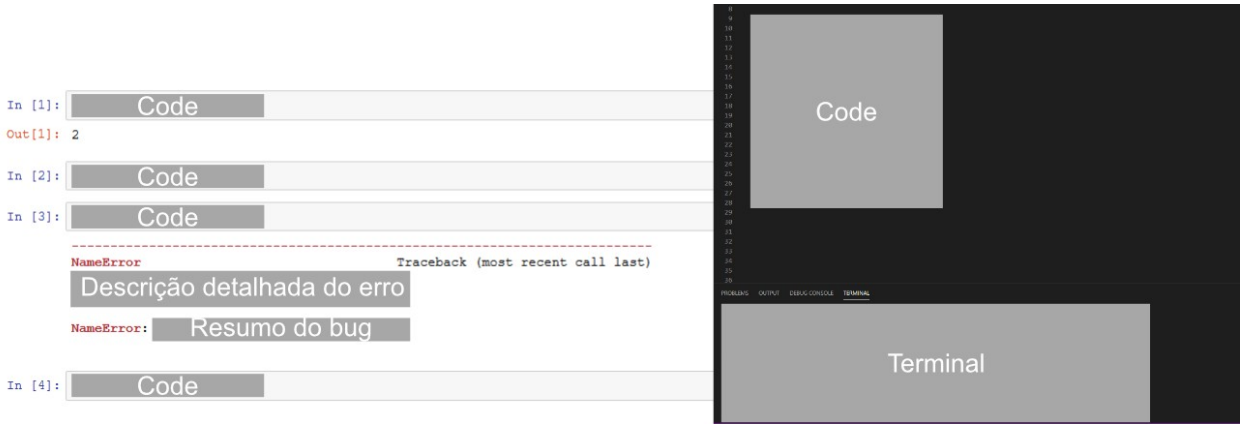
Introdução ao Python

Overview — Se habituando com o Jupyter

Como falado anteriormente, uma das principais vantagens do Jupyter notebook é a possibilidade de quebrar o código em células:



à esquerda, um Jupyter notebook. À direita um editor de código comum.

Como você pode ver na imagem anterior, diferente de quando escrevemos um bloco inteiro em um editor comum, no Jupyter é mais fácil isolarmos bugs em partes específicas do código. E não se preocupe com os erros, você encontrará muitos deles enquanto estiver programando (e, mesmo que seja experiente, eles ainda vão acontecer).

E também não se preocupe em decorar todas as respostas: se encontrar algum bug ou estiver procurando uma solução específica é só seguir este breve tutorial: https://bit.ly/2yPUE5M.

Além disso, podemos também observar o output de uma célula, sem necessidade de escrever um comando específico para que algo apareça no terminal:

Input x output

Observe a imagem acima. Um ponto importante para se ter em mente é que a lógica do seu código vai sempre respeitar a ordem em que as células forem rodadas. Eu posso, por exemplo, rodar a última célula, depois a primeira e por aí vai. Uma maneira de você observar a ordem que o seu código está respeitando é observar o número entre colchetes *In [**#**]*.

Bibliotecas

Imagine as bibliotecas como conjuntos ou pacotes de códigos e métodos criados por outras pessoas que programam e disponibilizados para nosso uso. Imagine ter que escrever um código do zero, utilizando apenas Python, para trabalhar com gráficos dinâmicos, interfaces ou com modelos estatísticos complexos, levaríamos muito mais tempo para construir as nossas soluções do que se pudéssemos, simplesmente, importar estes métodos prontos e apenas aplicá-los.

Ao longo do curso você vai se deparar com diversas dessas bibliotecas, mas o que você precisa saber neste momento é que existem muitas bibliotecas que já

vêm instaladas com o Anaconda e, caso precise de alguma outra biblioteca, basta escrever uma única vez no prompt do Anaconda:

```
pip install [nome da biblioteca]
```

Ou o seguinte em uma das células do Jupyter notebook:

```
!pip install [nome da biblioteca]
```

Uma vez dado este comando, a biblioteca já estará instalada no seu computador e não precisamos repetir esta operação.

Alguns comandos úteis:

- Desinstalar uma biblioteca: (!) `pip uninstall [package_name]`
- Atualizar a versão de uma biblioteca (sim elas têm atualizações o tempo todo): (!) `pip install [package_name] --upgrade`

> Tenha em mente que cada biblioteca possui a sua lógica, portanto existem maneiras específicas de trabalhar com cada uma delas, algo que você vai pegar ao longo do curso.

Importante: sempre que instalar uma biblioteca da maneira que ensinei acima, ela será instalada na sua versão mais atual. Tente não alterar as versões (a menos que necessário) porque, dependendo da biblioteca, ela pode usar outra como apoio e alterar as suas versões pode criar alguns conflitos.

Comentários

Uma maneira de você organizar bem o seu código é adicionando comentários. Comentários são como anotações no código e o Python não vai tentar processá-los. Mas eles são muito úteis se quisermos escrever algo para nos lembrar depois ou também caso outra pessoa veja seu código e precise de anotações para entender melhor algum aspecto.

É muito simples escrever um comentário em Python: basta colocar o caractere “#” no começo da linha. Exemplo:

```
# este é um comentário.
```

Preciso de ajuda!

Acredite em mim, em algum momento você pode se sentir assim 👆🏻

Mas não se preocupe! Para (quase) tudo existe uma solução rápida em tutoriais da internet:

- Cada biblioteca possui uma documentação caso você queira saber mais sobre alguma em específico. Basta procurar na internet. Um exemplo de documentação de uma biblioteca bem comum: https://matplotlib.org/3.2.1/contents.html
- Uma outra maneira de saber mais sobre uma biblioteca ou um método específico é usando o comando help.
- Tenho um bug no meu código: https://bit.ly/2yPUE5M.

Lembrem-se: Python possui uma comunidade incrível. Existe muita troca de informação, muita ajuda e a soluções prontas para muito bug. Se você está com alguma dificuldade, a chance de alguém já ter tido esse mesmo problema é muito grande. Existe alguns sites que concentram esse tipo de resolução, use e abuse 👉🏻 https://pt.stackoverflow.com/.

E claro, também conte com a nossa equipe para te dar uma força quando precisar :)

Começando a programar

Bom, agora que você já tem um overview, podemos começar. Então abra seu Jupyter notebook e vamos lá! É importante que você teste os conceitos passados aqui no seu notebook para absorver melhor o conteúdo.

Variáveis

As variáveis são maneiras de guardarmos informações e são utilizadas para armazenar valores em memória, nos permitindo gravar e ler esses dados com facilidade a partir de um nome definido por nós.

Para defini-las é simples: basta escrever o nome da variável seguida de "**=**". E tudo que estiver do outro lado do "=" será armazenado ali dentro. Posteriormente, podemos resgatar estas variáveis e utilizá-las em cálculos mais complexos. Alguns exemplos:

```
x = 2
y = 3
z = x + y + x + 5
```

Detalhe importante: essas variáeis podem ser redefinidas a qualquer momento.

```
x = 30
x = 10
```

Tipos básicos de dados

O primeiro tipo de informação são os dados compostos por caracteres numéricos (algarismo). Os números são divididos em 2 partes:

inteiros - chamados de integer

intponto flutuante - chamado de float ou double

Integer: números inteiros

Os números inteiros são chamados de integer ou int .

As operações básicas que podemos fazer aqui são: soma (+), subtração (-), multiplicação (*), divisão (/) e exponenciação (**).
Importante: tanto na divisão quanto na exponenciação com números negativos, o resultado será um float.

Exemplos: 1, 2, 4, 60, 120

Float: números com casas decimais

O float possui as mesmas operações básicas do integer. E uma função simples que podemos usar em um número neste formato é o round(#), que arredonda o float e o transforma em um integer.

Outras formas de conversão simples são

`int(float)` → converte o float em integer

`float(integer)` float(int) → converte o integer em float.

Exemplos: 1.5, 2.2, 4.0, 60.3, 120.65

Detalhe: em Python, a vírgula é utilizada para separar elementos e o ponto é utilizado para separar as casas decimais.

String: texto

Na programação, String representa um conjunto de caracteres disposto numa determinada ordem.

str(float ou integer) → converte o número em uma string

As strings normalmente são escritas entre aspas simples ou duplas e podemos fazer algumas operações com elas. Alguns exemplos:

- **Concatenar**

string + string = (strings concatenadas)

```
"a" + "b"
```

resultado:

```
"ab"
```

- **Imprimir**

O comando print(“texto”) imprime no resultado da célula ou no terminal (caso esteja utilizando um) o texto em específico. Além disto, o comando print nos permite alguns usos interessantes.

Vamos imaginar a seguinte situação: eu realizei um cálculo dentro do meu código e o armazenei na variável ***a***.

Caso eu queira imprimir um texto + o resultado do meu cálculo, poderia escrever o seguinte código:

```
b = 40
c = 2.0
a = round(40+2.0)
print(“somando {} e {} o cálculo é {}”.format(b,round(c),a))
```

resultado:

```
somando 40 e 2 o cálculo é 42
```

O que está acontecendo aqui é:

> Primeiro definimos as variáveis - note que o método `round()` é utilizado para arredondar o resultado da soma de um integer e um float. Depois utilizamos o comando `print()` para imprimir uma string, em que possuímos algumas chaves `{}.` Elas marcam os pontos da string onde vamos inserir algum conteúdo externo.
>
> Então, utilizamos o comando `.format()` para passar as informações que vão substituir as chaves `{}`. Dentro do `.format()` temos a variável `b`, que possui o valor de 40. Em seguida temos a variável `c` arredondada com o método `round()` e por último temos a variável `a`, que possui o resultado do nosso cálculo.
>
> Note que as informações externas da string estão separadas por vírgula e aparecem dentro do `.format()` na mesma ordem que devem aparecer na string substituindo as chaves `{}`.

Se quiser saber mais sobre o `round()` e o `.format()`, seguem alguns links:

Python round() Function, da W3 Schools

Using % and .format() for great good! da PyFormat

- **Recortar partes da string**

Podemos também selecionar partes específicas da string. Imagine a palavra "gramado". Logo após a string, podemos adicionar um número entre colchetes, que representa um index (sempre começando em 0), ou um índice, que corresponde a alguma letra específica.

Vamos a um exemplo mais palpável:

```
string - g r a m a d o
índice - 0 1 2 3 4 5 6
```

Portanto, se eu escrever `"gramado"[2]` o resultado será `"a"` .

Podemos também fazer recortes maiores seguindo o seguinte formato `"string"[primeiro index : segundo index]`. Por exemplo, se eu escrever `"gramado"[0:5]` o resultado será `"grama"`

Note que eu queria retornar o texto apenas do índice 0 ao 4, então por que tive que escrever 5 no segundo index?

Isso ocorre por que o segundo indica até onde será o corte, mas sem considerar aquele último caractere. Pois é.. 😅

Faça alguns testes no seu notebook! O que acontece se você escrever `"gramado"[1:2]`?

Lists

Usado para agrupar um conjunto de elementos.

Saber trabalhar bem com listas talvez seja uma das skills mais importantes neste começo. Você pode entender uma lista como uma coleção de elementos que podem ser strings, integers, floats, outras listas e diversos outros tipos de elementos.

Para definir uma lista, basta separar os elementos por vírgulas e colocá-los entre colchetes `[]`. Exemplo:

```
lista = [elemento1, elemento2, elemento3,..., elementoN]
```

A lista não precisa conter um único tipo de elemento, exemplo:

```
lista = [["texto", 1.2], 5, "texto2",[["texto3"],0.4], [x]]
```

> **Para você entender melhor, gravei um vídeo explicando a célula anterior (1 min).**

Como você pode ver no exemplo anterior, criei algumas listas dentro de listas, criando alguns "níveis de profundidade" dentro da lista original.

Podemos navegar nesta lista de forma muito simples. Basta escrevermos colchetes com um número de index referente ao elemento que queremos recuperar da lista, exemplo:

```
lista = [["texto", 1.2], 5, "texto2",[["texto3"],0.4], [x]]
index >         0        1      2            3          4
```

Portanto, se escrevermos `lista[0]` teremos como resultado `["texto", 1.2]`

E se quiséssemos recuperar a string "texto3"? Segue um passo a passo:

1. Primeiro temos que achegar ao elemento de índice 3: `lista[3]` o que retornará `[["texto3"], 0.4].`
2. Em seguida, temos que chegar no primeiro elemento desta nova lista. Portanto adicionamos mais um índice: `lista[3][0]`, o que nos retornará `["texto3"]` (que é mais uma lista com um único elemento).
3. Por fim, para chegar à "texto3", devemos adicionar um último índice: `lista[3][0][0]` e agora sim, chegamos à "texto3".

Podemos aplicar aqui mesma lógica que utilizamos para retornar partes de uma string. Portanto, considere a lista:

```
lista = ['a', 'b', 'c', 'd', 'e']
index >   0    1    2    3    4
```

Portanto, caso escrevermos `lista[1:3]` , o resultante será `['b', 'c']`. Mais alguns exemplos:

`lista[-1]` → `['e']`

`lista[:3]` → `['a', 'b', 'c']`

`lista[3:]` → `['d', 'e']`

Também podemos realizar algumas operações com as listas. Seguem alguns exemplos:

Para alterar um elemento da lista, basta escrever `lista[index]=novoelemento`, exemplo, se escrevêssemos `lista[0]='z'` o resultado seria `['z', 'b', 'c', 'd', 'e']`.

Se eu quiser fundir duas listas basta utilizar o sinal de `+`. Por exemplo `[1,2,3] + lista` resultaria em `[1, 2, 3, 'z', 'b', 'c', 'd', 'e']`.

Note que, neste caso, mesmo que o Anaconda retorne este resultado, não quer dizer que a lista original armazenada na variável `lista` foi alterada. Se este for o intuito deveríamos redefinir a nossa variável.

Exemplo:

```
lista = [1, 2, 3] + lista
```

Podemos também deletar um item da lista. Este caso também é bem simples, bastando utilizar a seguinte sintaxe: `del(lista[index do elemento a ser removido])`.

Então, neste caso, se escrevermos `del(lista[:4])` o resultado seria `['b', 'c', 'd', 'e']`.

Booleans

Booleans normalmente são resultantes de operações de comparação e assumem os valores de `True` ou `False` (atenção que a primeira letra é sempre maiúscula).

Exemplo de uma operação lógica em Python:

Igual a:

Para verificarmos se dois valores são iguais utilizamos `==`.

```
a = 1
b = 2
```

Portanto, neste caso, `a==b` resultaria em `False` e `a==1` resultaria em `True`

Outros operadores de comparação são:

`!=` ou `<>` — Diferente de

`>` — Maior que

`<` — Menor que

`>=` — Maior ou igual a

`<=` — Menor ou igual a

Métodos

No Python, temos algumas funções nativas, algumas que já vimos anteriormente e várias outras. Exemplos:

```
max( ) -> retorna o valor máximo de uma lista
len( ) -> retorna a quantidade de itens em uma lista ou a quantidade
de caracteres em ums string
round( ) -> arredonda números decimais
str( ) -> converte os dados em string
sum (lista) -> soma todos os números de uma lista
```

Cada um com um uso diferente e recebendo argumentos específicos. Quando falamos de métodos, entenda como funções "pertencentes" a um objeto python (listas, variáveis, strings, etc...). Um exemplo de método que já utilizamos por aqui é o próprio `.format()`.

Para utilizar os métodos basta digitar: `objeto_python.método()`.

Note que cada tipo de objeto possui métodos específicos que recebem argumentos específicos. Vale mencionar também que cada biblioteca possui suas próprias funções e métodos. *Acredito que neste ponto você já deva estar imaginando o quanto vai ter que estudar daqui pra frente, né?*

Alguns exemplos de métodos básicos:

```
string.upper() -> faz com que todas as letras fiquem maiúsculas
string.count("caractere") -> conta a ocorrência daqueles caracteres
na string
lista.count(item) -> conta a ocorrência daquele item na lista
lista.index(item) -> retorna o index correspondente ao item na lista
lista.append(item) -> adiciona o item à lista
lista.remove(item) -> remove o item da lista
lista.reverse() -> inverte a ordem dos itens na lista
```

Faça alguns testes com esses métodos no seu notebook 😄

Boa! Parabéns por chegar até aqui!

Mas não pense que acabou! Tenho um desafio para você:

Desafio!

Abaixo você vai encontrar um link para um arquivo que contém uma série de inputs e outputs. Pense nos inputs como os dados iniciais e os outputs como o resultado do seu código.

Sua tarefa é encontrar o caminho para transformar o input no output.

>> Baixe o arquivo aqui. <<

Minha sugestão é que você tente fazer os exercícios sem olhar para as respostas mas, caso trave ou queira comparar a resposta certa com a sua solução, no arquivo você também consegue encontrar o gabarito, basta seguir as instruções.

🌟 Good luck, champs.

Recap rápido

Bom, aqui fizemos um overview a respeito da linguagem e da ferramenta que você irá utilizar para desenvolver as suas soluções ao longo do curso. Além disso passamos pelos principais tipos de dados, alguns métodos e algumas funções.

No próximo texto vamos entrar ainda mais a fundo!