

Programação Orientada a Objetos II

Tiago Bacciotti Moreira



github
SOCIAL CODING

Escopo de trabalho

Conceitos Fundamentais

Python 3.8.1

Jupyter Notebook (Mini-curso)

MUITOS exercícios

Recursos

Slides

Conteúdo Web

Atendimento

Recursos

Replit.it

Google Colab

Get your ideas out there.

Stop wasting time setting up a development environment. Repl.it gives you an instant IDE to learn, build, collaborate, and host all in one place.

Sign up





Untitled0.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)



+ Code + Text



```
[ ] 1
```



```
1 1+1  
2
```



```
2
```

Link Classroom Repl.it

<http://bit.ly/uemg2020>

Link Entrega de Atividades

<http://bit.ly/entrega2020>



Entrega de trabalhos 2020-1

Use esse formulário para enviar seu trabalho. Siga o padrão de nomes para que seu trabalho seja facilmente identificado.

Para fins de organização coloque no nome do arquivo a SIGLA da disciplina seguido do número do trabalho. Não é necessário colocar seu nome no nome do arquivo pois o o google já adiciona seu nome.

Programação Orientada a Objetos II --> POO2

Gestão de TI --> GTI

Tópicos Avançados em Banco de Dados --> TABD

Formato: SIGLA - NÚMERO DO TRABALHO

Exemplos de nomes corretos: POO2 - Trabalho 1, GTI - Trabalho 2, TABD - Entrega de atividade 1

Exemplos de nomes inadequados: TRABALHO DO JOSÉ BUSCAPÉ, Atividade planilha, memória, etc

O nome e a foto associados à sua Conta do Google serão registrados quando você fizer upload de arquivos e enviar este formulário. Não é **baciotti@gmail.com**? [Alternar conta](#)

***Obrigatório**

Endereço de e-mail *

Seu e-mail

Período

- ☐ Quinto período Sistemas de Informação
- ☐ Sétimo período Sistemas de Informação

Anexe seu trabalho

 Adicionar arquivo

Uma cópia das suas respostas será enviada para o endereço de e-mail fornecido

Enviar

Github

**[https://github.com/baciotti/A
ulasUEMG](https://github.com/baciotti/ulasUEMG)**

Introdução

Fonte: https://www.tutorialspoint.com/python3/python_overview.htm

Algumas características

Interpretado

Interativo

Orientado a Objeto

Fácil de ler/escrever/manter

Vasta biblioteca

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constants or variables or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal is the beginning of a comment. All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.

```
#!/usr/bin/python3

# First comment
print ("Hello, Python!") # second comment
```

[Live Demo](#)

This produces the following result –

```
Hello, Python!
```

You can type a comment on the same line after a statement or expression –

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows –

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
This is a multiline
comment.
'''
```

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on a single line given that no statement starts a new code block. Here is a sample snip using the semicolon –

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Declarando variáveis

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print (counter)
print (miles)
print (name)
```

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example –

```
a, b, c = 1, 2, "john"
```

Here, two integer objects with values 1 and 2 are assigned to the variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Tipos de Dados

Números (int, float)

String

Listas

Tuplas

Dicionários

```
#!/usr/bin/python3

str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

This will produce the following result -

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

```
#!/usr/bin/python3

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)          # Prints complete list
print (list[0])        # Prints first element of the list
print (list[1:3])      # Prints elements starting from 2nd till 3rd
print (list[2:])        # Prints elements starting from 3rd element
print (tinylist * 2)    # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

This produces the following result –

```
['abcd', 786, 2.23, 'john', 70.200000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.

The main difference between lists and tuples are – Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as **read-only** lists. For example –

```
#!/usr/bin/python3

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print (tuple)           # Prints complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])      # Prints elements starting from 2nd till 3rd
print (tuple[2:])       # Prints elements starting from 3rd element
print (tinytuple * 2)    # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

[Live Demo](#)

This produces the following result –

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```


Python Dictionary

Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

```
#!/usr/bin/python3

dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john', 'code': 6734, 'dept': 'sales'}

print (dict['one'])      # Prints value for 'one' key
print (dict[2])         # Prints value for 2 key
print (tinydict)        # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the values
```

[Live Demo](#)

This produces the following result –

```
This is one
This is two
{'name': 'john', 'dept': 'sales', 'code': 6734}
dict_keys(['name', 'dept', 'code'])
dict_values(['john', 'sales', 6734])
```