The background of the slide is a faded image of the Golden Gate Bridge in San Francisco, spanning across the water with its iconic towers and suspension cables.

# Introdução à Programação em Python

-Tópicos preparados para áreas de Ciências e Engenharias-

Bartolomeu J. Ubisse  
email:[bartolomeujoaquim.ubisse@gmail.com](mailto:bartolomeujoaquim.ubisse@gmail.com)

Novembro, 2020

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Nota introdutória

Bem vindo(a) ao curso de introdução à programação em python. Neste curso aprenderemos a instalar o python, IPython e o Notepad ++ para escrevermos alguns scripts. Discutiremos sobre a estrutura e tipo de dados, módulos de python, arrays, funções e depois escreveremos os nossos programas(módulos) para realizar cálculos numéricos. Devido à necessidade de se ter que usar dados de diversos tipos e fontes, aprenderemos neste curso a importar dados quer num directório nos nossos computadores quer numa rede da internet, assim como também aprenderemos a exportar os nossos resultados em um spreadsheet usando pandas. É parte também deste curso fazermos data fitting usando scipy.optimize. Cada secção deste curso é acompanhada de exemplos concretos de exploração de modo a tornar claro o assunto discutido. Este curso tem uma ficha de exercícios cujo objectivo é para começarmos a desenvolver um raciocínio aplicado à formulação e resolução de um problema concreto.

## Porque Python?

Python tem vantagens em relação a algumas linguagens (ex: Matlab, IDL e mais) em seguintes aspectos:

- 1 É open-source;
- 2 Tem uma sintaxe simples e os seus códigos são bastantes legíveis;
- 3 Tem uma comunidade extremamente alta com um grupo de pessoas diversificadas e acolhedoras;
- 4 Tem uma imensa biblioteca padrão com classes, métodos e funções para realizar essencialmente qualquer tarefa quer de natureza científica, engenharia, finanças e mais.

## Objectivos

O objectivo fundamental deste curso é ajudar os participantes a começarem a desenvolver o raciocínio aplicado à formulação e resolução de problemas computacionais usando o python.

## Expectativas

No final deste curso, espera-se que todos os participantes consigam escrever códigos básicos para cálculos numéricos e que também, caso necessário, consigam apresentar os resultados em forma de gráficos.




# Conteúdos

- 1 Nota introdutória
- 2 **Instalação de python, jupyter-notebook e Notepad++**
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Instalação de python, jupyter-notebook e Notepad++

- 1 **Baixe Anaconda** (Faça o scroll até Anaconda Installers e escolha em função do seu sistema operativo)

Anaconda Installers

Windows 	MacOS 	Linux 
Python 3.8 64-Bit Graphical Installer (466 MB) 32-Bit Graphical Installer (397 MB)	Python 3.8 64-Bit Graphical Installer (462 MB) 64-Bit Command Line Installer (454 MB)	Python 3.8 64-Bit (x86) Installer (550 MB) 64-Bit (Power8 and Power9) Installer (290 MB)

Uma vez anaconda instalado, pode abrir e instalar o **jupyter notebook** assim como o **spyder**

- 2 **Baixe python aqui!**



# Execução do python em um Terminal

Abra o terminal do seu computador e escreva **python** e prima a tecla **Enter**. No windows, é possível que escrever python resulte em um erro, neste caso, escreva simplesmente a palavra **py** e depois prima a tecla **Enter**.

```
C:\Users\user>py
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Deste modo, consegue-se ver a versão do python, i.é., python 3.7.3. O sinal >>> é o prompt.

Instale o **Notepad++** no seu computador.

## Exemplo

- Vamos escrever um script para imprimir "Hello World!" no terminal;
- Vamos imprimir "Hello World!" no jupyter notebook.

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python**
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Operadores Matemáticos e lógicos no python

Tabela 1: Operadores Matemáticos

Operador	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**
Parte inteira	//
Módulo	%

Tabela 2: Operadores Lógicos

Símbol	Significado
==	igual a
!=	não igual a
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura**
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Tipo de dados e sua estrutura

O python reconhece dois tipos de dados, i.é., **string** e **números**.

**String** - é uma série de caracteres imutáveis. Em python, tudo o que estiver entre aspas é uma string.

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

```
Out[1]: Hello!
```

Com strings podemos fazer as seguintes operações:

- Concatenação (+);
- Retirar espaços em branco (**.rstrip()**, **.lstrip()**, **.strip()**);
- Capitalizar (**.title()**);
- Uppercase (**.upper()**);
- lowercase (**.lower()**<sup>1</sup>)

---

<sup>1</sup>Estes são os métodos usados para strings. Antes do ponto antecede a variável atribuída a string em causa.

# Tipo de dados e sua estrutura - Cont.

Quanto aos números, python reconhece quatro tipos, a saber:

- ① números inteiros simples (**até 32 bits**)
- ② números de ponto flutuante (*floating point numbers*)
- ③ números complexos (**Nº com a parte imaginária**)
- ④ números inteiros longos (**acima de 32 bits**)

Os números podem ser combinados com strings desde que se garanta que sejam também strings, como por exemplo:

```
In [2]: nome = "Bongane"
        idade = 6
        print(nome+" tem "+str(idade)+" anos de idade!")
```

```
Out[2]: Bongane tem 6 anos de idade!
```



# Tipo de dados e sua estrutura - Cont.

Os números podem ser convertidos de um tipo para outro bastando para tal, especificar para que tipo se pretende.

```
In [3]: float(9) #converte-se o numero inteiro em decimal.
```

```
In [4]: int(2.6) #converte-se o numero decimal em inteiro.
```

```
In [5]: complex(3) #converte-se o numero real em complexo.
```

```
In [6]: a = "2.3245" # esta e' uma string  
b = float(a) # converte-se a string em float  
print(b, type(b))
```

```
Out[6]: 2.3245, (class 'float')
```

**Nota:** Ném todas as strings são convertíveis em números!

# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados.....

Os dados no python são colecionados em uma sequência que por sua vez caracteriza-se em:

- 1 string
- 2 lista
- 3 tuple
- 4 dicionário

## String

In [7]:

```
a = "Aqui vai uma string"  
a.split()
```

```
['Aqui', 'vai', 'uma', 'string']
```



# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - String.....

In [8]:

```
b = "Nos somos fortes. Nos vamos sobreviver."  
c = b.split(".") # separa quando apanha ponto  
print(c,len(c)) #imprime o c e o seu tamanho
```

```
['Nos somos fortes', ' Nos vamos sobreviver', ''] 3
```

O método oposto de **.split()** é **.join()**. O join já leva o argumento e é antecedido pelo símbolo e/ou expressão de junção.

In [9]:

```
". ".join(c) # junta os termos pelo ponto  
" ".join(c) # junta os termos pelo espaco vazio
```



# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - **Lista** .....

## Lista

lista é uma coleção de itens de qualquer tipo em uma dada ordem.

A forma mais simples de criar uma lista é colocar os itens entre colchetes ([ ]). Porém, para listas numéricas muitas vezes usa-se a função **range()** (**range(start,stop,step)**).

```
In [10]: list1 = ["Bongane", "Emmanuel",6,1] # #s itens
list2 = [1,7,3,8] #itens do mesmo tipo
list3 = [1,"Maputo",3.14, ["Beira", 7,[5,9]], 10]
list4 = list(range(1,10,2))
```

## Manuseamento de uma lista:

- slicing (fatiamento);
- acrescentar ou remover itens de uma lista;
- ordenar os itens quer em ordem crescente quer em decrescente

# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - Lista.....

Métodos: `.insert()`; `.append()`; `del`; `.pop()`; `sorted()` ; `.remove()`

```
In [11]: list1 = ["Bongane", "Emmanuel",6,1] # #s itens
list1.insert(0,"Ubisse")
list1.remove("Ubisse")
list1.append("Celia")
list1.pop()
del list1[1]
```

```
In [12]: list_2 = [0, 1, 2, 3, 4, 5, 6, 7, 8]
sorted(list_2, reverse = True)
```

```
In [13]: list_2[:3] # obtem-se as primeiras tres entradas
list_2[2:] #obtem -se apartir do terceiro elemento
list_2[1:4] #obtem- se do segundo ao quarto
```

# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - **Tuplas**.....

As listas servem para armazenar um conjuntos de itens que podem mudar durante a vida de um programa. No entanto, às vezes, há uma necessidade de se criar uma lista de itens que não deve ser modificada, i.é., uma lista imutável. Para tal, é necessário usar-se as **tuplas**. As tuplas são criadas colocando-se os itens entre parênteses (( )).

```
In [14]: tupla1 = ("Pemba", 2020, "Monas", "Bogor", 2011, "Muembe")
         tupla1[3]
```

```
Out[14]: 'Bogor'
```

```
In [15]: tupla1[3] = "Jakarta"
```

```
TypeError: 'tuple' object does not support
         item assignment
```



# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - Dicionários.....

Dicionário em Python é uma colecção de pares **chave-valor**. Cada chave é conectada a um valor que pode ser número, string, lista ou até mesmo um dicionário. Um dicionário é criado colocando os seus pares, chave-valor, entre chavetas (`{}`), como assegurar:

```
In [16]: dic1 = {"pais": "Mocambique", "ind": 1975,
               "cap": "Maputo"}
          dic["pais"]
```

```
Out[16]: 'Mocambique'
```

```
In [17]: dic1["pop"] = str(30) + " milhares"
```

```
{'cap': 'Maputo', 'ind': 1975, 'pais': 'Mocambique',
 'pop': '30 milhares'}
```

# Tipo de dados e sua estrutura - Cont.

.....Estrutura de dados - Dicionários.....

Uma outra forma de criar dicionário é com recurso à função **dict()**:

```
In [18]: nome_provincias = ["Maputo", "Gaza", "Inhambane",  
                           "Sofala"]  
nome_capitais = ["Maputo", "Xai-Xai", "Inhambane",  
                 "Beira"]  
dic2 = dict(zip(nome_provincias, nome_capitais))
```

```
In [19]: dic3 = dict(um=1, dois=2, tres=3)
```

Há vários métodos que visam a fazer operações sobre os dicionários. [Aqui encontram-se alguns deles!](#)

Tanto como as listas também pode-se criar um dicionário vazio que depois vai-se preenchendo.

```
In [20]: dic4 = {}  
dic4[" Mondlane "] = " Mocambique "
```

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python**
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Módulos de python

Um módulo é um arquivo que contém uma coleção de funções relacionadas.

Antes de se usar um dado módulo é preciso que se importe usando a função **import**.

Alguns módulos:

- Numpy
- Scipy
- Math and CMath
- Matplotlib
- Pandas
- Scikit

Caso se pretenda ver o conteúdo, pode se usar a função **help()**. Por exemplo:

```
>>>import numpy
>>>help(numpy)
# ou simplesmente numpy?
```



# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução**
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Controle de fluxo de execução

Os computadores são muito usados para fazer trabalhos repetitivos e retornar os resultados. Porém, é importante notar que ainda no decorrer desse trabalho, os computadores tomam decisões em função às condições impostas pelo programador.

A capacidade de tomada de decisões em programas denomina-se **fluxo de execução** e, no python, esse fluxo é regulado pelos seguintes recursos:

- 1 Condicionais (if, elif, else) e
- 2 Laços de repetição (for e while loops);

### if, elif & else

Os recursos condicionais permitem seleccionar certos blocos de código que serão ou não executados dependendo de certas condições.

```
In [21]:  if cond1:
           executa bloco1 se cond1 for verdadeira
         elif cond2:
           executa bloco2 se cond1 for falsa e
           cond2 verdadeira
         else:
           executa bloco3 se todas anteriores forem falsas
```

# Controle de fluxo de execução - Cont.

.....for & while loops.....

## for loop

O laço **for** é usado com objectos iteráveis e permite percorrer todos os elementos da sequência (listas, dicionários, strings e tuplas).

```
In [22]:  for numero in range(3):  
          print(numero)
```

veja para um dicionário

```
In [23]:  dic_for = { "ind": 1975, "pais": "Mocambique",  
                     "pop": "30 milhoes"}  
for i in dic_for:  
    print(i,dic_for[i])
```



# Controle de fluxo de execução - Cont.

.....for & while loops.....

## while loop

O laço **while** permite repetir uma operação enquanto a condição for verdadeira.

```
In [24]: x=10
         while x>0:
             x=x-1
             print(x)
```

Todas estas instruções (if, for, while) podem ser combinadas para executar uma única tarefa.

Existem outras instruções auxiliares (eg: **break**, **continue**, **pass** ) que são incorporadas nestas já vistas com o objectivo de melhorar o fluxo e ainda acautelar situações de crash do programa em caso de encontrar inconsistências, como é o caso de o denominador for zero.

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções**
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Funções

Python possui uma série de funções matemáticas ([Veja aqui!](#)), porém, nem sempre essas funções resolvem cabalmente os nossos problemas, pelo que, temos que escrever as nossas próprias funções.

Pseudocódigo da sintaxe básica da definição de uma função:

```
def <nome_da_funcao>(<argumentos>):  
    <corpo>
```

```
In [25]: def conversor_celc_kelvin(x):  
         return 273.15+x
```

```
In [26]: ##Determinacao das posicoes x e y de projectil  
#vo-v.inicial, a - angulo, t - tempo  
def posicoes_x_y(v0,a,t):  
    from math import cos, sin, radians  
    return v0*cos(radians(a))*t,  
    v0*sin(radians(a))*t-(1/2)*9.81*t**2
```

## Número variável de argumentos numa função

É possível que não se conheça a priori o número de argumentos que uma função terá, assim no lugar de argumento coloca-se **\*args**.

```
In [27]:  ##Calcula a media aritmetica de numeros  
def media(*args):  
    print(args) #para imprimir todos os args.  
    sum = 0.0  
    for i in args:  
        sum += i  
    return sum/len(args)
```

**Nota:** Recorde-se que python tem um método para achar médias, **.mean()**.



# Funções - Cont.

## .....Funções anônimas ( $\lambda$ ).....

Existe casos em que dar nome a uma função é irrelevante e, nesses casos em que o nome da função é omitido, o python faz mímica do cálculo  $\lambda$ mbda<sup>2</sup>.

A sintaxe da função  $\lambda$ mbda é:

**lambda** <argumentos>: <expressao>

Características da função  $\lambda$ mbda:

- Não aceita instruções no seu corpo;
- É escrita em uma única linha de execução.

```
In [28]: g = lambda x: x**3
```

```
In [29]: a = list(map(lambda x: x**3, [1,2,3,4]))
```

---

<sup>2</sup>O cálculo  $\lambda$ mbda trata funções como cidadãos de primeira classe, isto é, entidades que podem ser utilizadas como argumentos e retornadas como valores de outras funções.



A recursão é um método de resolução de problemas cujo princípio baseia-se em redução de um certo problema em subproblemas cada vez mais menores de tal forma que a sua resolução seja trivial. Deste modo, a função definida chama a si própria.

```
In [30]: def facto(x):  
        """Determina factorial de um numero inteiro e + """  
        if x==0:  
            return 1  
        else:  
            return x*facto(x-1)
```

**Nota:** Este código não é robusto porque não acautela situações de números negativos e também para floats. Vamos melhorar durante a nossa discussão!

Existem casos em que a função que se define para executar uma dada tarefa, uma das suas entradas é determinada por uma outra função. Neste caso, a função corrente chama uma outra função previamente definida.

Exemplo, imaginemos a seguinte combinação:

$$C_b^a = \frac{a!}{(a-b)!b!}$$

Uma vez já escrito o código para determinar factorial, podemos usar

```
In [31]: def combinacao(a,b):  
        """Determina combinacao de a em b """  
        return facto(a)/(fact(a-b)*facto(b))
```

# Funções - Cont.

## .....Input & Output.....

O python possui duas funções (dependendo da versão) internas para ler a informação de entrada do usuário:

- `input()`
- `raw_input()`

```
In [32]: def conversor_celc_kelvin():  
          x = int(input("temperatura em graus celcius:"))  
          y = 273.15+x  
          print(str(x)+" oC ---> "+str(y)+" k")
```

Sempre que se faz uma operação numérica, o resultado deve ser um número limitado e com um formato adequado, como por exemplo:

```
In [33]: a = 4  
          b = a/3  
          c = 0.0456  
          print("%.2d, %.3f, %.2e, %.2f%%" %(a,b,c,c))
```

Módulos em python são ficheiros com extensão **.py** que contém um ou mais códigos com certas finalidades. Para se utilizar um dado código no módulo, primeiro importa-se o tal módulo e depois o respectivo código. Por exemplo, imaginemos que temos um módulo cujo nome é **matrix.py** e nesse módulo tem lá uma função que calcula o determinante e o seu nome é **determinante (x)**. Para usarmos essa função, procedemos:

```
In [34]: from matrix import determinate
```

Em caso de o módulo **matrix.py** tiver muitas funções que achamos que vamos usar todas, podemos importar desta forma:

```
In [35]: from matrix import *
```

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy**
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy

# Vectores (arrays) no Numpy

.....arange & linspace.....

Numpy tem muitas funções que são usadas para a computação numérica (entra **numpy?** no ipython ou mesmo **help(numpy)** no terminal). Quanto à criação de vectores, Numpy tem muitas funções para tal, porém, para a nossa discussão, vamos considerar arange, linspace.

```
In [36]: import numpy as np  
a = np.arange(1,10,2)  
b = np.linspace(1,10,100)
```

```
In [37]: list_1 =[0, 0, 1, 4, 7, 16, 31, 64, 127]  
vec_1 = array(list_1)
```

# Vectores (arrays) no Numpy - Cont.

.....ones & zeros.....

```
In [38]: z=list(map(lambda x: x**2, np.arange(1,10)))
          zz = np.array(z)
          z1=list(map(lambda x: x**2, np.linspace(1,10,10)))
          z2 = np.array(z1)
          N_inpar = list(filter(lambda x: x%2,np.arange(10)))
          N_inp = np.array(N_inpares)
```

```
In [39]: vec_2 = np.zeros(3,dtype = int)
          vec_3 = np.ones(4)
          vec_4 = np.ones((2,3), dtype = int)
          vec_5 = np.zeros((3,2))
```





# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas**
- 10 Visualizações com Matplotlib
- 11 SciPy

**Pandas** é uma livreria do python que permite importar/exportar dados em spreadsheet e em seguida efectuar várias operações de interesse. Esta livreria possui dois tipos de dados que são **Series** e **DataFrame**. Na nossa análise vamos usar DataFrame.

Para usarmos Pandas primeiro temos que importar:

```
import pandas
```

### Importar dados no spreadsheet com extensão CSV

```
In [40]: import pandas as pd  
df = pd.read_csv("nome do ficheiro")
```

Os dados podem também ser baixados de uma nuvem, pelo que, é preciso especificar-se a **url**.

```
In [41]: import pandas as pd
url='https://setscholars.net/wp-content/uploads/2020/07/AirPassengers.csv'
df = pd.read_csv(url, sep=",")
```

## Exportar dados para spreadsheet com extensão CSV

```
In [42]: import pandas as pd
dados = resultados.to_csv("resultados.csv")
```

ou, em caso de se ter efecturado cálculos com varios parâmetros de saída, por exemplo, p-pressão, u -viscosidade e d - densidade, podemos usar DataFrame como:

```
In [43]: dados = pd.DataFrame({"pressao": p,
"viscosidade": u, "densidade": d})
dados.to_csv("resultados.csv", index = False)
```

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib**
- 11 SciPy

# Visualizações com Matplotlib

&

Matplotlib é uma livreria multiforme de visualização de dados.

- Tipo de linhas, markers e cores: [Clique!](#)

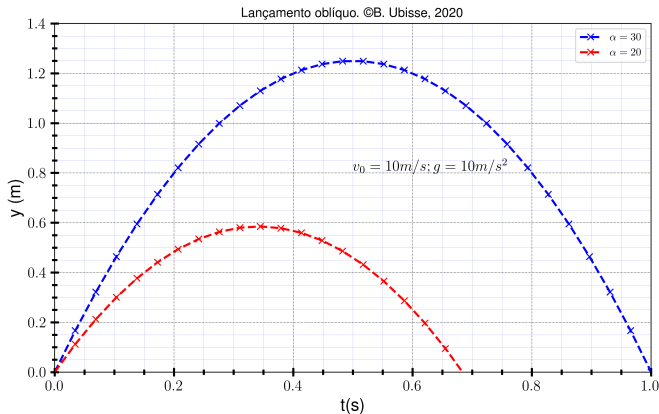


Figura 1: plot cartesiano simples



# Visualizações com Matplotlib

&

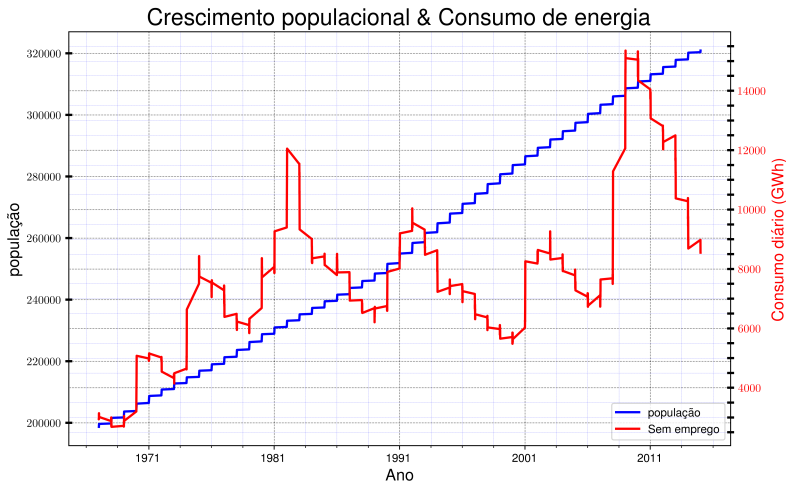


Figura 2: Dois plots com escalas diferentes

# Visualizações com Matplotlib

&

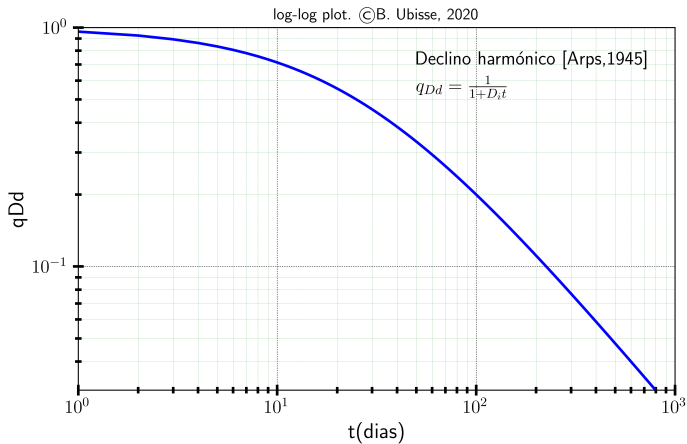


Figura 3: loglog plot

# Conteúdos

- 1 Nota introdutória
- 2 Instalação de python, jupyter-notebook e Notepad++
- 3 Operadores Matemáticos e lógicos no python
- 4 Tipo de dados e sua estrutura
- 5 Módulos de python
- 6 Controle de fluxo de execução
- 7 Funções
- 8 Vectores (arrays) no Numpy
- 9 Pandas
- 10 Visualizações com Matplotlib
- 11 SciPy



SciPy é uma livreria aberta (open-source) do Python destinado a resolver problemas científicos e com muitas funções especiais. Para ter mais detalhes de Scipy e seus pacotes, visite o seu [documento oficial aqui!](#)

Tabela 3: Alguns sub-módulos de Scipy

Sub-módulo	Função
<a href="#">scipy.cluster</a>	Quantização de vector /k-means
<a href="#">scipy.constants</a>	Constantes da Física e da Matemática
<a href="#">scipy.fftpack</a>	Transformadas de Fourier
<a href="#">scipy.integrate</a>	Routines de integração
<a href="#">scipy.interpolate</a>	Interpolação
<a href="#">scipy.linalg</a>	Routines da Algebra Linear
<a href="#">scipy.optimize</a>	Optimização
<a href="#">scipy.signal</a>	Processamento de sinais
<a href="#">scipy.special</a>	funções especiais de Matemática
<a href="#">scipy.stats</a>	Funções estatísticas

$$\text{Ex: } f(x) = \int_0^{\pi} \sin(x) dx$$

```
In [44]: from math import sin, pi
def seno(x):
    return sin(x)
inte_seno = quad(seno, 0, pi)
print(inte_seno)
```

$$\Gamma(\frac{1}{2}) = \int_0^{\infty} \exp(-t) t^{-1/2} dt$$

```
In [45]: def gama(t):
    return exp(-t)*t**(-0.5)
g = quad(gama, 0, np.inf)[0]
print(g)
```

**Nota:** A função `exp` foi importada do `math`. Porém em alguns casos pode-se ter um overflow e não se mostrar nenhum valor numérico. Por exemplo, para a integral:

$$I = \int_0^{\infty} \frac{x dx}{\exp(x) + 1} \text{ [Hornbeck, R.W. (1975). } \textit{Numeric Methods} \text{.p.173]}$$

tive esse caso e ultrapassei usando `exp` do `Numpy`, porém, o aviso do tempo de execução continua.

A livreria `scipy.integrate` possui duas rotinas poderosas para a resolução de equações diferenciais. Tais rotinas são **ode** e **odeint** e para mais detalhes pode visitar a página oficial do `scipy` ou também ver [Pine,D.(2013).*Introduction to python for science*]. Para as nossas considerações vamos ver **odeint**.

Consideremos o seg. exemplo (retirado no KREYSZING, E.(2011).*Advanced Engineering Mathematics*.10th edition. p.7)

$$\frac{dy}{dt} = -ky; \quad y(0) = 0.5; \quad k = 1.5$$

```
In [46]: import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
def n_derivative(y_der,t):
    return -1.5*y_der
t_vec = np.linspace(0,3,40)
s = odeint(n_derivative,y0 = 0.5, t = t_vec)
```

Solução:

$$\frac{dy}{dt} = -ky; \quad y(0) = 0.5; \quad k = 1.5$$

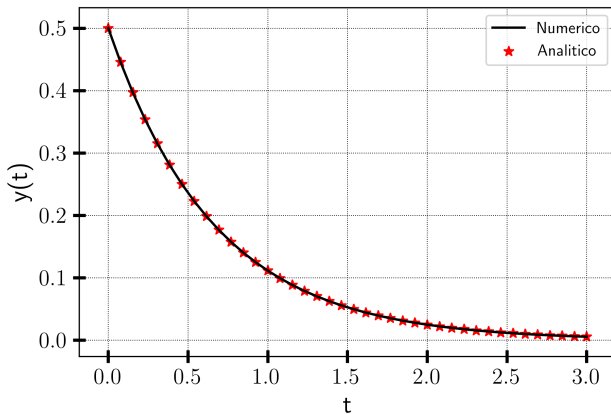


Figura 4: ODE

Quanto à ODEs da 2ª ordem de uma variável, primeiro temos que transformar a segunda derivada para um vector  $Y = (y, y')$  (Veja com maior profundidade no PINE, D.(2013).p.170).

Consideremos o seguinte exemplo de uma Oscilação amortecida

$$y'' + 2\epsilon\omega_0 y' + \omega_0^2 y = 0 \quad y(0) = 1 \quad y'(0) = 0.0$$

$$\omega_0^2 = k/m; \quad \epsilon = \frac{c}{2m\omega_0}$$

Onde:  $k$  - const.elastica da mola;  $c$  - coef. de amortecimento

```
In [47]: def calc_der(yvec, time, eps, omega):
            return (yvec[1], -2.0 * eps * omega * yvec[1]
                    - omega **2 * yvec[0])
            time_vec = np.linspace(0, 10, 100)
            yinit = (1, 0)
            yarr = odeint(calc_der, yinit, time_vec,
                          args=(eps, omega))
            kk = yarr[:,0]
            der = yarr[:,1]
```

Solução:

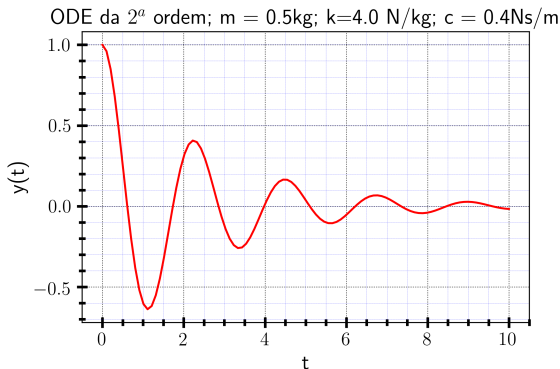


Figura 5: ODE da 2ª ordem

Consideremos a seg.equação:

$$u'' + w^2 u = 0; \quad u(0) = 1; \quad u'(0) = 0; \quad t \in [0, T]$$

Usando-se as diferenças finitas temos:

$$u_{n+1} = 2u_n - u_{n-1} - \Delta t^2 w^2 u_n$$

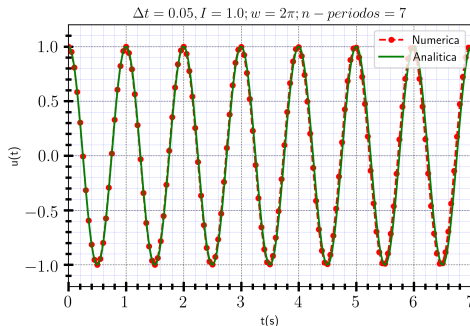


Figura 6: ODE D.finitas (Código nas sessões)



# Consideremos o ficheiro (cor\_tensao.csv) e encontremos o modelo que melhor caracteriza a relação entre as duas grandezas usando `scipy.optimize`.

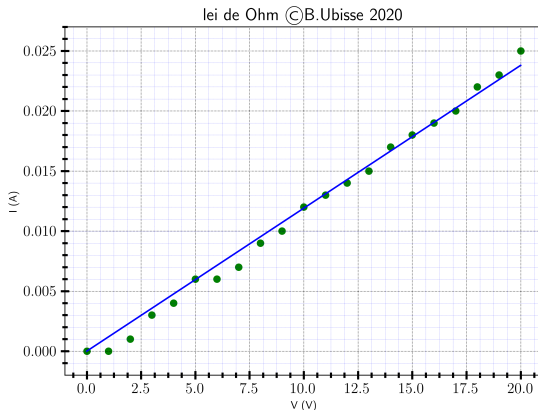


Figura 7: Curva de ajuste (Regressão linear)

$$\# 2x + \log(x) - 3 = 0$$

```
In [48]: def g(x):  
         return 2*x+mt.log(x)-3  
         resul = fsolve(g,3)  
         print(resul)
```

Podemos usar Newton-Raphson e comparar os resultados

```
In [49]: def n_raphson(fn,dfn,x,tol = 0.00001,maxiter=1000):  
         for i in range(maxiter):  
             xn = x-fn(x)/dfn(x)  
             if abs(xn-x)<tol and abs(fn(xn))<tol: break  
             x = xn  
         return x  
         y = lambda x: 2*x+log(x,10)-3  
         dy = lambda x: 2+1/(x*log(10))  
         n_raphson(y,dy,3)
```

# Considerações finais

Achamos que por agora devemos parar por aqui, porém isso não significa que esgotamos todos os assuntos de introdução à programação em python mas sim, iniciamos mais uma caminhada longa na nossa vida. Espero que cada um de nós comece a explorar o que mais lhe interessa nesta ou noutras linguagens pois, é assim que poderá melhorar a sua performance.

Muito obrigado pela coragem de aceitar que juntos podemos aprender mais uma coisa na nossa vida.

Brevemente começará um módulo de três dias sobre como escrever um documento longo (por exemplo, dissertação e texto de apoio) usando L<sup>A</sup>T<sub>E</sub>X. Caso isso também lhe interessa, faça-nos saber.

Landau, R. and Páez, M.(2018). *Computational Problems for Physics*. CRC Press

Langtangen, H.and Linge, S. (2017). *Finite Difference Computing with PDEs - A Modern Software Approach*.Springer.

Matthes, E. (2015). *Curso Intensivo de Python - Uma Introdução Prática e Baseada em Projetos à Programação*. Novatec, São Paulo, SP - Brasil.

Pine,D. (2013). *Introduction to Python for Science*. Release 0.9.23

Ramalho,L. (2015). *Python Fluente*. Novatec, São Paulo, SP - Brasil.

E todos os URLs em:

[https://www.kaggle.com/python10pm/  
plotting-with-python-learn-80-plots-step-by-step](https://www.kaggle.com/python10pm/plotting-with-python-learn-80-plots-step-by-step)