

11.Widgets

October 27, 2019

1 Widgets

Este notebook apresenta os seguintes tópicos

- Section 1.1 - Widgets interativos
- Section 1.2 - Exercício 14
- Section 1.3 - Widgets completos
- Section 1.4 - Exercício 15
- Section 1.5 - Exercício 16

1.1 Widgets Interativos

Finalmente, uma outra forma de interagir com o Jupyter é a partir de widgets interativos. Esses widgets podem ser usados para fazer formulários, dashboards e até mesmo variar rapidamente parâmetros de funções.

O widget a seguir interage com a função `fib`, definida no início deste notebook para variar os parâmetros dela.

```
[1]: from ipywidgets import interact

def fib(x):
    if x <= 1:
        return x
    return fib(x - 1) + fib(x - 2)

interact(fib, x=(1, 30))
```

```
interactive(children=(IntSlider(value=15, description='x', max=30, min=1), Output()), _dom_classes=)
```

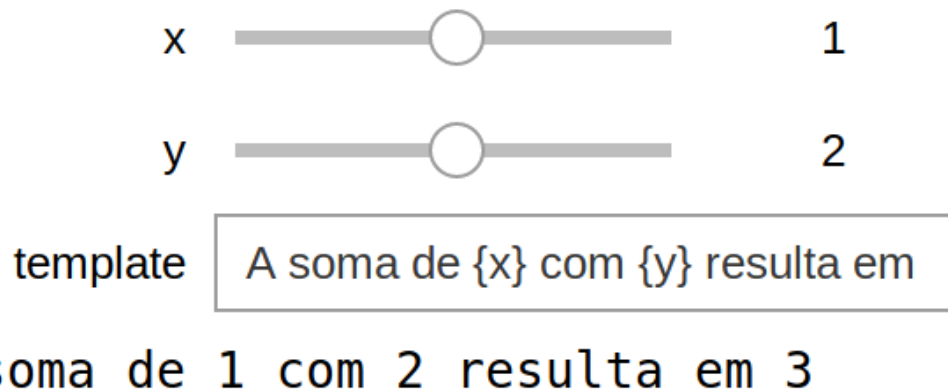
```
[1]: <function __main__.fib(x)>
```



Essa função também pode ser usada como um decorador.

```
[2]: @interact
def add(x=1, y=2, template="A soma de {x} com {y} resulta em {z}"):
    z = x + y
    print(template.format(x=x, y=y, z=z))
```

```
interactive(children=(IntSlider(value=1, description='x', max=3, min=-1), IntSlider(value=2, description='y', max=3, min=-1),
```



Podemos definir o intervalo dos valores.

```
[3]: @interact(x=(0, 100), y=(0, 100))
def add(x=1, y=2, template="A soma de {x} com {y} resulta em {z}"):
    z = x + y
    print(template.format(x=x, y=y, z=z))
```

```
interactive(children=(IntSlider(value=1, description='x'), IntSlider(value=2, description='y'))
```

x  1

y  2

template A soma de {x} com {y} resulta em

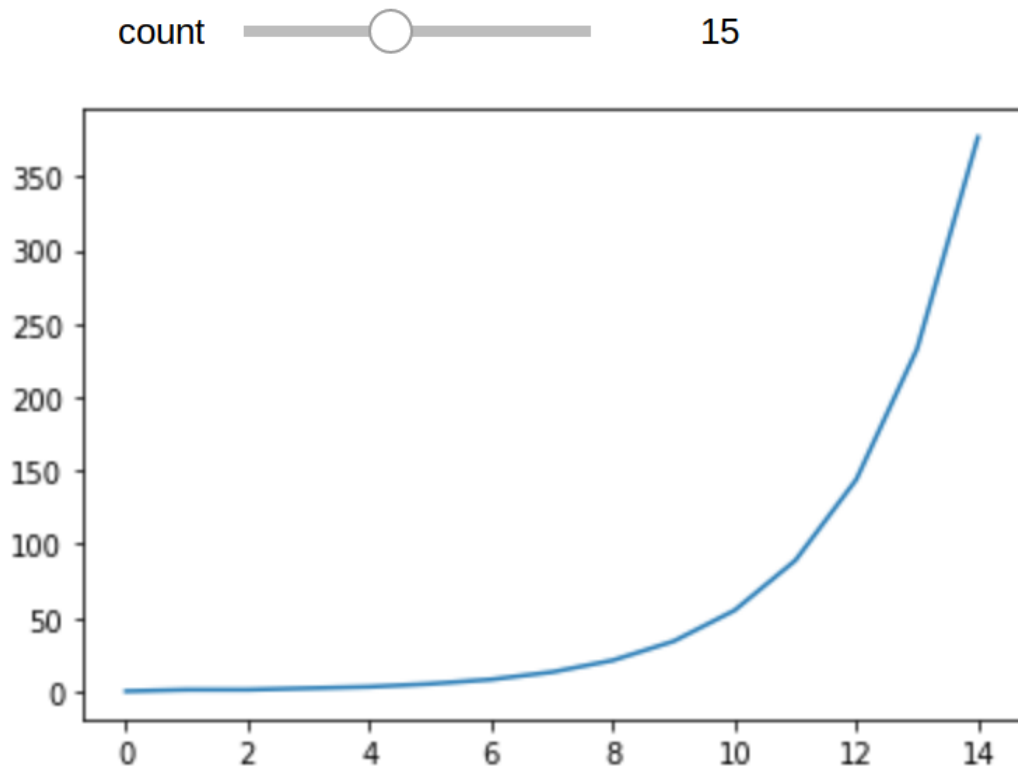
A soma de 1 com 2 resulta em 3

O widget também pode ser usado com visualizações ricas.

```
[4]: import matplotlib.pyplot as plt
      %matplotlib inline

      @interact(count=(1, 35))
      def generate_plot(count=15):
          x = range(count)
          y = [fib(n) for n in x]
          plt.plot(x, y)
```

```
interactive(children=(IntSlider(value=15, description='count', max=35, min=1), Output()), _dom=
```




Para números grandes, o uso do `interact` com a nossa implementação de fibonacci começou a não ser tão interativa.

Em funções de longa duração, a atualização automática do `interact` pode atrapalhar mais do que ajudar. Para resolver isso, podemos usar o `interact_manual`.

```
[5]: from ipywidgets import interact_manual

@interact_manual(count=(1, 35))
def generate_plot(count=15):
    x = range(count)
    y = [fib(n) for n in x]
    plt.plot(x, y)
```

```
interactive(children=(IntSlider(value=15, description='count', max=35, min=1), Button(description='Run Interact')))
```

count 

15

Run Interact

1.2 Exercício 14

Implemente uma função interativa que permita escolher um arquivo de código fonte usando um drop-down e imprima a quantidade de letras do arquivo após a escolha.

Dica: ao passar uma lista ou dicionário para o `interact`, é criado um elemento drop-down.

```
[6]: ...
```

```
[6]: Ellipsis
```

1.3 Widgets completos

O `interact` é uma simplificação do sistema de widgets para facilitar o uso em funções. Porém, quando estamos criando dashboards ou formulários mais completos, podemos usar o sistema mais completo.

A seguir, criaremos um slider que não depende de nenhuma função `interact`.

```
[7]: from ipywidgets import IntSlider

slider = IntSlider(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Test:'
)
slider
```

```
IntSlider(value=7, description='Test:', max=10)
```



Podemos acessar o valor do slider através do atributo `.value`.

```
[8]: slider.value
```

```
[8]: 7
```

Se quisermos ter o efeito do `interact` de executar alguma função ao alterar o slider, podemos definir funções de observação.

```
[9]: def add1(change):  
      if change.name == "value":  
          print(change.new + 1)  
  
      slider.observe(add1)  
      slider
```

```
IntSlider(value=7, description='Test:', max=10)
```

Test:  7

9 8 7 6 7 8

Note que verificamos o tipo da observação ao receber a mudança. Algumas mudanças no widget não ocorrem no valor e isso acaba mudando o resultado de `change.new`. Além de `change.new` e `change.name`, podemos acessar outros atributos de `change`, como `change.old`.

Note também que ao fazermos `print` nessa função, os outputs anteriores foram preservados.

Se quisermos ter um controle maior do output, podemos usar um widget específico de output.

```
[10]: from ipywidgets import Output  
  
out = Output()  
with out:  
    print("Dentro do output novo")  
print("Fora do output novo")  
out
```

Fora do output novo

Output()

Podemos usar o objeto de output para apagar o conteúdo.

```
[11]: out.clear_output()
```

Nota da versão em PDF: ao executar o `clear_output` na última célula, o resultado da célula anterior foi apagado. Portanto, não aparece no PDF estático.

Agora vamos combinar o slider com o output para gerar o efeito do `interact`.

Primeiro, precisamos limpar todos os eventos de observação que registramos no slider.

```
[12]: slider.unobserve_all()
```

Em seguida, podemos criar um novo evento que imprima dentro do objeto de output.

```
[13]: def add1(change):  
        if change.name == "value":  
            out.clear_output()  
            with out:  
                print(change.new + 1)  
  
        slider.observe(add1)
```

Por fim, criamos um widget que combine os dois no mesmo lugar usando VBox.

```
[14]: from ipywidgets import VBox  
  
        VBox([slider, out])
```

```
VBox(children=(IntSlider(value=7, description='Test:', max=10), Output()))
```

Test:  7

8

1.4 Exercício 15

Use o widget Button para simular o `interact_manual`. Esse widget possui um método `on_click` para definir funções de callback.

```
[ ]: from ipywidgets import Button  
  
        button = Button(description="run")  
        ...
```

1.5 Exercício 16

Faça um widget que imprima a soma acumulada de todas as suas execuções.

O objetivo desse exercício é pensar em como criar widgets com estados que continuem existindo além de uma execução da função observadora ou do `interact`.

```
[ ]: class AccWidget:  
        ...
```

```
[ ]: widget = AccWidget()  
      widget.view
```

Chegamos ao fim da apresentação principal do minicurso, mas não é o fim do conteúdo preparado. Temos mais dois notebooks que listam e explicam as magics do IPython e os ipywidgets disponíveis:

- [Extra/Lista.Magics.ipynb](#)
- [Extra/Lista.Widgets.ipynb](#)