

python

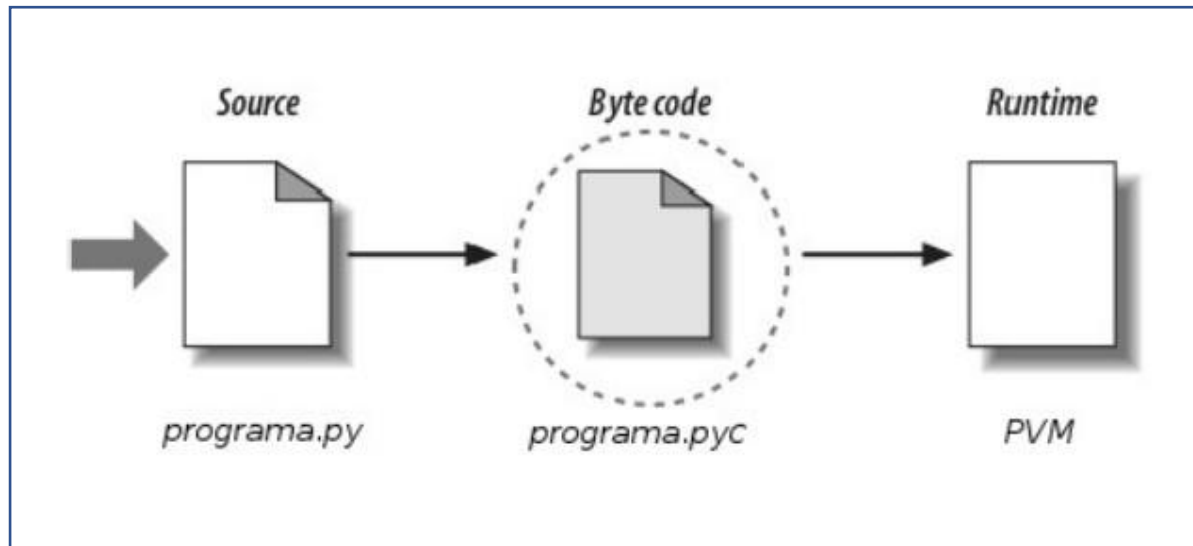
Introdução ao Python

Python é uma linguagem de programação interpretada, orientada a objetos, de alto nível e com semântica dinâmica. A simplicidade do Python reduz a manutenção de um programa. Python suporta módulos e pacotes, que encoraja a programação modularizada e reuso de códigos.



Interpretador Python

Python é uma linguagem interpretada a qual seu interpretador faz compilação instrução por instrução em tempo real de execução. Um compilador traduz linguagem Python em linguagem de máquina - código Python é traduzido em um código intermediário que deve ser executado por uma máquina virtual conhecida como PVM (Python Virtual Machine).



CPython é uma implementação da linguagem Python. Existem outras implementações como, por exemplo, **Jython** (Java), **IronPython** (C#) e **PyPy** (Python).

Breve História do Python

Python foi criada em **1990** por **Guido Van Rossum** no Centro de Matemática Stichting como uma sucessora da linguagem ABC. Guido é lembrado como o principal autor de Python, mas outros programadores ajudaram com muitas contribuições.

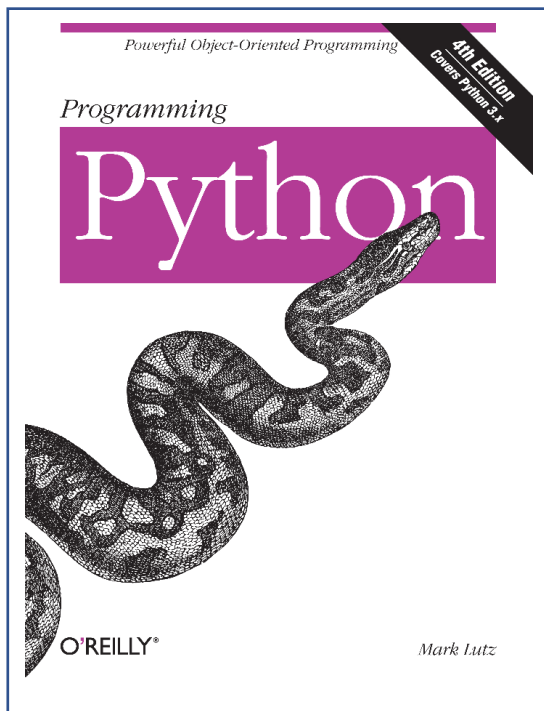
O criador da linguagem pretendia que **Python** fosse uma segunda linguagem para programadores C ou C++ e não uma **linguagem principal** para programadores. Atualmente o Python é a principal linguagem para não programadores.



Fonte: Github – Guido Van Rossum, 2022.

Breve História do Python

Python foi criada em **1990** por **Guido Van Rossum** no Centro de Matemática Stichting como uma sucessora da linguagem ABC. Guido é lembrado como o principal autor de Python, mas outros programadores ajudaram com muitas contribuições.



Fonte: Programming Python, 4th Edition.

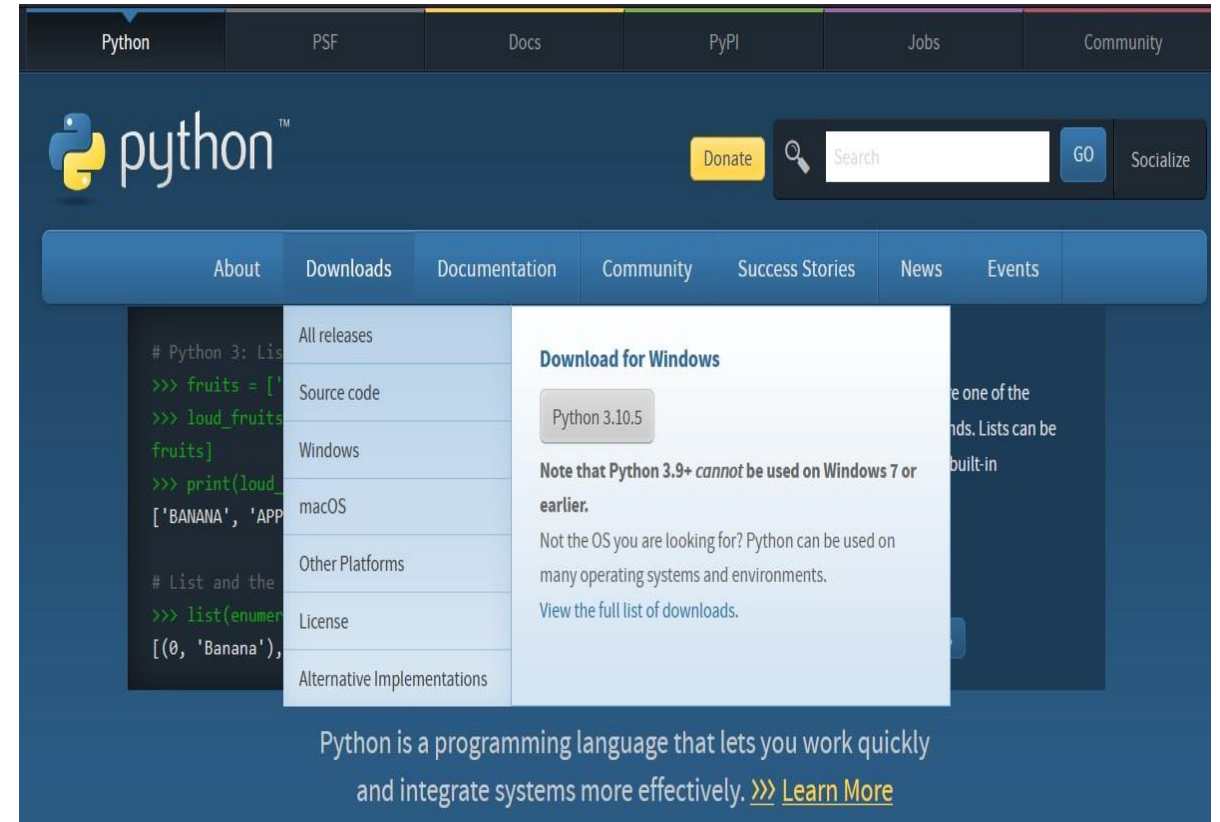


O nome da linguagem Python foi dado em homenagem a um programa de humor inglês.

Fonte: IMDb - Monty Python's Flying Circus.

Versão do Python

Neste curso utilizaremos o Google Colab, assim não é necessário instalar o Python 3 em sua máquina local. Caso queira, sempre utilize a versão do Python 3 mais recente. (Não é recomendado utilizar a versão 2 do Python). Obs: O Windows 7 não pode utilizar a versão do Python acima da 3.9++.



Documentação Python

Python » English » 3.10.5 » 3.10.5 Documentation » The Python Standard Library

Previous topic

10. Full Grammar specification

Next topic

Introduction

This Page

Report a Bug

Show Source

The Python Standard Library

While [The Python Language Reference](#) describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

- [Introduction](#)
 - [Notes on availability](#)
- [Built-in Functions](#)
- [Built-in Constants](#)
 - [Constants added by the `site` module](#)
- [Built-in Types](#)
 - [Truth Value Testing](#)
 - [Boolean Operations — `and`, `or`, `not`](#)
 - [Comparisons](#)
 - [Numeric Types — `int`, `float`, `complex`](#)

Fonte: python.org, 2022.

O que é um Programa ?

Um programa de computador ou programa informático é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador. Qualquer programa é formado pela união entre a **sintaxe** e **semântica**.

Sintaxe

```
import turtle

star = turtle.Turtle()
star.shape('turtle')

for i in range(50):
    star.forward(i)
    star.forward(100)
    star.right(144)

turtle.done()
```

VS

Semântica

```
import turtle

star = turtle.Turtle()
star.shape('turtle')

for i in range(50):
    star.forward(i)
    star.forward(100)
    star.right(144)

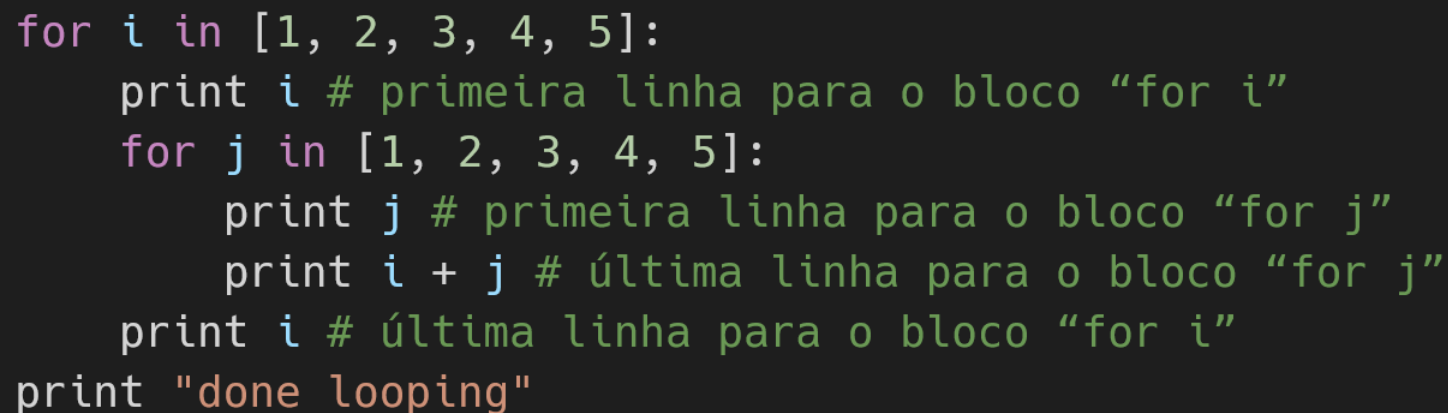
turtle.done()
```

- São as regras de como cada instrução é escrita.

- É real significado das declarações.

Indentação Python

O Python não tem “;” no final de cada linha como em outras linguagens (C++, C#, Java), isso é uma característica própria do Python. O código deve sempre ser mantida alinhado.



```
for i in [1, 2, 3, 4, 5]:  
    print i # primeira linha para o bloco "for i"  
    for j in [1, 2, 3, 4, 5]:  
        print j # primeira linha para o bloco "for j"  
        print i + j # última linha para o bloco "for j"  
    print i # última linha para o bloco "for i"  
print "done looping"
```

Primeiros passos



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/primeiros_passos_google_colab.ipynb

❏ Operadores Aritméticos

Existem em Python os seguintes operadores aritmético:

Operador	Nome	Função
+	Adição	Realiza a soma de ambos operandos.
-	Subtração	Realiza a subtração de ambos operandos.
*	Multiplicação	Realiza a multiplicação de ambos operandos.
/	Divisão	Realiza a Divisão de ambos operandos.
//	Divisão inteira	Realiza a divisão entre operandos e a parte decimal de ambos operandos.
%	Módulo	Retorna o resto da divisão de ambos operandos.
**	Exponenciação	Retorna o resultado da elevação da potência pelo outro.

Fonte: Autor próprio, 2022.

Operadores Aritméticos

```
numero_1 = 5
numero_2 = 2

soma = numero_1 + numero_2
subtracao = numero_1 - numero_2
multiplicacao = numero_1 * numero_2
divisao = numero_1 / numero_2
divisao_inteira = numero_1 // numero_2
modulo = numero_1 % numero_2
exponenciacao = numero_1 ** numero_2

print(soma) # 7
print(subtracao) # 3
print(multiplicacao) # 10
print(divisao) # 2.5
print(divisao_inteira) # 2
print(modulo) # 1
print(exponenciacao) # 25
```

Fonte: Autor próprio - Carbon, 2022.

❏ Operadores de Atribuição

Em Python existem os seguintes operadores de atribuição:

Operadores Atribuição

Operador	Nome	Função	Exemplos
=	Atribuição igual comum	Realiza atribuição	a = 5
+=	Operador atribuição igual com soma	Realiza a atribuição com a soma ()	a = a + 5 a += 5
-=	Operador atribuição igual com subtração	Realiza a atribuição com a subtração	a = a - 5 a -= 5
*=	Operador atribuição igual com multiplicação	Realiza a multiplicação de ambos operandos.	a = a + a * 4 a *= 4
/=	Operador atribuição igual com subtração	Realiza a Divisão de ambos operandos.	a = a / 10 a /= 10
%=	Operador atribuição igual com resto	Retorna o resto da divisão de ambos operandos.	a = a % 2 a %= 2

Fonte: Autor próprio, 2022.

```
numero = 5
numero += 5
print(numero) # 0 resultado será 10
numero = 5
numero -= 2
print(numero) # 0 resultado será 3
numero = 5
numero *= 5
print(numero) # 0 resultado será 5
numero = 5
numero /= 2
print(numero) # 0 resultado será 2.5
numero = 5
numero /= 2
print(numero) # 0 resultado será 2.5
numero = 5
numero %= 2
print(numero) # 0 resultado será 1
```

Fonte: Autor próprio - Carbon, 2022.

❏ Operadores de Comparação

Os operadores de comparação são usados para comparar valores, o que vai retornar True ou False, dependendo da condição. Em Python existem os seguintes operadores de comparação:

Operadores Comparação

Operador	Nome	Função	Exemplos
>	(Maior que)	Verifica se um valor é maior que outro	<code>x > 5</code>
<	(Menor que)	Verifica se um valor é menor que outro	<code>x < 5</code>
==	(Igual a)	Verifica se um valor é igual a outro	<code>x == 5</code>
!=	(Diferente de)	Verifica se um valor é diferente de outro	<code>x != 5</code>
>=	(Maior ou igual a)	Verifica se um valor é maior ou igual a outro	<code>x >= 5</code>
<=	(Menor ou igual a)	Verifica se um valor é menor ou igual a outro	<code>x <= 5</code>

Fonte: Autor próprio, 2022.

```
numero_1 = 2
numero_2 = 4
soma = numero_1 + numero_2

if soma < 10:
    print("soma não é maior que 10")
else:
    print("soma é maior ou igual a 10")
```

Fonte: Autor próprio - Carbon, 2022.

❑ Operadores Lógicos

Os operadores lógicos são usados para unir duas ou mais expressões condicionais.

Em Python existem os seguintes operadores de comparação:

Operadores Lógicos

Operador	Função	Exemplos
and	Retorna True se todas as condições for verdadeiras, caso contrário retorna False	$x > 1$ and $x < 5$
or	Retorna True se uma das condições for verdadeiras, caso contrário retorna False	$x > 1$ or $x < 5$
not	Inverte o resultado: se o resultado da expressão for True, o operador retorna false	not($x > 1$ and $x < 5$)
\wedge	Realiza a operação XOR.	$A \wedge B = (A \text{ and not } B) \text{ or } (\text{not } A \text{ and } B)$

Fonte: Autor próprio, 2022.

```
idade_mario = 21
idade_maria = 19

# OPERADOR OR
if idade_mario >= 18 or idade_maria >= 18:
    print("Pelo menos um dos dois é maior de idade")
else:
    print("Mario e Maria não são maiores de idade")

# OPERADOR AND
if idade_mario >= 18 and idade_maria >= 18:
    print("Mario e Maria são maiores de idade")
else:
    print("Pelo menos um dos dois não é maior de idade")
```

Fonte: Autor próprio - Carbon, 2022.

❏ Operadores de identidade

Os operadores de identidade servem para a comparação de objetos. Nessa comparação é verificado se eles ocupam a mesma posição na memória:

Operadores identidade

Operador	Função	Exemplos
is	Retorna True se as variáveis comparadas forem o mesmo objeto	nome is 'Marcos'
is not	Retorna True se as variáveis comparadas não forem o mesmo objeto	x is not 'Python'

Fonte: Autor próprio, 2022.

```
time_carlos = 'Santos'
time_luciano = 'Palmeiras'
time_fabricia = 'Palmeiras'

if time_carlos is time_luciano:
    print("time_carlos - time_luciano = mesmo objeto")
else:
    print("time_carlos - time_luciano = objetos diferentes")

if time_carlos is time_fabricia:
    print("time_carlos - time_fabricia = mesmo objeto")
else:
    print("time_carlos - time_fabricia = objetos diferentes")
```

Fonte: Autor próprio - Carbon, 2022.

❏ Operadores de associação

Os operadores de associação são utilizados para verificar se uma sequência contém um objeto.

Operadores associação

Operador	Função	Exemplos
in	Retorna True caso o valor seja encontrado na sequência	"2" in x
not in	Retorna True caso o valor não seja encontrado na sequência	"2" not in x

Fonte: Autor próprio, 2022.

```
frutas = ["banana", "laranja", "uva", "ameixa"]

fruta_1 = "ameixa"
fruta_2 = "melancia"

print(fruta_1 in frutas) # True
print(fruta_2 in frutas) # False
```

Fonte: Autor próprio - Carbon, 2022.

❑ Variáveis

Variáveis são pequenos espaços de memória, utilizados para armazenar e manipular dados. Em Python, os tipos de dados básicos são: tipo inteiro (armazena números inteiros), tipo float (armazena números em formato decimal), e tipo string (armazena um conjunto de caracteres). Cada variável pode armazenar apenas um tipo de dado a cada instante.

Exemplos:

```
>>> a = 10
>>> a
10
```

A variável a se torna uma variável do tipo inteiro.

```
>>> b = 1.2
>>> b
1.2
```

A variável b se torna uma variável do tipo float.

```
>>> c = "Olá Mundo"
>>> c
'Olá Mundo'
```

A variável c se torna uma variável do tipo string.

❑ Nomes de variáveis

Nome	Válido	Comentários
a3	Sim	Embora contenha um número, o nome a3 inicia com letra.
velocidade	Sim	Nome formado com letras.
velocidade90	Sim	Nome formado por letras e números, mas inicia com letras.
salario_médio	Sim	O símbolo (_) é permitido e facilita a leitura de nomes grandes.
salario médio	Não	Nomes de variáveis não podem conter espaços em branco.
_salário	Sim	O sublinha (_) é aceito em nomes de variáveis, mesmo no início.
5A	Não	Nomes de variáveis não podem começar com números.

☐ Tipos de variáveis

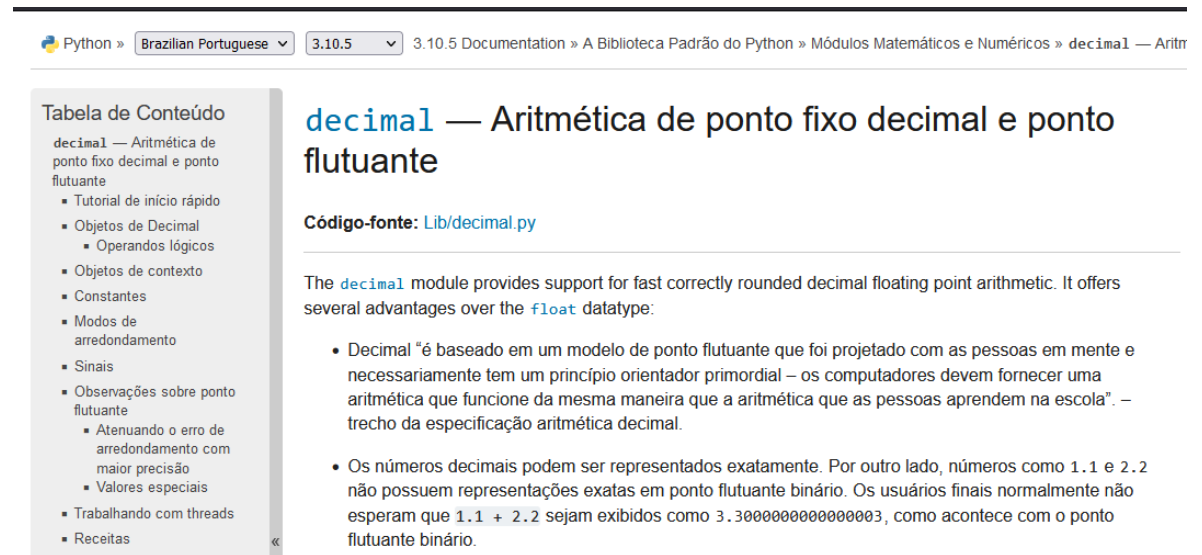
Python é uma linguagem dinamicamente tipada, o que significa que não é necessário declarar o tipo de variável ou fazer casting (mudar o tipo de variável), pois o Interpretador (CPython) se encarrega disso para nós. Isso significa também que o tipo da variável poder variar durante a execução do programa.

- **Inteiro (int)**
- **Ponto Flutuante ou Decimal (float)**
- **Tipo Complexo (complex)**
- **String (str)**
- **Boolean (bool)**
- **List (list)**
- **Tuple (tuple)**
- **Dictionary (dic)**

❏ Tipos de Numéricos

Os tipos de dados usados para números se dividem em três conjuntos:

- Inteiros (int)
- Números de ponto flutuante (float)
- Complexos (complex)



The screenshot shows the Python 3.10.5 documentation page for the `decimal` module. The page title is `decimal` — Aritmética de ponto fixo decimal e ponto flutuante. The left sidebar contains a table of contents for the `decimal` module, including sections like "Tutorial de início rápido", "Objetos de Decimal", "Objetos de contexto", "Constantes", "Modos de arredondamento", "Sinais", "Observações sobre ponto flutuante", "Trabalhando com threads", and "Receitas". The main content area starts with the heading `decimal` — Aritmética de ponto fixo decimal e ponto flutuante, followed by the source code link `Código-fonte: Lib/decimal.py`. Below this, a paragraph states: "The `decimal` module provides support for fast correctly rounded decimal floating point arithmetic. It offers several advantages over the `float` datatype:". This is followed by two bullet points: "Decimal "é baseado em um modelo de ponto flutuante que foi projetado com as pessoas em mente e necessariamente tem um princípio orientador primordial – os computadores devem fornecer uma aritmética que funcione da mesma maneira que a aritmética que as pessoas aprendem na escola". – trecho da especificação aritmética decimal." and "Os números decimais podem ser representados exatamente. Por outro lado, números como 1.1 e 2.2 não possuem representações exatas em ponto flutuante binário. Os usuários finais normalmente não esperam que 1.1 + 2.2 sejam exibidos como 3.3000000000000003, como acontece com o ponto flutuante binário."

Python Exercícios

Tipos numéricos



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_2_tipos_n%C3%BAmericos.ipynb

❑ Tipo Caracteres - String (str)

Strings são cadeia de caracteres, por exemplo “Hello World”, além disso, no Python não existe uma variável do tipo caractere (char) como, por exemplo, C, C++ ou Java. Por padrão qualquer entrada de dados dentro do Python é uma string. As strings podem ser delimitadas por aspas simples ou duplas (mas elas devem combinar):

```
>> value = input()  
>> type(value)  
<class 'str'>
```

```
single_quoted_string = 'data science'  
double_quoted_string = "data science"
```

❏ Manipulação de Strings

Em Python, existem várias funções (métodos) para manipular strings. Na tabela a seguir são apresentados os principais métodos para a manipulação as strings.

Método	Descrição	Exemplo
len()	Retorna o tamanho da string.	teste = "Apostila de Python" len(teste) 18
capitalize()	Retorna a string com a primeira letra maiúscula	a = "python" a.capitalize() 'Python'
count()	Informa quantas vezes um caractere (ou uma sequência de caracteres) aparece na string.	b = "Linguagem Python" b.count("n") 2
startswith()	Verifica se uma string inicia com uma determinada sequência.	c = "Python" c.startswith("Py") True
endswith()	Verifica se uma string termina com uma determinada sequência.	d = "Python" d.endswith("Py") False
isalnum()	Verifica se a string possui algum conteúdo alfanumérico (letra ou número).	e = "!@#\$%" e.isalnum() False
isalpha()	Verifica se a string possui apenas conteúdo alfabético.	f = "Python" f.isalpha() True

❏ Manipulação de Strings

<code>islower()</code>	Verifica se todas as letras de uma string são minúsculas.	<code>g = "pytHon"</code> <code>g.islower()</code> <code>False</code>
<code>isupper()</code>	Verifica se todas as letras de uma string são maiúsculas.	<code>h = "# PYTHON 12"</code> <code>h.isupper()</code> <code>True</code>
<code>lower()</code>	Retorna uma cópia da string trocando todas as letras para minúsculo.	<code>i = "#PYTHON 3"</code> <code>i.lower()</code> <code>'#python 3'</code>
<code>upper()</code>	Retorna uma cópia da string trocando todas as letras para maiúsculo.	<code>j = "Python"</code> <code>j.upper()</code> <code>'PYTHON'</code>
<code>swapcase()</code>	Inverte o conteúdo da string (Minúsculo / Maiúsculo).	<code>k = "Python"</code> <code>k.swapcase()</code> <code>'pYTHON'</code>
<code>title()</code>	Converte para maiúsculo todas as primeiras letras de cada palavra da string.	<code>l = "apostila de python"</code> <code>l.title()</code> <code>'Apostila De Python'</code>
<code>split()</code>	Transforma a string em uma lista, utilizando os espaços como referência.	<code>m = "cana de açúcar"</code> <code>m.split()</code> <code>['cana', 'de', 'açúcar']</code>

❏ Manipulação de Strings

<code>replace(S1, S2)</code>	Substitui na string o trecho S1 pelo trecho S2.	<code>n = "Apostila teste"</code> <code>n.replace("teste", "Python")</code> <code>'Apostila Python'</code>
<code>find()</code>	Retorna o índice da primeira ocorrência de um determinado caractere na string. Se o caractere não estiver na string retorna -1.	<code>o = "Python"</code> <code>o.find("h")</code> <code>3</code>
<code>ljust()</code>	Ajusta a string para um tamanho mínimo, acrescentando espaços à direita se necessário.	<code>p = " Python"</code> <code>p.ljust(15)</code> <code>' Python '</code>
<code>rjust()</code>	Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda se necessário.	<code>q = "Python"</code> <code>q.rjust(15)</code> <code>' Python'</code>
<code>center()</code>	Ajusta a string para um tamanho mínimo, acrescentando espaços à esquerda e à direita, se necessário.	<code>r = "Python"</code> <code>r.center(10)</code> <code>' Python '</code>
<code>lstrip()</code>	Remove todos os espaços em branco do lado esquerdo da string.	<code>s = " Python "</code> <code>s.lstrip()</code> <code>'Python '</code>
<code>rstrip()</code>	Remove todos os espaços em branco do lado direito da string.	<code>t = " Python "</code> <code>t.rstrip()</code> <code>' Python'</code>
<code>strip()</code>	Remove todos os espaços em branco da string.	<code>u = " Python "</code> <code>u.strip()</code> <code>'Python'</code>

Fonte: Autor próprio, 2022.

Python Exercícios

Strings



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_1_variaveis.ipynb

☐ Listas

Lista é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. O primeiro valor tem índice 0. Uma lista em Python é declarada da seguinte forma:

```
Nome_Lista = [ valor1, valor2, ..., valorN]
```

Uma lista pode ter valores de qualquer tipo, incluindo outras listas. Exemplo:

```
L = [3 , 'abacate' , 9.7 , [5 , 6 , 3] , "Python" , (3 , 'j')]
```

❑ Funções com Listas

Função	Descrição	Exemplo
len	retorna o tamanho da lista.	L = [1, 2, 3, 4] len(L) → 4
min	retorna o menor valor da lista.	L = [10, 40, 30, 20] min(L) → 10
max	retorna o maior valor da lista.	L = [10, 40, 30, 20] max(L) → 40
sum	retorna soma dos elementos da lista.	L = [10, 20, 30] sum(L) → 60
append	adiciona um novo valor na no final da lista.	L = [1, 2, 3] L.append(100) L → [1, 2, 3, 100]
extend	insere uma lista no final de outra lista.	L = [0, 1, 2] L.extend([3, 4, 5]) L → [0, 1, 2, 3, 4, 5]
del	remove um elemento da lista, dado seu índice.	L = [1,2,3,4] del L[1] L → [1, 3, 4]
in	verifica se um valor pertence à lista.	L = [1, 2 , 3, 4] 3 in L → True
sort()	ordena em ordem crescente	L = [3, 5, 2, 4, 1, 0] L.sort() L → [0, 1, 2, 3, 4, 5]
reverse()	inverte os elementos de uma lista.	L = [0, 1, 2, 3, 4, 5] L.reverse() L → [5, 4, 3, 2, 1, 0]

Fonte: Autor próprio, 2022.

Python Exercícios

Listas



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_4_listas.ipynb

❏ Tuplas

Tupla, assim como a Lista, é um conjunto sequencial de valores, onde cada valor é identificado através de um índice. A principal diferença entre elas é que as tuplas são imutáveis, ou seja, seus elementos não podem ser alterados. Dentre as utilidades das tuplas, destacam-se as operações de empacotamento e desempacotamento de valores. Uma tupla em Python é declarada da seguinte forma:

```
Nome_tupla = (valor1, valor2, ..., valorN)
```

❏ Tuplas

Segue abaixo exemplos operações com tuplas. Uma Tupla é estrutura muito utilizada em desempacotamento e empacotamento, que permite atribuir os elementos armazenados em uma tupla a diversas variáveis.

Operações com tuplas

```
>> T = (1,2,3,4,5)
>> print(T)
(1, 2, 3, 4, 5)
>> print(T[3])
4
>> T[3] = 8
Traceback (most recent call last):
  File "C:/Python34/teste.py", line 4, in <module>
    T[3] = 8
TypeError: 'tuple' object does not support item assignment
```

Desempacotamento tuplas

```
>> T = (10,20,30,40,50)
>> a,b,c,d,e = T
>> print("a=",a,"b=",b)
a= 10 b= 20
>> print("d+e=",d+e)
d+e= 90
```

Python Exercícios

Tuplas



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_6_tuplas.ipynb

❏ Dicionários

Dicionário é um conjunto de valores, onde cada valor é associado a uma chave de acesso. Um dicionário em Python é declarado da seguinte forma:

Exemplos declaração dicionário:

```
Nome_dicionario = { chave1 : valor1,  
                    chave2 : valor2,  
                    chave3 : valor3,  
                    .....  
                    chaveN : valorN  
                  }
```

ou

```
Nome_dicionario = dict()
```

❑ Funções com Dicionários

Abaixo são apresentados alguns comandos para a manipulação de dicionários.

Comando	Descrição	Exemplo	
del	Exclui um item informando a chave.	<pre>del D["feijão"] print(D) {'alface':3.4, 'tomate':8.8, 'arroz':17.3, 'carne':25.0}</pre>	
in	Verificar se uma chave existe no dicionário.	<pre>"batata" in D False</pre>	<pre>"alface" in D True</pre>
keys()	Obtém as chaves de um dicionário.	<pre>D.keys() dict_keys(['alface', 'tomate', 'carne', 'arroz'])</pre>	
values()	Obtém os valores de um dicionário.	<pre>D.values() dict_values([3.4, 8.8, 25.0, 17.3])</pre>	

Exemplo operação com dicionário:

```
>> Dx = {  
    2: "carro",  
    3: [4,5,6],  
    7: ('a','b'),  
    4: 173.8  
}  
>> print(Dx[7])  
( 'a', 'b' )
```

Python Exercícios

Dicionários



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_5_dicionarios.ipynb

❏ Estruturas Decisão (if else)

Uma estrutura condicional é aquela que faz uma verificação em uma determinada expressão para identificar se ela atende à condição especificada. A partir do resultado, uma ou mais instruções são executadas.

```
if (expressão_for_verdadeira):  
    executar_bloco_de_codigo()
```

```
if (expressão_for_verdadeira):  
    executar_primeiro_bloco_de_codigo()  
else:  
    executar_segundo_bloco_de_codigo()
```

❏ Estruturas Decisão (elif)

O comando elif é utilizado quando queremos realizar a verificação de outra expressão caso a primeira validação seja falsa.

```
if (expressão_for_verdadeira):  
    executar_primeiro_bloco_de_codigo()  
  
#else if  
elif (segunda_expressão_for_verdadeira):  
    executar_segundo_bloco_de_codigo()
```

❑ Estruturas Decisão (if, else, elif)



```
a = int(input("Informe um número entre 0 e 100: "))

if a > 50:
    print ("O número ", a, " é maior que 50")
elif a == 50:
    print ("O número informado é igual a 50")
else:
    print ("O número ", a, " é menor que 50")
```

Python Exercícios (If - Elif –Else)



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_7_If-Elif-Else.ipynb

❏ Estruturas Repetição

A estrutura de repetição é um recurso das linguagens de programação responsável por executar um bloco de código repetidas vezes enquanto determinada condição é atendida. No Python, possuímos dois tipos de estruturas de repetição: **for** e **while**.

Exemplo Bloco de Código - For

```
>> for i in range(4):  
    print(i)
```

```
0  
1  
2  
3
```

Exemplo Bloco de Código - While

```
>> contador = 0  
>> while contador < 5:  
    print(contador)  
    contador = contador + 1
```

```
0  
1  
2  
3  
4
```


Python Exercícios

Loops - For



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_8_For.ipynb

Python Exercícios

Loops -While



Google Colab

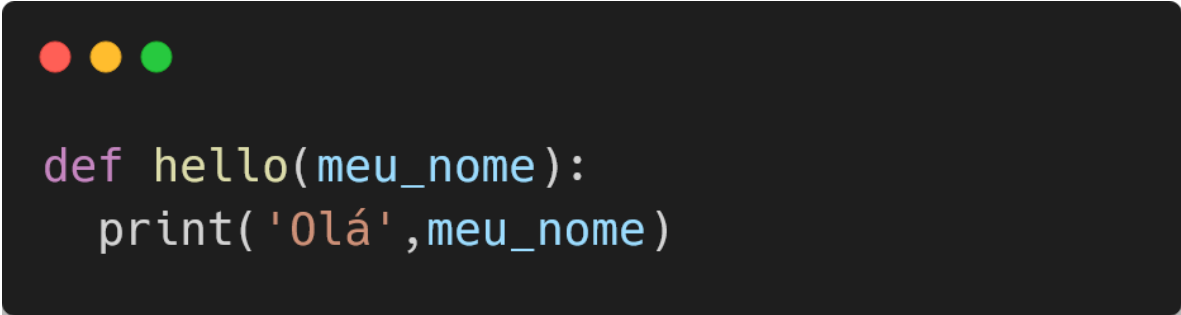
https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_8_For.ipynb

❏ Funções

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação.

Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Exemplo Declaração Função Python



```
def hello(meu_nome):  
    print('Olá', meu_nome)
```

Python Exercícios

Funções



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_11_Funcoes.ipynb

❑ Pandas

Pandas é a mais das principais bibliotecas para Ciência de Dados com Python. Pandas (<http://pandas.pydata.org>) fornece estruturas de dados adicionais para trabalhar com conjuntos de dados em Python. Os dois principais objetos dentro do Pandas são pandas DataFrame e Series. Se você quer usar Python para analisar, dividir, agrupar ou manipular conjuntos de dados, pandas uma ferramenta de valor inestimável.



Fonte: pydata.org

Pandas

Existem duas principais estruturas de dados dentro do pandas:

- DataFrame()
- Series()

`pandas.Series()`: *Series* é um array rotulado unidimensional capaz de armazenar quaisquer dados `type` (inteiros, strings, números de ponto flutuante, objetos Python, etc.). O eixo os rótulos são referidos coletivamente como o índice . O método básico para criar uma série é chamar:

Exemplo Declaração Serie do pandas

```
s = pd.Series(data, index=index)
```

Pandas

Pandas DataFrame

`pandas.DataFrame()`: DataFrame é uma estrutura de dados rotulada bidimensional com colunas de tipos potencialmente diferentes. Você pode pensar nisso como uma planilha ou SQL table ou um dict de objetos Series. Geralmente é o mais usado objeto pandas. Assim como o Series, o DataFrame aceita muitos tipos diferentes de entrada:

Exemplo Declaração pandas DataFrame()



```
df = pd.DataFrame(data, index=index)
```

Python Exercícios

Pandas



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_14_Pandas.ipynb

□ Numpy

NumPy é o pacote fundamental para computação científica em Python. É uma biblioteca Python que fornece um objeto array multidimensional, vários objetos derivados (**como matrizes e matrizes mascaradas**) e um variedade de rotinas para operações rápidas em arrays, incluindo matemática, lógica, manipulação de formas, classificação, seleção, E/S, transformadas discretas de Fourier, álgebra linear básica, estatística básica operações, simulação aleatória e muito mais.

No núcleo do pacote NumPy, está o *ndarray*. este encapsula n -dimensionais de tipos de dados homogêneos, com muitas operações sendo executadas em código compilado para desempenho.

Python Exercícios

Numpy



Google Colab

https://github.com/GabrielFonsecaNunes/data-science-with-python/blob/master/Aula%201/Notebook_15_NumPy.ipynb

Perguntas ?

Obrigado!