Tipos de objetos

- Criamos muitas variáveis até agora. Você lembra o tipo de cada uma? Para saber o tipo de um objeto ou variável, usamos a função type():

```
>>> x = 1
>>> type(x)
<class 'int’>

>>> y = 2.3
>>> type(y)
<class 'float’>

>>> type('Python')
<class 'str'>
>>> type(True)
<class 'bool'>
```

Tipos de objetos

- Python vem com alguns tipos básicos de objetos, dentre eles:

  bool: verdadeiro ou falso.

  int: números inteiros.

  float: números reais.

  complex: números complexos.

  str: strings (textos).

  list: listas. (Estudaremos em breve o que são.)

  dict: dicionários. (Estudaremos em breve o que são.)

Buscando ajuda rapidamente

- Está com dúvida em alguma coisa? Use a função help() e depois digite o que você busca.

```
>>> help()
```

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

help>

You are now leaving help and returning to the Python interpreter.

If you want to ask for help on a particular object directly from the interpreter, you can type "help(object)". Executing "help('string')" has the same effect as typing a particular string at the help> prompt.

Buscando ajuda rapidamente

- E para buscar ajuda em uma coisa específica?

```
>>> help(len)
```

Help on built-in function len in module builtins:

len(obj, /)

Return the number of items in a container.

Para sair do ambiente de ajuda, pressione a tecla q.

Listando variáveis criadas

- Em algum momento durante o seu código você pode querer saber quais variáveis já foram declaradas, ou até mesmo o valor atual delas. Podemos listar todas as variáveis declaradas no código usando o comando **dir().** Veja um exemplo

```
>>> a = 1
>>> b = 2
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__
↪', '__spec__', 'a', 'b']
```

Veja que nossas variáveis declaradas aparecem no final do resultado de dir(). Não se assuste com os outros elementos que aparecem nesse resultado. Essas variáveis são criadas e usadas pelo próprio Python, e não são importantes nesse momento.

Strings (sequência de caracteres)

- Strings são tipos que armazenam uma sequência de caracteres:

```
>>> "Texto bonito"
'Texto bonito'
>>> "Texto com acentos de cedilhas: hoje é dia de caça!"
'Texto com acentos de cedilhas: hoje é dia de caça!'
```

- As strings aceitam aspas simples também:

```
>>> nome = 'Silvio Santos'
>>> nome
'Silvio Santos'
```

- **Nota**: Strings aceitam os dois tipos de aspas, desde que seja consistente. Se começou com uma, termine com aquela!

Strings (sequência de caracteres)

- Também é possível fazer algumas operações com strings:

```
>>> nome * 3
'Silvio SantosSilvio SantosSilvio Santos'
>>> nome * 3.14
Traceback (most recent call last):
    ...
TypeError: can't multiply sequence by non-int of type 'float'
```

```
>>> canto1 = 'vem aí, '
>>> canto2 = 'lá '
```

```
>>> nome + ' ' + canto1 + canto2 * 6 + '!!'
'Silvio Santos vem aí, lá lá lá lá lá lá !!'
```

Strings (sequência de caracteres)

- Para strings em várias linhas, utilize 3 aspas:

```
>>> string_grande = '''Aqui consigo inserir um textão com várias linhas, posso␣
↪iniciar em uma...
... e posso continuar em outra
... e em outra
... e mais uma
... e acabou.'''
>>> string_grande
'Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...\ne posso␣
↪continuar em outra\ne em outra\ne mais uma\ne acabou.'
>>> print(string_grande)
Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...
e posso continuar em outra
e em outra
e mais uma
e acabou.
```

Strings (sequência de caracteres)

- Para strings em várias linhas, utilize 3 aspas:

```
>>> string_grande = '''Aqui consigo inserir um textão com várias linhas, posso␣
↪iniciar em uma...
... e posso continuar em outra
... e em outra
... e mais uma
... e acabou.'''
>>> string_grande
'Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...\ne posso␣
↪continuar em outra\ne em outra\ne mais uma\ne acabou.'
>>> print(string_grande)
Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...
e posso continuar em outra
e em outra
e mais uma
e acabou.
```

Tamanho

- A função embutida len() nos permite, entre outras coisas, saber o tamanho de uma string:

```
>>> len('Abracadabra')
11
>>> palavras = 'Faz um pull request lá'
>>> len(palavras)
22
```

Índices

Como visto anteriormente, o método len() pode ser utilizado para obter o tamanho de estruturas, sejam elas strings,listas, etc. Esse tamanho representa a quantidade de elementos na estrutura.

Para obter somente um caractere de dentro dessas estruturas, deve-se utilizar o acesso por índices, no qual o índice entre colchetes []representa a posição do elemento que se deseja acessar.

Nota: Os índices começam em zero.

Índices

Como visto anteriormente, o método len() pode ser utilizado para obter o tamanho de estruturas, sejam elas strings,listas, etc. Esse tamanho representa a quantidade de elementos na estrutura.

Para obter somente um caractere de dentro dessas estruturas, deve-se utilizar o acesso por índices, no qual o índice entre colchetes []representa a posição do elemento que se deseja acessar.

Nota: Os índices começam em zero.

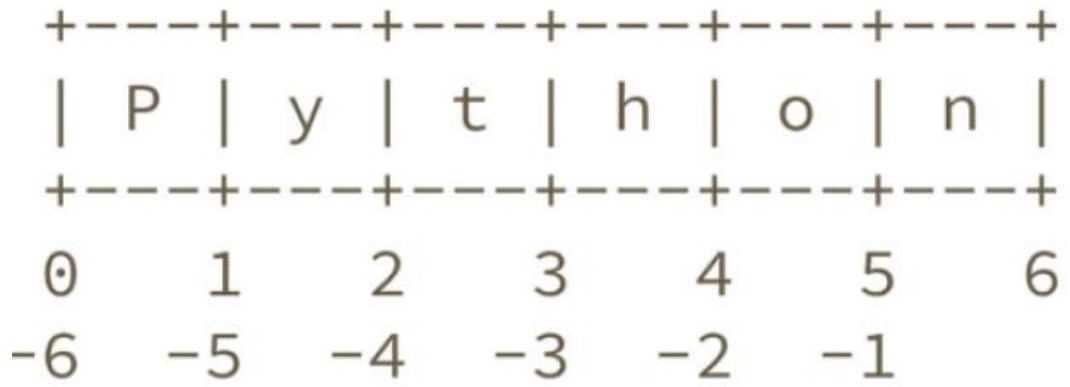
Índices

Como visto anteriormente, o método len() pode ser utilizado para obter o tamanho de estruturas, sejam elas strings,listas, etc. Esse tamanho representa a quantidade de elementos na estrutura.

Para obter somente um caractere de dentro dessas estruturas, deve-se utilizar o acesso por índices, no qual o índice entre colchetes []representa a posição do elemento que se deseja acessar.

Nota: Os índices começam em zero.

Índices




```
>>> palavra = 'Python'
>>> palavra[0] # primeira
'P'
>>> palavra[5] # última
'n'
```

Índices negativos correspondem à percorrer a estrutura (string, lista, . . .) na ordem reversa:

```
>>> palavra[-1] # última também
'n'
>>> palavra[-3] # terceira de tras pra frente
'h'
```

Fatias

- Se, ao invés de obter apenas um elemento de uma estrutura (string, lista, . . .), deseja-se obter múltiplos elementos,deve-se utilizar slicing (fatiamento). No lugar de colocar o índice do elemento entre chaves, deve-se colocar o índice do primeiro elemento, dois pontos (:) e o próximo índice do último elemento desejado, tudo entre colchetes.

Fatias

- Por exemplo:

```
>>> frase = "Aprender Python é muito divertido!"
>>> frase[0]
'A'
>>> frase[5]
'd'
>>> frase[0:5] # do zero até o antes do 5
'Apren'
>>> frase[:] # tudo!
'Aprender Python é muito divertido!'
>>> frase
'Aprender Python é muito divertido!'
>>> frase[6:] # Se omitido o segundo índice significa 'obter até o final'
'er Python é muito divertido!'
>>> frase[:6] # se omitido o primeiro indice, significa 'obter desde o começo'
'Aprend'
>>> frase[2:-3] # funciona com números negativos também
'render Python é muito diverti'
>>> frase[0:-5]
'Aprender Python é muito diver'
>>> frase[0:-6]
'Aprender Python é muito dive'
>>> frase[0:-7]
'Aprender Python é muito div'
>>> frase[2:-2]
'render Python é muito divertid'
```

Formatação de strings

- A formatação de string nos permite criar frases dinâmicas, utilizando valores de quaisquer variáveis desejadas. Por exemplo:

```
>>> nome = input('Digite seu nome ')
Digite seu nome Silvio Santos
>>> nome
'Silvio Santos'
>>> frase = 'Olá, {}'.format(nome)
>>> frase
'Olá, Silvio Santos'
```

Formatação de strings

É possível formatar uma quantidade arbitrária de valores:

```
>>> '{} x {} = {}'.format(7, 6, 7 * 6)
'7 x 6 = 42'
```

```
>>> palavra = 'Python'
>>> numero = 10
>>> booleano = False
>>> '{} é {}. E as outras linguagens? {}'.format(palavra, numero, booleano)
'Python é 10. E as outras linguagens? False'
```

Além disso, também é possível usar nomes para identificar quais valores serão substituídos:

```
>>> '{a}, {a}, {a}. {b}, {b}, {b}'.format(a='oi', b='tchau')
'oi, oi, oi. tchau, tchau, tchau'
```

Formatação de strings

- Alternativa ao .format()

Uma maneira mais recente de formatar strings foi introduzida a partir da versão 3.6 do Python: PEP 498 – Literal String Interpolation, carinhosamente conhecida como fstrings e funciona da seguinte forma:

```
>>> nome = 'Silvio'
>>> f'Olá, {name}.'
'Olá, Silvio.'
```

É também possível fazer operações:

```
>>> f'4654 * 321 é {4654 * 321}'
'4654 * 321 é 1493934'
```

Separação de Strings

Se tivermos a frase Sílvio Santos vem aí, oleoleolá! e quisermos separar cada palavra, como fazer? Pode-se usar o fatiamento:

```
>>> frase = "Sílvio Santos vem aí, oleoleolá!"
>>> frase[:6]
'Sílvio'
>>> frase[7:13]
'Santos'
>>> frase[14:17]
```

Mas também podemos usar a função `split()`:

```
>>> frase.split()
['Sílvio', 'Santos', 'vem', 'aí,', 'oleoleolá!']
```

Lendo valores do teclado

Em Python também é possível ler do teclado as informações digitadas pelo usuário. E isso é feito por meio da função embutida **input()** da seguinte forma:

```
>>> valor_lido = input("digite um valor: ")
digite um valor: 10

>>> type(valor_lido)  # deve-se notar que o valor lido é SEMPRE do tipo string
<class 'str'>
```

A função `input()` «termina» de ser executada quando pressionamos *enter*.

Nota: O valor lido é sempre do tipo string.

Lendo valores do teclado

Mas, como realizar operações com os valores lidos?

```
>>> valor_lido + 10  # para trabalhar com esse valor, é preciso converter para o tipo␣
↪correto
Traceback (most recent call last):
    ...
TypeError: must be str, not int
```

Para poder fazer isso pode-se usar os operadores `int()` e `float()`, que converte o valor lido para o tipo de dado esperado:

```
>>> valor_lido = int(input("digite um valor inteiro: "))
digite um valor inteiro: 10

>>> type(valor_lido)
<class 'int'>

>>> valor_lido + 10
20
```

Lendo valores do teclado

Mas, como realizar operações com os valores lidos?

```
>>> valor_lido + 10  # para trabalhar com esse valor, é preciso converter para o tipo␣
↪correto
Traceback (most recent call last):
    ...
TypeError: must be str, not int
```

Para poder fazer isso pode-se usar os operadores `int()` e `float()`, que converte o valor lido para o tipo de dado esperado:

```
>>> valor_lido = int(input("digite um valor inteiro: "))
digite um valor inteiro: 10

>>> type(valor_lido)
<class 'int'>

>>> valor_lido + 10
20
```