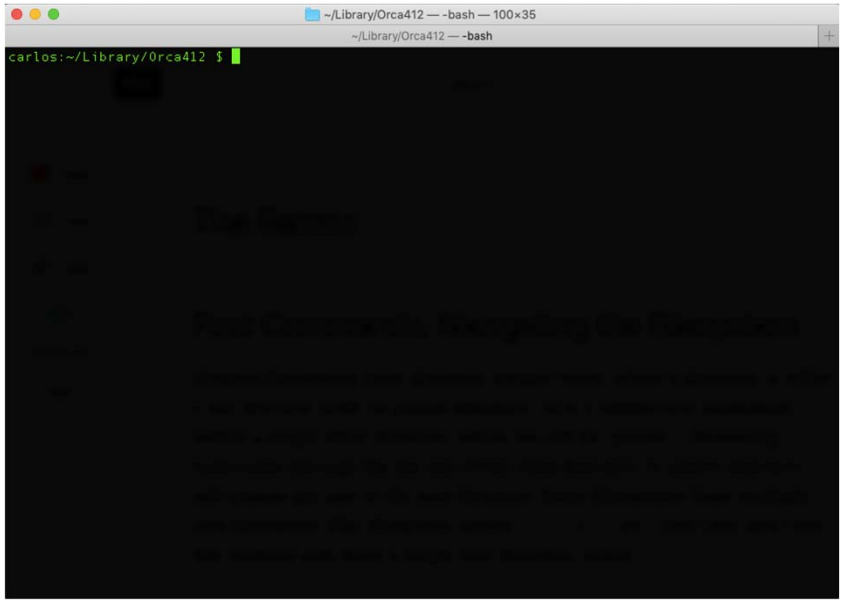
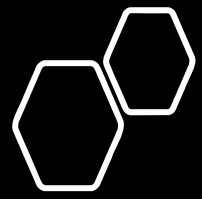

UNIX®
00011110 00011110 00011110 00011110 00011110 00011110 00011110 00011110
Linux™



INTRODUÇÃO AO AMBIENTE LINUX



PRIMEIROS COMANDOS DE NAVEGAÇÃO

pwd	Imprime o diretório de trabalho do utilizador
ls [args]	Lista o conteúdo deste diretório:
-l	Inclui informação adicional dos ficheiros
-l -h	Imprime memoria dos ficheiros em Byte, Kilobyte, Megabyte, ...
DIR/	Lista o conteúdo na pasta DIR
FILE*	Lista todos os ficheiros começados por FILE
*txt	Lista todos os ficheiros de texto terminados com extensão TXT
cd [args]	Muda para um diretório:
DIR	Muda para o pasta DIR
..	Muda para o pasta anterior
D1/D2	Muda para a pasta D2 que está dentro do diretório D1
~	Mudar para a “root” do utilizador. Igual a não utilizar argumentos
mkdir DIR	Cria uma diretória com o nome DIR

mv file1 file2	move/altera o nome de um ficheiro. Também pode ser usado para mover pastas de um local para o outro.
cp file1 file2	copia o ficheiro “file1” para “file2”
rm file	elimina o ficheiro “file”

args = argumentos.

EXERCÍCIO

a) Entrar na pasta “Documentos” e criar uma nova diretoria “Q_Computacional_2020”.

b) Entrar dentro da nova pasta e criar um segundo diretório “Pratica-0”

EDIÇÃO DE TEXTO

nano [FILE] Cria/edita o ficheiro FILE

Comandos úteis dentro da aplicação:

crtl + o Grava o ficheiro

crtl + x Sair do programa

VISUALIZAR CONTEÚDO DE FICHEIROS

cat [FILE]	Mostra todo o conteúdo do ficheiro
more [FILE]	Mostra todo o conteúdo do ficheiro com paginação
cat [FILE] | more	Equivalente à função **more**
grep 'TEXTO' [FILE]	Encontra a linha onde aparece "TEXTO" no ficheiro FILE
tail [FILE]	Mostra as últimas 10 linhas de um ficheiro
head [FILE]	Mostra as primeiras 10 linhas de um ficheiro

***Nota**: nas funções **grep**, **tail** e **head**, é possível alterar o número de linhas visualizadas utilizando "-n" antes do nome do ficheiro, onde n é o número de linhas pretendidas. No caso da função **grep**, serão mostradas a n linhas acima e abaixo da entrada "TEXTO*".

EXERCÍCIO

a) Abrir o navegador de internet e procurar no Google por “Química Computacional”.

b) Dentro da pasta que gerou no exercício anterior, crie um ficheiro “teste_nano” com o editor **nano**.

c) Copiar o resultado da pesquisa para o editor **nano**. Grave o ficheiro e saia do programa.

d) Verificar o conteúdo do ficheiro utilizando as funções **cat**, **more**, **head**, e **tail**.

e) Verifique o número de linhas copiadas em que aparece a palavra “computacional” utilizando a função **grep**.

CORRER PROGRAMAS

Program [args]	Corre o programa no terminal até terminar.
Program [args] &	Coloca o programa a funcionar em *background*.
nohup Program [args] **&**	Corre o programa, sendo possível fechar o terminal.
Program [args] > OUT	Cria um ficheiro OUT onde é gravada toda a informação que anteriormente seria visualizada no terminal.

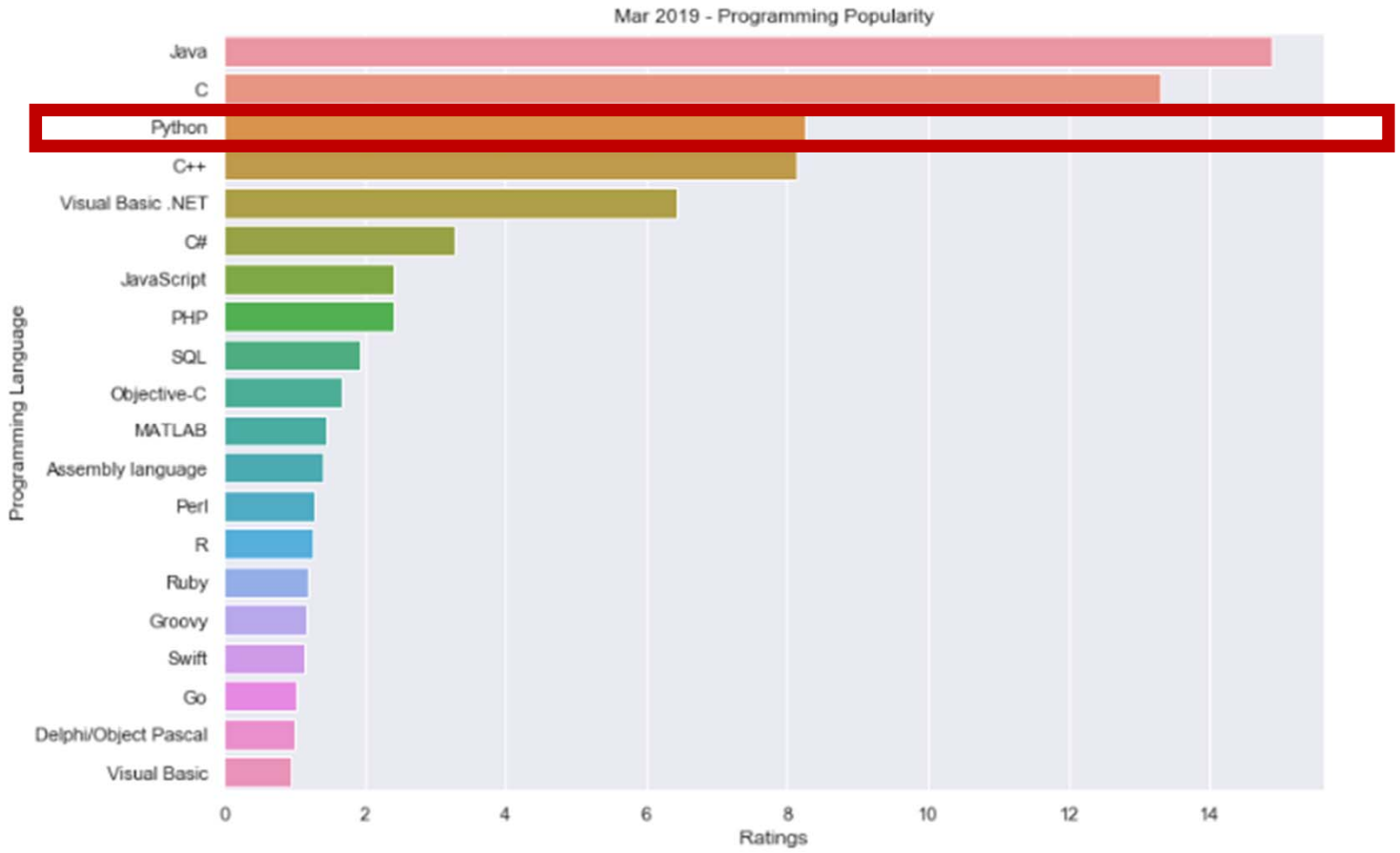
Exercício

Repita a alínea e) do exercício anterior, mas guarde o resultado da pesquisa num ficheiro com o nome “SEARCH”.


python™

Introdução à Programação em Python

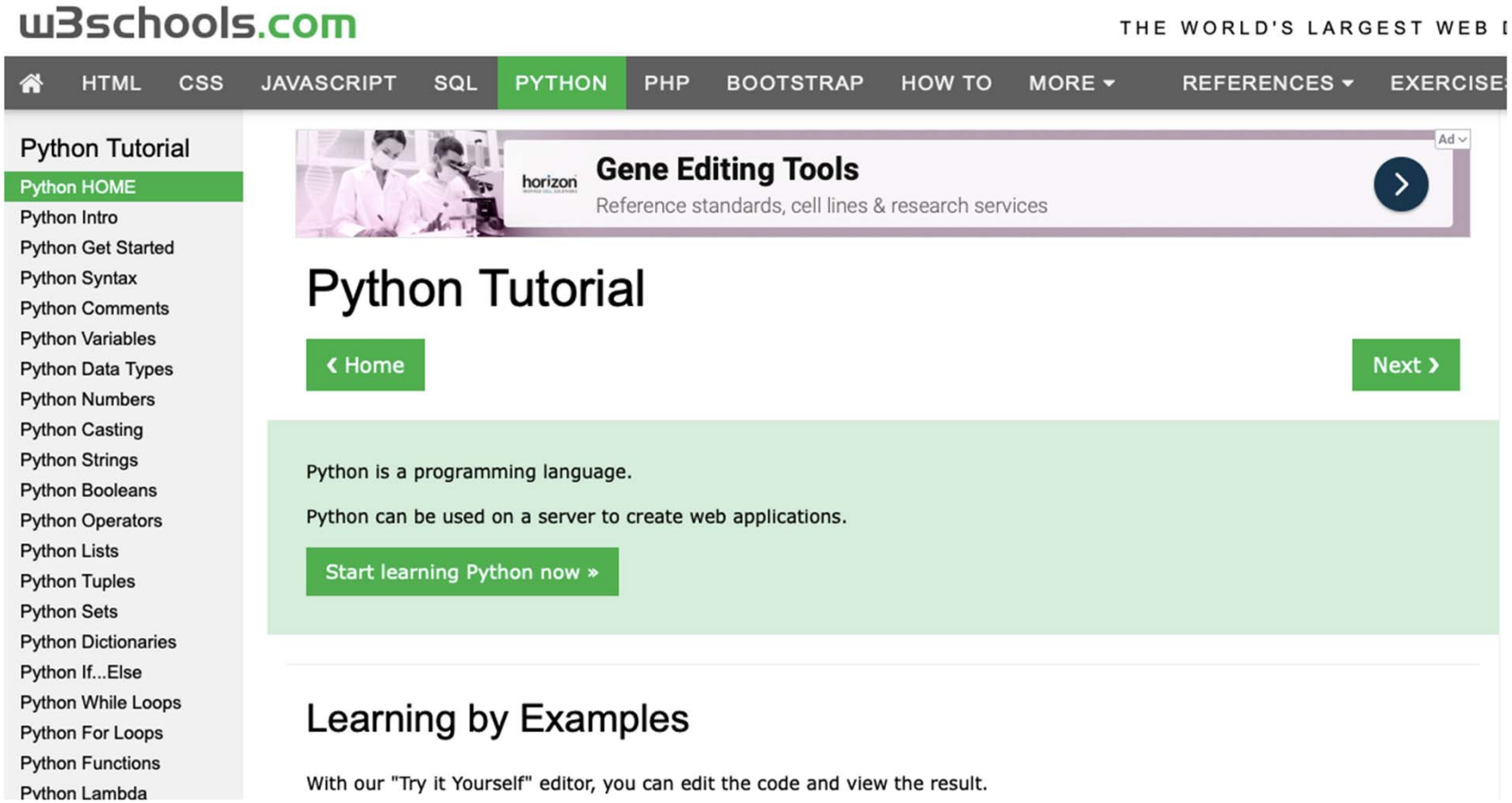
Porquê Python?



https://towardsdatascience.com/

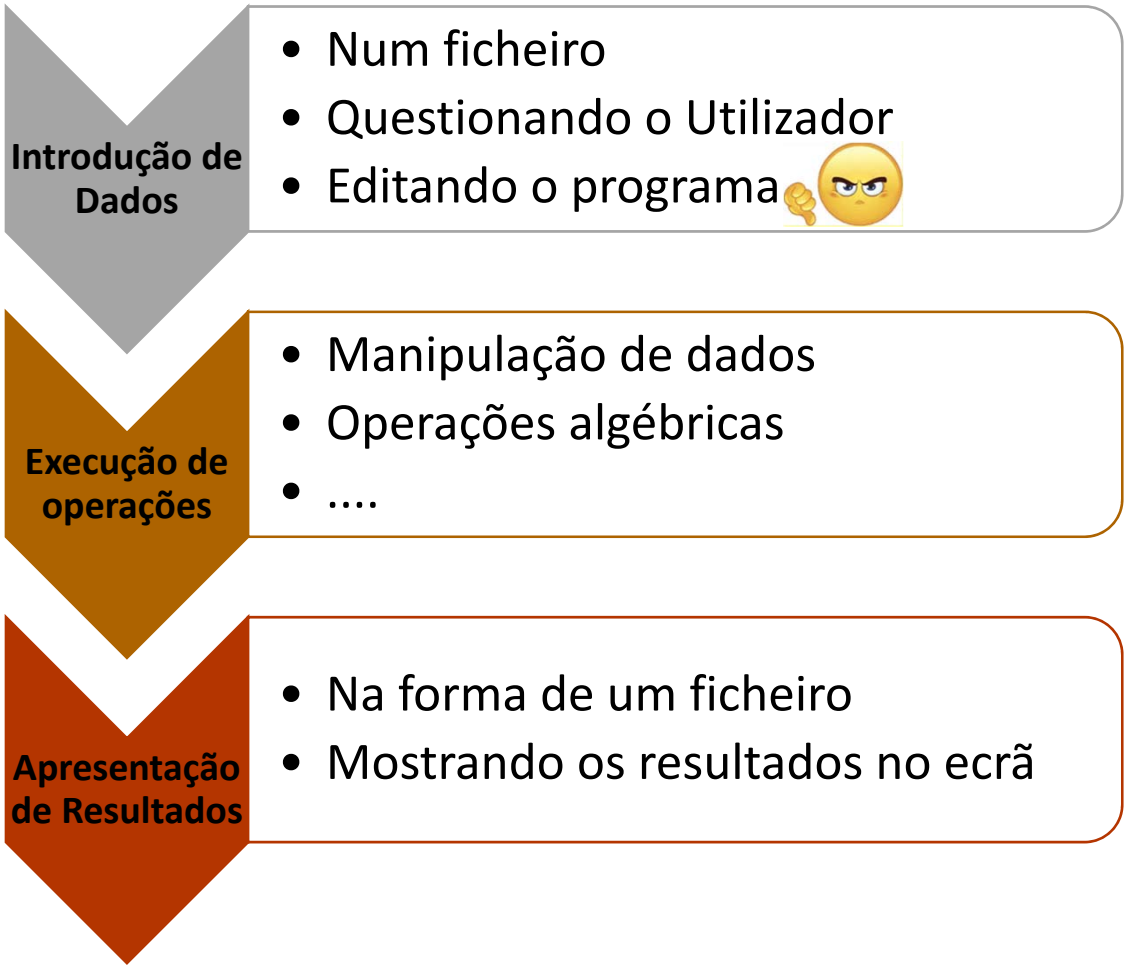
Documentação

WWW!



https://www.w3schools.com/python/default.asp

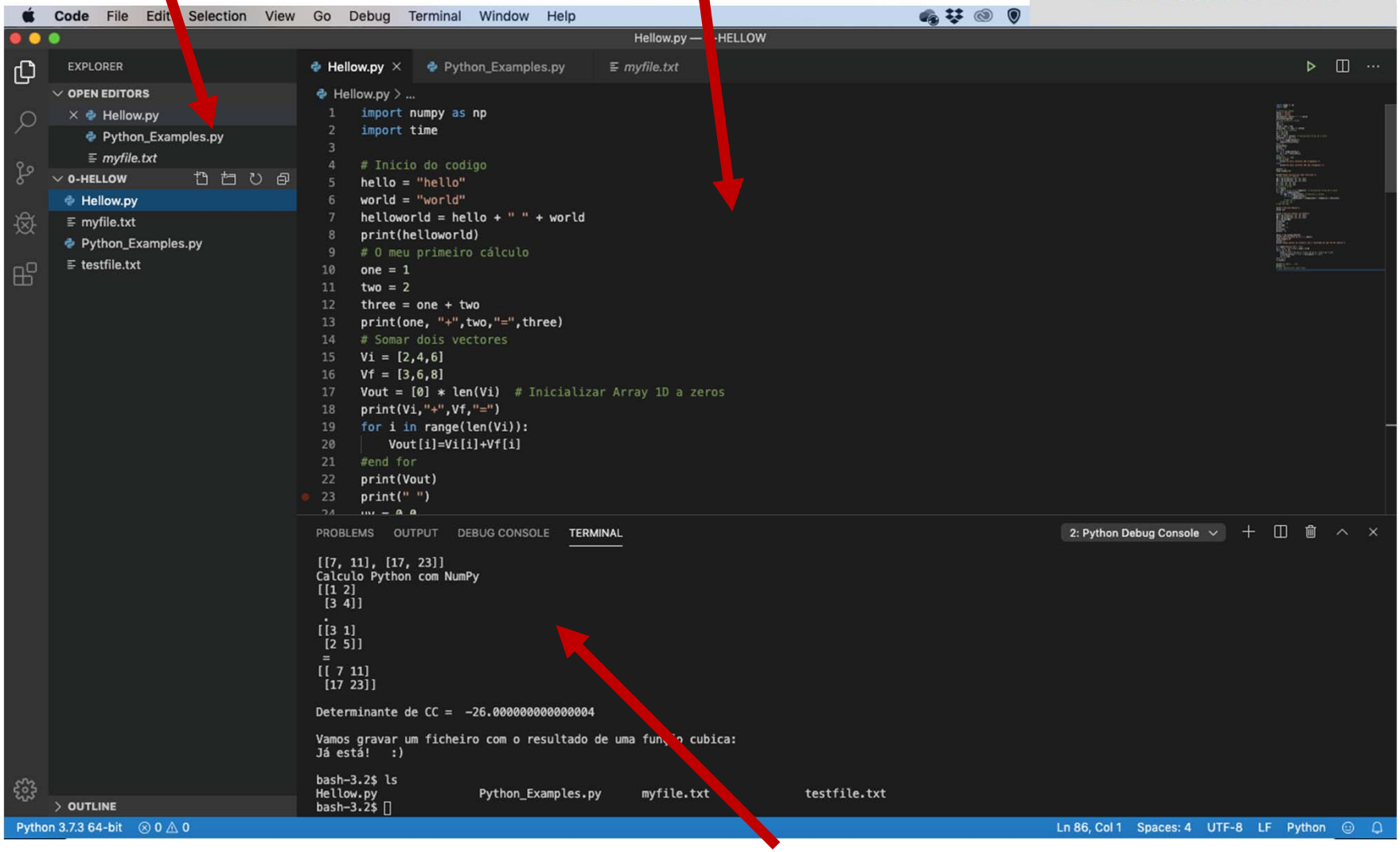
Introdução à Programação em Python



Gestor de Ficheiros

Editor de Programas

Visual Studio Code



Terminal e Consola de Erros

Primeiro Contacto com Python e VS CODE

1) Abrir o VS Code e criar um diretório “Pratica-1” dentro da pasta “Q_Computacional_2020”.
2) Criar um ficheiro “NOME_A_ESCOLHA.py” e escreva o código:

```
print("Hello World!")
```

3) Corra o programa.

ATENÇÃO: o nome do ficheiro deve terminar em “.py”

Operadores Aritméticos

Operador	Função	Exemplo
+	Adição	x + y
-	Subtração	x - y
*	Multiplicação	x * y
/	Divisão	x / y
%	Resto	x % y
**	Expoente	x ** y

Operadores de Atribuição

Operador	Exemplo	Igual a:
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
**=	x **= 3	x = x ** 3

Operadores de Comparação

Operador	Nome	Exemplo
==	Igual	x == y
!=	Diferente	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual	x >= y
<=	Menor ou igual	x <= y

Outros Operadores

Operador	Descrição	Exemplo
and	Devolve “true” se duas condições são verdadeiras	x < 5 and x < 10
or	Devolve “true” se uma das condições for verdadeiras	x < 5 or x < 4
not	Reverso do resultado, i.e., “False” se a condição é verdadeira	not(x < 5 and x < 10)
is	Devolve “true” se os valores são iguais	x is y
is not	Devolve “true” se os valores não são iguais	x is not y

Variáveis

Em Python não é necessário declarar variáveis.

Estas são inicializadas na primeira vez que são utilizadas, podendo conter texto, números inteiros, números reais, ou ser booleanas.

```
x = 5          (inteiro)
y = ”Joao”     (texto)
Z = 0.5        (número de virgula flutuante)
Z1 = Z2 = 6
Flag = True    (booleano)
```

Para pedir ao utilizador um valor para uma variável, *v*, pode usar-se:

```
v = input()
```

Arrays

Arrays são utilizados para guardar múltiplos valores numa única variável.

```
cars = ["Ford", "Volvo", "BMW"]
Numero = [1.0, 2.0, 3.1]
Numero[0] = 5.6
```

O valores, quando necessários, são “chamados” indicando a sua posição no array:

```
print(cars[1])  → Devolve "Volvo"
```

Para saber o tamanho de um array usa-se “len”:

```
len(cars) →  Devolve 3
```

Atenção: os índices dos valores num array são numerados entre 0 e N-1, onde N é o tamanho do array.

Arrays Multidimensionais

Array's NxM:

```
A = [[1, 2], [3, 4]]
```

Neste caso os valores são utilizados no código como, por exemplo:

```
print("A22 = ", A[1][1])
```
→ Devolve "4"

No entanto, na maioria das vezes não se sabe quantos valores serão colocados no array. Nestes casos, pode utilizar-se:

```
n = 2
D = [[0] for i in range(n)]
C = [[0] * n for i in range(n)]
```
→ Array com N entradas

→ Array N x N

Ciclos

Ciclos for:

```
for i in range(5)):
    Faz algo!!!
#end for
```

← **Linha comentada OPCIONAL**

Ciclos while:

```
xi = 0.0 ; xf = 3.0 ; step = 0.05
while xi <= xf:
    Faz algo !!!
    xi += step
#End while
```

← **Linha comentada OPCIONAL**

Cuidado com a Indentação!!!

Por exemplo:

```
xi = 0.0 ; xf = 3.0 ; step = 0.05
while xi <= xf:
    print (xi)
    xi += step
```

Não é igual a:

```
xi = 0.0 ; xf = 3.0 ; step = 0.05
while xi <= xf:
print (xi)
xi += step
```

Testes Lógicos

Função IF:

```
if Vi[i] > 0:
    # Faz algo se a condição é verdadeira
else:
    # Faz algo se a condição é falsa
#end if
```

Gravar Ficheiros

Gravar dados num ficheiro:

w – escreve o ficheiro
r – lê o ficheiro
a – adiciona dados a ficheiro já existente

1) Abre o ficheiro:

```
f = open("testfile.txt", "w")
```

Nome do ficheiro

2) Gravar uma linha de texto:

```
f.write(str(xi) + "\t" + str(valor) + "\n")
```

"Fecha" a linha

tab

Converte número para texto.
(a função inversa é **float**)

3) Fecha o ficheiro:

```
f.close()
```

Funções Matemáticas

É necessário importar bibliotecas de matemática. Para o efeito, no inicio do programa adicionar:

```
import math
```

Desta forma é possível utilizar funções como:

math.cos(x)	Cosseno de x, em radianos
math.sin(x)	Seno de x, em radianos
math.acos(x)	Arco cosseno de x, em radianos
math.asin(x)	Arco seno de x, em radianos
math.tan(x)	Tangente de x, em radianos
math.atan(x)	Arco tangente de x, em radianos
math.exp(x)	Exponencial de base *e*
math.log(x[, base])	Logaritmo de x (para a base *e*)
math.sqrt(*x*)	Raiz quadrada de x

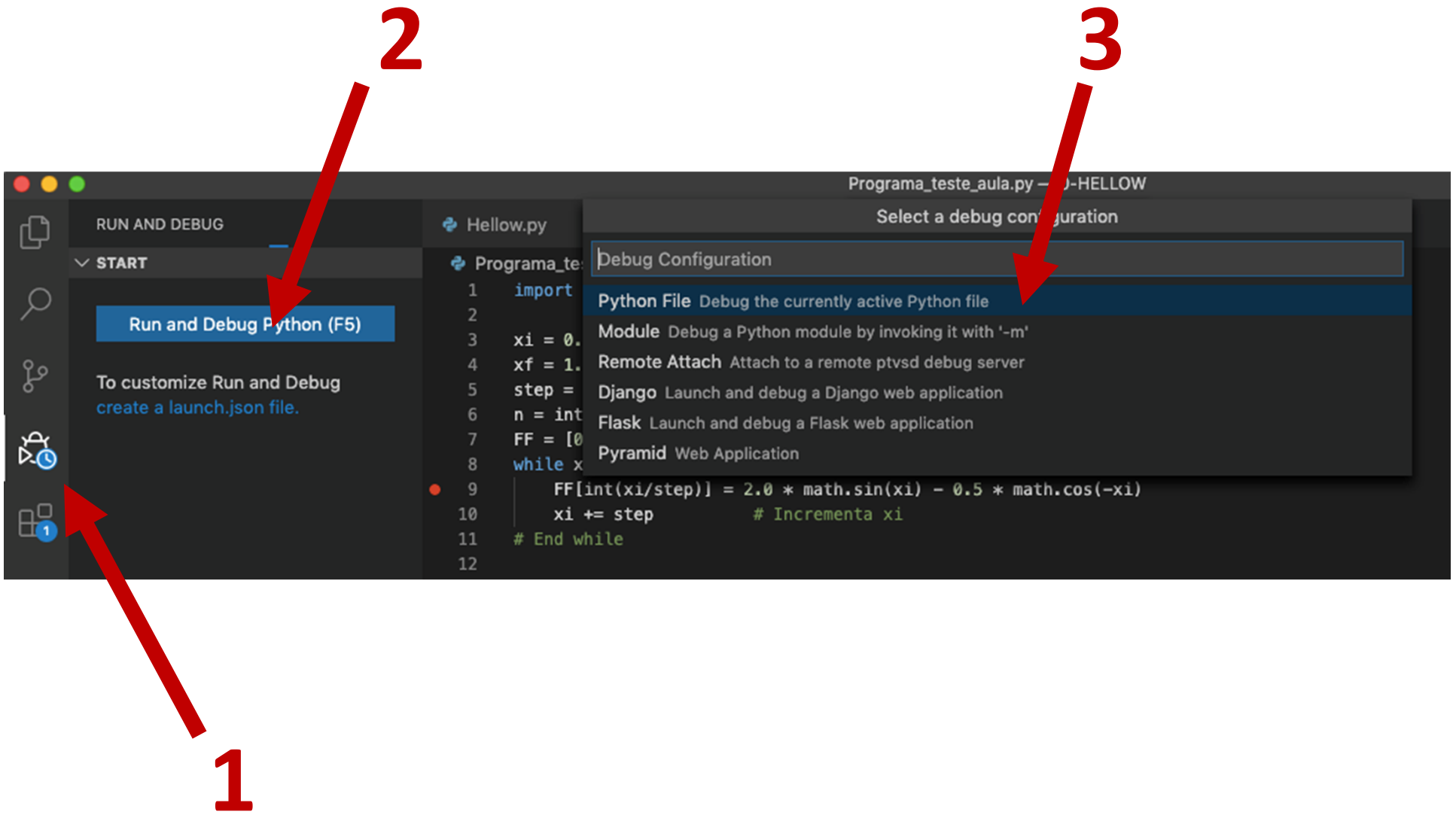
Programa de Exemplo

```
import math # Importa as bibliotecas de matemática

print ("Valor Xi = ") # Pergunta pelo valor de Xi
xi = float(input()) # Pede o valor Xi ao utilizador e converte para valor
print ("Valor Xf = ") # Pergunta pelo valor de Xf
xf = float(input()) # Pede o valor Xf ao utilizador e converte para valor
if xf < xi: # Verifica a validade dos valores introduzidos
        print ("xf deve ser menor que xi. Tente novamente!")
        quit() # Termina o programa
#end if
print ("Passo = ") # Pergunta intervalo entre valores calculados
step = float(input()) # Defini o step em cada ciclo e converte para valor
if xf-xi < step: # Verifica a validade do step introduzido
        print("Utilizar um step mais pequeno.")
        quit() # Termina o programa
#end if
n = int((xf-xi)/step) # Determina o tamanho do array que vai necessitar
FF = [0] * n # Aloca a memoria necessária para o array FF
i = 0 # Variável de controlo do ciclo
while i < n: # Calcula os valores para a função: 1.0 sin(x) - 0.5 cos(-2x)
        x = xi + i * step
        FF[i] = 1.0 * math.cos(x) - 0.5 * math.cos(-2*x)
        i += 1 # Incrementa i
# End while

# Grava os dados num fichero
f = open("testfile.txt", "w") # Abre o ficheiro
for i in range(len(FF)): # Vai correr todos os valores no array
        x = xi + i * step # Calcula o valor de x
        # Escreve cada linha com o valor de x e f(x)
        f.write(str(x) + "\t" + str(FF[i]) + "\n")
# End for
f.close() # Fecha o ficheiro
print ("DONE.")
```

DEBUG


2
3
1
RUN AND DEBUG
START
Run and Debug Python (F5)
To customize Run and Debug
create a launch.json file.
Hellow.py
Select a debug configuration
Debug Configuration
Python File Debug the currently active Python file
Module Debug a Python module by invoking it with '-m'
Remote Attach Attach to a remote ptvsd debug server
Django Launch and debug a Django web application
Flask Launch and debug a Flask web application
Pyramid Web Application
import
xi = 0.
xf = 1.
step =
n = int
FF = [0
while x
FF[int(xi/step)] = 2.0 * math.sin(xi) - 0.5 * math.cos(-xi)
xi += step # Incrementa xi
End while