

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Introdução

Apresentação

- Mario Benevides e Paulo Roma
 - Professores Titulares
 - Dept. Ciência da Computação - IM
 - COPPE/Sistemas
- Aulas fortemente baseadas nas notas de aula do Professores Claudio Esperança (COPPE/Sistemas) e Paulo Roma;
- <http://orion.lcg.ufrj.br/python/>
- **Agradecimento Especial ao Prof. Claudio Esperança** pela suas excelentes notas de aula.

Agenda

Primeira Parte:

- Introdução à Orientação a Objeto
- Classes
- Exceções
- Módulos
- Arquivos

Segunda Parte:

- Interfaces Gráficas

Orientação a Objetos

- É uma disciplina de programação assim como a Programação Estruturada
- Tenta unificar as idéias de algoritmos e estruturas de dados através do conceito de Objeto
 - Um objeto é uma unidade de software que encapsula algoritmos e os dados sobre o qual os algoritmos atuam
 - Um objeto é um **tipo abstrato de dados** na prática
- Os seguintes conceitos são importantes quando falamos de orientação a objetos:
 - Polimorfismo
 - Abstração
 - Herança

Polimorfismo

- É o que permite que dois objetos diferentes possam ser usados de forma semelhante
- Por exemplo, tanto listas quanto tuplas ou strings podem ser indexadas por um número entre colchetes e suportam o método len
- Assim, se escrevemos

```
for i in range(len(X)): print (i, X[i])
```

- Não é possível saber de antemão se X é uma tupla, uma lista ou uma string
- Desta forma, se escrevemos um algoritmo para ser aplicado um objeto X, então também pode ser aplicado a um objeto Y desde que Y seja suficientemente polimórfico a X

Abstração e Encapsulamento

- É o que permite que um objeto seja utilizado sabendo-se sobre ele apenas a sua interface
- Um objeto é uma unidade de software que encapsula algoritmos e os dados sobre o qual os algoritmos atuam
- Em OO a abstração tem mais alcance pois um objeto encapsula tanto dados como algoritmos
- Assim, podemos atribuir objetos ou passar objetos como argumentos, sem necessariamente saber como o objeto está implementado

Herança

- É o que permite construir objetos que são especializações de outro objeto
 - Isso permite o re-uso de software já que objetos especializados herdam dos objetos genéricos uma série de atributos comuns
- Por exemplo, considere um objeto que representa uma forma geométrica. Então, ele pode ter características tais como área, perímetro, centróide, etc.
- Um polígono é uma forma geométrica
 - Portanto, herda todas as características de formas geométricas
 - Deve suportar também características específicas como número de lados e comprimento de arestas

Objetos em Python

- Python suporta OO através de `classes`
- Uma classe pode ser entendida como uma fábrica de objetos, todos com as mesmas características
 - Diz-se que objeto fabricado por uma classe é uma instância da classe
- A rigor, uma classe é também um objeto
 - Encapsula dados e algoritmos
 - Entretanto, não é normalmente um objeto fabricado por uma classe, mas um objeto criado pela construção `class`
- Um objeto encapsula dados e algoritmos sob a forma de variáveis e métodos
 - É comum chamar esses elementos constituintes dos objetos de atributos

Motivação p/ Usar OO

- Exemplo: `Circulo`: centro (x,y) e um raio
- Implementar: `lista`: `circulo (x , y , raio)`
- Exemplo: `circulo=("3", "5", "7")`
 - `circulo[2] = "-5"`, Este erro só seria percebido quando formos usar o `circulo`
 - `circulo.sort()`, Este erro é difícil de perceber
- Usando OO
 - tratar exceções
 - encapsular métodos e dados
 - re-utilizar código

Declarando Classes

- A maneira mais simples é:

```
class nome (object):
    var = valor
    ...
    var = valor
    def metodo (self, ... arg):
        ...
    def metodo (self, ... arg):
        ...
```

- As variáveis e os métodos são escritos precedidos pelo nome da classe e por um ponto (.)
 - Assim, uma variável v definida numa classe c é escrita c.v
- Os métodos sempre têm self como primeiro argumento
- Uma nova instância da classe é criada usando nome ()

Exemplo 1

- Uma classe C com um método f:

```
class C (object):  
    a = 2  
    b = 3  
    def f(self, x):  
        return C.a*x+C.b  
exemplo = C()  
print ( exemplo.f(7))
```

- Executando o programa exemplo:

```
>>>
```

17

Atributos e Instâncias

- No exemplo anterior, a e b eram atributos da classe C e portanto usáveis por qualquer instância de C
- Um atributo attr associado a uma instância obj tem nome obj.attr

```
class C(object):  
    a = 2  
    b = 3  
    def f(self, x):  
        return C.a*x+C.b  
exemplo = C()  
print ( exemplo.f(7))
```

```
>>> C.a = 9  
>>> exemplo.f(7)
```

66

Atributos e Instâncias

- Freqüentemente, atributos associados a instâncias individuais
- Referir atributo `at` de objeto dentro de seus métodos: `self.at`

```
class C(object):
    def init(self,a=2,b=3):
        self.a = a
        self.b = b
    def f(self,x):
        return self.a*x+self.b
>>> obj1 = C()
>>> obj1.init(2,3)
>>> obj2 = C()
>>> obj2.init(8,1)
>>> obj1.f(7)
17
>>> obj2.f(7)
57
```

Exemplo 2: Classe C time de futebol

```
class Time(object):
    ## Classe dos times de futebol
    def __init__(self, nome, campeao, divisao1):
        self.nome = nome
        self.campeao = campeao
        self.divisao1 = divisao1

    def descricao(self):
        ff = "Eu sou %s. Somos %s" % (self.nome, self.campeao)
        print ( ff )

    def div(self):
        if self.divisao1:
            print ( "Somos da Primeira Divisão." )
        else:
            print ( "Estamos na Segunda Divisão." )
```

Exemplo 2: Classe C time de futebol

- Instanciando a classe Time para o objeto Fluminense

```
>>>Fluminense = Time("tricolor", "tetra-campeões", True)  
>>>Fluminense.descricao()
```

Eu sou tricolor. Somos tetra-campeões

```
>>>Fluminense.div()
```

Somos da Primeira Divisão.

Resumo

Nesta Aula : Introdução a OO e Classes

Aula Seguinte: Classes (Continuação)

- Atributos herdados da classe
- Método init
- Especialização de classes
- Herança Múltipla
- Métodos Mágicos
- Getters, Setters e Propriedades

Programação com Interfaces Gráfica

Mario Benevides

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Classes

Agenda

Aula Passada: Introdução a OO e Classes

Nesta Aula: Classes (Continuação)

- Atributos herdados da classe
- Método init
- Especialização de classes
- Herança Múltipla
- Métodos Mágicos
- Getters, Setters e Propriedades

Atributos herdados da classe

- Se uma classe define atributos de classe, as instâncias herdam esses atributos da classe como atributos de instância
- Exemplo:

```
>>> class C(object):
    a = 1
    def f(self,x):
        self.a += x

>>> c = C()
>>> c.f(2)
>>> c.a
3
>>> C.a
1
```

Método init

- Um método como `init` do exemplo da classe `Time` é bastante útil para inicializar atributos da instância e é conhecido como construtor da classe
- Na verdade, Python suporta construtores que podem ser chamados automaticamente na criação de instâncias
 - Basta definir na classe um método chamado `__init__`
 - Este método é chamado automaticamente durante a criação de uma nova instância da classe, sendo que os argumentos são passados entre parênteses após o nome da classe
- O método `__init__` é apenas um exemplo de “método mágico” que é invocado de maneira não padrão

Exemplo

```
>>> class C(object):
    def __init__(self,a=2,b=3):
        self.a = a
        self.b = b
    def f(self,x):
        return self.a*x+self.b

>>> obj1 = C()
>>> obj2 = C(8,1)
>>> obj1.f(7)
17
>>> obj2.f(7)
57
```

Especialização de classes

- Fazer uma classe C herdar de outra B, basta declarar C como:

```
class C(B):
```

- Diz-se que C é sub-classe (ou derivada) de B ou que B é super-classe (ou base) de C
- C herda todos os atributos de B
- A especialização de C se dá acrescentando-se novos atributos (variáveis e métodos) ou alterando-se métodos
- Se, um método de C, precisa invocar um método m de B, pode-se utilizar a notação B.m para diferenciar do m de C, referido como C.m

Exemplo: Especialização de classes

```
>>> class B(object):
n = 2
def f(self,x): return B.n*x
>>> class C(B):
def f(self,x): return B.f(self,x)**2
def g(self,x): return self.f(x)+1
>>> b = B()
>>> c = C()
>>> b.f(3)
6
>>> c.f(3)
36
>>> c.g(3)
37
>>> B.n = 5
>>> c.f(3)
225
```

Observação: Especialização de classes

- O parâmetro self não pode ser removido da chamada da função f de B, na classe C, do exemplo anterior:

```
>>> class C(B):  
def f(self,x): return B.f(x)**2  
def g(self,x): return self.f(x)+1
```

```
>>> c=C()  
>>> print (c.f(3))
```

- ERRO!!!**

Observação: Especialização de classes

- Para chamar um classe D derivada de uma classe C é preciso chamar C de maneira a inicializá-la.
- Isto não é feito automaticamente (por default).
- Permite inicializar os elementos de C que não são específicos de D
- Usa-se a notação C.__init__(self, ...)
- Exemplo:

```
>>> class C(object):
    def __init__(self):
        print ("Construtor de C")
        self.x = 1

>>> class D(C):
    def __init__(self):
        print ("Construtor de D")
        C.__init__(self)
        self.y = 2
```

Exemplo: Especialização de classes

```
>>> class C(object):
    def __init__(self):
        print ("Construtor de C")
        self.x = 1

>>> class D(C):
    def __init__(self):
        print ("Construtor de D")
        C.__init__(self)
        self.y = 2

>>> d=D()
Construtor de D
Construtor de C
>>> d.x
1
>>> d.y
2
```

Observação:

- A partir do Python 2.2, classes podem também ser declaradas no chamado “novo estilo”:
- Se uma classe não é derivada de nenhuma outra, ela deve ser declarada como derivada da classe especial chamada object.
- Exemplo:

```
class C(object):
```

- É possível construir uma classe que herda de duas ou mais outras.
Ex.: class C(A,B): ...
- Nesse caso, a classe derivada herda todos os atributos de ambas as classes-base
- Se ambas as classes base possuem um atributo com mesmo nome, aquela citada primeiro prevalece
- No exemplo acima, se A e B possuem um atributo x, então C.x se refere ao que foi herdado de A

Exemplo: Herança Múltipla

```
class C(object):
    def __init__(self,a,b):
        self.a, self.b = a, b
    def f(self,x):
        return self.a*x+self.b
class D(object):
    def __init__(self,legenda):
        self.legenda = legenda
    def escreve(self,valor):
        print (self.legenda,'=',valor)
class E(C,D):
    def __init__(self,legenda,a,b):
        C.__init__(self,a,b)
        D.__init__(self,legenda)
    def escreve(self,x):
        D.escreve(self,self.f(x))
```

Exemplo: Herança Múltipla

```
class C(object):
    def __init__(self,a,b):
        self.a = a
        self.b = b
    def f(self,x):
        return self.a*x+self.b
class D(object):
    def __init__(self,legenda):
        self.legenda = legenda
    def escreve(self,valor):
        print (self.legenda,'=',valor)
class E(C,D):
    def __init__(self,legenda,a,b):
        C.__init__(self,a,b)
        D.__init__(self,legenda)
    def escreve(self,x):
        D.escreve(self,self.f(x))
e = E("f",10,3)
print (e.escreve(4))
>>>
f = 43
```

Atributos Privados

- Em princípio, todos os atributos de um objeto podem ser acessados tanto dentro de métodos da classe como de fora.
- Quando um determinado atributo deve ser acessado apenas para implementação da classe, ele não deveria ser acessível de fora.
- Em princípio tais atributos não fazem parte da interface “pública” da classe.
- Atributos assim são ditos privados.
- Em Python, atributos privados têm nomes iniciados por dois caracteres “traço-embaixo”, isto é, `__`.

Exemplo: Atributos Privados

```
class C(object):
    def __init__(self,x): self.__x = x
    def incr(self):
        self.__x += 1
        return self.__x
a = C(5)
print (a.incr())
```

```
>>>
```

```
6
```

```
>>> a.__x
```

ERRO!!!

Métodos Mágicos

- São métodos que são invocados usando operadores sobre o objeto ao invés de por nome.
- Exemplo: o construtor `__init__`
- Outros:
- Adição: `__add__`
 - Chamado usando '+'
- Subtração: `__sub__`
 - Chamado usando '-'
- Adição: `__repr__`
 - Chamado quando objeto é impresso

Exemplo: Métodos Mágicos

```
class vetor:  
    def __init__(self,x,y):  
        self.x, self.y = x,y  
    def __add__(self,v):  
        return vetor(self.x+v.x, self.y+v.y)  
    def __sub__(self,v):  
        return vetor(self.x-v.x, self.y-v.y)  
    def __repr__(self):  
        return "vetor("+str(self.x)+","+str(self.y)+"")"  
  
    >>> a=vetor(1,2)  
    >>> a += vetor(3,5)  
    >>> a-vetor(2,2)  
vetor(2,5)  
    >>> print (a)  
vetor(4,7)
```

Getters, Setters e Propriedades

- Muitas vezes queremos que determinados atributos possam ser acessados de forma controlada, isto é, vigiados por métodos
- Os métodos que controlam o acesso a tais atributos são conhecidos como getters e setters , referindo-se a métodos de leitura e escrita, respectivamente
- Os atributos controlados são chamados de propriedades
- Úteis para **Encapsulamento** e proteger atributos

Exemplo 1: Getters, Setters e Propriedades

```
class C:

    def __init__(self,x):
        self.setX(x)

    def getX(self):
        return self.__x

    def setX(self, x):
        if x < 0: print ("Valor menor que 0")
        elif x > 1000: print ("Valor maior que 1000")
        else:
            self.__x = x
```

Exemplo 1: Getters, Setters e Propriedades

```
class C:

    def __init__(self,x):
        self.setX(x)

    def getX(self):
        return self.__x

    def setX(self, x):
        if x < 0: print ("Valor menor que 0")
        elif x > 1000: print ("Valor maior que 1000")
        else:
            self.__x = x

>>> c1= C(100)
>>> c2= C(150)
>>> c1.setX(c1.getX()+c2.getX())
>>> c1.getX()
250
>>>c1.setX(-10)
Valor menor que 0
```

Exemplo 2: Getters, Setters e Propriedades

```
class Retangulo:  
    def __init__(self,tamanho):  
        self.setTamanho(tamanho)  
    def setTamanho(self,tamanho):  
        if min(tamanho)<0: print ("Erro!")  
        self.__tamx,self.__tamy = tamanho  
    def getTamanho(self):  
        return (self.__tamx,self.__tamy)  
  
>>> r = Retangulo((20,30))  
>>> r.getTamanho()  
(20, 30)  
>>> r.setTamanho((-1,0))  
Erro!
```

Getters, Setters e Propriedades

- A função `property` pode ser usada para consubstanciar uma propriedade implementada por métodos de tal maneira que ela pareça um atributo da classe
- Ela é usada no corpo de uma declaração de classe com a forma:
`atributo = property(fget, fset, fdel, doc)`
- `fget`, `fset`, `fdel` são métodos para ler, escrever e remover o atributo
- `doc` é uma docstring para o atributo

Exemplo: Propriedades

```
class Retangulo:  
    def __init__(self,tamanho):  
        self.setTamanho(tamanho)  
    def setTamanho(self,tamanho):  
        if min(tamanho)<0: print ("Erro!")  
        self.__tamx,self.__tamy = tamanho  
    def getTamanho(self):  
        return (self.__tamx,self.__tamy)  
    tamanho = property(getTamanho,setTamanho)
```

```
>>> r = Retangulo((20,30))  
>>> r.tamanho  
(20, 30)  
>>> r.tamanho = (30,30)  
>>> r.tamanho  
(30, 30)  
>>> r.tamanho = (-1,0)  
Erro!
```

- Agrupe funções e dados que se referem a um mesmo problema
- Por exemplo, se uma função manipula uma variável global, é melhor que ambas sejam definidas numa classe como atributo e método
- Não permita promiscuidade entre classes e instâncias de classe
 - Por exemplo, se há necessidade de um objeto manipular um atributo de outro, escreva um método com essa manipulação e chame-o
 - Não escreva métodos extensos
 - Em geral, um método deve ser o mais simples possível

Agenda

Nesta Aula: Classes

Próxima Aula: Exceções

- Objeto de Exceção
- Avisos
- Comando *raise*
- Classes de Exceção
- Mais de um *except*
- Comando *else*
- Comando *finally*
- Iteradores e Geradores

Exceções

Mario Benevides

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Programação com Interfaces Gráfica

Agenda

Aula Passada: Classes

Nesta Aula: Exceções

- Objeto de Exceção
- Avisos
- Comando *raise*
- Classes de Exceção
- Mais de um *except*
- Comando *else*
- Comando *finally*
- Iteradores e Geradores

Exceções

- Quando um programa encontra dificuldades não previstas, diz-se que uma condição excepcional ou uma exceção ocorreu
 - Um erro é uma exceção mas nem toda exceção é um erro
- Para poder representar tais eventos, Python define os chamados objetos de exceção
- Se a condição excepcional não é prevista (e tratada), o programa termina com uma mensagem:

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Objetos de Exceção

- Cada exceção individual corresponde a um objeto de exceção, que por sua vez é uma instância de alguma classe de exceção
- Diz-se que o programa gerou ou levantou (*raised*) uma condição de exceção na forma de um objeto
- Um programa bem elaborado precisa capturar tais objetos e tratá-los para que a execução não seja abortada

Avisos

- Existem condições excepcionais menos sérias que não provocam o levantamento de um objeto de exceção, mas apenas são exibidas sob a forma de um aviso
- Por exemplo:

```
>>> import regex
```

Warning (from warnings module):

File "<__main__>", line 1

DeprecationWarning: the regex module is deprecated; please

- Neste caso, o interpretador nos sinaliza que o módulo regex é antigo e que foi substituído por outro mais atualizado chamado re
- O programa não falha, mas o programador fica ciente que provavelmente deve re-escrever seu programa usando o módulo re para evitar obsolescência

Comando *raise*

- Para sinalizar a ocorrência de uma condição excepcional, pode-se usar o comando *raise* que tem uma das formas:
 - *raise classe*
 - *raise classe (mensagem)*
- Onde classe é uma das classes de exceção definidas pelo Python
 - Para saber todos os tipos de exceção consulte o manual
 - Se quiser uma classe genérica use a classe `Exception`

Exemplo: *raise*

```
>>> raise Exception  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
Exception  
  
>>> raise Exception ("E agora?")  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
Exception: E agora?
```

Classes de Exceção

Classe	Descrição
Exception	Classe base para todas as exceções
AttributeError	Falha no acesso ou atribuição a atributo de classe
IOError	Falha no acesso a arquivo inexistente ou outros de E/S
IndexError	Índice inexistente de sequência
KeyError	Chave inexistente de dicionário
NameError	Variável inexistente
SyntaxError	Erro de sintaxe
TypeError	Operador aplicado a objeto de tipo errado
ValueError	Operador aplicado objeto tipo certo mas valor errado
ZeroDivisionError	Divisão por zero

Criando uma Classe de Exceção

- Basta criar uma classe da forma habitual derivando-a da classe Exception
- Não é preciso redefinir qualquer método
- Exemplo:

```
>>> class MinhaExcecao(Exception): pass  
...  
>>> raise MinhaExcecao("E agora?")
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
__main__.MinhaExcecao: E agora?
```

Capturando Exceções

- Para capturar uma exceção possivelmente levantada por um trecho de código, pode-se usar a construção try/except:

try:

Código

except Exceções:

Código de tratamento da exceção

- Sendo que Exceções pode ser:

- Classe
- Classe,var
- (Classe₁, ... , Classe_n)
- (Classe₁, ... , Classe_n) as var

- Onde:

- Classe₁, ... , Classe_n são nomes de classes de exceção
- var é uma variável à qual é atribuída um objeto de exceção

Exemplo 1:

```
>>> try:  
    a = eval(input("Entre com um numero:"))  
    b = eval(input("Entre com outro numero:"))  
    print (a, "/ ", b , "=" , a/b)  
except ZeroDivisionError:  
    print ("Erro, segundo numero não pode ser zero!")
```

Entre com um numero: 1

Entre com outro numero: 0

1 / 0 = Erro, segundo numero não pode ser zero!

Exemplo 2:

```
try:  
    a = eval(input("Entre com um numero:"))  
    b = eval(input("Entre com outro numero:"))  
    print (a, "/ ", b , " = " , a/b)  
except (ZeroDivisionError,TypeError):  
    print ("Erro, tipo errado ou divisao por zero!")
```

Entre com um numero:1

Entre com outro numero:"a"

Erro, tipo errado ou divisao por zero!

Erro por divisão por zero ou por tipo.

Exemplo 3:

```
try:  
    a = eval(input("Entre com um numero:"))  
    b = eval(input("Entre com outro numero:"))  
    print (a, "/ ", b , " = " , a/b)  
except (ZeroDivisionError,TypeError) as e:  
    print ("Erro:",e)
```

Entre com um numero:1

Entre com outro numero:"a"

Erro: unsupported operand type(s) for /: 'int' and 'str'

Coloca a mensagem de erro numa variável *e*.

Mais de um *except*

- É possível tratar diferentemente as diversas exceções usando duas ou mais cláusulas *except*
- Se quisermos nos prevenir contra qualquer tipo de erro, podemos usar uma cláusula *except* sem nome de classe
 - Outra opção é usar a classe `Exception`, que é base para todas as exceções e portanto casa com qualquer exceção
- Se não quisermos tratar um erro em uma cláusula *except*, podemos passá-la adiante usando o comando `raise`
 - Nesse caso, podemos usar um `raise` sem argumentos ou passar explicitamente um objeto de exceção

Exemplo 4:

```
try:  
    a = eval(input("Entre com um numero:"))  
    b = eval(input("Entre com outro numero:"))  
    print (a, "/ ", b , "=" , a/b)  
except (ZeroDivisionError):  
    print ("Erro: Divisao pro zero:")  
except (TypeError):  
    print ("Erro, tipo errado:")  
except :  
    print ("Erro!!!")
```

Entre com um numero:1

Entre com outro numero:fg12

Erro!!!

Captura um erro qualquer.

Exemplo 5:

```
try:  
    a = eval(input("Entre com um numero:"))  
    b = eval(input("Entre com outro numero:"))  
    print (a, " / ", b , " = " , a/b)  
except (TypeError):  
    print ("Erro, tipo errado:")  
except (Exception) as e:  
    print ("Erro:" , e)  
    raise
```

```
Entre com um numero:1  
Entre com outro numero:f  
Erro: name 'f' is not defined
```

```
Traceback (most recent call last):  
  File "<stdin>", line 3, in <module>  
  File "<string>", line 1, in <module>  
NameError: name 'f' is not defined
```

Captura um erro qualquer e imprime a mensagem de erro retornada em *e*.

Comando *else*

É possível completar um comando *try* com uma cláusula *else* que introduz um trecho de código que só é executado quando nenhuma exceção ocorre:

```
try:  
    Código  
except (Exceções):  
    Código de tratamento da exceção  
else:  
    Código executado se não ocorrem exceções
```

Exemplo: *else*

```
while True:  
    try:  
        a = eval(input("Entre com um numero:"))  
        b = eval(input("Entre com outro numero:"))  
        print (a, "/ ", b , " = " , a/b)  
    except (Exception) as e:  
        print ("Erro:" , e)  
        print ("Tente de novo!")  
    else:  
        break
```

```
Entre com um numero:1  
Entre com outro numero:f  
Erro: name 'f' is not defined  
Tente de novo!  
Entre com um numero:1  
Entre com outro numero:2  
1 / 2 = 0.5
```

Exemplo: *else*

```
while True:  
    try:  
        a = eval(input("Entre com um numero:"))  
        b = eval(input("Entre com outro numero:"))  
        print (a, "/", b , "=" , a/b)  
    except (Exception) as e:  
        print ("Erro:" , e)  
        print ("Tente de novo!")  
    else:  
        break
```

```
Entre com um numero:1  
Entre com outro numero:f  
Erro: name 'f' is not defined  
Tente de novo!  
Entre com um numero:1  
Entre com outro numero:2  
1 / 2 = 0.5
```

Repete enquanto não entra com 2 valores. No caso de erro, captura a mensagem em *e*.

Comando *finally*

- A cláusula *finally* pode ser usada para se assegurar que mesmo que ocorra algum erro, uma determinada seqüência de comandos vai ser executada
 - Pode ser usada para restabelecer alguma variável para um valor default, por exemplo
- A comando *finally* e comando *except* são **mutuamente exclusivas**
 - Neste caso exceções não são tratadas
 - É possível combinar ambas usando comandos *try* aninhados.

Exemplo: *finally*

```
try:  
    try:  
        x = eval(input("Entre com um numero: "))  
        print (x)  
    finally:  
        print ("Fazendo x igual ao valor default None")  
        x = None  
except:  
    print ("Ocorreu um Erro!" )
```

Entre com um numero: f

Fazendo x igual ao valor default None

Ocorreu um Erro!

Repare que o *finally* e o *except* estão em níveis de identação diferentes..

- São maneiras genéricas de implementar iterações com classes
 - Usados no comando *for*
 - É geralmente mais econômico do que usar uma lista pois não é preciso armazenar todos os valores, mas apenas computar um por vez
- Um iterador é uma classe que implementa o método mágico `__iter__`
 - É um método que, por sua vez, retorna um objeto que implementa um método chamado `__next__`
 - O método next deve retornar o “próximo” valor a ser iterado
 - Se não há próximo valor, *next* deve “levantar” a exceção *StopIteration*

Exemplo: Iteradores

```
class MeuIterador:  
    a = 0  
    def __iter__(self): return self  
    def __next__(self):  
        if self.a > 10: raise StopIteration  
        self.a += 1  
        return self.a  
itt = MeuIterador()  
for i in itt:  
    print (i , end=' ')
```

1 2 3 4 5 6 7 8 9 10 11

O iterador itt vai ser incrementado a cada passo. Se usamos range a lista fica na memória. Mais Eficiente!

Geradores

- Geradores são funções especiais que retornam iteradores
- Usa o comando *yield valor*
- usada para obter o *iterador* para um comando *for*
- O *for* automaticamente iterará sobre os valores que *yield* “retorna”
- Observe que o *iterador* produzido pela função geradora é tal que o código que gera os valores e o código dentro do *for* se sucedem alternadamente

Exemplo: Geradores

```
def gerador():
    for i in range(10):
        print ("i = ", i)
        yield i
for j in gerador():
    print ("j = ",j)
```

```
i = 0
j = 0
i = 1
j = 1
....
i = 9
j = 9
```

Repare na alternância entre os comandos *for*.

Projeto 1: Objeto Fração

Escreva uma classe para manipular objetos do tipo fração.

Frações são números racionais da forma n/d , onde n e d são inteiros.

- A sua classe deve fazer a sobrecarga dos operadores, $+, + =, *, -, ==, /$, e *print*.
- O operador de igualdade deve simplificar as frações antes de compará-las (use o MDC).
- Levante uma exceção se algum operador for aplicado a um tipo inválido, ou se for construída uma fração com denominador nulo.

Solução do Projeto 1: Objeto Fração

Resolva o Exercício

E só então retorne para assistir a explicação da solução.

É importante que você faça o projeto e compare com a nossa solução.

E só então retorne para assistir a explicação solução.

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Projeto 1 - Frações

Agenda

Aulas Passadas:

- Introdução a OO e Classes
- Classes
- Exceções

Nesta Aula: Projeto envolvendo estes conceitos

Projeto 1: Objeto Fração

Escreva uma classe para manipular objetos do tipo fração.

Frações são números racionais da forma n/d , onde n e d são inteiros.

- A sua classe deve fazer a sobrecarga dos operadores, $+, + =, *, -, ==, /$, e *str*.
- O operador de igualdade deve simplificar as frações antes de compará-las (use o MDC).
- Levante uma exceção se algum operador for aplicado a um tipo inválido, ou se for construída uma fração com denominador nulo.

Projeto 1: Dicas

Documentar Bem!!!

- Nós usamos Doxygen.
- Aplicativo para documentar programas.
- Gera documentação em HTML.
- Muito fácil de usar. Diagrama de Classes...
- Baixar: www.doxygen.org/
- Colocar no mesmo diretório arquivo *Doxyfile*. Configurações.
- Executar o *Doxygen* no mesmo diretório.
- Gera um diretório *html* com a documentação.

Usamos inglês para documentar.

- Nomes de variáveis.
- Comentários.

Projeto 1: Objeto Fração - Métodos

Métodos Mágicos

- `__init__`
- `__add__`
- `__iadd__`
- `__mul__`
- `__eq__`
- `__str__`
- `__truediv__`

Dois Métodos

- simplifica
- MDC

Classe Fracao

```
## A simple class for creating fraction objects (rational numbers).
#
class Fracao:
    ## Constructor.
    #
    #  @param num numerator.
    #  @param den denominator
    #
    def __init__(self, num=1, den=1):
        """Constructor"""

        if ( den == 0 ): raise ValueError ("Zero denominator")

        ### Numerator.
        self.num = num
        ### Denominator.
        self.den = den

        if self.den < 0:          # check for a negative denominator
            self.num = -self.num  # change the sign of the numerator
            self.den = -self.den
        self.simplifica()         # simplify the fraction
```

Função *simplifica*

```
## Simplifies this fraction, by dividing either the numerator
# or the denominator by its gcd.
#
def simplifica(self):
    max = 1
    if self.num != 0:                      # assert Fracao != 0
        max = mdc(self.num, self.den)      # find the gcd
    if max > 1:                          # reduce this fraction
        self.num //= max                  # integer division
        self.den //= max
    return self
```

Função mdc

```
## mdc is a general use function, defined outside the class.  
#  
# @param x first integer: numerator.  
# @param y second integer: denominator.  
# @return GCD: Greatest Common Divisor.  
#  
def mdc(x, y):  
    """Greatest Common Divisor (Maximo divisor comum)."""  
    while y != 0:  
        resto = x % y  
        x = y  
        y = resto  
    return x
```

Exemplo: $x = 30, y = 8 \rightarrow x = 8, y = 6 \rightarrow x = 6, y = 2 \rightarrow x = 2, y = 0$

Operador ==

```
## Operator ==
#
def __eq__(self, f):
    a=self.simplifica()
    b=f.simplifica()
    return (a.num == b.num and a.den == b.den)
```

Exemplo:

$$a = \frac{18}{12} \quad b = \frac{9}{6}$$
$$\downarrow \text{simplifica} \quad \downarrow \text{simplifica}$$
$$a = \frac{3}{2} \quad b = \frac{3}{2}$$

```
return (a.num == b.num and a.den == b.den) = true
```

Operador +

Revisão: soma de fração

$$a + b$$

- Exemplo 1: b inteiro

$$a = \frac{3}{5} \text{ e } b = 7$$

$$a + b = \frac{3}{5} + 7 = \frac{3 + (5 * 7)}{5} = \frac{38}{5}$$

- Exemplo 1: b é uma fração

$$a = \frac{4}{5} \text{ e } b = \frac{2}{3}$$

$$a + b = \frac{4}{5} + \frac{2}{3} = \frac{(4 * 3) + (5 * 2)}{5 * 3} = \frac{22}{15}$$

Operador +

```
## Operator +
#
def __add__(self, f):
    if isinstance (f,int):          # check f is integer
        num = self.num + f * self.den
        den = self.den
    elif isinstance (f,Fracao):    # check f is fraction
        den = self.den * f.den
        num = self.num * f.den + self.den * f.num
    else: raise TypeError ("__add__")
    # returns a copy, and therefore is slower than "+="
    return Fracao(num, den)      # Fraction is simplified
```

- Observação: retorna uma cópia. É menos eficiente.

Operador -

```
## Operator -
#
def __sub__(self, f):
    if isinstance (f,int):          # check f is an integer
        num = self.num - f * self.den
        den = self.den
    elif isinstance (f,Fracao):    # check f is a fraction
        den = self.den * f.den
        num = self.num * f.den - self.den * f.num
    else: raise TypeError ("__sub__")
    return Fracao(num, den)
```

- Observação: retorna uma cópia. É menos eficiente.

Operador `+=`

```
## Operator +=
#
def __iadd__(self, f):
    if isinstance (f,int):          # check f is an integer
        self.num += f * self.den
    elif isinstance (f,Fracao):    # check f is a fraction
        self.num = self.num * f.den + self.den * f.num
        self.den *= f.den
    else: raise TypeError ("__iadd__")
    return self.simplifica()
```

- Observação: retorna a própria fração. É mais eficiente.

Operador *

Revisão: multiplicação de fração

$a * b$

- Exemplo:

$$a = \frac{4}{5} \text{ e } b = \frac{2}{3}$$

$$a * b = \frac{4}{5} * \frac{2}{3} = \frac{4 * 2}{5 * 3} = \frac{8}{15}$$

Operador *

```
## Operator *
#
def __mul__(self, f):
    if isinstance (f,int):          # check f is an integer
        num = self.num * f
        den = self.den
    elif isinstance (f,Fracao):    # check f is a fraction
        num = self.num * f.num
        den = self.den * f.den
    else: raise TypeError ("__mul__")
    return Fracao(num, den)
```

- Observação: retorna uma cópia. É menos eficiente.

Operador /

Revisão: divisão de fração

a/b

- Exemplo:

$$a = \frac{4}{5} \text{ e } b = \frac{2}{3}$$

$$a / b = \frac{4}{5} / \frac{2}{3} = \frac{4 * 3}{5 * 2} = \frac{12}{10} = \frac{6}{5}$$

Operador /

```
## Operator /
#
def __truediv__(self, f):
    if isinstance (f,int):          # check f is an integer
        num = self.num
        den = self.den * f
    elif isinstance (f,Fracao):    # check f is a fraction
        num = self.num * f.den
        den = self.den * f.num
    else: raise TypeError ("__truediv__")
    return Fracao(num, den)
```

- Observação: retorna uma cópia. É menos eficiente.

Operador *str*

```
## Controls how a fraction is printed.  
#  
# @return a string: numerator/denominator, or  
# only the numerator, if the denominator is 1, after sim-  
# 0, if the numerator is null.  
def __str__(self):  
    if self.num == 0:  
        return "0"  
    elif self.den == 1:  
        return str(self.num)  
    else:  
        return str(self.num)+’/+ str(self.den)
```

Main - Parte 1

```
## Main program for testing.  
#  
def main ():  
    f = Fracao(15,45)  
    g = Fracao(50,75)  
    print ("f = 15/45 = %s" % f)  
    print ("g = 50/75 = %s" % g)  
    print ("f + g = %s" % (f + g))  
    h = Fracao (10,28)  
    print ("h = 10/28 = %s" % h)  
    print ("f * h = %s" % (f * h))  
    print ("f + g + h = %s" % (f + g + h))  
    print ("f + g * h = %s" % (f + g * h))  
    print ("g - f - f = %s" % (g - f - f))  
    print ("f * 2 = %s" % (f * 2))  
    print ("f + 2 = %s" % (f + 2))  
    print ("f / g = %s" % (f / g))  
    f += g*2  
    print ("f += g*2 = %s" % f)  
    f -= g*2  
    print ("f -= g*2 = %s" % f)
```

Main - Parte 2

```
try:  
    print ("2 + f =" )  
    print (2 + f)  
except (ValueError,TypeError) as e:  
    print ("Exception caught: %s" % e)  
print ("f == h %s" % (f==h))  
print ("f=g=h")  
f=g=h  
print ("f == h %s" % (f==h))  
try:  
    print ("f += \\'a\\'")  
    f += "a"  
except (ValueError,TypeError) as e:  
    print ("Exception caught: %s" % e)
```

Main - Parte 3

```
try:  
    print ("Fracao(2,0) = ")  
    print (Fracao(2,0))  
except (ValueError,TypeError) as e:  
    print ("Exception caught: %s" % e)  
  
try:  
    print ("Fracao(5,3) / Fracao(0,4)")  
    print (Fracao(5,3)/Fracao(0,4))  
except Exception as e:  
    print ("Exception caught: %s" % e)  
  
if __name__=="__main__":  
    sys.exit(main())
```

Main - Executando

```
f = 15/45 = 1/3
g = 50/75 = 2/3
f + g = 1
h = 10/28 = 5/14
f * h = 5/42
f + g + h = 19/14
f + g * h = 4/7
g - f - f = 0
f * 2 = 2/3
f + 2 = 7/3
f / g = 1/2
f += g*2 = 5/3
f -= g*2 = 1/3
2 + f =
Exception caught: unsupported operand type(s) for +: 'int' and 'Fracao'
f == h False
f=g=h
f == h True
f += 'a'
Exception caught: __iadd__
Fracao(2,0) =
Exception caught: Zero denominator
Fracao(5,3) / Fracao(0,4)
Exception caught: Zero denominator
```

O Arquivo

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package _01a_fracao
#
# A very simple fraction class.
#
# @author Miguel Jonathan e Paulo Roma
# @since 16/09/2009
# @see http://docs.python.org/library/fractions.html
# @see http://docs.python.org/reference/datamodel.html#emulating-nu

import sys
```

Classes Fracao + Main

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Módulos

Agenda

Nesta Aula: Módulos (Python)

- Introdução: Módulos em Python
- Onde Módulos são Buscados (Path)
- Variável `__name__`
- Pacotes

Introdução: Módulos

- Módulos são programas feitos para serem reaproveitados em outros programas
- Eles tipicamente contêm funções, variáveis, classes e objetos que provêm alguma funcionalidade comum
- Por exemplo, já vimos que o módulo math contém funções matemáticas como sin, exp, etc, além da constante π
- Toda a biblioteca padrão do Python é dividida em módulos e pacotes
- Alguns dos mais comuns são: sys, os, time, random, re, shelve

Introdução: Módulos

- Qualquer programa que você escreva e salve num arquivo pode ser importado como um módulo
- Por exemplo, se você salva um programa com o nome prog.py, ele pode ser importado usando o comando *import* prog
- A a “importação” só ocorre uma vez.
- Python assume que o código do módulo serve meramente para inicializar variáveis e funções.

Introdução: Módulos

- Após a importação de um módulo, este é compilado, gerando um arquivo .pyc correspondente
- No exemplo, um arquivo prog.pyc será criado
- Python só recompila um programa se o arquivo .py for mais recente que o arquivo .pyc

Exemplo: Módulo

- Criamos um arquivo *teste_mod* contendo

```
def f():
    print ("Oi!")
f()
```

- Execute o *python*

```
$python
>>> import teste_mod
Oi!
>>> import teste_mod
>>> teste_mod.f()
Oi!
```

- Encerre o *python*

```
$dir
teste_mod.py  teste_mod.pyc  ...
```

Fazendo Módulos Disponíveis

Onde os módulos são buscados durante a importação?

- No diretório corrente
- Nos diretórios da lista `sys.path`

Mudar lugar onde os módulos residem

- Alterar diretamente a variável `sys.path` **Não Recomendado**;
- Alterar a variável de ambiente `PYTHONPATH` **Recomendado**.
- Não requer que o programa que importará o módulo seja alterado

Exemplo: Módulo - Path

- Criamos um diretório *meu_dir*;
- Movemos o programa *teste_mod* para lá;

```
$ mkdir meu_dir  
$ mv teste_mod.py meu_dir/  
$ more meu_dir/teste_mod.py  
def f():  
    print ("Oi!")  
f()
```

```
$ export PYTHONPATH=~/meu_dir
```

- Execute o *python*

```
$python  
>>> import teste_mod  
Oi!
```

Variável __name__

Programa pode ser executado por si só ou importado dentro de outro, como distinguir as duas situações?

- A variável __name__ é definida para cada programa:
- Se é um módulo, retorna o nome do módulo
- Se é um programa sendo executado, retorna '__main__'
- Saber se o código está sendo executado como módulo, testar:
- If __name__ == '__main__': código
- Isto é útil em diversas circunstâncias
- Por exemplo, para colocar código de teste, código para instalação do módulo ou exemplos de utilização

Exemplo: Variável `__name__`

- Criamos um arquivo `teste_mod` contendo

```
def f():
    print ("Oi!")
if __name__ == '__main__':
    f()
```

- Execute o `python`

```
$ python teste_mod
Oi!
>>> print (__name__)
__main__
$python
....
>>> import teste_mod
>>> print (teste_mod.__name__)
teste_mod
```

São hierarquias de módulos

- É um diretório que contém um arquivo chamado `__init__.py`
- O pacote deve estar em um dos diretórios nos quais o Python busca por módulos
- Para importar o pacote, use o nome do diretório
 - O programa correspondente ao pacote é `__init__.py`

São hierarquias de módulos

- Os demais arquivos e diretórios dentro do pacote são encarados recursivamente como módulos

`__init__.py`

- Por exemplo, se um pacote se chama p e contém um arquivo chamado m.py, então podemos importar
 - p (o arquivo p/`__init__.py`)
 - p.m (o arquivo p/m.py)
- Semelhantemente, p poderia ter um outro pacote sob a forma de outro diretório contendo um arquivo `__init__.py`

Exemplo: Pacote

- Criamos um diretório *meu_dir*;

```
$ export PYTHONPATH=~/meu_dir
```

- Criamos um diretório dentro do *meu_dir* chamado *meu_pacote*;

- Programa *teste1* com: `print ("teste 1")`

- Prog. *teste2* dentro do diretório *meu_pacote*: `print ("teste 2")`

- Programa *__int__.py* dentro do diretório *meu_pacote*:

```
print ("pacote")
$dir meu_dir
meu_pacote teste1
$dir meu_dir/meu_pacote
__int__.py teste2
```

- Execute o *python*

```
>>> import teste1
teste 1
>>> import meu_pacote
pacote
>>> import meu_pacote.teste2
teste 2
```

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Arquivos
Entrada e Saída

Agenda

Nesta Aula: Arquivos (Python)

- Introdução: Arquivos
- Arquivos Default
- Abrindo arquivos
- O Objeto file
- Métodos Read, Write e Close
- Lendo e escrevendo linhas

Introdução: Arquivos

Arquivos

- Entrada e saída são operações de comunicação de um programa com o mundo externo
- Essa comunicação se dá usualmente através de arquivos
- Arquivos estão associados a dispositivos
 - Por exemplo, disco, impressora, teclado
- Em Python, um arquivo pode ser lido/escrito através de um objeto da classe *file*

Arquivos Default

Já usamos, sem saber, três arquivos default

- Sempre que um comando *print* é executado, o resultado vai para um arquivo chamado *sys.stdout*
- Sempre que lemos um dado através do comando *input* ou *raw_input*, na verdade estamos lendo de um arquivo chamado *sys.stdin*
- Mensagens de erro ou de rastreamento de exceções são enviadas para um arquivo chamado *sys.stderr*

Exemplo: Arquivo

```
$python
>>> import sys
>>> sys.stdout.write("Oi!")
Oi!
>>> print ("Oi!")
Oi!
>>> sys.stdin.readline()
fluminense
'fluminense\n'
>>> raw_input()
fluminense
'fluminense'
```

Abrindo arquivos

open (name, mode, buffering)

- name : nome do arquivo a abrir
- mode : (opcional) modo de abertura – string contendo
 - r : leitura (default)
 - w : escrita
 - b : binário
 - a : escrita a partir do final
 - + : (usado com r) indica leitura e escrita
- buffering : (opcional) indica se memória (buffers) é usada para acelerar operações de entrada e saída
 - 0 : buffers não são usados
 - 1 (ou número negativo): um buffer de tamanho padrão (default)
 - 2 ou maior: tamanho do buffer em bytes

O Objeto *file*

- O comando *open* retorna um objeto do tipo *file* (arquivo)
- O objeto retornado é usado para realizar operações de entrada e saída:

```
>>> arq = open ("teste", "w")
>>> arq.write("Oi!")
>>> arq.close()
>>> arq = open ("teste")
>>> x = arq.read()
>>> x
"Oi!"
```

Métodos Read, Write e Close

read(num)

- Lê num bytes do arquivo e os retorna numa string;
- Se num não é especificado, todos os bytes desde o ponto atual até o fim do arquivo são retornados.

write(string)

- Escreve string no arquivo;
- Devido ao uso de buffers, a escrita pode não ser feita imediatamente;
- Use o método flush() ou close() para assegurar a escrita física.

close()

- Termina o uso do arquivo para operações de leitura e escrita.

Fim de Linha

- Arquivos de texto são divididos em linhas usando caracteres especiais
 - Linux/Unix: \n
 - Windows: \r\n
 - Mac: \r
- Python usa sempre \n para separar linhas
 - Ao se ler/escrever um arquivo aberto em modo texto (não binário) faz traduções de \n para se adequar ao sistema operacional;
 - Em modo binário, entretanto, a conversão não é feita

Interação com o Sistema Operacional

- Operações de entrada e saída são na verdade realizadas pelo sistema operacional
- O módulo *OS* possui diversas variáveis e funções que ajudam um programa Python a se adequar ao sistema operacional, por exemplo:
 - *os.getcwd()* retorna o diretório corrente;
 - *os.chdir(dir)* muda o diretório corrente para *dir*;
 - *os.path.exists(path)* diz se *path* se refere ao nome de um arquivo existente.

Lendo e escrevendo linhas

readline(n)

- Se n não é especificado, retorna exatamente uma linha lida do arquivo;
- Caso contrário, lê uma linha, mas busca no máximo n caracteres pelo final de linha.

readlines(n)

- Se n não é especificado, retorna o restante do conteúdo do arquivo em uma lista de strings ;
- Caso n seja especificado, a leitura é limitada a n caracteres no máximo.

writelines(sequência)

- Escreve a lista (ou qualquer sequência) de strings, uma por uma no arquivo;

Acesso direto

- É possível ler e escrever não seqüencialmente em alguns tipos de arquivo;
 - Devem estar associados a dispositivos que permitem acesso direto, como discos, por exemplo.
- `seek(offset, whence)`
 - offset indica o número do byte a ser lido e escrito pela próxima operação de entrada e saída;
 - whence indica a partir de onde offset será contado
 - 0 (default) : do início
 - 1 : do ponto corrente
 - 2 : do final
- `tell()`
 - Indica a posição corrente (número de bytes a partir do início do arquivo)

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Projeto 2 - Frações com Arquivos

Agenda

Aulas Passadas:

- Introdução a OO e Classes
- Classes
- Exceções
- Módulos
- Arquivos

Nesta Aula: Projeto envolvendo estes conceitos

Projeto 2: Fração com Arquivos

Escreva um programa para ler uma série de frações de um arquivo e imprimir a sua soma e produto.

Por exemplo:

Arquivo de Entrada: **Saída do programa:**

2 3	Fracao 0: 2/3
5 7	Fracao 1: 5/7
1 4	Fracao 2: 1/4
8 3	Fracao 3: 8/3
2 4	Fracao 4: 1/2
1 7	Fracao 5: 1/7
3 8	Fracao 6: 3/8

Soma: 893/168

Produto: 5/588

Projeto 1: Objeto Fracos - Métodos

Métodos Mágicos

- `__init__`
- `__str__`
- `__repr__`

Método

- Reader

O Arquivo

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package _01e_fracoes
#
# Reads a file with a series of fractions, and prints their sum
# and product.
#
# @author Paulo Roma
# @since 25/09/2014

import sys

from _01a_fracao import Fracao
```

Classe Fracos + Main

Classe Fracoes

```
# Process fractions on a given file.
class Fracoes:
    ##
    # Constructor.
    # Opens filename and calls Reader for inputting the fraction readings.
    # Raises an exception if filename does not exist.
    #
    # @param filename fraction file name.
    #
    def __init__(self, filename):
        """ lfracoes - a list of objects of type Fracao.
        self.lfracoes = []

try:
    f = open(filename, 'r')
except IOError:
    print ('Fracoes: Cannot open file %s for reading' % filename)
    raise
self.Reader(f)
```

O Método *Reader*

```
##  
# Reads a file with a numerator and denominator per line.  
# Creates a Fracao object for each line and inserts it in lfracoes.  
#  
# @param f fraction file object.  
#  
def Reader (self, f):  
    for line in f:  
        temp = line.split(None)  
        if len(temp) == 2:  
            try:  
                self.lfracoes.append(Fracao(int(temp[0]),int(temp[1])))  
            except:  
                print ('Fração Inválida: %s\n' % temp)  
                continue  
    f.close()
```

O Método `__str__`

```
##  
#     Returns the sum and product of all entries of "lfracoes".  
#  
#     @return a string: sum and product of all fractions.  
#  
def __str__(self):  
    sb = ""  
    f = Fracao(0,1)  
    g = Fracao(1,1)  
    for i in range(0, len(self.lfracoes)):  
        f += self.lfracoes[i]  
        g *= self.lfracoes[i]  
    sb += "Soma: %s\nProduto: %s\n" % (f,g)  
  
    return sb
```

O Método `__repr__`

```
##  
#     Returns each fraction in list "lfracoes".  
#  
#     @return a string: a series of fractions, one per line.  
#  
def __repr__(self):  
    sb = ""  
    for i in range(0, len(self.lfracoes)):  
        sb += "Fracao %d: %s\n" % (i, self.lfracoes[i])  
  
    return sb
```

Main

```
#     Reads a series of pairs  and prints the sum and product of all fractions.
#
def main(argv=None):
    f = "fracoes.txt"
    if argv is None:
        argv = sys.argv

    if ( len(argv) > 1 ):
        f = argv[1]

    try:
        m = Fracoes(f)
        print (repr(m))
        print (m)
    except IOError:
        sys.exit ( "File %s not found." % f )

if __name__=="__main__":
    sys.exit(main())
```

Main - Executando

```
$ more fracoes.txt  
5 7  
8 10  
12 23  
2 9  
4 8  
10 33  
11 99
```

```
$python fracoes.py  
Fracao 0: 5/7  
Fracao 1: 4/5  
Fracao 2: 12/23  
Fracao 3: 2/9  
Fracao 4: 1/2  
Fracao 5: 10/33  
Fracao 6: 1/9
```

```
Soma: 56183/17710  
Produto: 160/143451
```

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Projeto 3 - Impressora

Agenda

Aulas Passadas:

- Introdução a OO e Classes
- Classes
- Exceções
- Módulos
- Arquivos

Nesta Aula: Projeto envolvendo estes conceitos

Projeto 3: Controle de Tinta e Papel numa Impressora

Escreva um programa para implementar uma classe *impressora* para controlar a quantidade de tinta e papel na impressora.

- Papel pode ser adicionado à impressora a qualquer momento, e assume-se que não há nenhuma capacidade máxima para papel.
- Uma impressora recém construída tem um cartucho de tinta completo contendo a quantidade de tinta dado pela constante INK_CAPACITY.
- A impressora pode imprimir em um lado ou em frente e verso.
- Para cada lado impresso, uma pequena quantidade da tinta é usada, como determinado pela constante INK_USAGE.
- O cartucho de tinta pode ser substituído em qualquer altura, restaurando a quantidade da tinta para o valor INK_CAPACITY.

Projeto 1: Objeto Printer - Métodos

Métodos

- `__init__`
- `addPaper`
- `getCurrentPaper`
- `getTotPaperUsed`
- `isInkOut`
- `replacelnk`
- `printOneSided`
- `printTwoSided`
- `__str__`

Classe Printer

```
## Models the usage of paper and ink in a printer.  
class printer:  
##  
# Capacity, in ounces, of a new ink cartridge.  
#  
INK_CAPACITY = 2.0 # 56.699 gramas  
  
##  
# Amount of ink, in ounces, used per printed page.  
#  
INK_USAGE = 0.0023
```

O Método `__init__`

```
## Printer initially contains a given number of paper sheets
# and a full ink cartridge.
#
# @param givenNumberOfSheets initial number of paper sheets.

def __init__(self,givenNumberOfSheets):
    ## number of sheets available
    self.__numberOfSheets = abs(givenNumberOfSheets)

    ## total paper used since construction
    self.__totalPaperUsed = 0

    ## quantity of ink available
    self.__inkQuantity = printer.INK_CAPACITY
```

Método *addPaper*

```
## Adds the given number of sheets of paper to this printer.  
# We assume that there is no maximum capacity.  
#  
# @param additionalSheets number of sheets to be added  
# to the printer.  
  
def addPaper(self,additionalSheets):  
    if additionalSheets > 0:  
        self.__numberOfSheets += additionalSheets
```

Os Métodos *getCurrentPaper* e *getTotPaperUsed*

```
## Returns the number of sheets of paper currently in this printer.  
#  
# @return number of sheets available.  
  
def getCurrentPaper(self):  
    return self._numberOfSheets  
  
## Returns the total number of sheets of paper printed by this printer.  
# Sheets used for two sided printing still count as just one sheet.  
#  
# @return number of sheets of paper used since construction.  
  
def getTotPaperUsed(self):  
    return self._totalPaperUsed
```

Os Métodos *isInkOut* e *replaceInk*

```
## Check if the ink has run out. Returns true if the amount
# of ink left is smaller than the quantity INK_USAGE.
#
# @return True if the ink is over, and False otherwise.

def isInkOut(self):
    return self.__inkQuantity < printer.INK_USAGE

## Simulates replacement of the ink cartridge, restoring the
# quantity of ink in the printer to INK_CAPACITY.

def replaceInk(self):
    self.__inkQuantity = printer.INK_CAPACITY
```

O Método *printOneSided*

Imprimir um Lado da Folha

- Usando a quantidade apropriada de **folhas** e de **tinta**
- Se quantidade de papel é insuficiente ela imprimi até acabar o papel
 - usando a quantidade de tinta necessária;
- Se quantidade de tinta é insuficiente, ela usa a tinta até acabar
 - e imprimi folhas em branco até o final.

O Método *printOneSided*

```
## Simulates printing pages in one-sided mode, using the appropriate number of sheets and a
# corresponding quantity of ink. If there is not enough paper, the printer will use up all
# remaining paper and will only use the quantity of ink needed for the sheets actually
# printed. If there is not enough ink, the printer will use up all the ink, and will still
# use up the specified number of sheets of paper
# (i.e., it just prints a bunch of blank pages after the ink runs out).
#
#
# @param numberOfPages number of sheets of paper to be printed
#
def printOneSided(self,numberOfPages):
    np = max(numberOfPages,0)
    np = min(np, self.__numberOfSheets)

    self.__totalPaperUsed += np
    self.__inkQuantity -= printer.INK_USAGE*np
    self.__inkQuantity = max(self.__inkQuantity,0)
    self.__numberOfSheets -= np
```

O Método *printTwoSided*

Imprimir Dois Lado da Folha

- Similar ao *printOneSided*;
- Usando a quantidade apropriada de **folhas** e de **tinta**
- Se quantidade de papel é insuficiente ela imprimi até acabar o papel
 - usando a quantidade de tinta necessária;
- Se quantidade de tinta é insuficiente, ela usa a tinta até acabar
 - e imprimi folhas em branco até o final.
- Precisamos determinar quantos folhas serão necessárias:
 - exemplo: 4 páginas → 2 folhas e 5 páginas → 3 folhas.
- Precisamos determinar quantidade de tinta necessária:
 - **Ou** é o número de páginas requisitadas;
 - **Ou** (talvez duas vezes) o número de folhas disponíveis.
 - **Escolhemos o menor.**

O Método *printTwoSided*

```
## Simulates printing pages in two-sided mode, using the appropriate... #
# This is similar to printOneSided() method, but you first need to determine
# how many sheets of paper are needed. For 1 or 2 pages, you need 1 sheet;
# for 3 or 4 pages, you need 2 sheets; and so on. You can use integer division
# (and/or the modulus operator) for this. Then, you have to figure out how many sheets
# of paper will actually be used (as in printOneSided()). Finally, to calculate the ink
# needed, you need to know how many pages will really be printed: this must be either
# the original number os pages requested, or (maybe twice) the number of sheets of paper
# available in the printer, whichever is smaller.

# @param numberOfPages num. sheets printed in double side.

def printTwoSided(self,numberOfPages):
    np = max(numberOfPages,0)
    nf = min(np//2+np%2, self.__numberOfSheets)
    # rnp is real number pages printed
    rnp = min(numberOfPages, 2*self.__numberOfSheets)
    self.__totalPaperUsed += nf
    self.__inkQuantity -= printer.INK_USAGE*rnp
    self.__inkQuantity = max(self.__inkQuantity,0)
    self.__numberOfSheets -= nf
```

O Método `__str__`

```
## Print printer statistics.  
#  
def __str__(self):  
    return "Total paper = %d\nInk quantity = %f\nNum.Sheets = %d\n%" \  
(self.getTotPaperUsed(), self.__inkQuantity, self.getCurrentPaper())
```

Main - Parte 1

```
## Main program for testing.  
def main():  
    print ("Constructed(50)")  
    prt = printer(50)  
    print (prt)  
  
    print ("printTwoSided(3)")  
    prt.printTwoSided(3)  
    print (prt)  
  
    print ("printOneSided(2)")  
    prt.printOneSided(2)  
    print (prt)  
  
    print ("printOneSided(60)")  
    prt.printOneSided(60)  
    print (prt)  
  
    print ("addPaper(2000)")  
    prt.addPaper(2000)  
    print (prt)
```

Main - Parte 2

```
print ("Sheets used = %d" % prt.getTotPaperUsed())
print ("Out of ink = %s" % prt.isInkOut())
print ("Sheets available = %d\n" % prt.getCurrentPaper())

print ("printOneSided(870)")
prt.printOneSided(870)
print (prt)

print ("Out of ink = %s\n" % prt.isInkOut())

prt.replaceInk()
print ("replaceInk()")
print (prt)

print ("printTwoSided(101)")
prt101 = printer(50)
prt101.printTwoSided(101)
print (prt101)

if __name__ == "__main__":
    sys.exit(main())
```

O Arquivo

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package printer
#
# The Printer class models the usage of paper and ink in a printer. Paper can be added to the
# printer at any time, and we assume that there is no maximum capacity for paper. A newly
# constructed printer has a full ink cartridge containing the quantity of ink given by constant
# INK_CAPACITY. The printer can print one-sided or two-sided. For each side printed, a small
# quantity of ink is used, as given by constant INK_USAGE. The ink cartridge can be replaced
# at any time, restoring the ink quantity to the value INK_CAPACITY.
#
# Please note that you do not need any conditional statements (which we start next week) to
# complete this assignment. There will be a few places where you need to choose the smaller of
# two numbers, which can be done with the method min().
#
# @author Paulo Roma
# @since 20/06/2016
# @see http://radek.io/2011/07/21/private-protected-and-public-in-python/

import sys
```

Classe Printer + Main

Main - Executando 1

```
$python printer.py
Constructed(50)
Total paper = 0
Ink quantity = 2.000000
Num. Sheets = 50

printTwoSided(3)
Total paper = 2
Ink quantity = 1.993100
Num. Sheets = 48

printOneSided(2)
Total paper = 4
Ink quantity = 1.988500
Num. Sheets = 46

printOneSided(60)
Total paper = 50
Ink quantity = 1.882700
Num. Sheets = 0
```

Main - Executando 2

```
addPaper(2000)
Total paper = 50
Ink quantity = 1.882700
Num. Sheets = 2000

Sheets used = 50
Out of ink = False
Sheets available = 2000

printOneSided(870)
Total paper = 920
Ink quantity = 0.000000
Num. Sheets = 1130

Out of ink = True

replaceInk()
Total paper = 920
Ink quantity = 2.000000
Num. Sheets = 1130

printTwoSided(101)
Total paper = 50
Ink quantity = 1.770000
Num. Sheets = 0
```

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Projeto 4 - Calcular Consumo de um Carro

Agenda

Aulas Passadas:

- Introdução a OO e Classes
- Classes
- Exceções
- Módulos
- Arquivos

Nesta Aula: Projeto envolvendo estes conceitos

Projeto 4: Calcular Consumo de um Carro

Escreva um programa para implementar uma classe para controlar o consumo de um carro a partir de uma arquivo contendo um série de medidas de distância e quantidade de combustível consumida.

- Construa uma classe para ler uma série de leituras de hodômetro e galões consumidos após encher o tanque de gasolina de um veículo, e calcular o consumo entre cada paragem no posto de gasolina.

Duas Classes

- MileageCalculator

Lê, calcula e imprime consumo de gasolina em km/litro e milhas/galões

- FillUp

Manipula objetos do tipo FillUp, que são leituras do odômetro e de galões consumidos

Classe FillUp

```
class FillUp:  
    ##  
    # Odometer reading when the tank was filled.  
    #  
    odometer = 0.0  
  
    ##  
    # Gallons needed to fill the tank.  
    #  
    gallons = 0.0
```

Classe FillUp

- *__init__*
- *getOdometer* - retorna a leitura do odômetro
- *getGallons* - retorna a leitura de galões consumidos

Método `__init__` da Classe FillUp

```
##  
# Constructs a new FillUp object with the given data.  
# @param givenOdometer  
#   odometer reading  
# @param givenGallons  
#   number of gallons  
#  
def __init__(self, givenOdometer, givenGallons):  
    self.odometer = givenOdometer  
    self.gallons = givenGallons
```

Métodos *getOdometer* e *getGallons* da Classe FillUp

- Método *getOdometer*

```
##  
# Returns the odometer reading.  
# @return  
#     the odometer reading  
#  
def getOdometer(self):  
    return self.odometer
```

- Método *getGallons*

```
##  
# Returns the number of gallons.  
# @return  
#     number of gallons  
#  
def getGallons(self):  
    return self.gallons
```

Classe MileageCalculator

```
## Class for controlling the gasoline consumption of a car.  
#  
class MileageCalculator:  
  
    ### fillup list, which aggregates two values.  
    fillup = []  
  
    ### debugging state.  
    debug = False
```

Métodos da Classe MileageCalculator

- *__init__*
- *Reader* - lê no arquivo leituras de distância e gasolina
- *consumption* - calcula o consumo
- *__repr__*
- *__str__*

Método `__init__` da Classe MileageCalculator

```
##  
# Constructor.  
# Opens filename and calls Reader for inputting the mileage re  
# Raises an exception if filename does not exist.  
#  
# @param filename mileage file name.  
#  
def __init__(self, filename):  
    try:  
        f = open(filename, 'r')  
    except IOError:  
        print ('Cannot open file %s for reading' % filename)  
        raise  
    self.Reader(f)
```

Método *Reader* da Classe MileageCalculator

```
##  
#   Reads a file with mileage and gasoline per line.  
#   Creates FillUp object for each line and inserts in the fillup list.  
#  
#   @param f mileage file object.  
#  
def Reader (self, f):  
    for line in f:  
        tempwords = line.split(None)  
        if len(tempwords) == 2:  
            try:  
                self.fillup.append(FillUp(float(tempwords[0]),float(tempwords[1])))  
            except:  
                print ('Invalid reading: %s\n' % tempwords)  
  
f.close()
```

Método *consumption* da Classe MileageCalculator

```
##  
# Calculates the consumption of the k-th entry of fillup list.  
#  
# @param k fillup index.  
# @return (odometer[k]-odometer[k-1])/gallons[k] .  
#  
def consumption ( self, k ):  
    if k < 1 or k >= len(self.fillup): return None  
  
    previous = self.fillup[k-1].getOdometer()  
    current = self.fillup[k].getOdometer()  
    gallons = self.fillup[k].getGallons()  
  
    if MileageCalculator.debug:  
        print("current %f\nprevious%f\ngallons %f\n"%(current,previous,gallons))  
  
    return (current-previous)/gallons
```

Método `__repr__` da Classe MileageCalculator

```
##  
# Returns the consumption (in mi/gal) corresponding to each  
# entry of "fillup", by calling the method "consumption".  
#  
# @return a string: a series of consumptions.  
#  
def __repr__(self):  
    sb = "Miles per gallon\n"  
    for i in range(1, len(self.fillup)):  
        sb += "Consump. %d: %.3f\n" % (i, self.consumption(i))  
  
    return sb
```

Método `__str__` da Classe MileageCalculator

```
##  
# Returns the consumption (in km/lt) corresponding to each entry  
# of "fillup", by calling the method "consumption".  
#  
# 1 gallon = 3.7854118 litres  
#  
# 1 mile    = 1.609344  kilometers  
#  
# @return a string: a series of consumptions.  
#  
def __str__(self):  
    sb = "Kilometers per litre\n"  
    for i in range(1, len(self.fillup)):  
        sb += "Consump.%d: %.3f\n"%(i,self.consumption(i)*1.609344/3.7854118)  
  
    return sb
```

Main - Parte 1

```
##  
# Main method. Reads a series of pairs of mileage and number  
# the average consumption: (current-previous)/gallons.  
#  
def main(argv=None):  
    f = "mileage.txt"  
    d = False  
    if argv is None:  
        argv = sys.argv  
  
    if ( len(argv) > 2 ):  
        f = argv[1]  
        d = argv[2]=='True'
```

Main - Parte 2

```
try:  
    m = MileageCalculator(f)  
    MileageCalculator.debug = d  
    print (m)  
    print (repr(m))  
except IOError:  
    sys.exit ( "File %s not found." % f )  
  
if __name__=="__main__":  
    sys.exit(main())
```

O Arquivo

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package MileageCalculator
#
# Class for reading a series of odometer and gallons data
# from filling up the gas tank of a vehicle, and calculating
# the consumption between each gas station stop.
#
# @author Paulo Roma
# @date 24/08/2014
#
import sys
```

Classes FillUp + MileageCalculator + Main

Executando

```
$ more mileage.txt  
91183 12.878  
91538 11.007  
91884 10.351  
92164 9.644  
92400 8.125  
92812 12.629  
123abc xyz
```

```
$ python MileageCalculator.py  
Invalid reading: ['123abc', 'xyz']
```

```
Kilometers per litre  
Consump. 1: 13.712  
Consump. 2: 14.211  
Consump. 3: 12.343  
Consump. 4: 12.349  
Consump. 5: 13.870  
Consump. 6: 13.407  
Consump. 7: 12.115
```

```
Miles per gallon  
Consump. 1: 32.252  
Consump. 2: 33.427  
Consump. 3: 29.034  
Consump. 4: 29.046  
Consump. 5: 32.623  
Consump. 6: 31.534  
Consump. 7: 28.497
```

Programação com Interfaces Gráfica

Mario Benevides e Paulo Roma

Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil

Projeto 5 - Ordenar Linhas num Arquivo

Agenda

Aulas Passadas:

- Introdução a OO e Classes
- Classes
- Exceções
- Módulos
- Arquivos

Nesta Aula: Projeto envolvendo estes conceitos

Projeto 5 - Ordenar Linhas num Arquivo

Faça programa para ordenar linhas num arquivo e imprimi-las ordenadas.

- Ordene as linhas fornecidas num arquivo de entrada e dê como saída as linhas ordenadas no stdout.
- Por Exemplo (linha de comando)

```
sort_file.py unsorted.txt > sorted.txt
```

Arquivo de Entrada: **Saída do programa:**

Fluminense	Av. Irene Lopes
Rua Linda 23	Babilonia
Av. Irene Lopes	Carla Silva
Carla Silva	Escola Nova
O grilo Cantor	Fluminense
Babilonia	O grilo Cantor

O Arquivo

Módulo `fileinput`

- Implementa uma classe e funções para ajudar e iterar sobre arquivos de entrada;
- principalmente múltiplos arquivos de entrada;
- veja <http://docs.python.org/2/library/fileinput.html>

```
#!/usr/bin/env python
# coding: UTF-8
#
## @package sort_file
#
# Sort the lines of a file.
#
# @see http://docs.python.org/2/library/fileinput.html

import fileinput, sys
```

Método *sort_file*

```
## Sort the lines in the file named on standard input,
# outputting the sorted lines on stdout.
#
# Example call (from command line)
# sort_file.py unsorted.txt > sorted.txt
#
def sort_file():
    lines=[] # list of file lines
    for line in fileinput.input():
        lines.append(line.rstrip())
    lines.sort()
    for line in lines:
        print (line)

if __name__ == "__main__":
    sys.exit(sort_file())
```

Executando

```
$ python sort_file.py test_in.txt > test_out.txt
$ more test_in.txt
Fluminense
Rua Linda 23
Av. Irene Lopes
Carla Silva
O grilo Cantor
Babilonia
$ more test_out.txt
Av. Irene Lopes
Babilonia
Carla Silva
Escola Nova
Fluminense
O grilo Cantor
```

Python: Interfaces Gráficas com Tk

UFRJ

Interfaces Gráficas

- Também chamadas de Graphical User Interfaces (GUI)
- Usadas em aplicações modernas que requerem uma interação constante com o usuário
 - Maior usabilidade e naturalidade do que interfaces textuais
- Aplicação apresenta uma ou mais janelas com elementos gráficos que servem para comandar ações, especificar parâmetros, desenhar e exibir gráficos, etc
- Bibliotecas (*toolkits*) para construção de interfaces como
 - Qt
 - Gtk
 - wxWindows
 - Tk

Interfaces Gráficas em Python

- Python possui camadas de portabilidade (*bindings*) para várias bibliotecas de construção de interfaces. Ex.:
 - PyQt (Qt)
 - PyGtk (Gtk)
 - wxPython (wxWindows)
 - Tkinter (Tk)
- Multiplataforma (MS-Windows, Unix/Linux, OSX)

Tk

- Toolkit originalmente criado para utilização com a linguagem script Tcl
- Bastante leve, portátil e robusto
- Um tanto obsoleto frente a outros toolkits mais modernos como Qt ou Gtk
- Camada Tkinter normalmente distribuída com o Python
 - Inicia um processo Tcl que toma conta dos elementos de interface
 - Classes e funções do Tkinter se comunicam com o interpretador Tcl para especificar aspecto e comportamento da interface

Usando Tkinter

- Importar o módulo Tkinter
 - from Tkinter import *

- Elementos de interface (*widgets*) correspondem a objetos de diversas classes. Por exemplo:

- Frame (Área retangular)
- Button (botão)
- Label (rótulo)
- Text (caixa de texto)
- Canvas (caixa de desenho)

- Posição e tamanho dos elementos controlados por gerentes de geometria

- Pack (mais comum), Place, Grid

Usando Tkinter (2)

- Para criar um widget, tem-se que informar o widget-pai (parâmetro *master*) onde geometricamente deverá ser encaixado e as opções de configuração para o widget. Ex.:
`w=Button(pai,text="Cancelar",command=cancelar)`
- Tk já define por default uma janela principal
 - `master=None` (default) indica que o widget será filho da janela principal
 - Outras janelas podem ser criadas instanciando-se objetos da classe `Toplevel`
- A função `mainloop` tem que ser invocada para que a aplicação entre no modo de tratamento de eventos

Exemplo

```
from Tkinter import *

class Application(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.msg = Label(self, text="Hello World")
        self.msg.pack()
        self.bye = Button (self, text="Bye", command=self.quit)
        self.bye.pack()
        self.pack()

app = Application()
mainloop()
```

Exemplo

```
from Tkinter import *\n\n\nclass Application(Frame):\n    def __init__(self, master=None):\n        Frame.__init__(self, master)\n        self.msg = Label(self, text="Hello World")\n        self.msg.pack ()\n        self.bye = Button (self, text="Bye", command=self.quit)\n        self.bye.pack ()\n        self.pack()\n\napp = Application()\nmainloop()
```



Exemplo

```
from Tkinter import *
```

Elemento principal
derivado de Frame

```
class Application(Frame):
```

```
    def __init__(self, master=None):
```

Construtor da classe base

```
        Frame.__init__(self, master)
```

```
        self.msg = Label(self, text="Hello World")
```

```
        self.msg.pack()
```

```
        self.bye = Button (self, text="Bye", command=self.quit)
```

```
        self.bye.pack()
```

```
        self.pack()
```

Janela tem um
rótulo e um botão

Interface é
instanciada

```
app = Application()
```

```
mainloop()
```

Laço de tratamento de
eventos é iniciado

Classes de componentes

- Button Um botão simples usado para executar um comando
- Canvas Provê facilidades de gráficos estruturados
- Checkbutton Representa uma variável que pode ter dois valores distintos (tipicamente um valor booleano). Clicando no botão alterna-se entre os valores
- Entry Um campo para entrada de uma linha de texto
- Frame Usado como agrupador de widgets
- Label Mostra um texto ou uma imagem
- Listbox Mostra uma lista de alternativas. Pode ser configurado para ter comportamento de checkbutton ou radiobutton

Classes de componentes (cont.)

- Menu Um painel de menu. Implementa menus de janela, pulldowns e popups
- Message Similar ao widget Label, mas tem mais facilidade para mostrar texto quebrado em linhas
- Radiobutton Representa um possível valor de uma variável que tem um de muitos valores. Clicando o botão, a variável assume aquele valor
- Scale Permite especificar um valor numérico através de um ponteiro em uma escala linear
- Scrollbar Barra de rolamento para widgets que têm superfície útil variável (Text, Canvas, Entry, Listbox)
- Text Exibe e permite editar texto formatado. Também suporta imagens e janelas embutidas
- Toplevel Uma janela separada

A Classe Tk

- É a que define uma janela principal e o interpretador Tcl
- Em geral, nunca precisa ser instanciada
 - É instanciada automaticamente quando um widget filho é criado
- Pode ser instanciada explicitamente
- Possui vários métodos, entre os quais
 - `title(string)` Especifica o título da janela
 - `geometry(string)` Especifica tamanho e posição da janela
 - String tem a forma *largura*x*altura*+*x*+*y*

Exemplo

```
from Tkinter import *

class Application(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.msg = Label(self, text="Hello World")
        self.msg.pack ()
        self.bye = Button (self, text="Bye", command=self.quit)
        self.bye.pack ()
        self.pack()

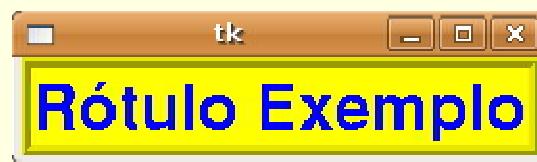
app = Application()
app.master.title("Exemplo")
app.master.geometry("200x200+100+100")
mainloop()
```

Opções de *Widgets*

- *Widgets* (elementos de interface) têm opções com nomenclatura unificada. Ex.:
 - text Texto mostrado no elemento
 - background cor de fundo
 - foreground cor do texto
 - font fonte do texto
 - relief relevo da borda ('flat', 'raised', 'ridge', 'sunken', 'groove')
- Opções são especificadas
 - No construtor
 - Através do método configure

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
rotulo = Label (top, text="Rótulo Exemplo", foreground="blue")
rotulo.pack ()
rotulo.configure(relief="ridge", font="Arial 24 bold", border=5,
    background="yellow")
mainloop()
```



O método configure

- Usado com pares do tipo *opção=valor*, modifica os valores dos atributos
- Usado com uma string “*nomeopção*” retorna a configuração da opção com esse nome
 - A configuração é uma tupla com 5 valores
 - nome do atributo
 - nome do atributo no banco de dados (X11)
 - nome da classe no banco de dados (X11)
 - objeto que representa a opção
 - valor corrente da opção
- Se configure é usado sem argumentos, retorna um dicionário com todas as opções
- Pode-se obter diretamente o valor de uma opção usando o método cget

Exemplo

```
>>> rotulo.configure(relief="ridge")
>>> rotulo.configure("relief")
('relief', 'relief', 'Relief', <index object at 0x85f9530>, 'ridge')
>>> rotulo.configure()["relief"]
('relief', 'relief', 'Relief', <index object at 0x85f9530>, 'ridge')
>>> rotulo.configure("relief")[4]
'ridge'
>>> rotulo.cget("relief")
'ridge'
```

Gerenciando geometrias

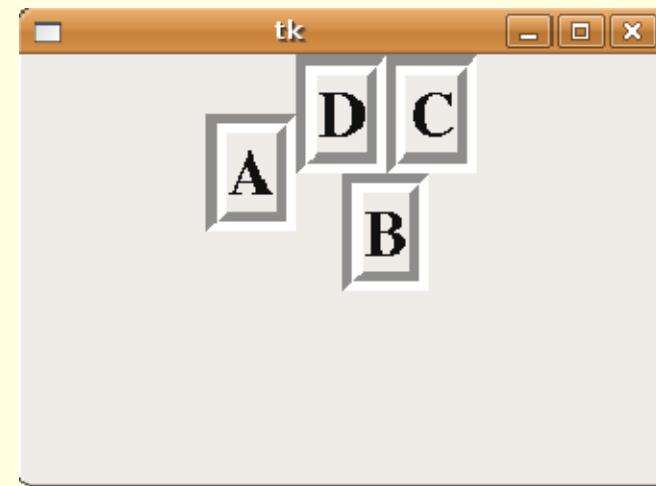
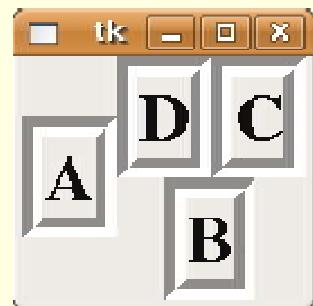
- Todos os elementos de interface ocupam uma área retangular na janela
- A posição e tamanho de cada elemento é determinada por um gerenciador de geometria
 - O elemento não “aparece” enquanto não for informado ao gerenciador
- A geometria resultante depende de
 - Propriedades dos elementos (tamanho mínimo, tamanho da moldura, etc)
 - Opções do gerenciador
 - Algoritmo usado pelo gerenciador
- O gerenciador mais usado em Tk é o **pack**

Usando o pack

- Para informar que um elemento deve ser gerenciado pelo pack, use o método pack (opções)
- O pack considera o espaço do elemento “pai” como uma cavidade a ser preenchida pelos elementos filhos
- O algoritmo usado pelo pack consiste em empacotar os filhos de um elemento “pai” segundo o lado (side) especificado
 - Os lados possíveis são 'top', 'left', 'right' e 'bottom'
 - Deve-se imaginar que sempre que um elemento filho escolhe um lado, a cavidade disponível fica restrita ao lado oposto

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left")
b = Label (top, text="B") ; b.pack (side="bottom")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10,
                      font="Times 24 bold")
top.mainloop()
```



Redimensionamento

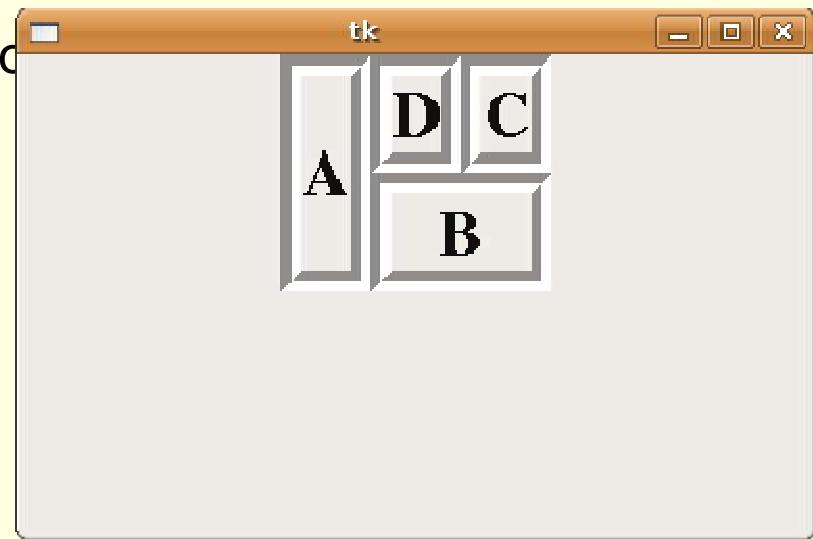
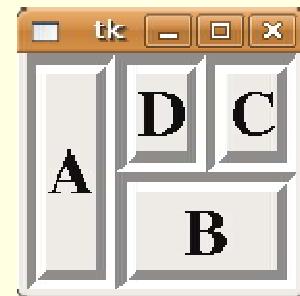
- Por default, o pack não redimensiona os filhos quando o pai é redimensionado
- Duas opções controlam o redimensionamento dos filhos
 - expand (booleano)
 - Se verdadeiro, indica que o filho deve tomar toda a cavidade disponível no pai
 - Caso contrário, toma apenas o espaço necessário (default)
 - fill ('none', 'x', 'y' ou 'both')
 - Indica como o desenho do elemento irá preencher o espaço alocado
 - 'x' / 'y' indica que irá preencher a largura / altura
 - 'both' indica preenchimento de todo o espaço
 - 'none' indica que apenas o espaço necessário será ocupado (default)

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left", fill="y")
b = Label (top, text="B") ; b.pack (side="bottom", fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10, font="Times 24 bold")
top.mainloop()
```

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack()
a = Label (top, text="A") ; a.pack (side="left", fill="y")
b = Label (top, text="B") ; b.pack (side="bottom", fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", borderwidth=2)
top.mainloop()
```

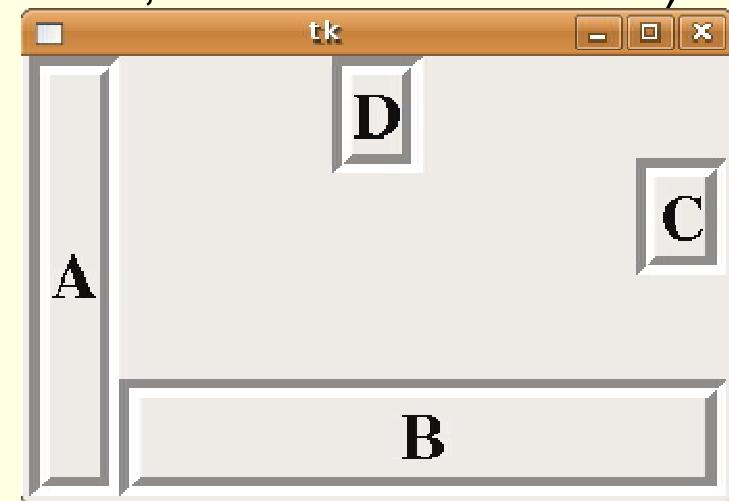
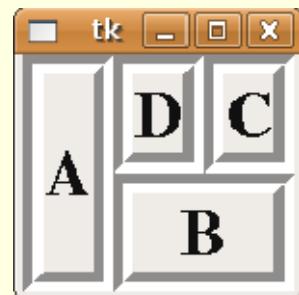


Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack (side="left",fill="y")
b = Label (top, text="B") ; b.pack (side="bottom",fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10, font="Times 24 bold")
top.mainloop()
```

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack (side="left",fill="y")
b = Label (top, text="B") ; b.pack (side="bottom",fill="x")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10, font="Times 24 bold")
top.mainloop()
```

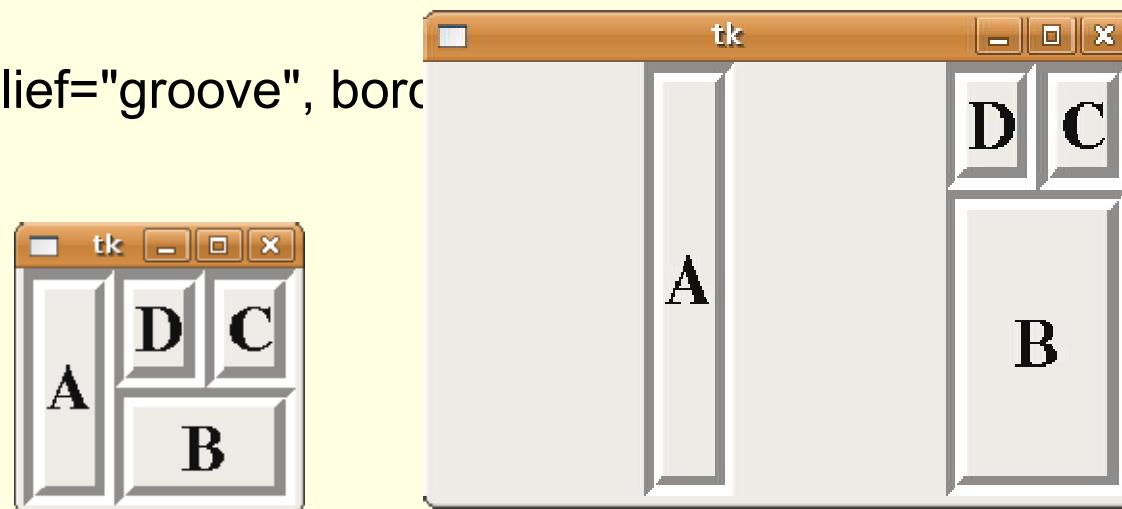


Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack (side="left",expand=True,fill="y")
b = Label (top, text="B") ; b.pack
    (side="bottom",expand=True,fill="both")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", border=10, font="Times 24 bold")
top.mainloop()
```

Exemplo

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
a = Label (top, text="A") ; a.pack (side="left",expand=True,fill="y")
b = Label (top, text="B") ; b.pack
    (side="bottom",expand=True,fill="both")
c = Label (top, text="C") ; c.pack (side="right")
d = Label (top, text="D") ; d.pack (side="top")
for widget in (a,b,c,d):
    widget.configure(relief="groove", borderwidth=2)
top.mainloop()
```



Usando frames

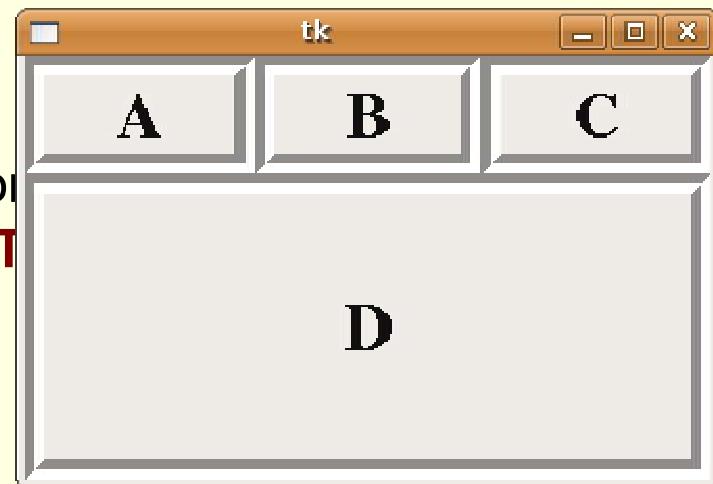
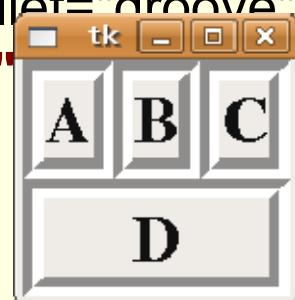
- Frames podem ser usados para auxiliar no layout dos elementos com pack. Ex.:

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
f = Frame (top); f.pack (fill='x')
a = Label (f, text="A")
b = Label (f, text="B")
c = Label (f, text="C")
d = Label (top, text="D")
for w in (a,b,c,d):
    w.configure(relief="groove", border=10, font="Times 24 bold")
    w.pack(side="left", expand=True, fill="both")
top.mainloop()
```

Usando frames

- Frames podem ser usados para auxiliar no layout dos elementos com pack. Ex.:

```
from Tkinter import *
top = Frame() ; top.pack(fill='both', expand=True)
f = Frame (top); f.pack (fill='x')
a = Label (f, text="A")
b = Label (f, text="B")
c = Label (f, text="C")
d = Label (top, text="D")
for w in (a,b,c,d):
    w.configure(relief="groove", borderwidth=1)
    w.pack(side="top", fill="x", expand=True)
top.mainloop()
```



Programação com eventos

- Diferente da programação convencional
- O programa não está sob controle 100% do tempo
 - Programa entrega controle ao sistema
 - Em Tk: método(função) mainloop
- Interação gera eventos. Ex:
 - Acionamento de um menu ou de um botão
 - Mouse arrastado sobre uma janela
 - Uma caixa de texto teve seu valor alterado
- O tratamento de um evento é feito por uma rotina “Callback”

A opção *command*

- Muitos componentes do Tk suportam a opção *command* que indica uma função a ser invocada sempre que o widget é acionado
- Tipicamente, a função (ou método) usado obtém valores de outros widgets para realizar alguma operação

Exemplo

```
from Tkinter import *\n\ndef inc():\n    n=int(rotulo.configure("text")[4])+1\n    rotulo.configure(text=str(n))\n\nb = Button(text="Incrementa",command=inc)\nb.pack()\nrotulo = Label(text="0")\nrotulo.pack()\nmainloop()
```

Exemplo

```
from Tkinter import *\n\ndef inc():\n    n=int(rotulo.configure("text")[4])+1\n    rotulo.configure(text=str(n))\n\nb = Button(text="Incrementa",command=inc)\nb.pack()\nrotulo = Label(text="0")\nrotulo.pack()\nmainloop()
```



Exemplo

```
from Tkinter import *\n\ndef inc():\n    n=int(rotulo.configure("text")[4])+1\n    rotulo.configure(text=str(n))\n\nb = Button(text="Incrementa",command=inc)\nb.pack()\nrotulo = Label(text="0")\nrotulo.pack()\nmainloop()
```



Eventos e *Bind*

- Widgets que não dispõem da opção command também podem receber eventos e responder a eles
- O método bind permite especificar um padrão de eventos ao qual o widget será sensível e uma rotina callback para tratá-lo

`bind(padrão,rotina)`

- *padrão* é uma string que descreve quais eventos a rotina irá tratar
- *rotina* é uma função ou método com exatamente um parâmetro: o evento que deve ser tratado

Exemplo

```
from Tkinter import *\n\ndef clica (e):\n    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)\n    r.configure(text=txt)\n\nr = Label()\n\nr.pack(expand=True, fill="both")\nr.master.geometry("200x200")\nr.bind("<Button-1>", clica)\nmainloop()
```

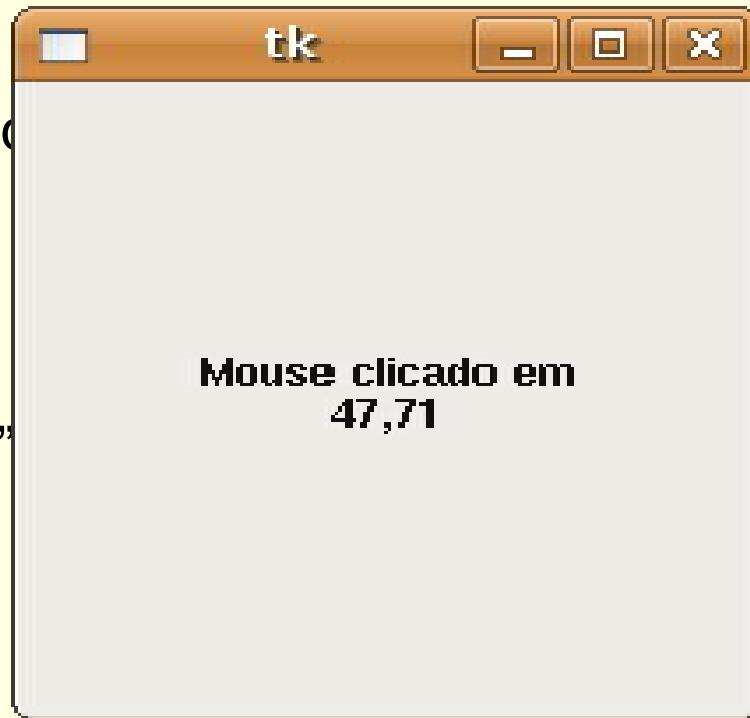
Exemplo

```
from Tkinter import *\n\ndef clica (e):\n    txt = "Mouse clicado em\n%o\n%o" % (e.x, e.y)\n    r.configure(text=txt)\n\nr = Label()\n\nr.pack(expand=True, fill="both")\nr.master.geometry("200x200")\nr.bind("<Button-1>", clica)\nmainloop()
```



Exemplo

```
from Tkinter import *\n\ndef clica (e):\n    txt = "Mouse clicado em\n%0.1f,%0.1f" % (e.x, e.y)\n    r.configure(text=txt)\n\nr = Label()\n\nr.pack(expand=True, fill="both")\nr.master.geometry("200x200")\nr.bind("<Button-1>", clica)\nmainloop()
```



Campos do objeto evento

- x,y : posição do mouse com relação ao canto superior esquerdo do widget
- x_root, y_root: posição do mouse com relação ao canto superior esquerdo da tela
- char: caractere digitado (eventos de teclado)
- keysym: representação simbólica da tecla
- keycode: representação numérica da tecla
- num: número do botão – 1/2/3 = Esquerdo/Meio/Direito – (eventos de mouse)
- widget: o objeto que gerou o evento
- width,height: largura e altura do widget (evento Configure)

Padrões de evento (mouse)

- <Button-*i*> para $i = 1, 2, 3$: botão *i* do mouse pressionado sobre o widget
- <Motion> : mouse arrastado sobre o widget
- <B*i*-Motion> : mouse arrastado sobre o widget com o botão *i* pressionado
- <ButtonRelease-*i*> : botão *i* do mouse solto sobre o widget
- <Double-Button-*i*>: botão *i* do mouse clicado duas vezes em seguida
- <Enter>: O mouse entrou na área do widget
- <Leave>: O mouse saiu da área do widget

Padrões de evento (teclado)

- *caracter* : O *caracter* foi digitado sobre o widget
- <Key>: Algum caracter foi digitado sobre o widget
- <Return>: Tecla *enter* foi digitada
- <Tab>, <F1>, <Up>...: A tecla correspondente foi digitada
- <Shift-Tab>, <Alt-F1>, <Ctrl-Up>...: Tecla com modificador
- Para os eventos serem gerados, é preciso que o *foco* de teclado esteja sobre o widget
 - Depende do sistema de janelas
 - O foco para um widget pode ser forçado usando o método *focus*

Exemplo

```
from Tkinter import *
def clica (e):
    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeyCode=%s\nChar=%s"\n
        %(e.keysym,e.keyCode,e.char)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
r.bind("<Key>", tecla)
mainloop()
```

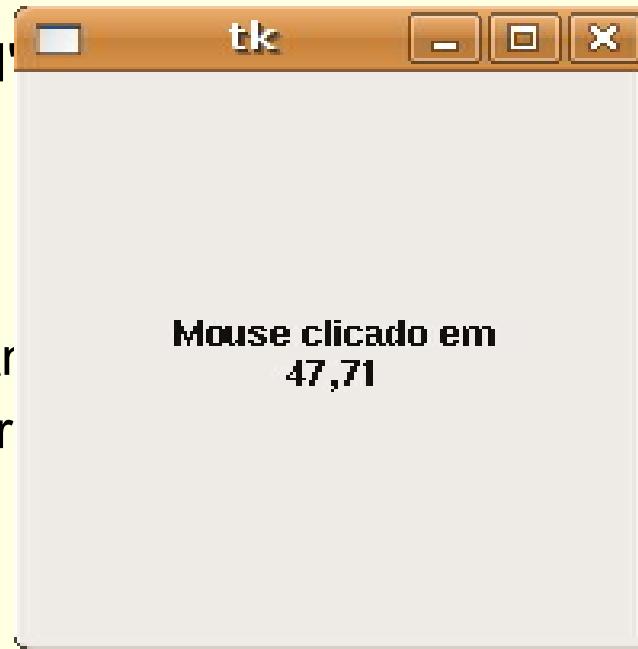
Exemplo

```
from Tkinter import *
def clica (e):
    txt = "Mouse clicado em\n%d,%d"
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeycode=%s"
    txt+= "%(e.keysym,e.keycode,e.char)"
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
r.bind("<Key>", tecla)
mainloop()
```



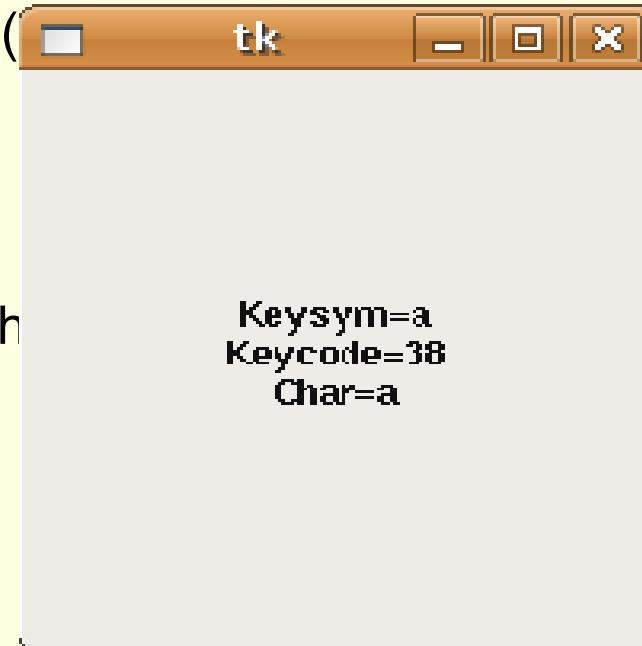
Exemplo

```
from Tkinter import *
def clica (e):
    txt = "Mouse clicado em\n%d,%d"
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeycode=%s\r
        %(e.keysym,e.keycode,e.char)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
r.bind("<Key>", tecla)
mainloop()
```



Exemplo

```
from Tkinter import *
def clica (e):
    txt = "Mouse clicado em\n%d,%d"%(e.x,e.y)
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeyCode=%s\nChar=%s"
    txt=txt%(e.keysym,e.keyCode,e.char)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
r.bind("<Key>", tecla)
mainloop()
```



Exemplo

```
from Tkinter import *
def clica (e):
    txt = "Mouse clicado em\n%d,%d"%e.x,e.y
    r.configure(text=txt)
    r.focus()
def tecla(e):
    txt="Keysym=%s\nKeyCode=%s\nChar-%s"
    txt=txt%(e.keysym,e.keyCode,e.char)
    r.configure(text=txt)
r = Label()
r.pack(expand=True, fill="both")
r.master.geometry("200x200")
r.bind("<Button-1>", clica)
r.bind("<Key>", tecla)
mainloop()
```



Menus

- Podem ser associados a uma janela (menus toplevel), pulldown, popup e em cascata a partir de outro menu
- Todos são instâncias da classe Menu
- Um menu é composto de itens que podem ser
 - command quando pressionado executa uma callback
 - checkbox parecido com command, mas tem um valor booleano associado
 - radiobutton como command, mas representa um de vários estados mutuamente exclusivos
 - cascade ativa um outro menu em cascata
- Para adicionar um item a um menu, use métodos da forma add (“*tipo*”, opções) ou add_*tipo*(opções)

Menu de janela (toplevel)

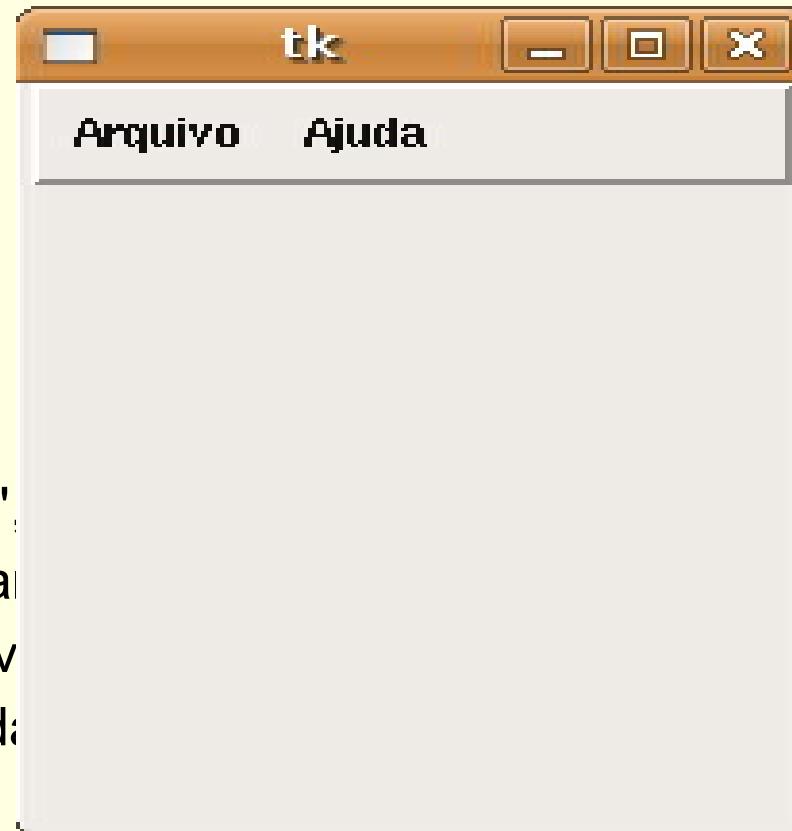
- É tipicamente exibido horizontalmente no topo da janela
 - Aspecto depende do sistema operacional
- Se um outro menu é associado como item cascade, ele é tratado como *pulldown*, isto é, é exibido sob o item do menu de janela
- Assim como outros menus, não necessita ter sua geometria gerenciada (e.g., pack ou grid)
- Para associar a uma janela, usa-se a opção menu do objeto janela.

Exemplo

```
from Tkinter import *
def abrir(): print "abrir"
def salvar(): print "salvar"
def ajuda() : print "ajuda"
top=Tk()
principal=Menu(top)
arquivo=Menu(principal)
arquivo.add_command(label="Abrir",command=abrir)
arquivo.add_command(label="Salvar",command=salvar)
principal.add_cascade(label="Arquivo",menu=arquivo)
principal.add_command(label="Ajuda",command=ajuda)
top.configure(menu=principal)
top.mainloop()
```

Exemplo

```
from Tkinter import *
def abrir(): print "abrir"
def salvar(): print "salvar"
def ajuda() : print "ajuda"
top=Tk()
principal=Menu(top)
arquivo=Menu(principal)
arquivo.add_command(label="Abrir")
arquivo.add_command(label="Salvar")
principal.add_cascade(label="Arquivo", menu=arquivo)
principal.add_command(label="Ajuda")
top.configure(menu=principal)
top.mainloop()
```



Exemplo

```
from Tkinter import *
def abrir(): print "abrir"
def salvar(): print "salvar"
def ajuda() : print "ajuda"
top=Tk()
principal=Menu(top)
arquivo=Menu(principal)
arquivo.add_command(label="Abrir"
arquivo.add_command(label="Salvar")
principal.add_cascade(label="Arquivo")
principal.add_command(label="Ajuda")
top.configure(menu=principal)
top.mainloop()
```



Menus Popup

- Um menu popup é aquele que é exibido numa janela independente
- Para que o menu seja exibido, é preciso invocar o método `post`:
 - `post (x, y)`
 - onde x e y são as coordenadas do canto superior esquerdo do menu com relação ao canto superior esquerdo da tela

Exemplo

```
from Tkinter import *

def alo(): print "Alo!"

root = Tk()
menu = Menu(root, tearoff=0)
menu.add_command(label="Alo 1", command=alo)
menu.add_command(label="Alo 2", command=alo)

def popup(e): menu.post(e.x_root, e.y_root)

frame = Frame(root, width=200, height=200)
frame.pack()
frame.bind("<Button-3>", popup)
root.mainloop()
```

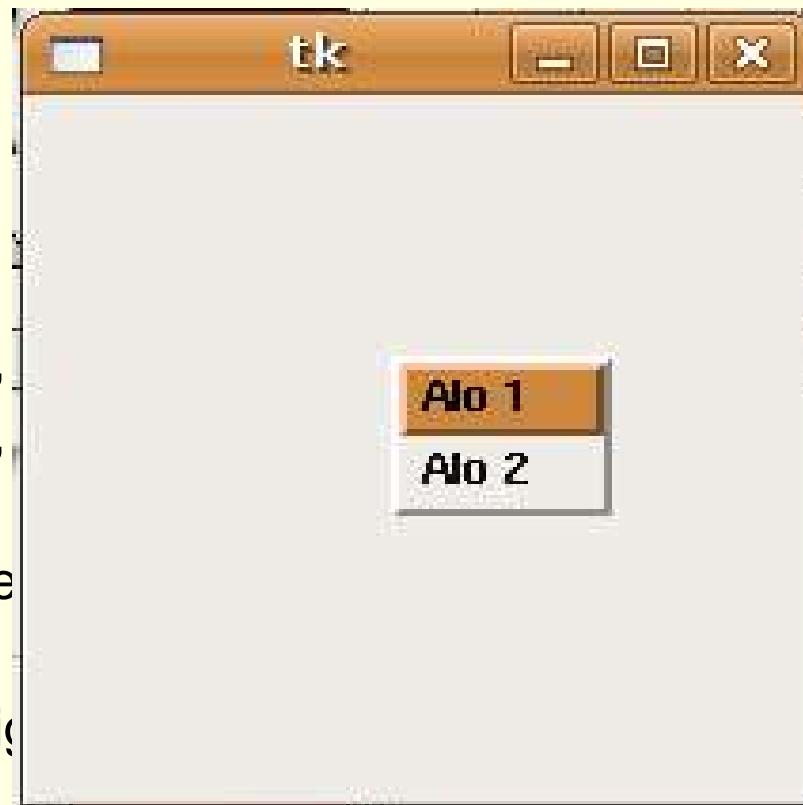
Exemplo

```
from Tkinter import *

def alo(): print "Alo!"

root = Tk()
menu = Menu(root, tearoff=0)
menu.add_command(label="Alo 1",
menu.add_command(label="Alo 2",
def popup(e): menu.post(e.x_root, e

frame = Frame(root, width=200, hei
frame.pack()
frame.bind("<Button-3>", popup)
root.mainloop()
```



Variáveis

- Tk é controlado por um interpretador Tcl (e não diretamente pelo python)
- Em alguns casos, deseja-se usar variáveis na interface
 - Por exemplo, é possível especificar que o texto exibido em um Label é o valor de uma variável (e não uma constante)
 - Nesse caso, usa-se a opção `textvar` ao invés de `text`
- Variáveis Tcl são expostas à aplicação Python através das classes `StringVar`, `IntVar` e `DoubleVar`
 - O construtor é da forma `StringVar(master)` onde `master` é uma janela ou widget
- Instâncias dessas classes possuem os métodos `get` e `set` que podem ser usados para acessar os valores armazenados no interpretador Tcl

Exemplo

```
from Tkinter import *

root = Tk()
soma = DoubleVar(root)
parcela = DoubleVar(root)
def aritmetica (e):
    soma.set(soma.get()+parcela.get())

lsoma = Label(textvar=soma)
eparcela = Entry(textvar=parcela)
eparcela.bind("<Return>", aritmetica)
lsoma.pack()
eparcela.pack()
root.mainloop()
```

Exemplo

```
from Tkinter import *\n\nroot = Tk()\nsoma = DoubleVar(root)\nparcela = DoubleVar(root)\ndef aritmetica (e):\n    soma.set(soma.get()+parcela.get())\n\nl soma = Label(textvar=soma)\ne parcela = Entry(textvar=parcela)\ne parcela.bind("<Return>", aritmetica)\nl soma.pack()\ne parcela.pack()\nroot.mainloop()
```



Exemplo

```
from Tkinter import *\n\nroot = Tk()\nsoma = DoubleVar(root)\nparcela = DoubleVar(root)\ndef aritmetica (e):\n    soma.set(soma.get()+parcela.get())\nl soma = Label(textvar=soma)\ne parcela = Entry(textvar=parcela)\ne parcela.bind("<Return>", aritmetica)\nl soma.pack()\ne parcela.pack()\nroot.mainloop()
```



Exemplo

```
from Tkinter import *\n\nroot = Tk()\nsoma = DoubleVar(root)\nparcela = DoubleVar(root)\ndef aritmetica (e):\n    soma.set(soma.get()+parcela.get())\nl soma = Label(textvar=soma)\ne parcela = Entry(textvar=parcela)\ne parcela.bind("<Return>", aritmetica)\nl soma.pack()\ne parcela.pack()\nroot.mainloop()
```



Checkbuttons

- Checkbutton Representa uma variável que pode ter dois valores distintos (tipicamente um valor booleano). Clicando no botão alterna-se entre os valores
- A callback especificada pela opção command é chamada sempre que a variável muda de valor
- Estado é armazenado pela variável Tcl especificada pela opção variable
- Se a variável é inteira, o valor correspondente ao checkbutton “desligado”/“ligado” é 0/1
- É possível usar um checkbutton com uma variável string
 - Nesse caso, os valores correspondentes a “desligado”/“ligado” são especificados com as opções offvalue e onvalue

Exemplo

```
from Tkinter import *

root = Tk()
v1 = IntVar(root)
v2 = StringVar(root)

def exibe():
    l.config(text="v1=%d,v2=%s"%(v1.get(),v2.get()))

c1 = Checkbutton(text="V1", var=v1, command=exibe)
c2 = Checkbutton(text="V2", var=v2, command=exibe,\n                 onvalue="Sim", offvalue="Nao")
l = Label()
for w in (c1,c2,l):w.pack()
exibe()
mainloop()
```

Exemplo

```
from Tkinter import *
```

```
root = Tk()  
v1 = IntVar(root)  
v2 = StringVar(root)
```

```
def exibe():  
    l.config(text="v1:
```

```
c1 = Checkbutton(t  
c2 = Checkbutton(t
```

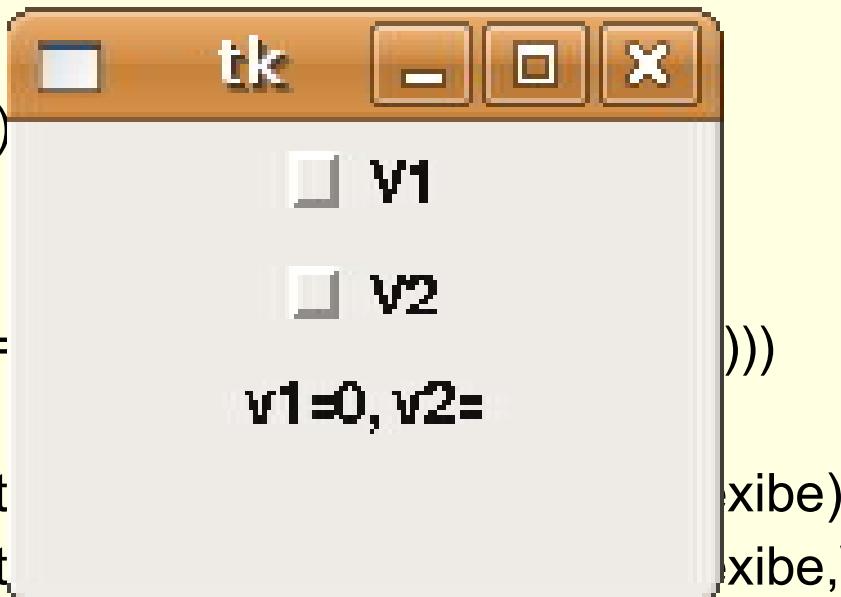
```
        onvalue="Sim", offvalue="Nao")
```

```
l = Label()
```

```
for w in (c1,c2,l):w.pack()
```

```
exibe()
```

```
mainloop()
```



Exemplo

```
from Tkinter import *
```

```
root = Tk()  
v1 = IntVar(root)  
v2 = StringVar(root)
```

```
def exibe():  
    l.config(text="v1:
```

```
c1 = Checkbutton(t  
c2 = Checkbutton(t
```

```
        onvalue="Sim", offvalue="Nao")
```

```
l = Label()
```

```
for w in (c1,c2,l):w.pack()
```

```
exibe()
```

```
mainloop()
```



Exemplo

```
from Tkinter import *
```

```
root = Tk()  
v1 = IntVar(root)  
v2 = StringVar(root)
```

```
def exibe():  
    l.config(text="v1:
```

```
c1 = Checkbutton(t  
c2 = Checkbutton(t  
    onvalue="Sim", offvalue="Nao")
```

```
l = Label()  
for w in (c1,c2,l):w.pack()  
exibe()  
mainloop()
```



Radiobuttons

- Radiobutton representa um possível valor de uma variável que tem um de muitos valores. Clicando o botão, a variável assume aquele valor
- A variável é especificada com a opção variable e o valor associado com a opção value
- Os radiobuttons que se referem à mesma variável funcionam em conjunto
 - Ex.: ligar um faz com que outro seja desligado
- Um radiobutton é mostrado com um indicador ao lado
 - Pode-se desabilitar o indicador usando a opção indicatoron=False
 - Nesse caso, é mostrado como um botão normal

Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                ("vermelho","red"),
                ("azul","blue"),
                ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta).pack(anchor=W)
mainloop()
```

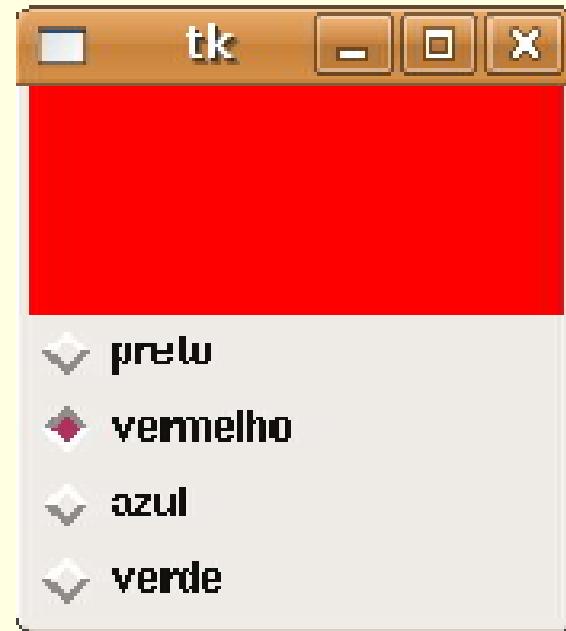
Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                ("vermelho","red"),
                ("azul","blue"),
                ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta).pack(anchor=W)
mainloop()
```



Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                ("vermelho","red"),
                ("azul","blue"),
                ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta).pack(anchor=W)
mainloop()
```



Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                ("vermelho","red"),
                ("azul","blue"),
                ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta).pack(anchor=W)
mainloop()
```



Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                 ("vermelho","red"),
                 ("azul","blue"),
                 ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta,indicatoron=False).pack(fill='x')
mainloop()
```

Exemplo

```
from Tkinter import *
root=Tk()
cor = StringVar(root)
cor.set("black")
l = Label(background=cor.get())
l.pack(fill='both',expand=True)
def pinta(): l.configure(background=cor.get())
for txt,val in (("preto","black"),
                ("vermelho","red"),
                ("azul","blue"),
                ("verde","green")):
    Radiobutton(text=txt,value=val,variable=cor,
                command=pinta,indicatoron=False).pack(fill='x')
mainloop()
```



Entry

- Um Entry permite entrada/edição de uma linha de texto
- O texto associado ao Entry é normalmente armazenado numa variável indicada pela opção textvariable
 - Se não indicada, é usada uma variável interna cujo valor pode ser obtido usando o método get()
- Há diversos métodos para manipular diretamente o texto
 - Usam o conceito de índices (não confundir com os índices usado pelo Python)
 - Por exemplo, o índice INSERT indica a posição do texto onde o cursor de inserção se encontra, 0 a posição antes do primeiro caractere e END a posição ao final do texto

Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa",command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```

Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa",command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```



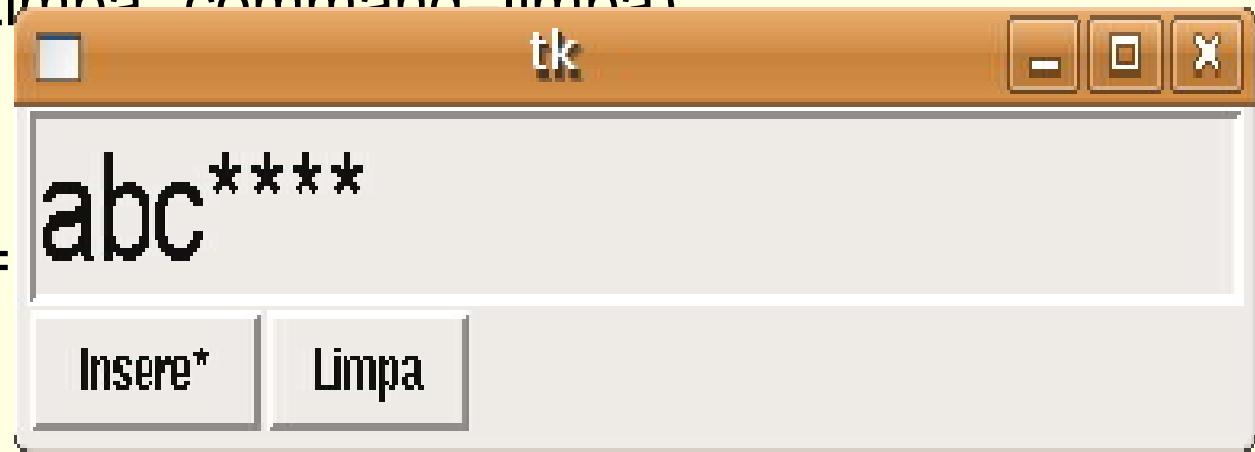
Exemplo

```
from Tkinter import *
def insere(): e.insert(0, "*")
def limpa(): e.delete(0, END)
e=Entry(font="Arial 24")
i=Button(text="Insere*", command=insere)
l=Button(text="Limpa", command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```



Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa", command=limpa)
e.pack()
for w in (i,l):
    w.pack(side=
mainloop()
```



Canvas

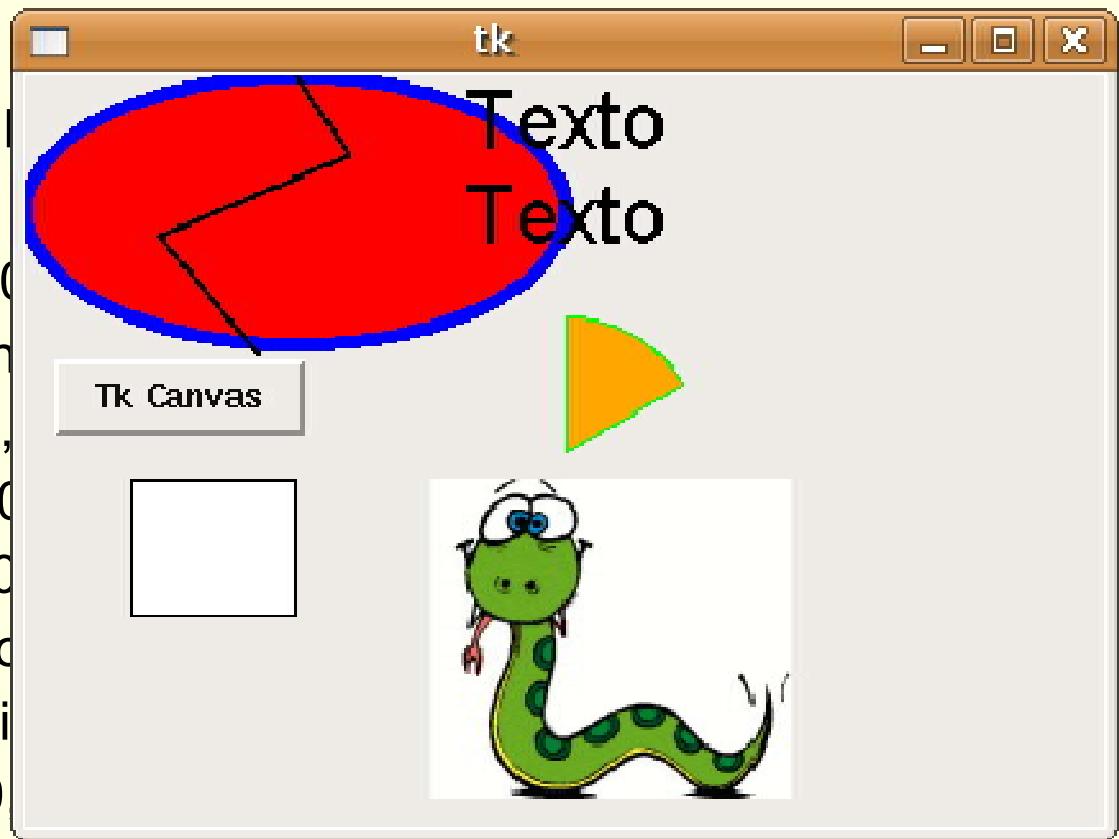
- Permite a exibição e edição de gráficos estruturados 2D
- Elementos gráficos (itens) são introduzidos usando métodos da forma `create_`*tipo* (...), onde *tipo* pode ser
 - arc arco de círculo
 - bitmap imagem binária
 - image imagem colorida
 - line linha poligonal
 - oval círculos e elipses
 - polygon polígonos
 - rectangle retângulo
 - text texto
 - window um widget tk

Exemplo

```
from Tkinter import *
root = Tk()
root.geometry("512x512")
c = Canvas(root, width=512, height=512)
c.pack()
o = c.create_oval(1,1,200,100,outline="blue",width=5,fill="red")
widget = Button(text="Tk Canvas")
w = c.create_window(10,120>window=widget,anchor=W)
l = c.create_line(100,0,120,30,50,60,100,120,fill="black",width=2)
r = c.create_rectangle(40,150,100,200,fill="white")
img = PhotoImage(file="python.gif")
i = c.create_image (150,150,image=img,anchor=NW)
a = c.create_arc (150,90,250,190,start=30,extent=60,\outline="green",fill="orange")
t = c.create_text(200,35,text="Texto\nTexto",font="Arial 22")
mainloop()
```

Exemplo

```
from Tkinter import *
root = Tk()
root.geometry("512x512")
c = Canvas(root, width=512, height=512)
c.pack()
o = c.create_oval(1,1,200,100, outline="blue", fill="red")
widget = Button(text="Tk Canvas")
w = c.create_window(10,120, width=100, height=50)
l = c.create_line(100,0,120,30)
r = c.create_rectangle(40,150,150,250)
img = PhotoImage(file="python.gif")
i = c.create_image (150,150,image=img)
a = c.create_arc (150,90,250,270,
                  outline="green",fill="orange")
t = c.create_text(200,35,text="Texto\nTexto",font="Arial 22")
mainloop()
```



Coordenadas de Itens

- Todos os métodos `create_item` têm como primeiros argumentos um par de coordenadas x,y do item
 - Os itens oval e rectangle requerem mais um par de coordenadas para delimitar a extensão (caixa envolvente)
 - Os itens line e polygon podem ser seguidos por outros pares de coordenadas que especificam demais vértices
- As coordenadas referem-se a um sistema de coordenadas próprio que pode ser diferente do da janela
 - A área do canvas que deve ser mostrada na janela pode ser modificada pela opção
`scrollarea=(xmin,ymin,xmax,ymax)`
 - Para obter as coordenadas do canvas dadas as coordenadas da janela, usam-se os métodos `canvasx(x)` e `canvasy(y)`

Identificação de Itens

- Todo item de um canvas tem um identificador numérico que é retornado pelo método `create_item`
- Pode-se também associar tags (etiquetas) a itens
 - Usa-se a opção `tags=tags` onde `tags` pode ser uma string ou uma tupla com várias strings
 - Uma mesma etiqueta pode ser associada a mais de um item
- O identificador ALL refere-se a todos os itens do canvas
- O identificador CURRENT refere-se ao item do canvas sob o cursor do mouse
 - Usado em callbacks de canvas para alterar propriedades dos itens clicados

Métodos de Canvas

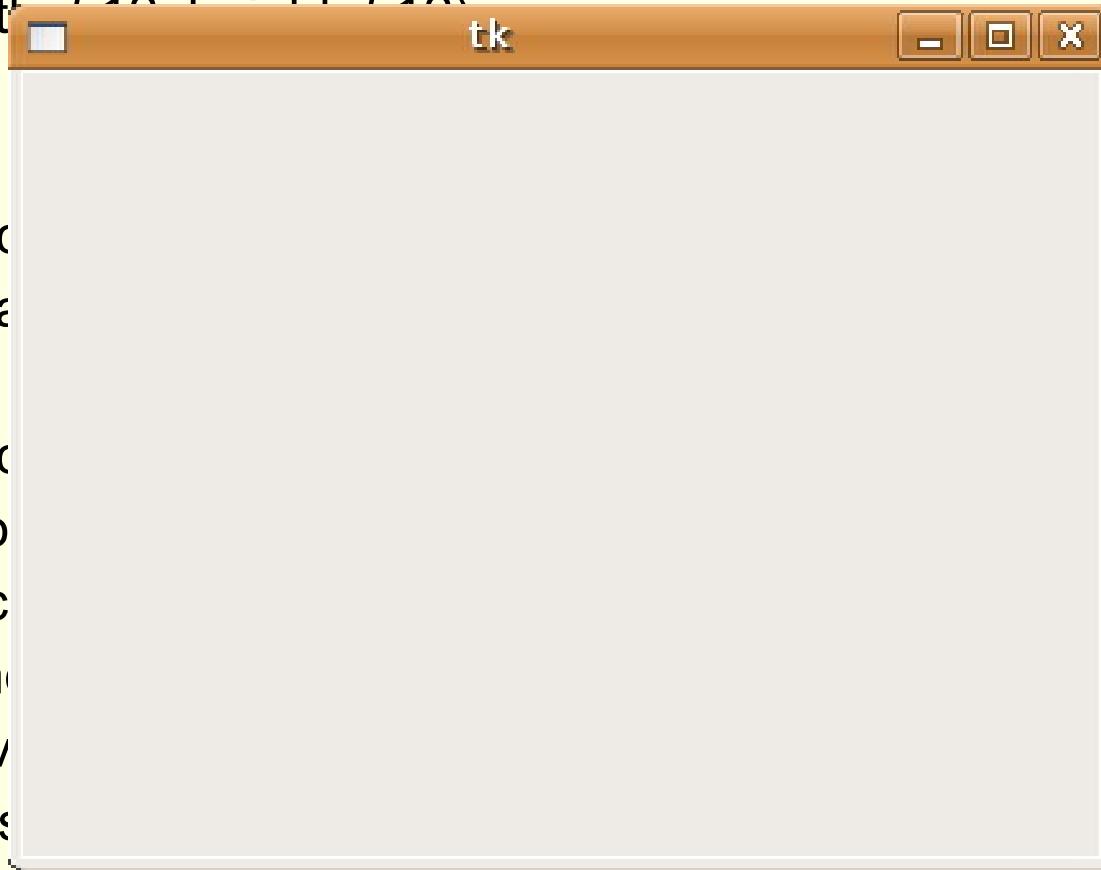
- `itemconfig(itemOuTag, ...)` altera opções do(s) item(s)
- `tag_bind(itemOuTag, padrão, callback)` associa uma *callback* a um *padrão* de eventos sobre o(s) item(s)
- `delete(itemOuTag)` remove o(s) item(s)
- `move(itemOuTag, dx,dy)` translada o(s) item(s)
- `coords(itemOuTag, x1,x2,..xN,yN)` altera as coordenadas do(s) item(s)
- `coords(item)` retorna as coordenadas do item
- `bbox(itemOuTag)` retorna uma tupla com a caixa envolvente dos itens
- `itemcget(item,opção)` retorna o valor da *opção* dada do *item*

Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=512, height=512)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tags="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente") + [x,y]
    c.coords("corrente",*coords)
def fechalinha(e): c.itemconfig("corrente",tags=())
c.bind("<Button-1>", novalinha)
c.bind("<B1-Motion>", estendelinha)
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```

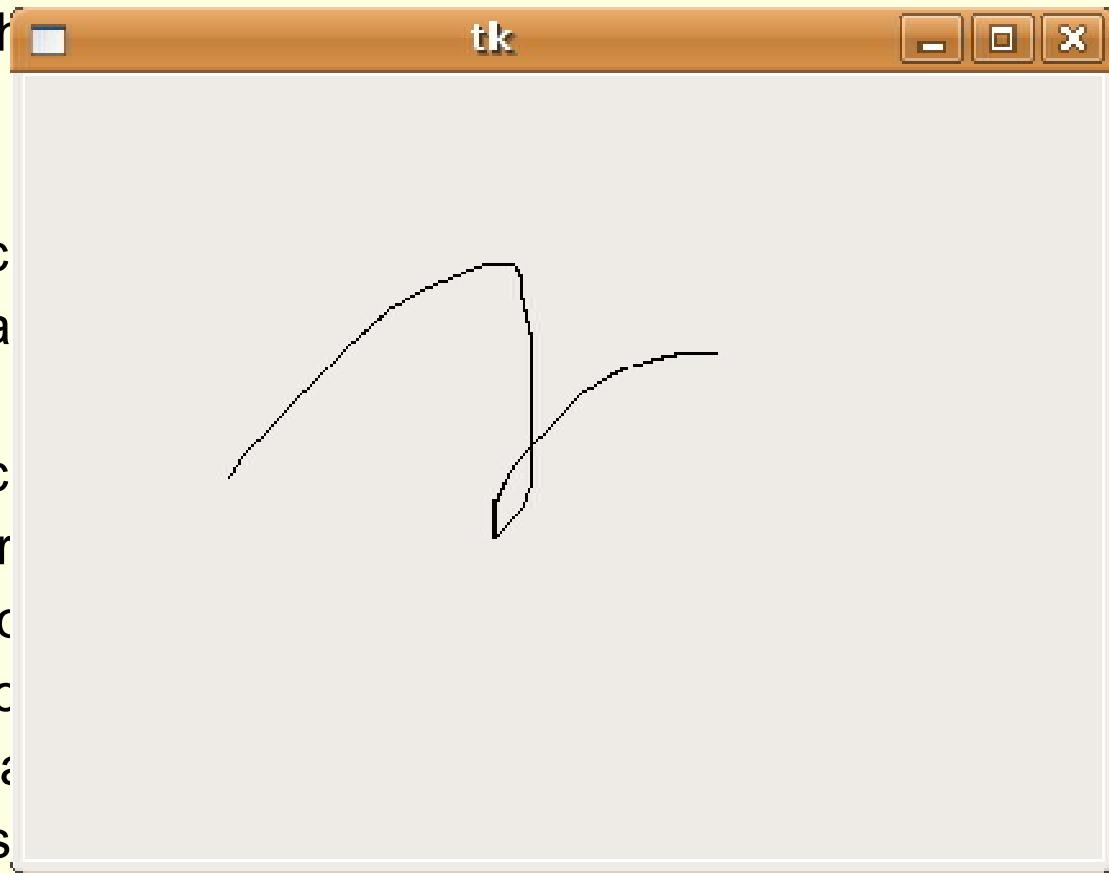
Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=510, height=510)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tags="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente",*coords+[x,y])
def fechalinha(e): c.itemconfig("corrente", fill="black")
c.bind("<Button-1>", novalinha)
c.bind("<B1-Motion>", estendelinha)
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```



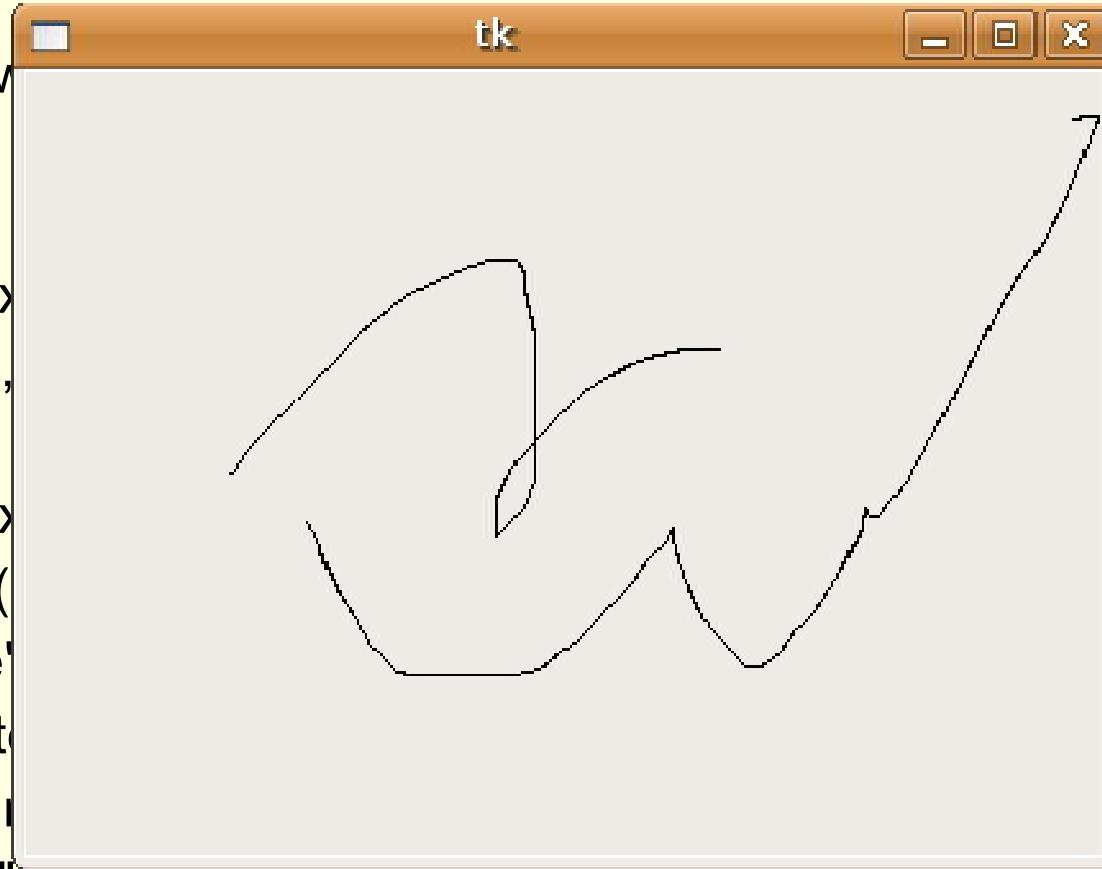
Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=400, height=300)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tag="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente", *coords + [x,y])
def fechalinha(e):
    c.itemconfig("corrente", close=True)
    c.bind("<Button-1>", novalinha)
    c.bind("<B1-Motion>", estendelinha)
    c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```



Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=400, height=400)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y)
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente", coords[0], coords[1], x,y)
def fechalinha():
    c.itemconfig("corrente", fill="white", outline="black")
    c.bind("<Button-1>", novalinha)
    c.bind("<B1-Motion>", estendelinha)
    c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```

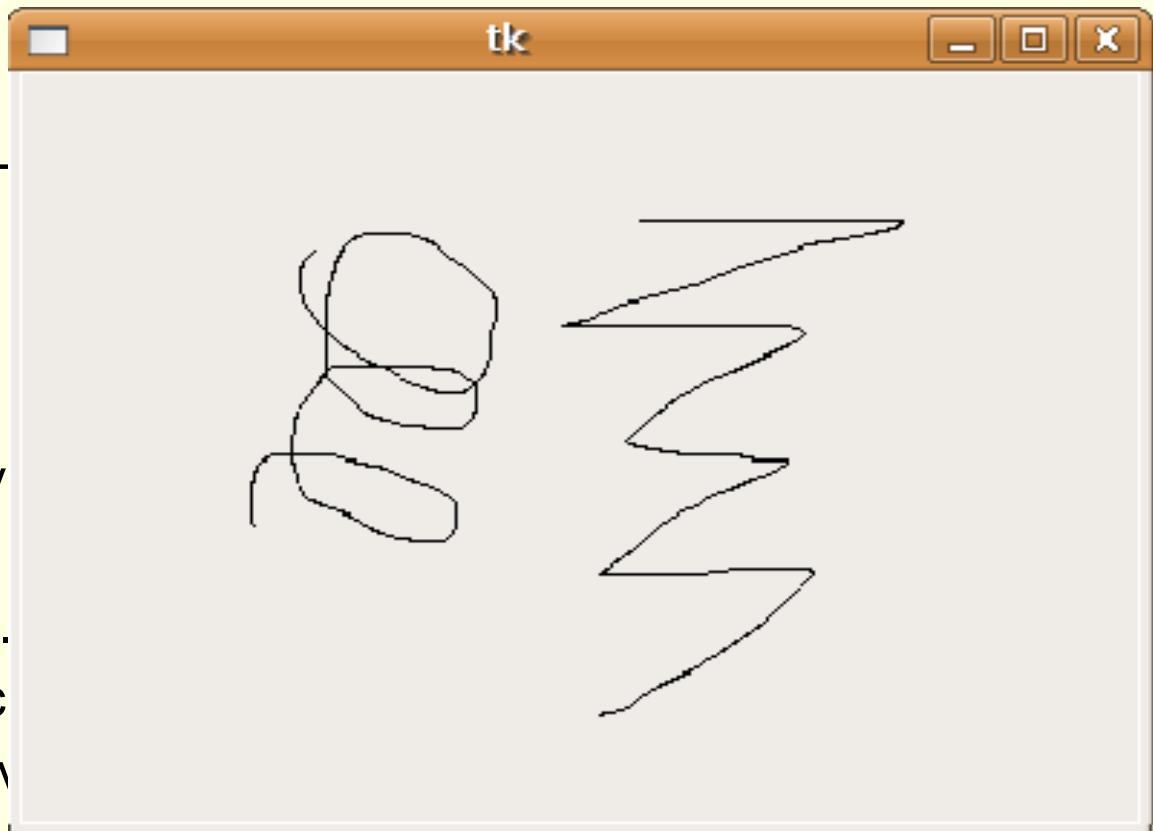


Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x), c.canvasy(e.y)
    c.itemconfig(CURRENT, tags="sel")
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x), c.canvasy(e.y)
    c.move("sel",x1-x0,y1-y0)
    x0,y0=x1,y1
def deselecionalinha(e): c.itemconfig("sel", tags=())
c.bind("<Button-3>", selecionalinha)
c.bind("<B3-Motion>", movelinha)
c.bind("<ButtonRelease-3>", deselecionalinha)
c.pack()
mainloop()
```

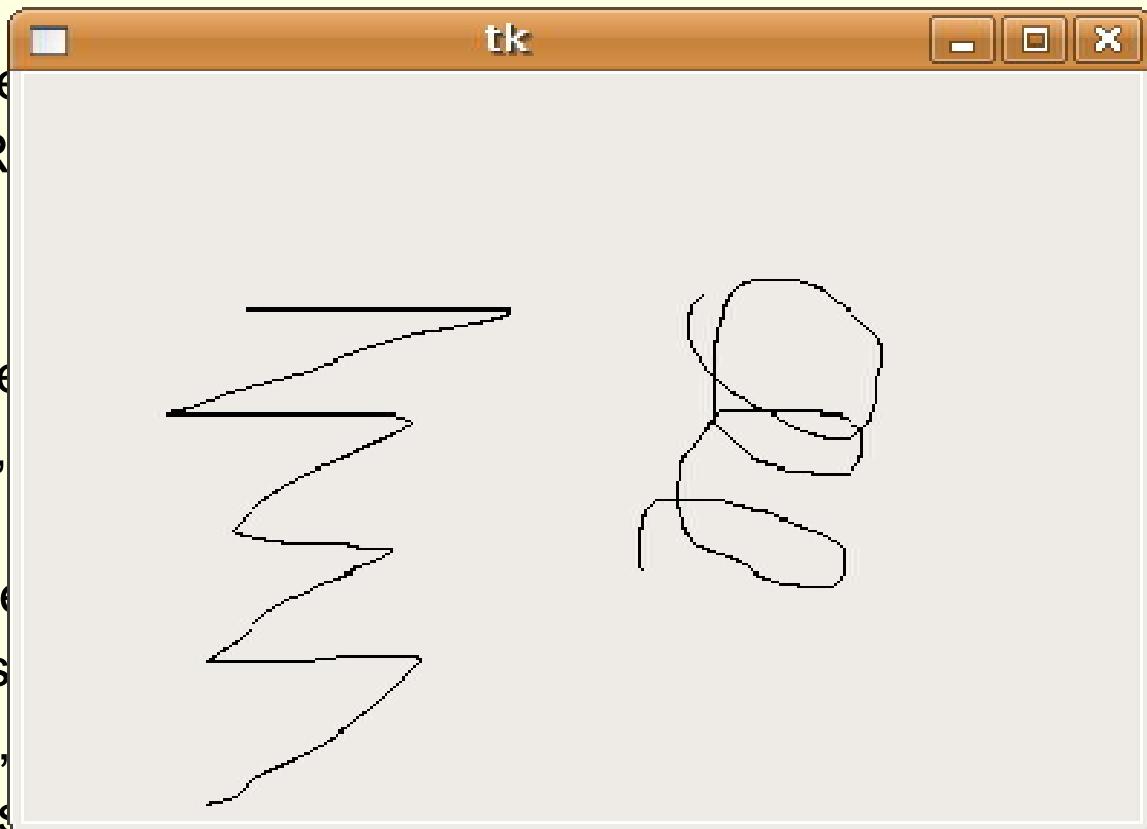
Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x),
    c.itemconfig(CURRENT
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x),
    c.move("sel",x1-x0,y1-y
    x0,y0=x1,y1
def deselecionalinha(e): c.
c.bind("<Button-3>", selec
c.bind("<B3-Motion>", mov
c.bind("<ButtonRelease-3> , aselecionalinha)
c.pack()
mainloop()
```



Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x),c.canvasy(e.y)
    c.itemconfig(CURRENT,strokeWidth=2)
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x),c.canvasy(e.y)
    c.move("sel",x1-x0,y1-y0)
    x0,y0=x1,y1
def deselecionalinha(e):
    c.bind("<Button-3>", selecionalinha)
    c.bind("<B3-Motion>", movelinha)
    c.bind("<ButtonRelease-3>", deselecionalinha)
c.pack()
mainloop()
```



Scrollbar

- Barras de rolamento são usadas com outros widgets com área útil maior do que pode ser exibida na janela (Canvas, Text, Listbox, Entry)
- Uma barra de rolamento horizontal (vertical) funciona chamando o método `xview` (`yview`) do widget associado
 - Isto é feito configurando a opção `command` da barra
- Por outro lado, sempre que a visão do widget muda, a barra de rolamento precisa ser atualizada
 - Isto é feito configurando a opção `xscrollcommand` (ou `yscrollcommand`) do widget ao método `set` da barra

Exemplo

```
from Tkinter import *
lb = Listbox()
lb.pack(side=LEFT,expand=True,fill="both")
sb = Scrollbar()
sb.pack(side=RIGHT,fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(100):
    lb.insert(END,i)
```

Exemplo

```
from Tkinter import *
lb = Listbox()
lb.pack(side=LEFT,expand=True)
sb = Scrollbar()
sb.pack(side=RIGHT,fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(100):
    lb.insert(END,i)
```



Entry

- Um Entry permite entrada/edição de uma linha de texto
- O texto associado ao Entry é normalmente armazenado numa variável indicada pela opção textvariable
 - Se não indicada, é usada uma variável interna cujo valor pode ser obtido usando o método get()
- Há diversos métodos para manipular diretamente o texto
 - Usam o conceito de índices (não confundir com os índices usado pelo Python)
 - Por exemplo, o índice INSERT indica a posição do texto onde o cursor de inserção se encontra, 0 a posição antes do primeiro caractere e END a posição ao final do texto

Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa",command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```

Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa",command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```



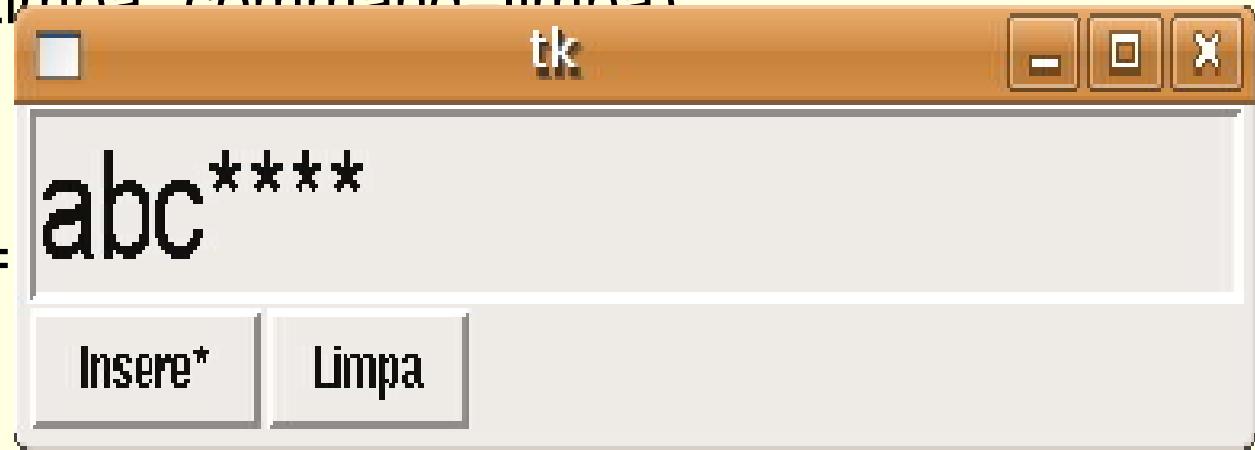
Exemplo

```
from Tkinter import *
def insere(): e.insert(0, "*")
def limpa(): e.delete(0, END)
e=Entry(font="Arial 24")
i=Button(text="Insere*", command=insere)
l=Button(text="Limpa", command=limpa)
e.pack()
for w in (i,l):
    w.pack(side='left')
mainloop()
```



Exemplo

```
from Tkinter import *
def insere(): e.insert(INSERT,"*")
def limpa(): e.delete(INSERT,END)
e=Entry(font="Arial 24")
i=Button(text="Insere*",command=insere)
l=Button(text="Limpa", command=limpa)
e.pack()
for w in (i,l):
    w.pack(side=
mainloop()
```



Canvas

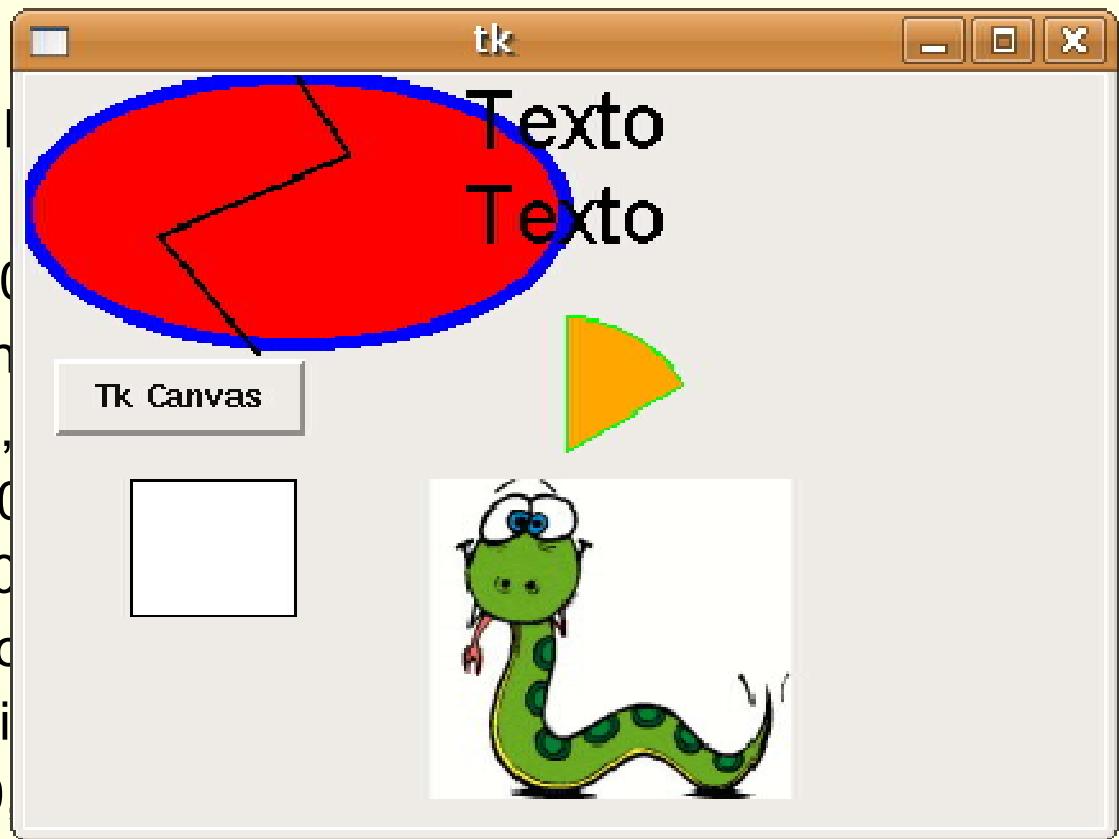
- Permite a exibição e edição de gráficos estruturados 2D
- Elementos gráficos (itens) são introduzidos usando métodos da forma `create_`*tipo* (...), onde *tipo* pode ser
 - arc arco de círculo
 - bitmap imagem binária
 - image imagem colorida
 - line linha poligonal
 - oval círculos e elipses
 - polygon polígonos
 - rectangle retângulo
 - text texto
 - window um widget tk

Exemplo

```
from Tkinter import *
root = Tk()
root.geometry("512x512")
c = Canvas(root, width=512, height=512)
c.pack()
o = c.create_oval(1,1,200,100,outline="blue",width=5,fill="red")
widget = Button(text="Tk Canvas")
w = c.create_window(10,120>window=widget,anchor=W)
l = c.create_line(100,0,120,30,50,60,100,120,fill="black",width=2)
r = c.create_rectangle(40,150,100,200,fill="white")
img = PhotoImage(file="python.gif")
i = c.create_image (150,150,image=img,anchor=NW)
a = c.create_arc (150,90,250,190,start=30,extent=60,\n                  outline="green",fill="orange")
t = c.create_text(200,35,text="Texto\nTexto",font="Arial 22")
mainloop()
```

Exemplo

```
from Tkinter import *
root = Tk()
root.geometry("512x512")
c = Canvas(root, width=512, height=512)
c.pack()
o = c.create_oval(1,1,200,100, outline="blue", fill="red")
widget = Button(text="Tk Canvas")
w = c.create_window(10,120, width=100, height=50)
l = c.create_line(100,0,120,30)
r = c.create_rectangle(40,150,150,250)
img = PhotoImage(file="python.gif")
i = c.create_image (150,150,image=img)
a = c.create_arc (150,90,250,270,
                  outline="green",fill="orange")
t = c.create_text(200,35,text="Texto\nTexto",font="Arial 22")
mainloop()
```



Coordenadas de Itens

- Todos os métodos `create_item` têm como primeiros argumentos um par de coordenadas x,y do item
 - Os itens oval e rectangle requerem mais um par de coordenadas para delimitar a extensão (caixa envolvente)
 - Os itens line e polygon podem ser seguidos por outros pares de coordenadas que especificam demais vértices
- As coordenadas referem-se a um sistema de coordenadas próprio que pode ser diferente do da janela
 - A área do canvas que deve ser mostrada na janela pode ser modificada pela opção
`scrollarea=(xmin,ymin,xmax,ymax)`
 - Para obter as coordenadas do canvas dadas as coordenadas da janela, usam-se os métodos `canvasx(x)` e `canvasy(y)`

Identificação de Itens

- Todo item de um canvas tem um identificador numérico que é retornado pelo método `create_item`
- Pode-se também associar tags (etiquetas) a itens
 - Usa-se a opção `tags=tags` onde `tags` pode ser uma string ou uma tupla com várias strings
 - Uma mesma etiqueta pode ser associada a mais de um item
- O identificador ALL refere-se a todos os itens do canvas
- O identificador CURRENT refere-se ao item do canvas sob o cursor do mouse
 - Usado em callbacks de canvas para alterar propriedades dos itens clicados

Métodos de Canvas

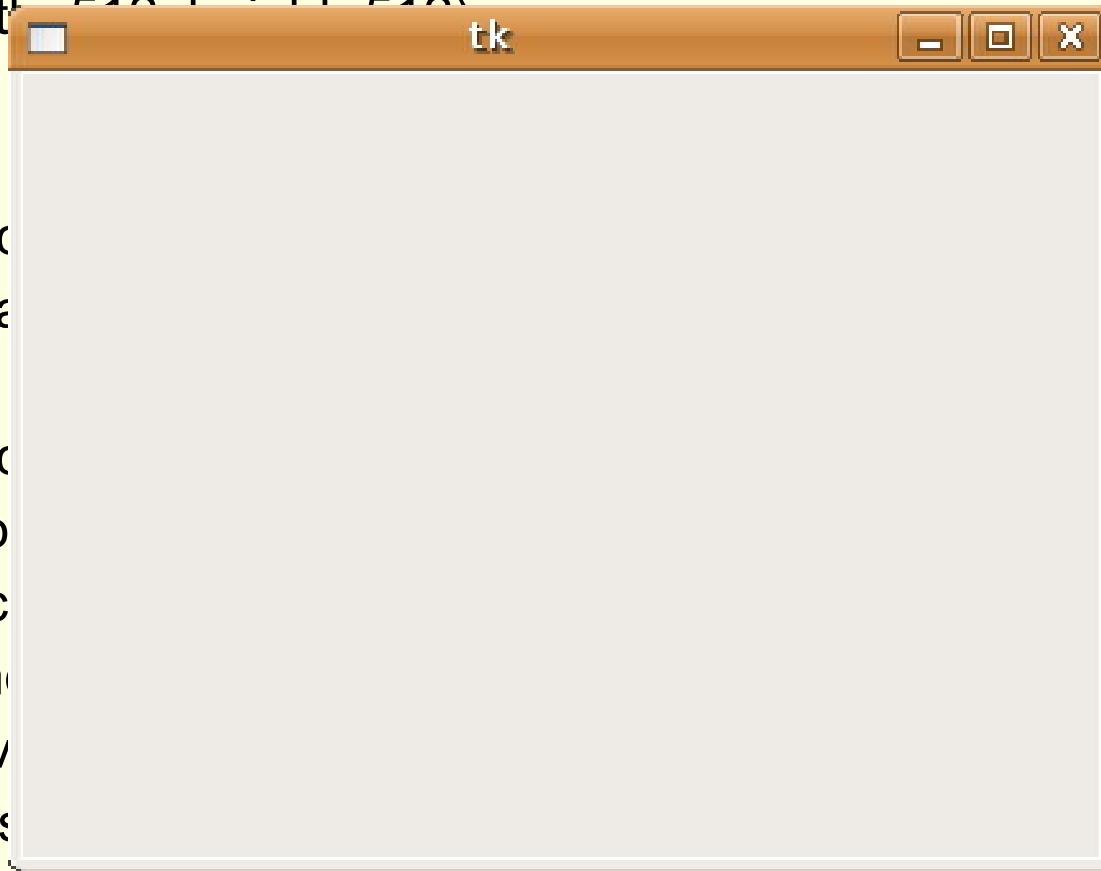
- `itemconfig(itemOuTag, ...)` altera opções do(s) item(s)
- `tag_bind(itemOuTag, padrão, callback)` associa uma *callback* a um *padrão* de eventos sobre o(s) item(s)
- `delete(itemOuTag)` remove o(s) item(s)
- `move(itemOuTag, dx,dy)` translada o(s) item(s)
- `coords(itemOuTag, x1,x2,..xN,yN)` altera as coordenadas do(s) item(s)
- `coords(item)` retorna as coordenadas do item
- `bbox(itemOuTag)` retorna uma tupla com a caixa envolvente dos itens
- `itemcget(item,opção)` retorna o valor da *opção* dada do *item*

Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=512, height=512)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tags="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente") + [x,y]
    c.coords("corrente",*coords)
def fechalinha(e): c.itemconfig("corrente",tags=())
c.bind("<Button-1>", novalinha)
c.bind("<B1-Motion>", estendelinha)
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```

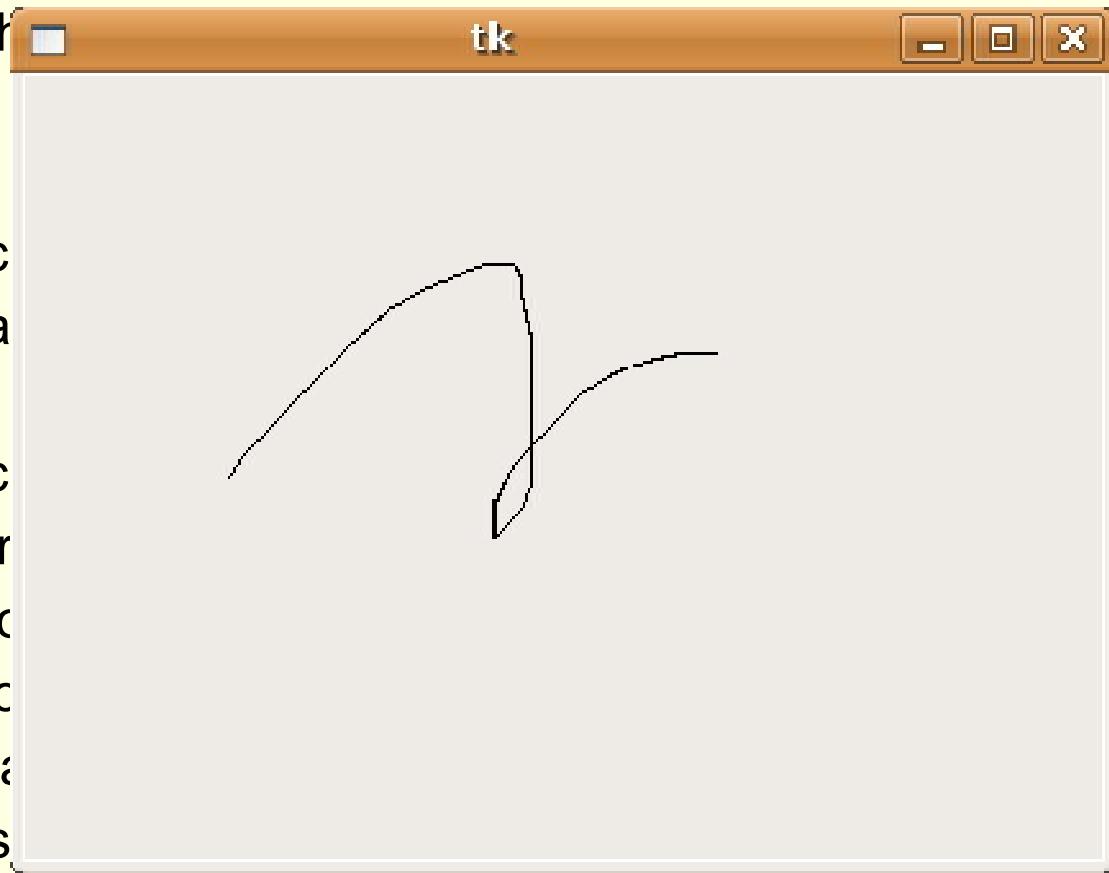
Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=510, height=510)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tags="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente",*coords+[x,y])
def fechalinha(e): c.itemconfig("corrente", fill="black")
c.bind("<Button-1>", novalinha)
c.bind("<B1-Motion>", estendelinha)
c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```



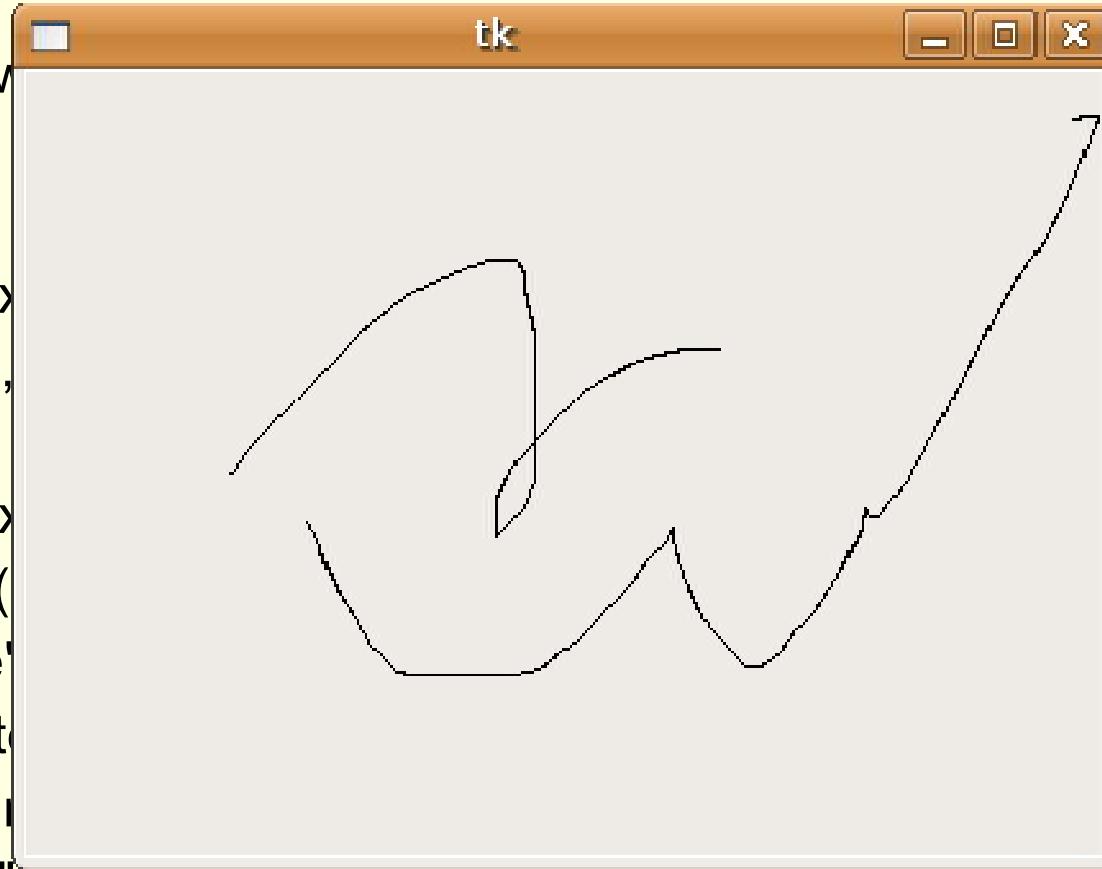
Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=400, height=300)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y,tag="corrente")
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente", *coords + [x,y])
def fechalinha(e):
    c.itemconfig("corrente", close=True)
    c.bind("<Button-1>", novalinha)
    c.bind("<B1-Motion>", estendelinha)
    c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```



Exemplo

```
from Tkinter import *
master = Tk()
c = Canvas(master, width=400, height=400)
c.pack()
def novalinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    c.create_line(x,y,x,y)
def estendelinha(e):
    x,y = c.canvasx(e.x), c.canvasy(e.y)
    coords = c.coords("corrente")
    c.coords("corrente", coords[0], coords[1], x,y)
def fechalinha():
    c.itemconfig("corrente", fill="white", outline="black")
    c.bind("<Button-1>", novalinha)
    c.bind("<B1-Motion>", estendelinha)
    c.bind("<ButtonRelease-1>", fechalinha)
c.pack()
```



Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x), c.canvasy(e.y)
    c.itemconfig(CURRENT, tags="sel")
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x), c.canvasy(e.y)
    c.move("sel",x1-x0,y1-y0)
    x0,y0=x1,y1
def deselecionalinha(e): c.itemconfig("sel", tags=())
c.bind("<Button-3>", selecionalinha)
c.bind("<B3-Motion>", movelinha)
c.bind("<ButtonRelease-3>", deselecionalinha)
c.pack()
mainloop()
```

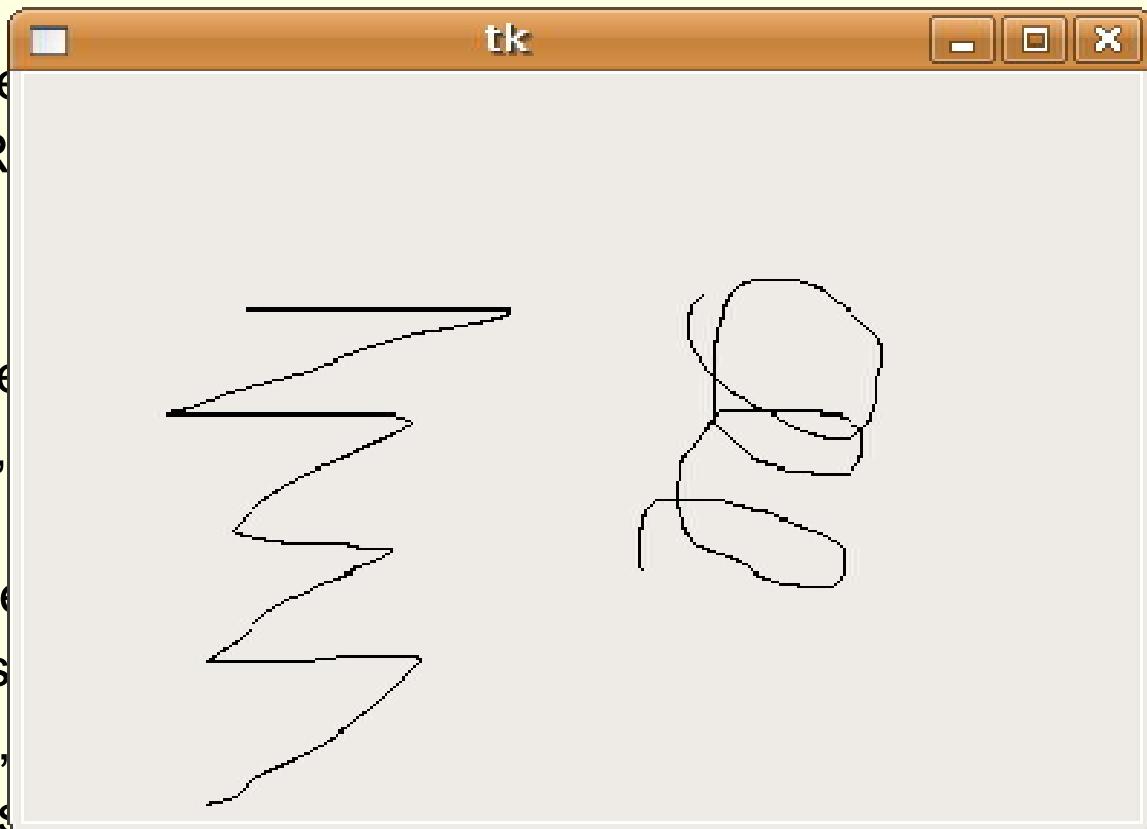
Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x),
    c.itemconfig(CURRENT
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x),
    c.move("sel",x1-x0,y1-y
    x0,y0=x1,y1
def deselecionalinha(e): c.
c.bind("<Button-3>", selec
c.bind("<B3-Motion>", mov
c.bind("<ButtonRelease-3> , aselecionalinha)
c.pack()
mainloop()
```



Exemplo

```
def selecionalinha(e):
    global x0,y0
    x0,y0 = c.canvasx(e.x),c.canvasy(e.y)
    c.itemconfig(CURRENT,strokeWidth=2)
def movelinha (e):
    global x0,y0
    x1,y1 = c.canvasx(e.x),c.canvasy(e.y)
    c.move("sel",x1-x0,y1-y0)
    x0,y0=x1,y1
def deselecionalinha(e):
    c.bind("<Button-3>", selecionalinha)
    c.bind("<B3-Motion>", movelinha)
    c.bind("<ButtonRelease-3>", deselecionalinha)
c.pack()
mainloop()
```



Scrollbar

- Barras de rolamento são usadas com outros widgets com área útil maior do que pode ser exibida na janela (Canvas, Text, Listbox, Entry)
- Uma barra de rolamento horizontal (vertical) funciona chamando o método `xview` (`yview`) do widget associado
 - Isto é feito configurando a opção `command` da barra
- Por outro lado, sempre que a visão do widget muda, a barra de rolamento precisa ser atualizada
 - Isto é feito configurando a opção `xscrollcommand` (ou `yscrollcommand`) do widget ao método `set` da barra

Exemplo

```
from Tkinter import *
lb = Listbox()
lb.pack(side=LEFT,expand=True,fill="both")
sb = Scrollbar()
sb.pack(side=RIGHT,fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(100):
    lb.insert(END,i)
```

Exemplo

```
from Tkinter import *
lb = Listbox()
lb.pack(side=LEFT,expand=True)
sb = Scrollbar()
sb.pack(side=RIGHT,fill="y")
sb.configure(command=lb.yview)
lb.configure(yscrollcommand=sb.set)
for i in range(100):
    lb.insert(END,i)
```

