
PRG203402 - Lógica de Programação:

Introdução ao Python

João Cláudio Elsen Barcellos

Engenheiro Eletricista
Formado na Universidade Federal de Santa Catarina
campus Florianópolis
joaoclaudiobarcellos@gmail.com

23 de Abril de 2025

** Créditos ao Prof. Emerson Ribeiro de Mello, o qual criou e disponibilizou o template aqui usado, via ShareLaTeX*

*** Créditos ao Prof. Renan Augusto Starke, o qual forneceu parte do conteúdo usado nesta apresentação*



Na aula de hoje veremos...





Python é uma das **cinco** mais populares linguagens de programação (segundo índice TIOBE).

É mais fácil de entender do que Java ou C.

Menos trabalho para fazer gráficos.

Tudo que se aprende em Python é aplicado a outras linguagens.





Há duas versões principais de Python: 2 e 3.

Há diferenças entre as duas, utilizaremos a versão 3.

Sempre verifique a versão utilizada.



Python: por onde começar

- 1 Put worm on hook.
- 2 Cast line into pond.
- 3 Watch the bobber until it goes underwater.
- 4 Hook and pull in fish.
- 5 If done fishing, then go home; otherwise, go back to step 1.

Cria-se o algoritmo.

```
def hook_fish():  
    print('I got a fish!')  
  
def wait():  
    print('Waiting...')  
    print('Get worm')  
    print('Get worm on hook')  
    print('Throw in lure')  
  
while True:  
    response = input('Is bobber underwater? ')  
    if response == 'yes':  
        is_moving = True  
        print('I got a bite!')  
        hook_fish()  
    else:  
        wait()
```

Escreve-se o código.

Testa-se!



Python: por onde começar

No MacOS e Linux já está instalado nativamente.

Windows, sugere-se instalar o Anaconda.

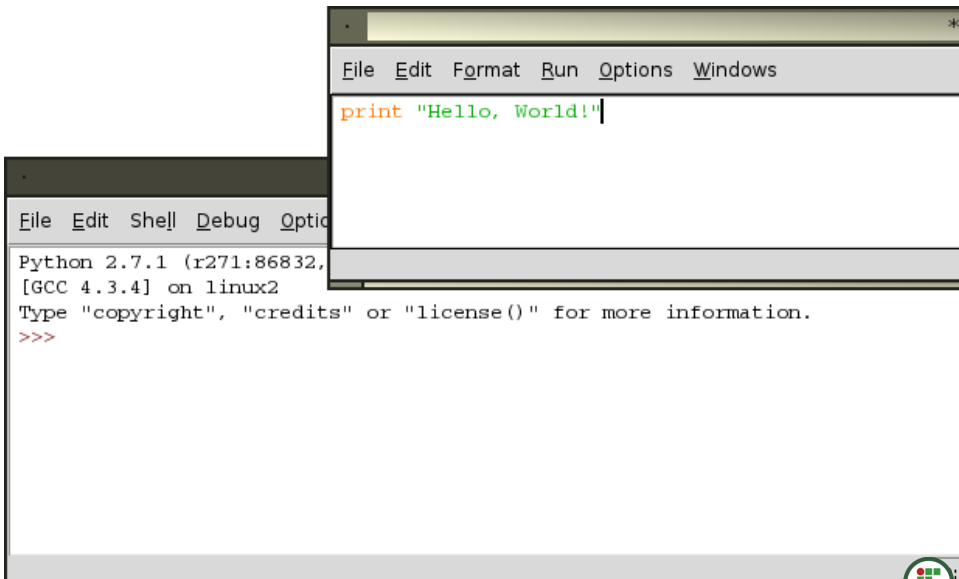
Um programa em Python é um arquivo de texto simples, pode-se escrever em qualquer programa.

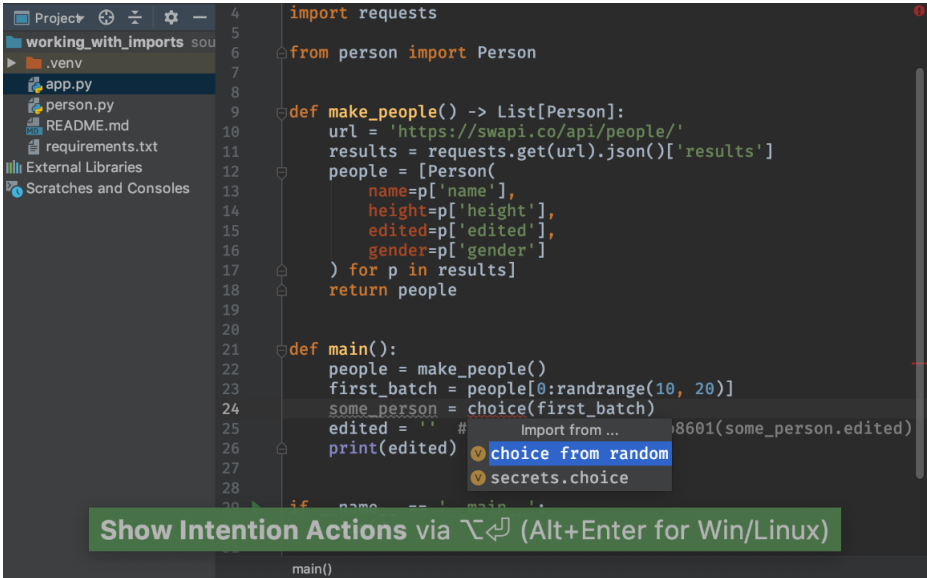
Mas há editores especiais que ajudam na codificação.

Inclusive online: Python Online



Idle3





The screenshot shows the PyCharm IDE interface. On the left is the Project Explorer with a tree view containing 'working_with_imports', '.venv', 'app.py', 'person.py', 'README.md', and 'requirements.txt'. The main editor displays a Python script. The script imports 'requests' and 'Person' from 'person'. It defines a 'make_people()' function that fetches data from 'https://swapi.co/api/people/' and returns a list of 'Person' objects. A 'main()' function calls 'make_people()', selects a random batch, and prints the 'edited' attribute of a chosen person. A tooltip is visible over the 'choice' attribute access, showing a list of intention actions: 'Import from ...', 'choice from random' (highlighted), and 'secrets.choice'.

```
4 import requests
5
6 from person import Person
7
8
9 def make_people() -> List[Person]:
10     url = 'https://swapi.co/api/people/'
11     results = requests.get(url).json()['results']
12     people = [Person(
13         name=p['name'],
14         height=p['height'],
15         edited=p['edited'],
16         gender=p['gender']
17     ) for p in results]
18     return people
19
20
21 def main():
22     people = make_people()
23     first_batch = people[0:randrange(10, 20)]
24     some_person = choice(first_batch)
25     edited = '' # Import from ... 8601(some_person.edited)
26     print(edited)
27     choice from random
28     secrets.choice
29
30 if __name__ == '__main__':
31     main()
```

Show Intention Actions via `⌘↵` (Alt+Enter for Win/Linux)



C:\Users\TestUser\Documents\Spyder - Spyder (Python 3.6)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Project explorer

Spyder

- Data
- spyder
 - github
 - conda.recipe
 - continuous_integration
 - doc
 - img_src
 - requirements
 - rose_profiling
 - scripts
 - spyder
 - app
 - tests
 - __init__.py
 - cl_options.py
 - mac_stylesheets.py
 - mainwindow.py
 - restart.py
 - start.py
 - tour.py
 - config
 - defaults
 - fonts
 - images
 - locale
 - plugins
 - tests
 - utils
 - widgits
 - windows
 - workers
 - __init__.py
 - dependencies.py
 - interspider.py
 - otherplugins.py
 - pl_gatch.py
 - pycompat.py
 - pyplot.py
 - requirements.py
 - spyder_bnaekpoints
 - spyder_is_dom
 - spyder_in_hdfs
 - spyder_profiler
 - spyder_pyint
 - checkignore
 - clocheck
 - cincoynight
 - codecs.yml
 - coveragegc
 - gitignore
 - pepspeaks.yml
 - project
 - travis.yml

temp.py | interpolation.py | __init__.py | und_helper.py | und_main.py | README.md

```

6
7 import pylab
8 from numpy import cos, linspace, pi, sin, random
9 from scipy.interpolate import splprep, splev
10
11 ## Generate data for analysis
12
13 # Make ascending spiral in 3-space
14 t = linspace(0, 1.75 * 2 * pi, 100)
15
16 x = sin(t)
17 y = cos(t)
18 z = t
19
20
21 # Add noise
22 x += random.normal(scale=0.1, size=x.shape)
23 y += random.normal(scale=0.1, size=y.shape)
24 z += random.normal(scale=0.1, size=z.shape)
25
26 ## Perform calculations
27
28 # Spline parameters
29 smoothness = 3.0 # Smoothness parameter
30 k_param = 2 # Spline order
31 nests = -1 # Estimate of number of knots needed (-1 = maximal)
32
33 # Find the knot points
34 knot_points, u = splprep([x, y, z], s=smoothness, k=k_param, nests=-1)
35
36 # Evaluate spline, including interpolated points
37 knew, ynew, znew = splev(linspace(0, 1, 400), knot_points)
38
39
40 ## Plot results
41
42 # TODO: Rewrite to avoid code smell
43 pylab.subplot(2, 2, 1)
44 data = pylab.plot(x, y, 'bo-', label='Data with X-Y Cross Section')
45 fit = pylab.plot(knew, ynew, 'r-', label='Fit with X-Y Cross Section')
46 pylab.legend()
47 pylab.xlabel('x')
48 pylab.ylabel('y')
49
50
51 pylab.subplot(2, 2, 2)
52 data = pylab.plot(x, z, 'bo-', label='Data with X-Z Cross Section')
53 fit = pylab.plot(knew, znew, 'r-', label='Fit with X-Z Cross Section')

```

Outline

- interpolation.py
 - Generate data for analysis
 - Perform calculations
 - Plot results
 - imputerlib
 - Queue
 - __init__
 - appendleft
 - pop
 - with open(data_path + output_file_name, 'w') as f:
 - Example Biter class
 - print(f)
 - Example Biter class
 - DataSet
 - __init__
 - prepare_dataset
 - Series
 - something
 - Dataframe
 - something
 - foo
 - __init__
 - spam
 - with open(file) as f:
 - Base
 - Derived
 - for c, bar in zip(radii, bars):
 - with np.load(filename) as demc:

Variable explorer

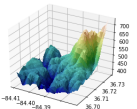
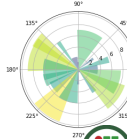
Name	Type	Size	Min:	Value
array_int8	int8	(2, 3)	Min: -7	
array_uint32	uint32	(2, 2, 3)	Min: 1	
bars	container.BarContainer	20	Max: 7	BarContainer object of matplotlib
df	DataFrame	(3, 2)	Column names: bools, ints	
filename	str	1		C:\ProgramData\Anaconda3\Li
list_test	list	2		[Dataframe, Numpy array]
nrows	int	1		344
r	float64	1		7.611882589334796
radii	float64	(20,)	Min: 0.4983036638535687	
region	tuple	2		(slice, slice)
rgb	float64	(45, 45, 4)	Min: 0.0	
series	Series	(1,)	Max: 1.0	Series object of pandas.com
test_none	NoneType	1		NoneType object of builtins

Python console

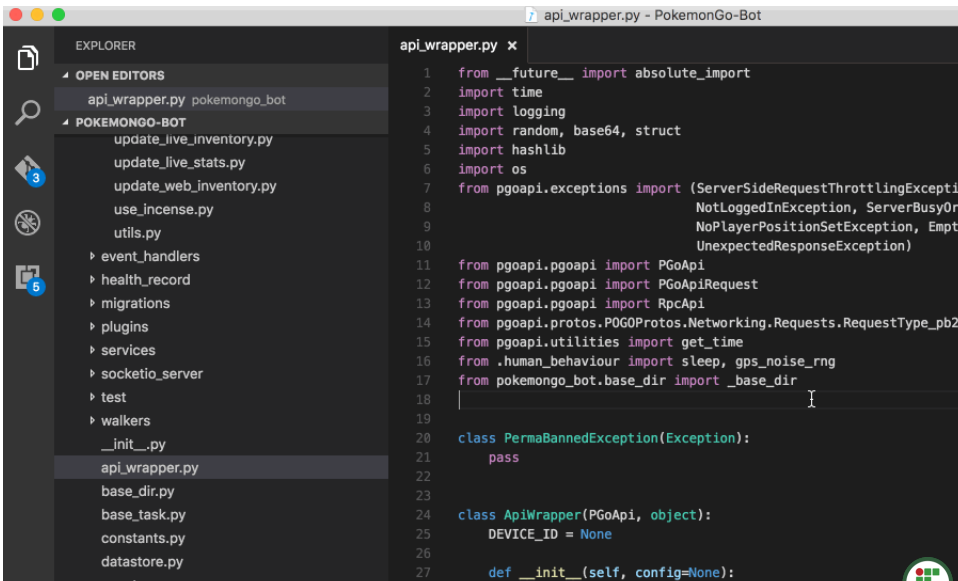
```

...: ls = LightSource(270, 45)
...: # To use a custom hillshading mode, override the built-in sh
...: # in the rgb colors of the shaded surface calculated from "s
...: rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_m
...: surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, shade
...: linewidth=0, antialiased=False, shade
...:
...: plt.show()

```


Visual Code



Jupyter Notebook

File Edit View Insert Cell Kernel Help



```
In [2]: print(1)
```

```
1
```

```
In [3]: import sys
        sys.version
```

```
Out[3]: '2.7.12 (default, Nov 19 2016, 06:48:10) \n[GCC 5.4.0 20160609]'
```

```
In [4]: sys.path
```

```
Out[4]: ['',
          '/usr/lib/python2.7',
          '/usr/lib/python2.7/plat-x86_64-linux-gnu',
          '/usr/lib/python2.7/lib-tk',
          '/usr/lib/python2.7/lib-old',
          '/usr/lib/python2.7/lib-dynload',
          '/home/FDSM_lhn/.local/lib/python2.7/site-packages',
          '/usr/local/lib/python2.7/dist-packages',
          '/usr/lib/python2.7/dist-packages',
          '/usr/local/lib/python2.7/dist-packages/IPython/extensions',
          '/home/FDSM_lhn/.ipython']
```

```
In [ ]: ›
```



Meu primeiro programa

Tradicionalmente o primeiro programa escrito é um “Olá Mundo”.

Para introduzir você ao mundo em Python:

```
print("Olá mundo, meu nome é Juquinha")
```



Comentários

Comentários são ignorados pelo computador quando um programa é executado.

Eles ajudam na compreensão do código.

Comentários em Python são ativados pelo caractere: #

```
# Isso é um comentário  
print("Olá!") # A partir daqui também é!  
  
print("# isso não é comentário")  
# Um novo comentário pode ser colocado aqui!
```



A função print

A função `print` imprime dados do programa.

Funções em Python são parecidas com as matemáticas:

`seno(x)`

`cosseno(x)`

`f(x)`

Funções são sempre seguidas por parênteses ()

Os dados colocados entre parênteses são conhecidos como **argumentos**.



Personalizando o print

Há alguns caracteres especiais:

```
# Se eu precisar imprimir o caractere "  
# Devo colocar uma barra na frente  
print("Desejo imprimir o \" por algum motivo.")  
  
# E se eu quiser o caractere \ ?  
print("Esse arquivo está em C:\\minha_pasta")
```



Personalizando o print

Caractere	Descrição
\'	Aspas simples
\"	Aspas duplas
\t	Tabulação
\n	Nova linha

```
print("Isso\né\num\nexemplo.")
```

```
Isto
é
um
exemplo.
```



Uma **variável** é utilizada para guardar dados que serão reutilizados.

Para criar uma variável em Python, utilizamos o operador de atribuição

=

```
x = 5

print(x) # Isto irá imprimir 5

mensagem = "Olá"

print(mensagem) # Isto irá imprimir Olá
```



Regras para nomes de variáveis

Bons exemplos:

temperatura_em_celsius

posicao_tres

velocidade_carro

numero_de_itens

simpsom

OK, mas não muito bom:

temperaturaEmCelsius: melhor manter tudo minúsculo

x: pode ser muito curto

Simsom: evitar letras maiúsculas

Não funciona:

posição três: acentos e espaços não são permitidos

4tercos: não pode começar com números ou conter caracteres especiais (ç?)



Tipos de dados nas Variáveis

Existem três tipos básicos de variáveis em Python:

Textos e caracteres: dados vindos do teclado, por exemplo.

Conhecidos como **string**

Números inteiros: números sem ponto. Conhecidos como **int** ou **integer**.

Números reais: números **com** ponto. Conhecidos como **float**.

```
numero_inteiro = 5          # inteiro, int ou integer
numero_virgula  = 9.69      # real ou float
mensagem = "Olá"           # texto ou string
```



Tipos de dados nas Variáveis

Para pedir para o usuário dados, utilizamos a **função** `input`.

```
x = input('Digite um número por gentileza: ')\nprint(type(x))      # Com a função type podemos descobrimos o tipo de x!
```

```
<class 'str'>
```

Como os dados são oriundos do teclado, é texto... **string**.

Textos não são números, portanto não conseguiremos fazer operações aritméticas.



Tipos de dados nas Variáveis

```
x = input('Digite um número por gentileza: ')\ny = x + 5\n\nprint(y)
```

TypeError Traceback (most recent call last)

```
<ipython-input-15-3430d491b4b6> in <module>()\n    1 x = input('Digite um número por gentileza: ')\n    2 y = x + 5\n    3\n    4 print(y)
```

TypeError: can only concatenate **str** (**not** "int") to **str**



Tipos de dados nas Variáveis

Computador não gosta que misture tipos, portanto temos que converter:

```
# Obtém dados do teclado e os converter para inteiro
x = int(input('Digite um número por gentileza: '))
y = x + 5
print(y)
```

```
Digite um número por gentileza: 58
63
```



Tipos de dados nas Variáveis

Para converter, usa-se as funções `str()`, `int()` e `float()`:

`int(dado)`: tenta converter `dado` para o tipo inteiro.

`float(dado)`: tenta converter `dado` para o tipo real.

`str(dado)`: tenta converter `dado` para uma string.

```
print(type(int(5)))  
print(type(float(5)))  
print(type(str(5)))
```

```
<class 'int'>  
<class 'float'>  
<class 'str'>
```



Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
**	Exponenciação (elevado a)
/	Divisão
//	Divisão inteira (parte após a vírgula é ignorada)
%	Módulo (resto de uma divisão inteira)



```
x = 3 + 4    # x vale = 7
```

```
x = 5
```

```
y = 6
```

```
z = x - 2 * y    # z = 5 - 6 * 2 = -7 (cuidado com a  
                # precedência dos operadores)
```

```
x = 8
```

```
y = 6
```

```
a = x / y        # a = 1.3333333333333333
```

```
b = x // y       # b = 1
```

```
c = x % y        # c = 2
```



Operadores de comparação

Operador	Descrição
>	Maior
<	Menor
>=	Maior igual
<=	Menor igual
==	Igual
!=	Diferente



```
x = 3    # x vale = 7
```

```
y = 7
```

```
a = x > y # x é maior que y?
```

```
    # (comparação retornar verdadeiro ou falso)
```

```
# a = Falso
```

```
b = x == y # x é igual ao y ?
```

```
# b = Falso
```

```
c = x != y # x é diferente de y?
```

```
# c = Verdadeiro
```





Python Arcade

PyCharm Edu

FORBELLONE, A. L. V. **Lógica de Programação: a construção de algoritmos e estruturas de dados.** 3.ed. São Paulo: Prentice Hall, 2005.

Eric Freeman. **Head First Learn to Code.** O'Reilly, 2018.

