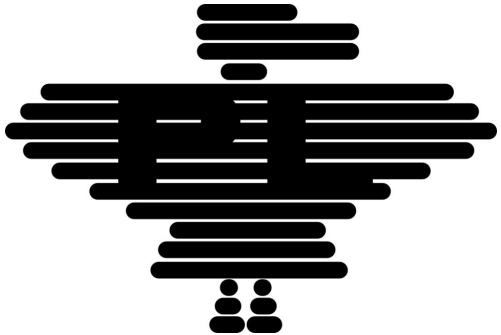
Iniciando com a Lingutagem Python

Emanoeli Madalosso
Jean Massucatto



Por que Python?

- É uma linguagem de sintaxe limpa, fácil de entender;
- Permite focar mais esforços no problema a ser resolvido do que aprendendo a linguagem;
- É um linguagem de propósito geral:
  - Aplicações web;
  - Aplicações científicas;
  - Aplicações desktop.
- Grande suporte da comunidade.

Quem usa Python?

- Nasa;
- Google;
- Yahoo;
- Disney;
- IBM.

Um pouco de história...

- Iniciada em 1989, por Guido Van Rossum, baseada na linguagem ABC;
- O nome vem da série Monty Python’s Flying Circus, da qual Guido era fã;

Zen of Python

- Bonito é melhor que feio;
- Explícito é melhor que implícito;
- Simples é melhor que complexo;
- Complexo é melhor que complicado;
- Plano é melhor que alinhado;
- ...

Vamos começar!

Identação

- “:” em vez de “{“ para indicar um bloco de código:

```
x = 5
if x > 0:
   print(True)
else:
   print(False)
```

Comentários

- “#” indica um comentário de uma linha;
- Aspas triplas indicam um comentário de múltiplas linhas.

```
# este é um comentário curto

"""
este
é
um
comentário
longo
"""
```

Tipos de dados

- Números:
  - Declaração:

```
# inteiro
x = 5

# float
x = 5.1

# complexo
x = 1+2j
```

  - Operações básicas:

```
x = 3 + 2
x = 3 - 2
x = 3 * 2
x = 3 / 2
x = 3 % 2
x = 3 ** 2
```

  - Operações mais avançadas:
    - Módulo math:

```
import math

x = math.sqrt(25)
x = math.pi
x = math.cos(90)
x =
math.factorial(6)
```

Tipos de dados

- Strings:
  - Sequência de caracteres unicode;

```
x = 'Esta é uma string curta'

x = '''Esta é uma string
com várias linhas'''
```

Tipos de dados

- Strings:
  - Operações básicas:

```
x = 'Python'

# acessar uma posição
print(x[1])
'y'

# slicing
print(x[1:3])
'yt'

# última letra
print(x[-1])
'n'
```

```
# concatenação
linguagem = 'Python'
versao = '3.5'
print(linguagem + ' ' + versao)
'Python 3.5'

# verificar se uma substring está
contida na string
'Py' in 'Python'
True

'Pi' in 'Python'
False
```

Tipos de dados

- Strings:
  - Métodos comuns:
    - ***lower()***: converte para minúsculas;
    - ***upper()***: converte para maiúsculas;
    - ***join()***: junta uma lista de strings;
    - ***split()***: quebra a string de acordo com um caracter ou sequência de caracteres;
    - ***find()***: encontra o índice onde um caractere ou uma sequência de caracteres está localizado;
    - ***replace()***: substitui um caractere ou uma sequência de caracteres por um outro caractere ou sequência de caracteres.

Tipos de dados

- Listas:
  - São uma sequência **ordenada** de itens;
  - Um dos tipos de dados mais utilizados na linguagem Python;
  - Os elementos de uma lista não precisam ser todos do mesmo tipo;

```
lista = [5, 5.1, 'Aprendendo Python']
```

  - O que significa dizer que a sequência é **ordenada**?
    - A ordem dos itens tem uma relevância. Podemos acessar um de seus elementos a partir de um **índice**:

```
primeiro_item = lista[0]
```

Tipos de dados

- Listas:
  - Operações básicas semelhante a strings:
    - Acesso de elementos por índices;
    - Slicing;
    - Concatenação;
    - Operador *‘in’* para verificar se um item está na lista;

Tipos de dados

- Listas:
  - Métodos comuns:
    - ***insert()***: adiciona elemento na lista em um índice definido;
    - ***remove()***: remove um item da lista;
    - ***append()***: adiciona um elemento no fim da lista;
    - ***clear()***: limpa a lista;
    - ***count()***: retorna a quantidade de elementos;
    - ***reverse()***: inverte a lista;
    - ***sort()***: ordena a lista;
    - ***index()***: retorna o item do primeiro elemento correspondente;

Tipos de dados

- Tuplas:
  - São uma sequência ordenada de itens, assim como a lista, representadas por “( )” em vez de “[ ]”

```
tupla = (5, 5.1, 'Aprendendo Python')
```

  - Navegação por índices, slicing e concatenação semelhante a lista;
  - Qual a diferença?
    - As **listas** são objetos **mutáveis**.
    - As **tuplas** são objetos **imutáveis**.
    - Mas o que isto significa?

Tipos de dados

- Tipos mutáveis e imutáveis:

```
lista = [1, 2, 3]
lista[1] = '2'
print(lista)
[1, '2', 3]

tupla = (1, 2, 3)
tupla[1] = '2'
TypeError: 'tuple' object does not support item assignment
```

  - Da mesma forma, não consigo remover ou adicionar um elemento;

Tipos de dados

- Sets:
  - São uma sequência **não ordenada** de itens;
  - Também pode conter tipos diferentes de dados, como a lista e a tupla;
  - Declaração é feita com chaves;

```
x = {1, 2, 2, 3, 3, 3}
```

  - O que significa dizer que é uma sequência **não ordenada**?
    - Não é possível acessar um elemento através de um índice, uma vez que a ordem dos itens é irrelevante.
  - Para o que serve?

Tipos de dados

- Sets:
  - Operações com conjuntos: união, intersecção; diferença, etc.
    - Exemplo:

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = A.union(B)
print(C)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Tipos de dados

- Dicionários:
  - São uma sequência não ordenada de itens;
  - Cada item é formado por um par de dados: chave e valor. Ideal para recuperar dados;

```
dados_pessoais = {'nome': 'Emanoeli',  'e-mail':'emanoelimadalosso@gmail.com'}
```

  - Elementos são acessados através de sua chave:

```
dados_pessoais['nome']
'Emanoeli'
```

  - Operações comuns:
    - Adicionar um item:

```
dados_pessoais['telefone'] = 123456
```

  - Métodos comuns:
    - keys(): retorna todas as chaves do dicionário;
    - values(): retorna todos os valores do dicionário;

Conversões entre tipos

- Entre números:

```
x = 5
float(x)
5.0

x = 5.1
int(x)
5

x = 5
complex(5)
5 + 0j
```

- Strings -> números

```
x = '5'
int(x)
5

float(x)
5.0

complex(x)
5+0j

x = '5a'
int(x)
ValueError: invalid literal for
int() with base 10: '5a'
```

- Números -> strings:

```
x = 5
str(x)
'5'

x = 5.1
str(x)
'5.1'

x = 1+2j
str(x)
'(1+2j)'
```

Conversão de tipos

- Conversão entre sequências:

```
# lista -> string
x = [1, 2, 3]
str(x)
'[1, 2, 3]'
# tupla -> string
x = (1, 2, 3)
str(x)
'(1, 2, 3)'
# set -> string
x = {1, 2, 3}
str(x)
'set([1, 2, 3])'
# dicionário -> string
x = {'key1': 1, 'key2': 2}
str(x)
'{'key1': 1, 'key2': 2}'
```

```
# string -> lista
x = 'ola'
list(x)
['o', 'l', 'a']
# tupla -> lista
x = (1, 2, 3)
list(x)
[1, 2, 3]
# set -> lista
x = {1, 2, 3}
list(x)
[1, 2, 3]
```

```
# string -> tupla
x = 'ola'
tuple(x)
('o', 'l', 'a')
# lista -> tupla
x = [1, 2, 3]
tuple(x)
(1, 2, 3)
# set -> tupla
x = {1, 2, 3}
tuple(x)
(1, 2, 3)
```

```
# string -> set
x = 'ola'
set(x)
set(['a', 'l', 'o'])
# lista -> set
x = [1, 2, 3]
set(x)
set([1, 2, 3])
# tupla -> set
x = (1, 2, 3)
set(x)
set([1, 2, 3])
```

```
# lista -> dicionário
x = [['key1', 1], ['key2', 2]]
dict(x)
{'key2': 2, 'key1': 1}
# tupla -> dicionário
x = (('key1', 1), ('key2', 2))
dict(x)
{'key2': 2, 'key1': 1}
```

Controles de fluxo

- If:

```
x = 5
if x > 0:
   print('Maior')
elif x < 5:
   print('Menor')
else:
   print('Igual')
```

Controles de fluxo

- While:

```
i = 0
while i < 5:
   i += 1
```

- For:
  - Objetivo de iterar sobre uma sequência de dados:

```
for i in range(0, 5):
   print(i)

for item in [0, 5, 4, 6, 2, 1]:
   print(item)
```

Operadores

- Aritiméticos: +, -, *, /, *, //, **;
- Comparação: <, >, <=, >=, ==, !=;
- Lógicos: and, or, not;
- Bitwise: &, |, ~, ^, <<, >>;
- Atribuição: =, +=, -=, *=, /=, %=, //=, **=;
- Especiais: is, is not, in, in not;

Funções

- Declarando e chamando funções:

```
def imprime_msg(msg):
    print(msg)

imprime_msg('Hello')
Hello
```

- Funções com argumento padrão:

```
def funcao_arg_padrao(p=1000):
    print(p)

funcao_arg_padrao()
1000
funcao_arg_padrao(5000)
5000
```

Funções

- Funções com argumento arbitrário:
  - Argumentos arbitrários devem sempre vir ao final:

```
def funcao_arg_arbitrario(x, y=None):
   if y:
      print(x + y)
   else:
      print(x)

funcao_arg_arbitrario(1)
1
funcao_arg_arbitrario(1, 2)
3
```

Funções

- Função Lambda (função anônima):
  - Definida sem precisar de um nome;
  - Usadas quando precisamos executar uma função simples por um curto período de tempo;

```
def double(x):
    return x * 2

double = lambda x: x * 2
print(double(2))
4
```

Argumentos

Expressão

  - Bastante usada com a função ***filter(função, objeto iterável)***;

```
lista = [1, 2, 3, 4, 5, 6]
pares = list(filter(lambda x: (x % 2 == 0), lista))
print(pares)
[2, 4, 6]
```

Vamos praticar!

- Dojo Fizz-Buzz:
  - O objetivo é escrever um programa que imprime "Fizz" para números divisíveis por 3, "Buzz" para números divisíveis por 5 e "FizzBuzz" quando o número é divisível por ambos.

```
import unittest

def fizzbuzz(numero):
     pass()

class TestFizzBuzz(unittest.TestCase):
     def teste1(self):
          self.assertEqual(fizzbuzz(numero), resultado_esperado)
```

Onde aprender mais?

https://www.programiz.com/python-programming

https://docs.python.org/3/

https://github.com/pyladiespatobranco

Iniciando com a Lingutagem Python

Emanoeli Madalosso
Jean Massucatto