

Python

introdução

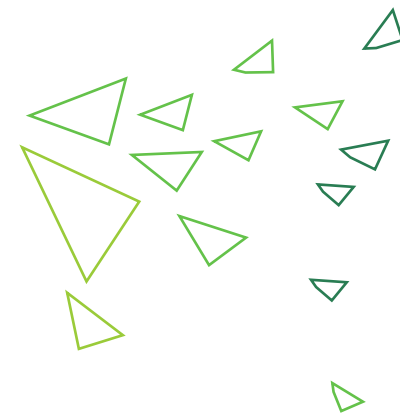
Eduardo Silva – easilva91@gmail.com

```
31 def __init__(self, settings):
32     self.file = None
33     self.fingerprints = set()
34     self.logdups = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, 'requests.log'),
39                           'a')
40         self.file.seek(0)
41         self.fingerprints.update(retrieved)
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getbool('SUPERLITE_DEBUG')
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```




Quem é o Eduardo

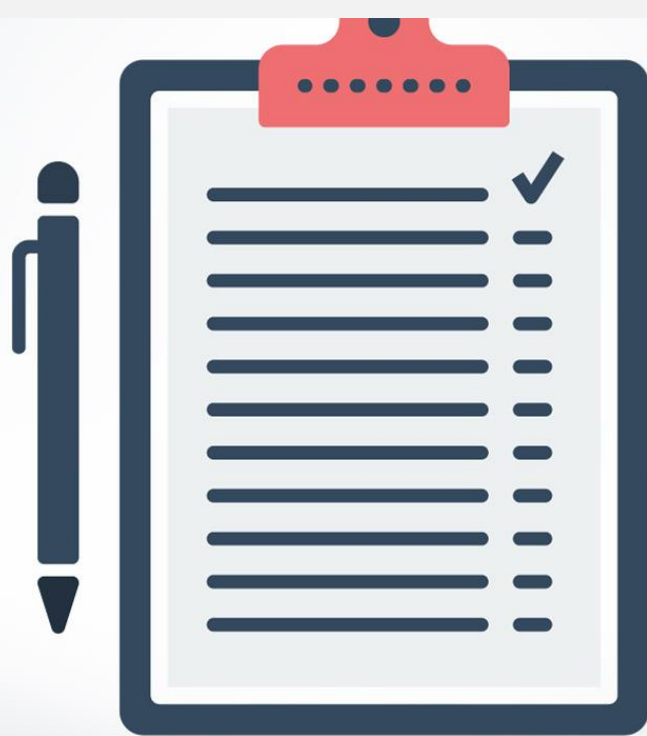
Do que se tratam as aulas?



Bacharel em Gestão da Informação pela UFG;

Mestrando em Gestão da Informação pela NOVA IMS;

- As aulas tem por objetivo principal apresentar conceitos teóricos e práticos de mineração de dados com foco na linguagem Python.



Agenda

Reverendo conceitos

1. Introdução ao Python

- I. O que é o Python?
- II. Porque Python?
- III. Versões do Python e IDE

2. Noções básicas de Python

3. Variáveis, tipos de dados e operadores

4. Estrutura de Dados

5. Loops e Condicionais

O Que é o Python



“Python é uma linguagem de programação que permite trabalhar rapidamente e integrar sistemas de forma mais eficaz”

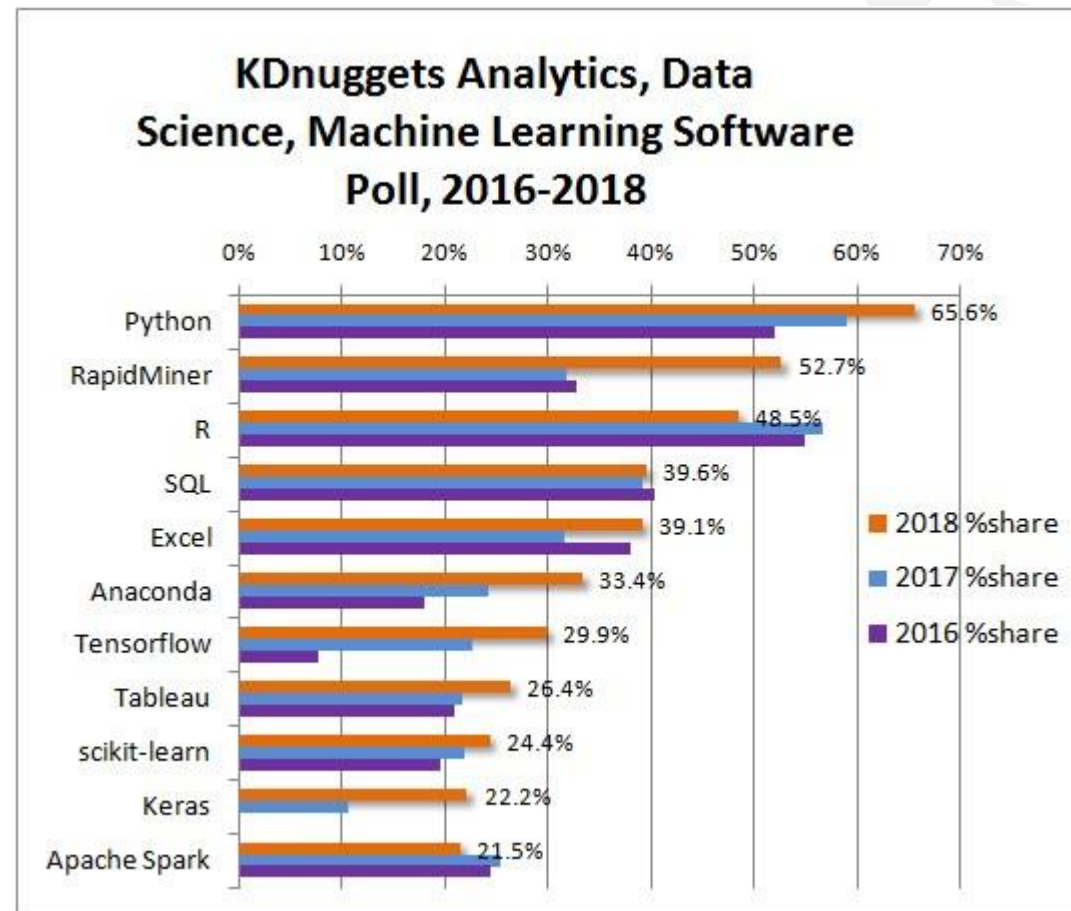
www.python.org

“Python é uma linguagem de programação interpretada de alto nível e de propósito geral [...] Python tem uma filosofia de design que enfatiza a legibilidade do código e uma sintaxe que permite aos programadores expressarem conceitos em menos linhas de código que podem ser usados em linguagens como C ++ e Java”

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

Porque Python?

1. Fácil de aprender e poderosa.
2. Vastamente utilizada para programação e propósitos em geral.
3. Desenvolvida sobre a licença open source, se tornando de fácil utilização e distribuição.
4. Flexível e escalável.
5. Linguagem de programação popular e com uma comunidade gigantesca.
6. Python está em todo lugar: projetos de ciências de dados, aprendizagem de máquina, aprendizagem profunda, aplicações científicas e muito mais.



Versões do Python e IDE

A maior parte das aplicações atuais ainda utilizam a versão 2.7 do Python. Entretanto, essa versão só irá receber updates de segurança até 2020

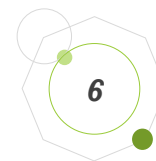
O Python 3.0 foi lançado em 2008, e iremos utilizar essa versão durante as aulas.

Python IDE



Iremos utilizar o Spyder durante as aulas, uma IDE multi plataforma, que está disponível juntamente com o ambiente Anaconda, para programação científica com Python.

O Spyder integra por padrão livrarias como Numpy, Scipy, Matplotlib e Ipython.



Versões do Python e IDE

Download do Anaconda e do Spyder

<https://www.anaconda.com/distribution/>



Windows



macOS



Linux

Anaconda 5.0.0 For Windows Installer

Python 3.6 version *

64-Bit Graphical Installer (535 MB) ?

Download

[Download 32-Bit \(436 MB\)](#)

Python 2.7 version *

64-Bit Graphical Installer (522 MB) ?

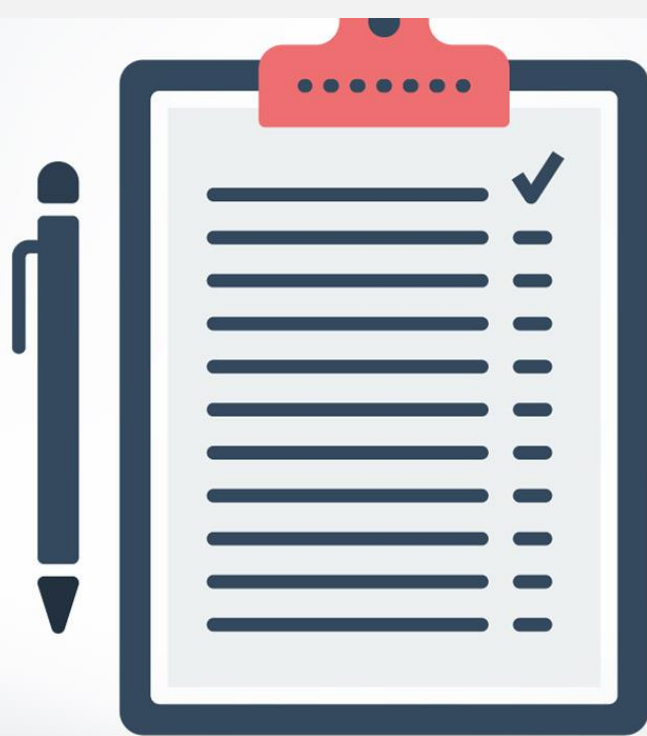
Download

[Download 32-Bit \(421 MB\)](#)

[Behind a firewall?](#)

[*How to get Python 3.5 or other Python versions](#)

[How to Install ANACONDA](#)

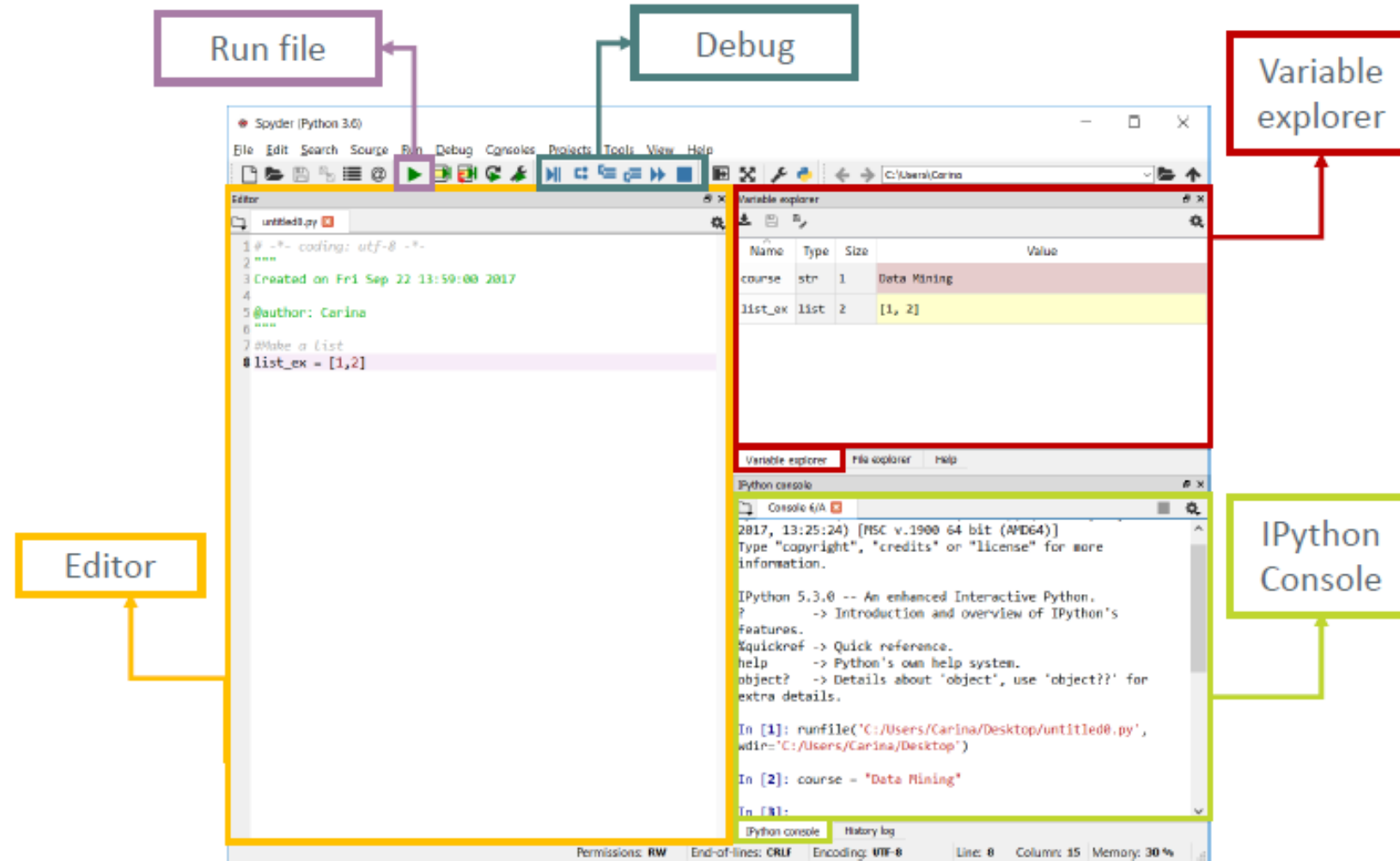


Agenda

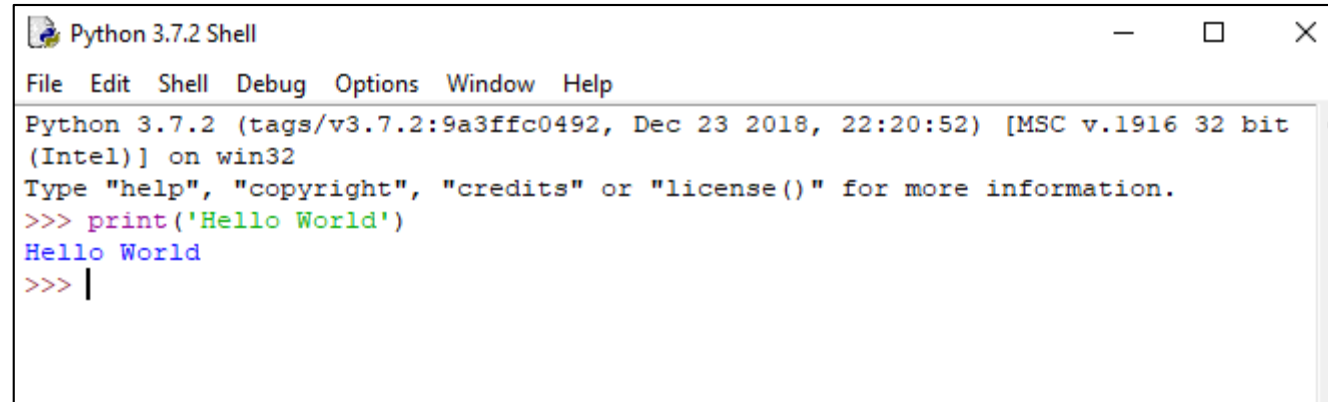
Reverendo conceitos

1. Introdução ao Python
2. **Noções básicas de Python**
 - I. Interface do Spyder
 - II. Primeiro Programa: Hello World
 - III. Como executar
 - IV. Sintaxe e semântica
 - V. Print() e Comentários
 - VI. Input do Usuário
 - VII. Instalando novos modulos
3. Variáveis, tipos de dados e operadores
4. Estrutura de Dados
5. Loops e Condicionais

Interface do Spyder



Primeiro programa: Hello World!



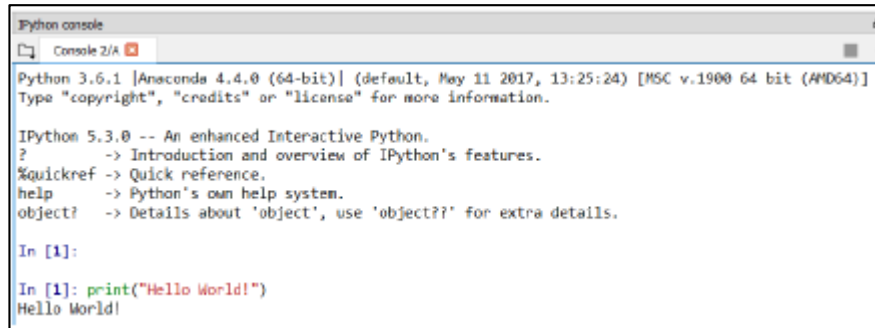
```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello World')
Hello World
>>> |
```

```
In []: # "Hello world!" program in Python 3.x
      print("Hello World!")
```

```
Out[]: Hello World!
```

Como Executar?

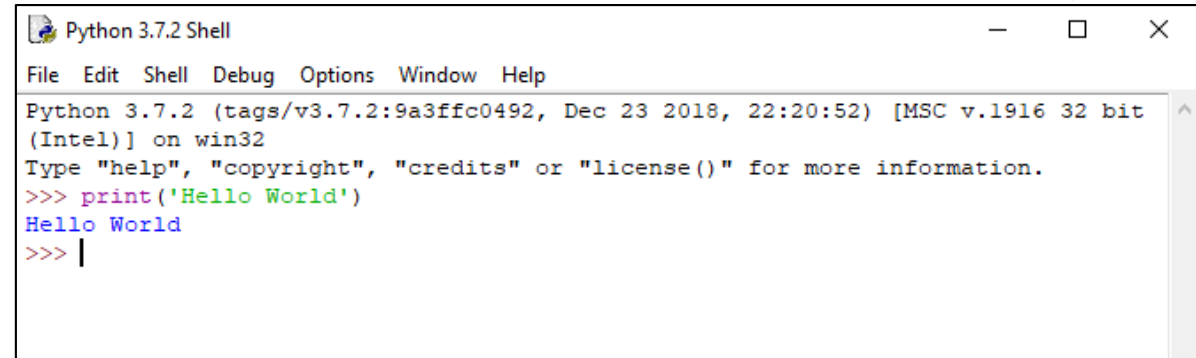
1 - Diretamente na console do Ipython dentro do Anacando, ou no IDLE do Python.



```
Python console
Console 2/A
Python 3.6.1 [Anaconda 4.4.0 (64-bit)] (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

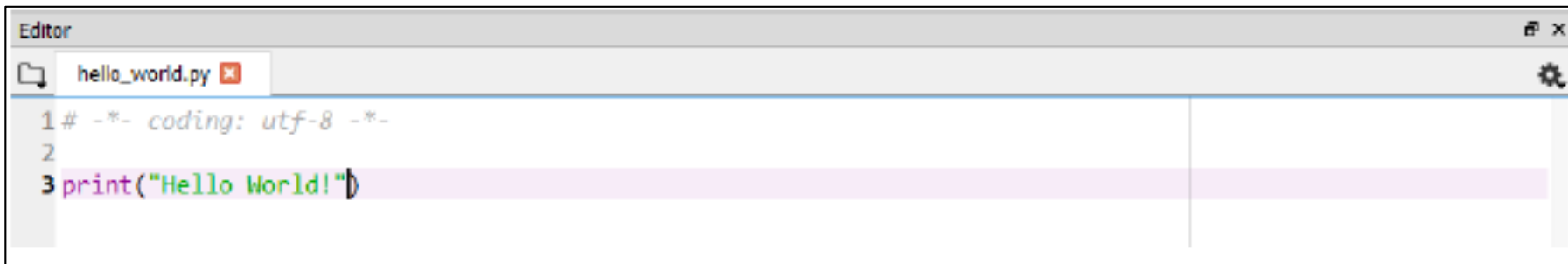
IPython 5.3.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]:
In [1]: print("Hello World!")
Hello World!
```



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello World')
Hello World
>>> |
```

2 - Método comum de Script



```
Editor
hello_world.py
1 # -*- coding: utf-8 -*-
2
3 print("Hello World!")
```

Sintaxe e Semântica

O Python traz algumas diferenças no seu uso, quando comparado com linguagens como C, entre outras.

- 1 - Não existe o uso de ponto e vírgula.
- 2 - Não se usam chaves { }
- 3 - Indentação é parte importante da sintaxe

Notas:

- Python é uma linguagem case sensitive (A a)
- Os índices em pythons se iniciam em 0
- Python usa espaços em branco (tabs ou espaço) para indentar o código, ao invés de usar chaves ou parenteses, um código em python ficaria da seguinte forma:

```
In []: """
Generates Fibonacci series
"""
def fibonacci(n):
    x,y = 0,1
    while x < n:
        print(x)
        x, y = y, x + y

fibonacci(5000)
```


Print() e Comentários

```
In []: # This prints Hello Data Mining!
print("Hello Data Mining!")

# This is a single line comment

"""
And this is a multiline comment.
It can take more than one line.
Like this.
"""
```

Out[]: Hello Data Mining!

```
In []: print("Two times two is", 2 * 2)
print("4 is less than or equal to 5 is", 4<=5)
subject = "Data Mining"
print("This is", subject)
print("This is " + subject + " in NOVA IMS")
```

Out[]: Two times two is 4
4 is less than or equal to 5 is True
This is Data Mining
This is Data Mining in NOVA IMS

Input dos Usuários

Receber inputs do usuário é importante em quase todo o programa, o python tem uma função de input que possibilita receber dados do usuário e usá-lo posteriormente.

```
In []: # Get the input and store it in a variable  
name=input("Please enter your name: ")
```

```
Out[]: Please enter your name: Maria
```

```
In []: # Print the entered value  
print("Hi" + name + "!")
```

```
Out[]: Hi Maria!
```

Instalando novos módulos

Quase sempre que utilizamos um script em python, fazemos uso de alguma livreria ou módulo dessa linguagem, nesse contexto existem diferentes formas de instalar essas livrerias no seu computador.

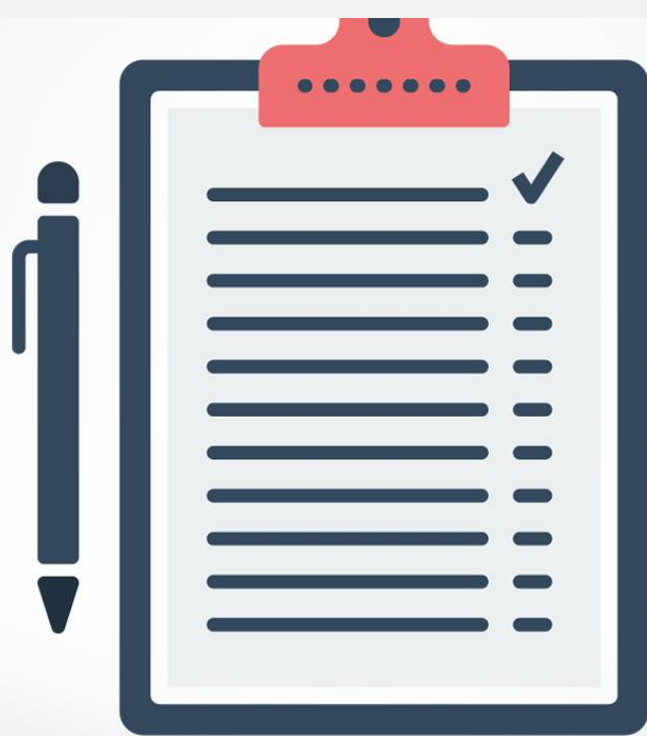
1 – Utilizando o spyder/anaconda para instalar módulos

```
In [1]: !pip install seaborn
```

```
In [1]: !pip install seaborn  
(base) C:\Users\eduar>conda install -c spyder-ide qdarkstyle_
```

2 – Utilizando o python instalado na sua máquina

```
C:\> Linha de comandos  
Microsoft Windows [Version 10.0.17763.379]  
(c) 2018 Microsoft Corporation. Todos os direitos reservados.  
C:\Users\eduar>pip install spacy
```



Agenda

Reverendo conceitos

1. Introdução ao Python
2. Noções básicas de Python
3. **Variáveis, tipos de dados e operadores**
 - I. Atribuição de variáveis
 - II. Tipos de Dados
 - III. Operadores
 1. Operadores aritméticos
 2. Operadores relacionais/comparação
 3. Operadores lógicos
 4. Operadores de associação
 5. Operadores de Identidade
4. Estrutura de Dados
5. Loops e Condicionais

Atribuição de variáveis

Para criar variáveis em python basta definir um nome para a variável e o valor dessa variável.

```
# Integer
a = 2
print(a)
# Output: 2

# Integer
b = 9223372036854775807
print(b)
# Output: 9223372036854775807

# Floating point
pi = 3.14
print(pi)
# Output: 3.14
```

*algo importante ao definir o nome de uma variável em python é a utilização de padronização dos nomes, utilizando letras maiúsculas ou underscore para definir uma variável:

minhaVariavel = 20

minha_variavel = 20

Tipos de Dados

Numérico: int, float

Boleano: true, false

Sequência: str, list, tuple

Maps: dict

Sets: set

- Dados imutáveis e mutáveis.

Tipos imutáveis (não podem ser alterados)

- int, float, long, str, tuple...

Tipos mutáveis (podem ser alterados)

- list, dict, set,...



Operadores Aritméticos

<http://www.w3big.com/pt/python/python-operators.html>



Operador	Descrição	Exemplo
+	Faz adição de dois valores	$7 + 2 = 9$
-	Faz subtração entre valores	$7 - 2 = 5$
*	Faz multiplicação entre dois valores	$7 * 2 = 14$
/	Faz a divisão entre dois valores	$7 / 2 = 3.5$
%	Retorna o resto de uma divisão	$7 \% 2 = 1$
**	Faz o calculo de expoente	$7 ** 2 = 49$
//	Retorna a parte inteira de uma divisão	$7 // 2 = 3$



Operadores Relacionais e de Comparação

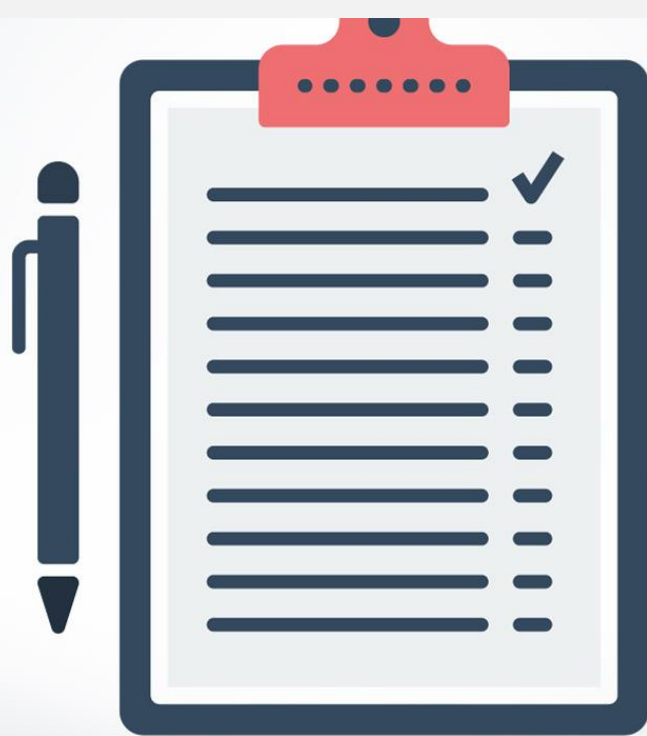
<http://www.w3big.com/pt/python/python-operators.html>

Operador	Descrição	Exemplo
==	Valores iguais	3 == 4 é Falso
!=	Valores diferentes	3 != 4 é Verdadeiro
>	Valor maior que.	3 > 4 é Falso
<	Valor menor que.	3 < 4 é Verdadeiro
>=	Valor maior ou igual.	3 >= 4 é Falso
<=	Valor menor ou igual.	3 <= 4 é verdadeiro

Operadores Lógicos/Associação e Identidade

<http://www.w3big.com/pt/python/python-operators.html>

Operador	Descrição	Exemplo
and	Determina quando dois valores são verdadeiros.	Verdade and Verdadeiro é Verdade Verdadeiro and Falso é Falso
or	Determina quando um de dois valores é verdadeiro.	Verdadeiro or Verdadeiro é Verdadeiro Verdadeiro or Falso é Verdadeiro
not	Nega a veracidade de um único operador. Um valor verdadeiro se torna falso e vice versa.	not Verdadeiro é Falso not Falso é Verdadeiro
in	Determina se um valor do primeiro operador aparece no segundo.	“World” in “Hello World” é Verdadeiro
not in	Determina se o valor do primeiro operador não aparece no segundo.	“World” not in “Hello World” é Falso
is	Avalia se o tipo do primeiro valor é igual o tipo do segundo valor	Type(1) is int é Verdadeiro
is not		Type(1) is not int é Falso



Agenda

Reverso conceitos

1. Introdução ao Python
2. Noções básicas de Python
3. Variáveis, tipos de dados e operadores
4. **Estrutura de Dados**
 - I. Listas
 - II. Dicionários
 - III. Strings
5. Loops e Condicionais

Estrutura de Dados

Existem 6 estruturas de dados diferentes em python, mas não iremos falar de set e tuples.

1 – String

As strings são identificadas como grupos de caracteres em conjunto, ou de forma mais simples, texto ou palavras. Strings são tipos de dados sequenciais imutáveis, ou seja, cada vez que alguém faz alterações para uma string, um objeto de string completamente novo é criado.

```
a_str = 'Hello World'
print(a_str)      #output will be whole string. Hello World
print(a_str[0])   #output will be first character. H
print(a_str[0:5]) #output will be first five characters. Hello
```

2 – Numérico

Existem quatro tipos de dados numéricos em python. (na maior parte do tempo só lidamos com os 2 primeiros).

```
int_num = 10      #int value
float_num = 10.2   #float value
complex_num = 3.14j #complex value
long_num = 1234567L #long value
```

Estrutura de Dados

Existem 6 estruturas de dados diferentes em python.

3 – Listas

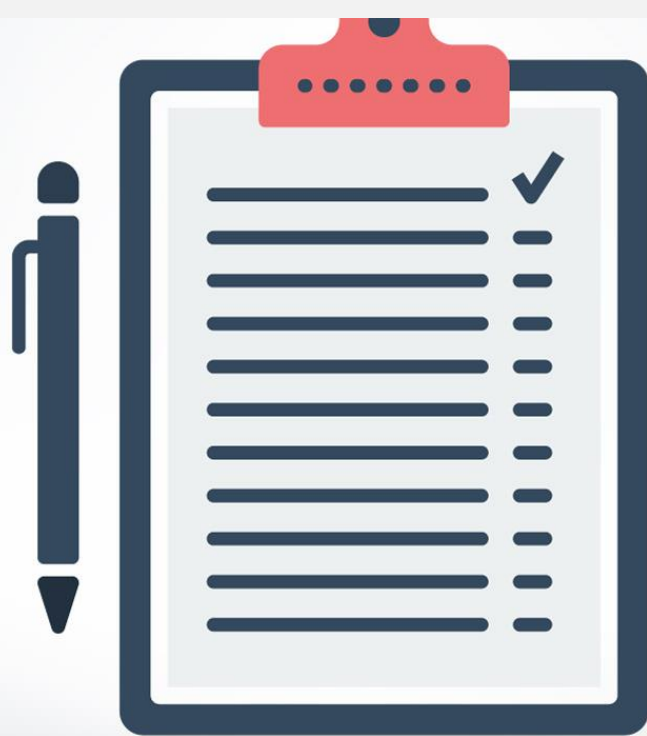
Uma lista contém itens separados por vírgulas e entre colchetes []. Listas são quase semelhantes às matrizes em C. Uma diferença é que todos os itens pertencentes a uma lista podem ser de tipos de dados diferentes.

```
list = [123, 'abcd', 10.2, 'd']    #can be an array of any data type or single data type.
list1 = ['hello', 'world']
print(list)    #will output whole list. [123, 'abcd', 10.2, 'd']
print(list[0:2])    #will output first two element of list. [123, 'abcd']
print(list1 * 2)    #will gave list1 two times. ['hello', 'world', 'hello', 'world']
print(list + list1)    #will gave concatenation of both the lists.
[123, 'abcd', 10.2, 'd', 'hello', 'world']
```

4 – Dicionários

Dicionário consiste em pares de chave-valores. É colocado entre chaves {} e os valores podem ser atribuídos e acessados usando colchetes [].

```
dic={'name':'red', 'age':10}
print(dic)    #will output all the key-value pairs. {'name':'red', 'age':10}
print(dic['name'])    #will output only value with 'name' key. 'red'
print(dic.values())    #will output list of values in dic. ['red', 10]
print(dic.keys())    #will output list of keys. ['name', 'age']
```



Agenda

Reverendo conceitos

1. Introdução ao Python
2. Noções básicas de Python
3. Variáveis, tipos de dados e operadores
4. Estrutura de Dados
5. **Loops e Condicionais**
 - I. Teste Condicional
 - II. If Statements
 - III. While Loop

Teste Condicional if/elif

Em Python você pode definir uma série de condicionais usando **if** para o primeiro, **elif** para o resto, e **else** para qualquer coisa não capturada pelos outros condicionais.

```
#Checking for equality -
    a single equal sign assigns a value to a variable.
#A double equal sign (==) checks wheter two values are equal.
university = 'NOVAIMS'
print(university == 'NOVAIMS')

course = 'Data Mining'
print(course == 'Business Intelligence')
print(course != 'Business Intelligence')

#Checking multiple conditions
#Using and tyo check multiple conditions
first_age = 30
second_age = 20
print(first_age >= 28 and second_age >= 22)
print(first_age >= 28 and second_age <= 22)

#Using or to check multiple conditions
first_age = 30
second_age = 20
print(first_age >= 28 or second_age >= 22)
print(first_age <= 28 or second_age >= 22)
```

```
number = 5

if number > 2:
    print("Number is bigger than 2.")
elif number < 2: # Optional clause (you can have multiple elifs)
    print("Number is smaller than 2.")
else: # Optional clause (you can only have one else)
    print("Number is 2.")
```

While Loop

Um loop **while** fará com que as instruções de loop sejam executadas até que a condição de loop seja interrompida. O código a seguir execute as instruções de loop no total de 4 vezes.

```
i = 0
while i < 4:
    #loop statements
    i = i + 1
```

```
#Counting to 10
current_number = 1
while current_number <=10:
    print(current_number)
    current_number +=1

#Letting the user choose when to quit
message = ""
while message != "quit":
    message = input("What's your message? ")
    print(message)
```



Real Python