



Sobre a caixa revisada (Mac OS X)

Ajuda online

O IDLE também tinha uma caixa de diálogo de ajuda (desnecessário dizer, modal) que exibia informações sobre o uso do programa.

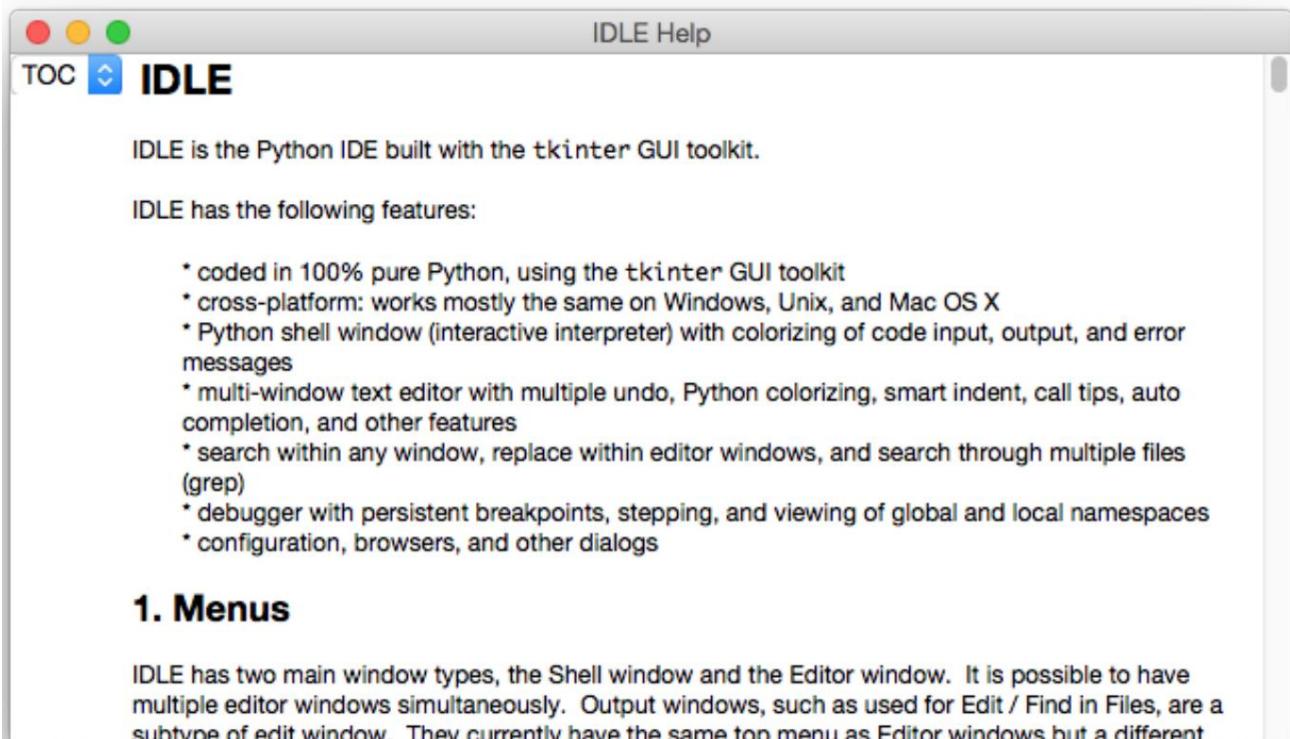
Isso exibia um arquivo de texto simples semelhante à janela 'Sobre - Leiamer' acima.

Ao mesmo tempo, como o restante do Python, havia [documentação de referência](#), criado como "texto reestruturado" que pode ser formatado usando uma ferramenta chamada [Sphinx](#) em HTML, texto, etc.

A documentação na caixa de diálogo de ajuda do IDLE foi baseada em um arquivo de texto simples separado, mas semelhante.

Manter os dois em sincronia era um problema. Eles foram separados porque a renderização de texto simples do Sphinx não parecia muito boa, e toda a navegação extra etc. [Problema nº 16893](#)

O widget de texto do Tk é inteligente o suficiente para lidar com a formatação básica usada na documentação do IDLE, e o Python inclui um analisador de HTML em sua biblioteca padrão. Colocar os dois juntos facilitou a exibição de uma versão simplificada da documentação de referência HTML.



The screenshot shows a Mac OS X window titled "IDLE Help". Inside, the title "IDLE" is displayed above a text area. The text area contains the following content:

IDLE is the Python IDE built with the `tkinter` GUI toolkit.

IDLE has the following features:

- * coded in 100% pure Python, using the `tkinter` GUI toolkit
- * cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- * Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- * multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- * search within any window, replace within editor windows, and search through multiple files (grep)
- * debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- * configuration, browsers, and other dialogs

1. Menus

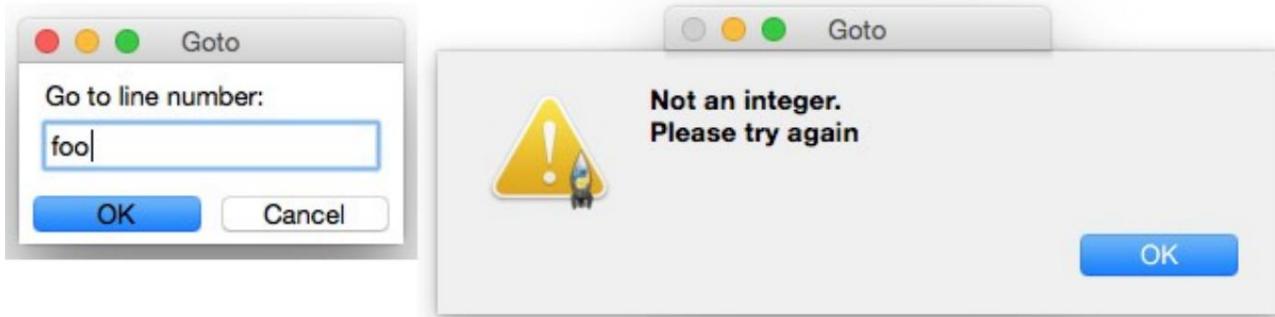
IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit window. They currently have the same top menu as Editor windows but a different

Ajuda online usando widget de texto (Mac OS X)

É fácil se deixar levar por aqui. Em meados da década de 1990, um pequeno [analisador de HTML em Tcl](#) gerou uma série de aventuras de "navegador da web em um widget de texto", primeiro em Tcl e depois em outros idiomas, por exemplo Graal de Python. Tentar acompanhar tudo o que grandes equipes de desenvolvedores estão colocando em navegadores da Web reais é uma missão tola. No entanto, para subconjuntos muito limitados e restritos de HTML, como pode ser encontrado na ajuda on-line, é uma abordagem totalmente razoável.

diálogos de consulta

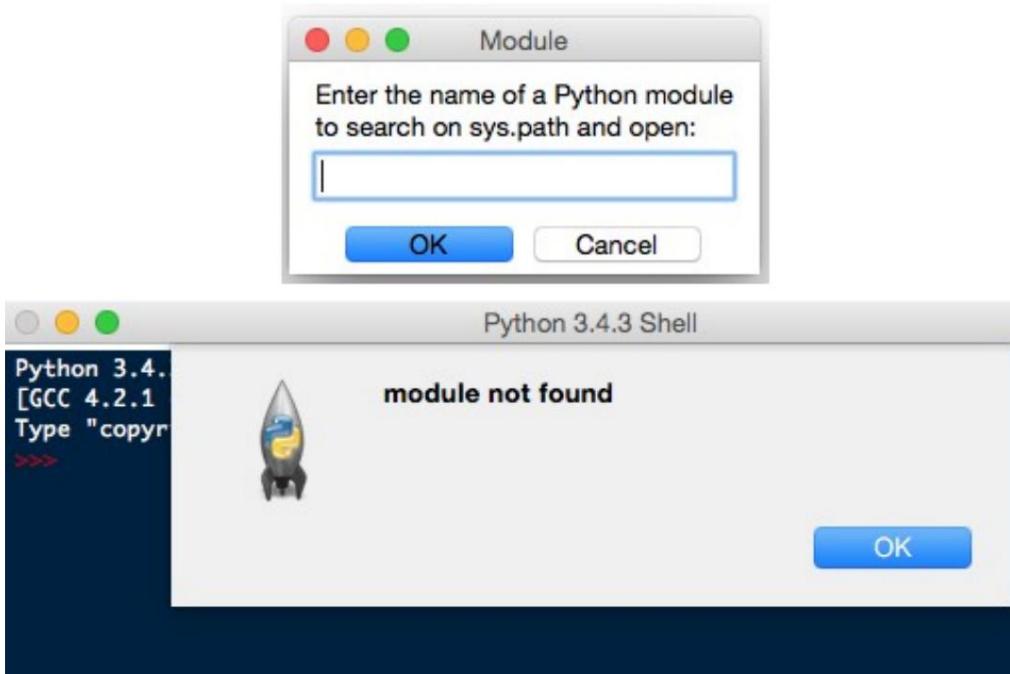
Ainda com o tema do diálogo modal, o IDLE usou o pacote "simpledialog" que é distribuído como parte do Tkinter para solicitar certas pequenas informações do usuário por meio de diálogos modais. Um exemplo é o comando "Goto line...". Isso, juntamente com um exemplo do alerta que é apresentado se você digitar algo inválido, é mostrado abaixo.



Caixa de diálogo Ir para linha e tratamento de erros (Mac OS X)

O alerta na caixa de diálogo não é tão ruim quanto o padrão de caixa de diálogo que vimos antes. Mas esses diálogos certamente poderiam ser atualizados cosmeticamente e alguns outros ajustes feitos. Por exemplo, embora eles interpretassem corretamente as teclas Return ou Escape como sinônimo de pressionar OK ou Cancelar, eles também não permitiam as convenções alternativas do Mac (tecla Enter no teclado numérico e ponto de comando) [Problema nº 24812]. Havia algumas outras personalizações que seria bom ter, por exemplo, alterar o nome do botão OK, que simpledialog não apoia.

Em relação ao tratamento de erros, em alguns casos as coisas foram tratadas pior. Por exemplo, existe um comando para abrir uma janela do editor contendo a fonte de um módulo do stdlib do Python. Veja o que acontece se você cometer um erro de digitação.



Diálogo de módulo aberto com erro (Mac OS X)

Como a caixa de diálogo não sabe se o nome do módulo está correto, a validação não é feita até que a caixa de diálogo seja descartada. Portanto, o alerta de erro é anexado a outra janela e, para tentar novamente, você deve ignorá-lo e ir ao menu para reabrir a caixa de diálogo e tentar novamente (do zero).

Havia alguns lugares no código onde era realmente necessária mais validação na caixa de diálogo.

Como o código "simpledialog" fazia parte do Tkinter e não era prontamente extensível nas formas necessárias, os desenvolvedores tiveram que recorrer à "herança por editor de texto" (ou seja, copiar todo o código simpledialog e modificar a cópia). Duas vezes, separadamente.

Como a aparência precisava ser atualizada de qualquer maneira, generalizamos as coisas e ignoramos completamente o que estava no módulo simpledialog do Tkinter. Em vez disso, foi criada uma única substituição de uso geral que poderia ser usada em todo o IDLE (e ainda resultaria em menos código).



Nova caixa de diálogo de consulta (Mac OS X)

Além dos widgets e alinhamento atualizados, observe como as mensagens de erro de entrada inválida agora são mostradas na própria caixa de diálogo (uma técnica vista com frequência em aplicativos da Web), em vez de um alerta separado. Para o OS X, também adicionamos atalhos de teclado para a tecla Enter do teclado numérico e o ponto de comando, e também garantimos que a janela se parecesse com uma caixa de diálogo modal por meio desta pequena joia:

```
self.tk.call('::tk::unsupported::MacWindowStyle', 'estilo', self._w, 'moveableModal', '')
```

No que diz respeito à validação, a caixa de diálogo de consulta foi estruturada para aceitar um retorno de chamada de validação, que poderia lidar com critérios arbitrários. Por exemplo, aqui está o código de validação usado quando as pessoas inserem o nome de um novo tema, para garantir que ele atenda a certos requisitos sintáticos e também não tenha sido usado.

```
def newtheme(self): def
    valid_theme(s): if not s: raise
        ValueError('Não pode
            ficar em branco') if len(s) > 30: raise
        ValueError('Não pode ter mais de 30 caracteres') if s
            em self.all_theme_names(): raise ValueError('Nome já usado') new_theme
            = querydialog.askstring(parent=self, prompt='...', title='Criar Novo Tema', validcmd=validar_tema)
            se new_theme não for None:
                ...
            ...
```

Um retorno de chamada de validação 'inteiro' genérico, com mínimo e máximo opcionais, foi adicionado como parte do módulo de diálogo de consulta para ser usado em diálogos como o diálogo 'Ir para a linha...'.

Posicionamento da Caixa de Diálogo

Várias caixas de diálogo, incluindo alertas, salvamento de arquivo etc., apareciam no meio da tela,

em vez de perto da janela à qual estavam associados. [\[Problema nº 25173\]](#) _____

Escolher a janela certa como pai da caixa de diálogo é importante, porque garante que a janela de diálogo apareça perto dessa janela. No Mac OS X, essas caixas de diálogo geralmente são anexadas à barra de título da janela principal (consulte os alertas de erro na seção anterior).

As extremidades frontais dessas caixas de diálogo no Tkinter suportam tanto um mestre quanto um pai. Embora na maioria das vezes, "mestre" e "pai" sejam usados de forma intercambiável no Tkinter, esse não é o caso aqui. Se você fornecer apenas o mestre, a caixa de diálogo não será anexada a essa janela (mas a caixa de diálogo ainda precisa de uma janela existente para criar a caixa de diálogo, e é por isso que o parâmetro mestre está lá). Se a caixa de diálogo estiver associada a outra janela, certifique-se de usar o parâmetro pai.

Integração de janela

Várias pessoas esperavam tornar possível ter tudo exibido em uma única janela, para evitar os problemas de gerenciamento de janelas que às vezes podem atrapalhar as pessoas, por exemplo, [\[Problema#9262\]](#), [\[Problema nº 24826\]](#), [\[Problema nº 24818\]](#).

Abaixo está uma maquete inicial, parcialmente funcional, de algumas das coisas que queríamos ser capazes de realizar.

The screenshot shows the IDLE Python IDE interface. The code editor window displays `uipreferences.py` with a yellow selection bar highlighting the line `x = 5/0`. The status bar at the bottom right shows "Ln: 478 Col: 8". A message bar indicates "Program stopped by exception." Below the code editor is a stack trace:

```

Type "copyright", "credits" or "license()" for more information.
>>> [DEBUG ON]
>>>
===== RUN /Users/roseman/vboxshare/cpython/Lib/idlelib/uipreferences.py =====

```

To the right of the code editor is a "Locals" variable viewer window. It lists the following variables and their values:

	Locals
<code>bg</code>	'#ffffff'
<code>element</code>	'Shell Normal Text'
<code>elt</code>	'console'
<code>fg</code>	'#770000'
<code>self</code>	<__main__.ThemesPane
<code>theme</code>	'IDLE Classic'

Modelo inicial de integração de janela (Mac OS X)

Neste ponto, quase tudo aqui foi concluído e acabou parecendo quase idêntico à maquete original. Veja, de vez em quando acontece!

Editor com guias

Mesmo os programadores iniciantes precisam fazer malabarismos com vários arquivos de origem diferentes. Se cada um obtiver sua própria janela, como era o caso originalmente no IDLE, as coisas podem ficar confusas e/ou perdidas rapidamente. Usar guias para organizar vários arquivos em uma única janela é uma solução familiar e eficaz.

Ao arquitetar seu aplicativo, não construa componentes grandes como subclasses de nível superior ou assuma que eles serão a única coisa na janela no futuro. Contornar essa suposição no código deu muito trabalho. Se os componentes forem construídos como quadros, eles podem ser mais facilmente inseridos em um nível superior, uma janela panorâmica, um notebook com guias, outro quadro, etc.

Felizmente, podemos contar com o widget `ttk.Notebook` para fornecer a interface de usuário com guias, assim como fizemos em Preferências.

Ou talvez não.

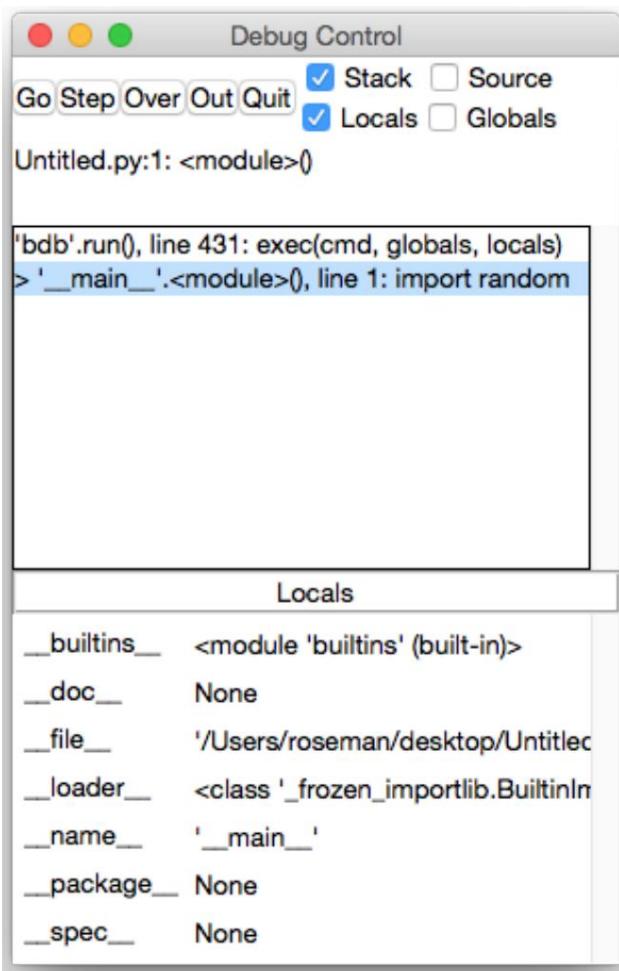
Infelizmente, o widget `ttk.Notebook` (e os widgets de plataforma subjacentes que ele usa) suporta apenas a exibição e alternância entre um pequeno número fixo de guias. Não há nada integrado para oferecer suporte à adição ou fechamento de guias da interface do usuário, o que definitivamente precisamos aqui. E como todo programador sabe, é mais do que possível precisar de um grande número de abas.

Como você viu em diferentes editores e processadores de texto, todos fazem as coisas de maneira ligeiramente diferente. Fizemos nosso próprio widget de guia personalizado (suspiro...) para o IDLE. O design foi fortemente emprestado [do TextMate](#) editor no Mac. Ele permite criar novas abas, fechar as antigas, arrastar para reorganizar a ordem, dicas de ferramentas em cada aba, indicar se o arquivo contido precisa ser salvo, etc. menu na última guia visível.

A implementação do widget com guias depende de uma única tela do Tkinter para exibir a linha de guias e lidar com todas as interações. A troca real do conteúdo da janela é realizada completamente fora do widget, com um mecanismo de retorno de chamada simples usado para coordenar tudo.

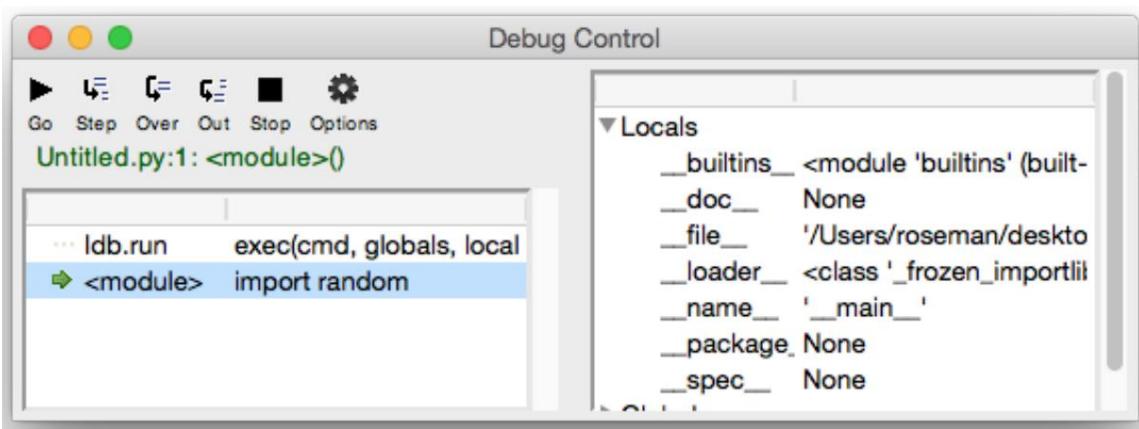
depurador

O design do depurador original, além de ter seu próprio conjunto de falhas [\[Problema nº 17942\]](#), [era muito alto para ser integrado](#) como queríamos fazer.



Depurador IDLE original (Mac OS X)

A interface do usuário foi substancialmente revisada, com um layout que funcionaria tanto em janela autônoma quanto integrada. A nova versão usa uma janela em painel para separar os controles e a pilha à esquerda da exibição variável à direita. Tanto a pilha quanto a exibição de variáveis são implementadas usando widgets de visualização em árvore. Isso também fornece um grande controle quando se trata de quanto espaço cada elemento usará.



Novo depurador IDLE (Mac OS X)

Shell e depurador integrados

Para obter a integração do shell e do depurador com o editor de guias, outra janela em painel foi usada. Controles adicionais serão adicionados para mostrar/ocultar os painéis à medida que a implementação avança.

O shell embutido também é interessante. Lembre-se de que o IDLE normalmente tem uma única janela de shell Python executando outro processo Python; quando os módulos em um editor são "executados", eles o fazem por meio desse shell. É bom ter esse shell grande disponível e não queremos necessariamente iniciar um shell separado no editor.

Novidade no Tk 8.5, o widget de texto na verdade suporta a capacidade de criar "pares", que são widgets separados, mas compartilham o mesmo back-end de texto. Isso significa que quando algo muda em um, muda no outro. É uma maneira perfeita de resolver nosso problema aqui.

Soluções alternativas, hacks e muito mais

Sendo este um capítulo sobre experiências reais que modernizam um aplicativo real, seria uma mentira dizer que o kit de ferramentas subjacente da interface do usuário (Tk e o wrapper Tkinter) sempre funcionou exatamente como deveria. Assim como IDLE, Tkinter e Tcl/Tk são resultados de incríveis esforços voluntários. Dito isso...

Tk e Tkinter têm alguns bugs e arestas. Eu sei, você está chocado.

Nesta seção, tentarei catalogar apenas algumas das "pegadinhas" específicas que encontramos, além de fornecer algumas pequenas dicas que não se encaixam necessariamente em outro lugar, mas ajudam a fornecer um pouco mais de polimento à interface do usuário. .

Dicas de ferramentas

- dicas de ferramentas quebradas no mac [\[Problema#24570\]](#)
- Ajuda MacWindowStyle para dicas de ferramentas, vs. wm overrideredirect em outro lugar

Menus de Contexto

- vinculações falsas de widgets de texto interferem nos cliques pop-up [\[Problema nº 24801\]](#) • dois vínculos de clique pop-up diferentes necessários no Mac [\[Problema#24801\]](#)

Widgets de texto de mesmo nível

- API de widget de texto de mesmo nível quebrada no Tkinter [\[Problema#17945\]](#)

Janelas Modais

- o mac não funciona totalmente modal, além do estilo de janela [\[Problema#24760\]](#)