Computação II (MAB 225)

Fabio Mascarenhas - 2015.1

http://www.dcc.ufrj.br/~fabiom/pythonoo

Tkinter

- Tkinter é uma biblioteca para criação de interfaces gráficas em Python

- As aplicações Tkinter usam uma estrutura de *componentes* parecida com a do Editor Gráfico: uma aplicação tem uma *janela*, onde adicionamos *controles* como botões, texto, caixas de desenho, sliders, caixas de entrada de texto...

- Os controles respondem a *eventos*: manipulação do mouse, cliques, teclas apertadas no teclado

- Na última aula, vimos como criar janelas, botões e labels com o Tkinter, hoje vamos ver mais alguns controles, e recriar a interface do Editor Gráfico

Cores

- Podemos mudar a cor dos componentes do Tkinter com dois atributos: “bg” controla a cor de fundo, e “fg” controla a cor do texto

- Essa cor é uma string com um código especial que representa os componentes vermelho, verde e azul "#ffffff" · branco

- Podemos converter uma tripla de cor no formato que estamos usando para esse código com a função abaixo:

```
def cor_tkinter(cor):
    return "#%02x%02x%02x" % (int(cor[0]*255),
                              int(cor[1]*255),
                              int(cor[2]*255))
```

Paleta de cores

- A paleta de cores do editor gráfico pode ser simplesmente um rótulo com um texto vazio e uma cor de fundo

- Vamos encapsular isso em uma classe `Paleta`:

```
class Paleta(Tkinter.Label):
    def __init__(self, janela, x, y, larg, alt, cor):
        Tkinter.Label.__init__(self, janela)
        self["text"] = ""
        self["bg"] = cor_tkinter(cor)
        self.place({ "x": x, "y": y, "width": larg, "height": alt })

    def mudou(self, cor):
        self["bg"] = cor_tkinter(cor)
```

Sliders

- O componente Scale é um slider como o que construímos para o editor
- Seus parâmetros “`from`” e “`to`” dão o intervalo do slider, e o parâmetro “`resolution`” dá o incremento (-1 se quiser simplesmente qualquer valor decimal)
- O método `get(self)` retorna o valor, e o método `set(self, valor)` muda o valor, movendo a marca do slider
- O atributo “`showvalue`” (0 ou 1) controla se o slider mostra seu valor atual ao lado dele ou não
- A largura do slider não é dada pelo método `place`, mas pelo atributo “`width`”

Encapsulando sliders

- Vamos usar a classe abaixo para encapsular sliders em uma interface mais amigável

- Como exemplo, vamos conectar três sliders a uma paleta de cores via um modelo com um campo cor

```
class Slider(Tkinter.Scale):
    def __init__(self, janela, x, y, larg, alt, inicio, fim):
        Tkinter.Scale.__init__(self, janela)
        self["from"] = inicio
        self["to"] = fim
        self["resolution"] = -1
        self["showvalue"] = 0
        self["width"] = larg
        self.set(fim)
        self.place({ "x": x, "y": y, "height": alt })

    def valor(self):
        return self.get()
```

Canvas

- Para poder recriar a interface do Editor Gráfico precisamos apenas de uma área de desenho
- O controle `Canvas` do Tkinter é uma área de desenho bem sofisticada, e vamos usar alguns de seus recursos
- Podemos desenhar no canvas com métodos como `create_oval` e `create_rectangle`, que recebem as quatro coordendas dos cantos superior esquerdo e inferior direito da figura, e um dicionário de opções
- Vamos usar duas dessas opções, “`fill`” e “`outline`”, para dar cor às figuras
- As figuras do canvas são persistentes, mas podemos usar o método `delete(Tkinter.ALL)` para apagar o canvas

Eventos

- O canvas pode responder a eventos do mouse (cliques e arrastos)
- Para isso usamos o método bind, passando o código do evento e o método que vai responder a ele
- Os códigos que nos interessam são “`<ButtonPress-1>`” (botão esquerdo apertado), “`<ButtonRelease-1>`” (botão esquerdo solto) e “`<B1-Motion>`” (botão esquerdo arrastado)
- O método que responde ao evento recebe um objeto que descreve o evento, com coordendas `x` e `y` que dão o ponto onde ele aconteceu
- Essas coordenadas são coordendas de tela, o canvas tem métodos `canvasx` e `canvasy` que convertem elas para coordendas do canvas

Encapsulando o Canvas

- Vamos encapsular o canvas em uma classe abstrata:

```
class Canvas(Tkinter.Canvas):
    def __init__(self, janela, x, y, larg, alt, cor_fundo):
        Tkinter.Canvas.__init__(self, janela)
        self["bg"] = cor_tkinter(cor_fundo)
        self.bind("<ButtonPress-1>", self.apertou)
        self.bind("<ButtonRelease-1>", self.soltou)
        self.bind("<B1-Motion>", self.arrastou)
        self.place({ "x": x, "y": y, "width": larg, "height": alt })

    def limpar(self):
        self.delete(Tkinter.ALL)

    def retangulo(self, x, y, larg, alt, cor):
        self.create_rectangle(x, y, x+larg, y+alt,
                              { "fill": cor_tkinter(cor),
                                "outline": cor_tkinter(cor) })

    def circulo(self, x, y, raio, cor):
        self.create_oval(x, y, x+raio*2, y+raio*2,
                         { "fill": cor_tkinter(cor),
                           "outline": cor_tkinter(cor) })
```

Recriando o Editor

- Podemos usar os controles do Tkinter para recriar a interface do Editor Gráfico usando o mesmo modelo que já tínhamos, com pequenas mudanças para poder conectar observadores a mudanças no estado do Editor

- As mudanças não afetam o funcionamento da interface antiga, é só não conectarmos nenhum observador, já que não eram necessários

Menus

- Ao invés de botões, os comandos do editor poderiam estar em menus na janela principal como, os menus do IDLE

- O controle `Menu` do Tkinter permite implementar menus

- Os itens do menu são adicionados pelo método add, que recebe o tipo do menu (“`command`” para uma ação, “`cascade`” para um submenu) e um dicionário de opções

- As opções que vamos usar são “`label`”, para o texto do menu, “`command`”, o método associado a uma ação, e “`menu`”, para o controle menu associado a um submenu