Programação Orientada a Objetos II

Tiago Bacciotti Moreira

Ementa

Objetivo Geral:

Objetivos Específicos:.

Bibliografia

Quatro horários semanais

Teórica e Prática

Divisão de Notas					
Disciplina	Prova 1	Prova 2	Trabalhos	Prova Seme	Total
POO 2	15	15	30	40	100

Datas de Provas			
Disciplina	Prova 1	Prova 2	Prova Semestral
POO 2	30/mar.	28/abr.	29/jun.

POO 2			
Aula	Data	Dia	Obs
1	17/fevereiro	Seg	
2	18/fevereiro	Ter	
3	02/março	Seg	
4	03/março	Ter	
5	09/março	Seg	
6	10/março	Ter	
7	16/março	Seg	
8	17/março	Ter	
9	23/março	Seg	
10	24/março	Ter	
11	30/março	Seg	Prova 1
12	31/março	Ter	
13	06/abril	Seg	
14	07/abril	Ter	
15	13/abril	Seg	
16	14/abril	Ter	
17	20/abril	Seg	
18	21/abril	Ter	Feriado Tiradentes
19	27/abril	Seg	
20	28/abril	Ter	Prova 2
21	04/maio	Seg	
22	05/maio	Ter	
23	11/maio	Seg	
24	12/maio	Ter	
25	18/maio	Seg	
26	19/maio	Ter	
27	25/maio	Seg	
28	26/maio	Ter	
29	01/junho	Seg	
30	02/junho	Ter	
31	08/junho	Seg	
32	09/junho	Ter	
33	15/junho	Seg	
34	16/junho	Ter	
35	22/junho	Seg	
36	23/junho	Ter	
37	29/junho	Seg	Prova Semestral
38	30/junho	Ter	
39	06/julho	Seg	
40	07/julho	Ter	

39 aulas

Plataforma Lattes | CNPq

github
SOCIAL CODING

Escopo de trabalho

Conceitos Fundamentais

Python 3.8.1

Jupyter Notebook (2 mini-cursos)

MUITOS exercícios

Recursos

Slides

Conteúdo Web

Atendimento

Recursos

Replit.it

Google Colab

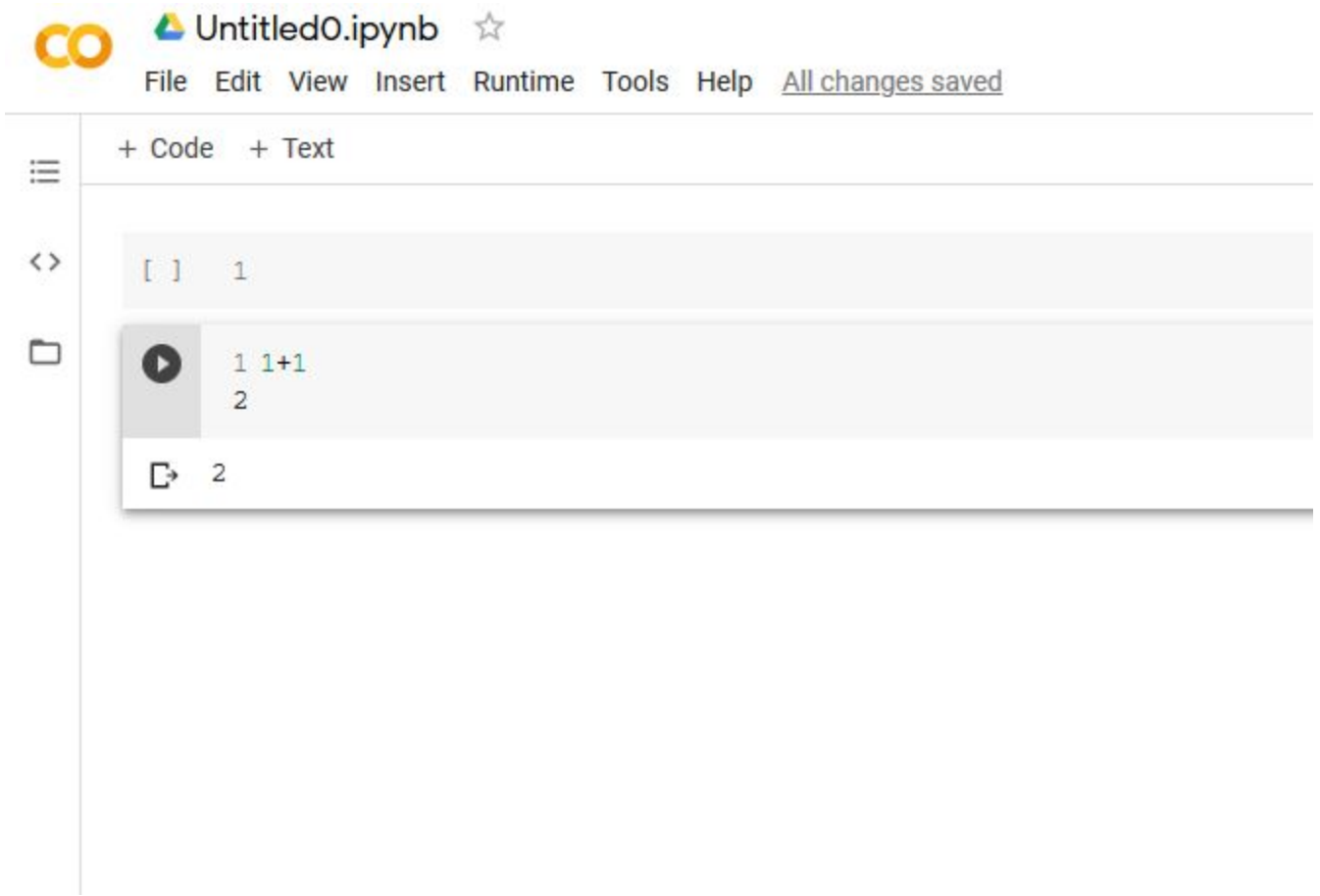
repl.it jobs blog pricing features challenge

+ new repl Log In

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

Github

https://github.com/baciotti/AulasUEMG

Introdução

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

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		

Lines and Indentation

Python does not use braces({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

```
if True:
   print ("True")

else:
   print ("False")
```

Multi-Line Statements

Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
   item_two + \
   item_three
```

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.

Live Demo

```
#!/usr/bin/python3

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

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, complex)

String

Listas

Tuplas

Dicionários

Live Demo

```
#!/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
```

Live Demo

```
#!/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 −

Live Demo

```
#!/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
```

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 −

Live Demo

```
#!/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
```

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])
```

Resolver exercício 1

http://bit.ly/uemg2020

https://www.baciotti.com