

PYTHON

Nota : Torna-se necessário tirar notas a partir das explicações verbais do exemplos apresentados. Todas as linhas de Código são explicadas detalhadamente e como tal os Alunos deverão tirar notas sobre essas explicações.

“Let’s Start Programming...”

Módulos

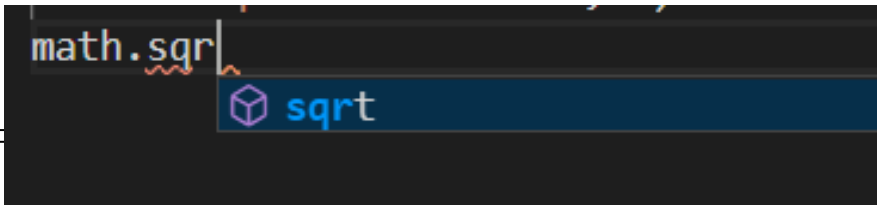
Módulos

Existem várias bibliotecas que podem ser utilizadas em Python. Essas bibliotecas (módulos) contêm diversos métodos (funções) com muita utilidade e nas mais variadas áreas, por exemplo na matemática nós utilizamos o módulo 'math'

Em Python para utilizar esses módulos utiliza-se o comando "import" seguido do nome da biblioteca que se quer importar, por exemplo no caso da biblioteca 'math' faz-se:

```
import math
```

Para utilizar os métodos do modulo basta colocar o nome do módulo seguido de um '.' e o vscode irá mostrar todos os métodos disponíveis no módulo. Basta selecionar o método pretendido e usar o mesmo com os respeito(s) parâmetro(s). Por exemplo no caso da raiz quadrada :



```
math.sqr|  
  sqrt
```

```
math.sqrt(12)
```

Módulos

Se fizer “print dir()” verifica quais são os módulos e atributos que o Python está a usar no momento. Repare que depois de utilizarmos o modulo “math” o comando ‘dir’ mostrou-o também:

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
```

```
'__name__', '__package__', '__spec__', 'a', 'b', 'c', 'math']
```



Módulos

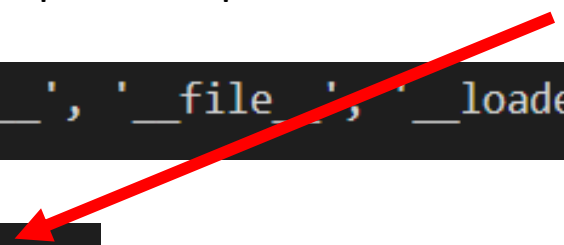
No entanto se quisermos apenas usar um dos métodos do módulo `math`, por exemplo método `'sqrt'` devermos utilizar o comando:

```
from math import sqrt
```

Nota: naturalmente que desta forma ao utilizar o método `'sqrt'` não necessita utilizar o `'math.sqrt'`, basta apenas utilizar o `'sqrt(x)'`

Ao fazer `"print(dir())"` irá verificar que em vez do módulo `'math'` o Python apenas disponibiliza o método `'sqrt'`:

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',  
['__name__', '__package__', '__spec__', 'a', 'b', 'c', 'sqrt']
```



Para remover o `sqrt`, fazer `del sqrt`.

Módulos

Para entender a questão do ‘__name__’, vamos escrever os seguintes códigos:

Ex000a.py

```
Ex000a.py > ...
1  # Módulo
2  import math
3  c = (math.sqrt(81))
4  print('Módulo:', __name__)
5  |
```

Ex000b.py

```
Ex000b.py > ...
1  # Módulo
2  import math
3  c = (math.sqrt(9))
4  print('Módulo:', __name__)
5  |
```

Ex000c.py

```
Ex000c.py > ...
1  # Módulo
2  import Ex000a
3  import Ex000b
4  print('Módulo:', __name__)
5  print(Ex000a.c)
6  print(Ex000b.c)
7  |
```

Se executar o programa Ex000a.py, verá que o print indica que o “__name__” é : “__main__”

Se executar o programa Ex000b.py, verá que o print indica que o “__name__” é : “__main__”

Módulos

Para entender a questão do ‘__name__’, vamos escrever os seguintes códigos:

Ex000a.py

```
Ex000a.py > ...
1  # Módulo
2  import math
3  c = (math.sqrt(81))
4  print('Módulo:', __name__)
5  |
```

Ex000b.py

```
Ex000b.py > ...
1  # Módulo
2  import math
3  c = (math.sqrt(9))
4  print('Módulo:', __name__)
5  |
```

Ex000c.py

```
Ex000c.py > ...
1  # Módulo
2  import Ex000a
3  import Ex000b
4  print('Módulo:', __name__)
5  print(Ex000a.c)
6  print(Ex000b.c)
7  |
```

Se executar o programa Ex000c.py, verá que o ‘__name__’ continua = ‘__main__’, no entanto os módulos que são importados (Ex000a.py e Ex000b.py, assumem o ‘__name__’ de ‘Ex000a’ e ‘Ex000b’ respetivamente. Isto porque são módulos importados pelo programa Ex000c.py que é quem tem agora o ‘__name__’ igual a ‘__main__’.

Módulos

Para entender a questão do ‘__name__’, vamos escrever os seguintes códigos:

Ex000a.py

```
Ex000a.py > ...  
1  # Módulo  
2  import math  
3  c = (math.sqrt(81))  
4  print('Módulo:', __name__)  
5
```

Ex000b.py

```
Ex000b.py > ...  
1  # Módulo  
2  import math  
3  c = (math.sqrt(9))  
4  print('Módulo:', __name__)  
5
```

Ex000c.py

```
Ex000c.py > ...  
1  # Módulo  
2  import Ex000a  
3  import Ex000b  
4  print('Módulo:', __name__)  
5  print(Ex000a.c)  
6  print(Ex000b.c)  
7
```

Portanto conclui-se que se podem incluir módulos dentro de módulos e que o programa que importa esses módulos será sempre o ‘main’ (porta de entrada da solução (programa)).

Repare-se também que foi possível utilizar as variáveis ‘c’ do módulos ‘Ex000a’ e ‘Ex000b’ cujos valores mostram-se diferentes; 9 e 3 respetivamente. (Ex000a.c e Ex000b.c)

Módulos

Para entender a questão do ‘__name__’, vamos escrever os seguintes códigos:

Ex000a.py

```
Ex000a.py > ...  
1  # Módulo  
2  import math  
3  c = (math.sqrt(81))  
4  print('Módulo:', __name__)  
5
```

Ex000b.py

```
Ex000b.py > ...  
1  # Módulo  
2  import math  
3  c = (math.sqrt(9))  
4  print('Módulo:', __name__)  
5
```

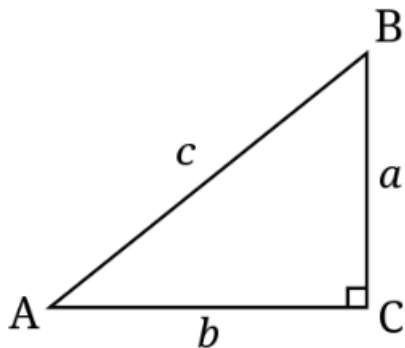
Ex000c.py

```
Ex000c.py > ...  
1  # Módulo  
2  import Ex000a  
3  import Ex000b  
4  print('Módulo:', __name__)  
5  print(Ex000a.c)  
6  print(Ex000b.c)  
7
```

Resultado do Programa Ex000c.py

```
Módulo: Ex000a  
Módulo: Ex000b  
Módulo: __main__  
9.0  
3.0
```

Teorema de Pitágoras



$$c^2 = a^2 + b^2$$

$$c = \sqrt{b^2 + a^2}, b = \sqrt{c^2 - a^2} \text{ e } a = \sqrt{c^2 - b^2}$$

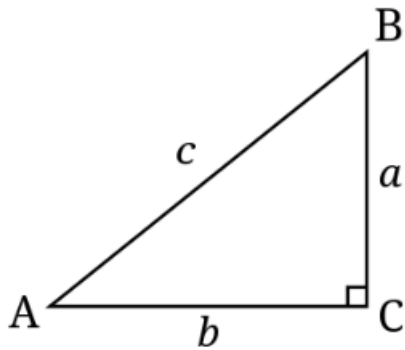


20 ‘

Exercício
(Ex001_teoremaPitagoras.py)

Com os conhecimentos que já adquiriu (pode consultar os apontamentos). Tente programar em Python o teorema de Pitágoras (Atribua os valores que entender a ‘a’ e ‘b’).

Teorema de Pitágoras



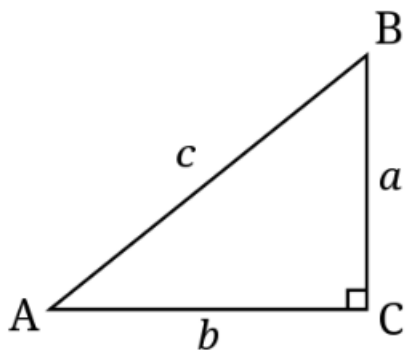
$$c^2 = a^2 + b^2$$

$$c = \sqrt{b^2 + a^2}, b = \sqrt{c^2 - a^2} \text{ e } a = \sqrt{c^2 - b^2}$$

```
1  # Ex001 - Teorema de Pitágoras
2  import math
3
4  ...
5      c = hipotenusa
6      a e b = catetos
7
8      c2 = a2 + b2
9
10     a = sql (c2 + b2);
11     b= sqr (c2 - a2) ;
12     c = sqr (a2 + b2);
13     ...
14
```

```
15  a = 10
16  b = 20
17  c = math.sqrt(a**2 + b**2)
18  print('Para cateto "a" = ' + str(a)
19        + ' e cateto "b" = ' + str(b)
20        + ' hipotenusa "c" = ' + str(c))
21
22  print('Para ceteto "a" =', a,
23        'e cateto "b" =', b,
24        'hipotenusa "c" =', c)
25
```

Teorema de Pitágoras



$$c^2 = a^2 + b^2$$

$$c = \sqrt{b^2 + a^2}, b = \sqrt{c^2 - a^2} \text{ e } a = \sqrt{c^2 - b^2}$$

```

1  # Ex001 - Teorema de Pitágoras
2  import math
3
4  ...
5      c = hipotenusa
6      a e b = catetos
7
8      c2 = a2 + b2
9
10     a = sql (c2 + b2);
11     b= sqr (c2 - a2) ;
12     c = sqr (a2 + b2);
13     ...
14

```

```

15  a = 10
16  b = 20
17  c = math.sqrt(a**2 + b**2)
18  print('Para cateto "a" = ' + str(a)
19        + ' e cateto "b" = ' + str(b)
20        + ' hipotenusa "c" = ' + str(c))
21
22  print('Para ceteto "a" =', a,
23        'e cateto "b" =', b,
24        'hipotenusa "c" =', c)
25

```

Para executar fazer 'alt + ctrl + n' ou botão



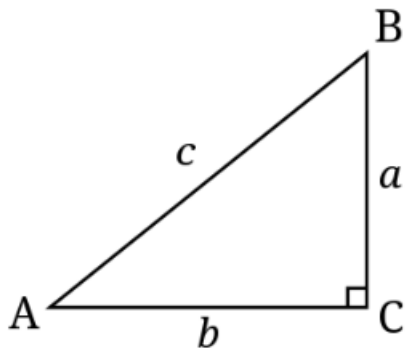
```

Para cateto "a" = 10 e cateto "b" = 20 hipotenusa "c" = 22.360679774997898
Para ceteto "a" = 10 e cateto "b" = 20 hipotenusa "c" = 22.360679774997898

[Done] exited with code=0 in 0.145 seconds

```

Teorema de Pitágoras



$$c^2 = a^2 + b^2$$

$$c = \sqrt{b^2 + a^2}, b = \sqrt{c^2 - a^2} \text{ e } a = \sqrt{c^2 - b^2}$$

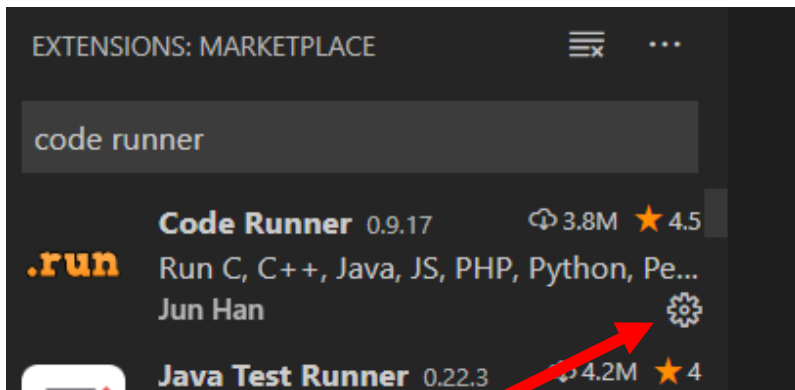
Também se podem executar o programas Python a partir do terminal.
Experimente:

- 1) menu 'view' opção 'terminal' (ctrl + c)
- 2) faça o comando 'dir' e verificará em que diretório se encontra e os respetivos ficheiros.
- 3) faça Python 'Ex001.py' e obterá o resultado do programa.

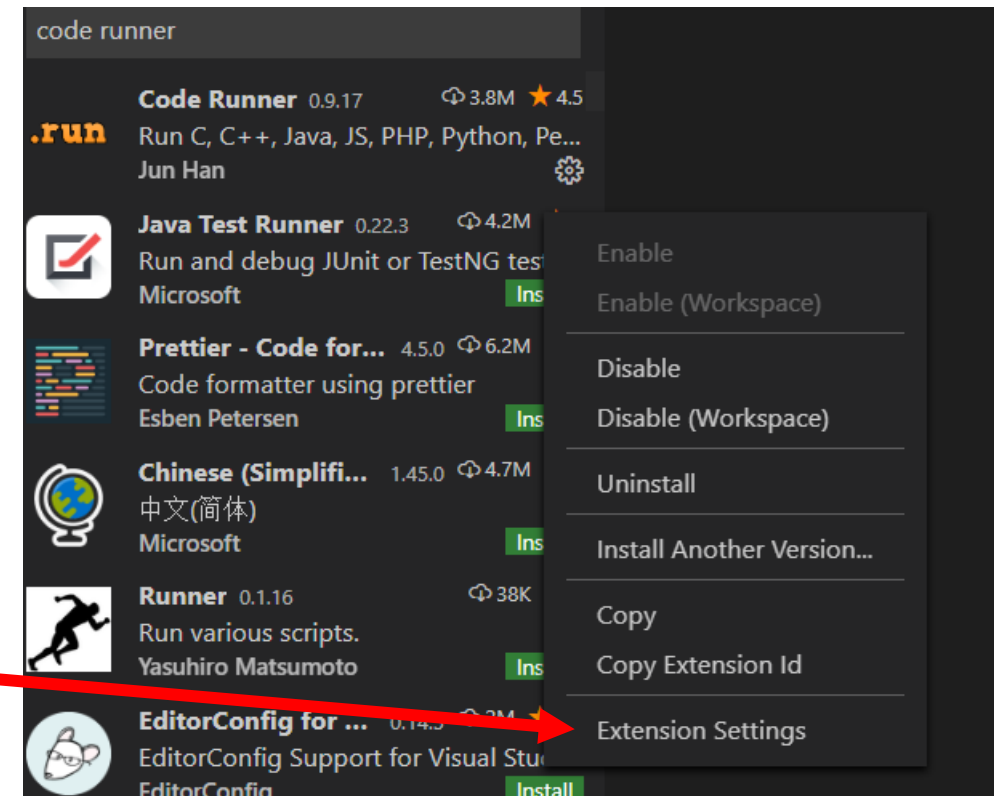
```
PS D:\Docencia\SegundoSemestre\LinguagemProgramacao1\Exercicios> python Ex001.py
Para cateto "a" = 10 e cateto "b" = 20 hipotenusa "c" = 22.360679774997898
Para ceteto "a" = 10 e cateto "b" = 20 hipotenusa "c" = 22.360679774997898
```

Eventual problema com o 'encoding' do Visual Studio Code : (utf-8)

- 1) Ir a menu 'view' e seleccionar a opção 'extensions' (ctrl + shift + x)
- 2) Procurar por 'Code Runner'



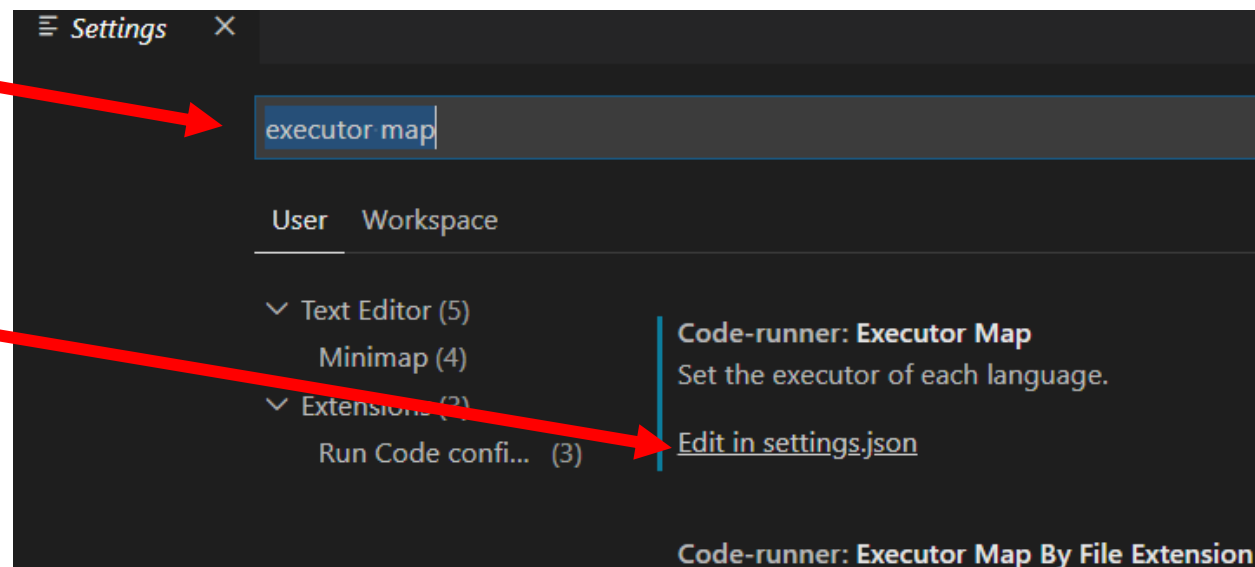
- 3) Clicar na 'roda dentada' (manage) e seleccionar a opção "Extension Settings"



Eventual problema com o 'encoding' do Visual Studio Code : (utf-8)

4) Na barra de procurar 'executor map'

5) Clicar em 'Edit in settings.json'



Eventual problema com o 'encoding' do Visual Studio Code : (utf-8)

5) Poderá encontrar um texto eventualmente com o seguinte texto :

```
{  
  "window.zoomLevel": 1,  
  "python.linting.flake8Enabled": true,  
  "files.autoSave": "afterDelay",  
  "markdown.preview.breaks": true,  
  "code-runner.executorMap": {  
  }  
}
```

Eventual problema com o 'encoding' do Visual Studio Code : (utf-8)

6) Deverá substituir o texto (do slide anterior) com este texto (configurações) :

```
{
  "git.ignoreMissingGitWarning": true,
  "python.linting.flake8Enabled": true,
  "window.zoomLevel": 0,
  "explorer.confirmDelete": false,
  "code-runner.executorMap": {

    "python": "set PYTHONIOENCODING=utf8 && python",
  },
  "python.dataScience.sendSelectionToInteractiveWindow": true,
  "explorer.confirmDragAndDrop": false,
  "git.confirmSync": false,
  "[sql]": {},
  "files.associations": {
    "*.sql": "sql"
  },
  "files.autoSave": "off",
  "editor.formatOnSave": true,
  "workbench.colorTheme": "Monokai Pro",
  "workbench.activityBar.visible": true,
  "workbench.statusBar.visible": true,
  "editor.minimap.enabled": true,
  "diffEditor.ignoreTrimWhitespace": false
}
```

Ficheiro 'ProblemaVscodeUTF8.txt' disponível na moodle, para poder copiar o texto que está dentro do mesmo (não esquecer de incluir o texto na integra).

Eventual problema com o 'encoding' do Visual Studio Code : (utf-8)

7) Gravar

PROCESSO CONCLUÍDO.

Nota adicional, em versões anterior à 3 do Python, pode-se ainda inserir uma linha de código, que na prática é uma linha de comentário especial que resolve o problema do 'encoding', forçando no programa o encoding que se pretende.

Por exemplo no caso pode-se fazer : **"# -*- coding: utf-8 -*-"** (sem as aspas) e automaticamente o Python forçará o encoding utf-8. Na versão 3 e superiores do Python não é necessário utilizar esta linha de comentário especial.

Como fazer com que o utilizador introduza dados via teclado em Python ?

```
3  # Ex001-a - Teorema de Pitágoras
4
5  # import math
6  from math import sqrt
7
8  a = float(input('Valor para cateto "a" = '))
9  b = float(input('Valor para cateto "b" = '))
10 c = sqrt(a**2 + b**2)
11
12
13 ✓ print('Para cateto "a" =', a,
14       'e cateto "b" =', b,
15       'hipotenusa "c" =', c)
16
```

- O 'input' é a instrução que o Python utiliza para capturar dados de “fora para dentro”, isto é a partir do teclado (e até de outros dispositivos de input).
- A necessidade de converter para 'float' os valores de 'a' e de 'b' prende-se com o facto de garantir que o que chega a 'c' são números.
- Se tentar correr este código a partir do vscode (output) irá verificar que o código fica a correr continuamente (terá que para-lo com “ctrl + alt + m”, por isso só deve correr código com 'input' num terminal (que pode ser do vscode), neste caso com `python Ex001a.py` ou `python .\Ex001a.py`.

Como fazer com que o utilizador introduza dados via teclado em Python ?

- Resultado:

```
3 # Ex001-a - Teorema de Pitágoras
```

```
4
```

```
5 # import math
```

```
6 from math import sqrt
```

```
Valor para cateto "a" = 10
```

```
Valor para cateto "b" = 30
```

```
Para cateto "a" = 10.0 e cateto "b" = 30.0 hipotenusa "c" = 31.622776601683793
```

```
10 c = sqrt(a**2 + b**2)
```

```
11
```

```
12
```

```
13 print('Para cateto "a" =', a,
```

```
14      'e cateto "b" =', b,
```

```
15      'hipotenusa "c" =', c)
```

```
16
```



Podemos usar ‘__name__’ (modulo) para fazer com que um determinado código execute apenas partes especificas do mesmo, caso seja por exemplo ‘__main__’. Observe os seguintes códigos :

```
6  from math import sqrt
7  |
8  a = 20
9  b = 30
10 |
11 if __name__ == '__main__':
12 |     a = float(input('Valor para cateto "a" = '))
13 |     b = float(input('Valor para cateto "b" = '))
14 |
15 c = sqrt(a**2 + b**2)
16 |
17 if __name__ == '__main__':
18 |     print('Para cateto "a" =', a,
19 |         'e cateto "b" =', b,
20 |         'hipotenusa "c" =', c)
21 |
```

- Desvalorize para já a instrução ‘if’ e entenda-a apenas como uma restrição condicional.
- Neste caso o código só irá executar as linhas 12, 13 e 18 a 20 se o nome do ‘modulo’ for igual a ‘main’ (__name__ == ‘__main__’), isto é se não se tratar de uma importação e sim do programa principal (“porta de entrada”)
- Portanto de executar este código tal como está e a partir dele próprio, todas as linhas irão ser executadas.

Nota: Os blocos de if no Python são definidos pela indentação (assunto para ver em aulas dedicadas á instrução if.



Podemos usar ‘__name__’ (modulo) para fazer com que um determinado código execute apenas partes específicas do mesmo, caso seja por exemplo ‘__main__’. Observe os seguintes códigos :

- Agora analise este segundo código :

```
6  import Ex001b
7
8  print(Ex001b.c)
9  |
```

“ex001c.py”



Podemos usar ‘__name__’ (modulo) para fazer com que um determinado código execute apenas partes específicas do mesmo, caso seja por exemplo ‘__main__’. Observe os seguintes códigos :

- Resultado:

```
6  import Ex001b
7
8  print(Ex001b.c)
9  |
```

“ex001c.py”

36.05551275463989

- Os ‘Input’ não ser verificaram porque a penas são executados pelo código quando se tratar do módulo principal e não a partir de importação (tal como codificação do código Ex001b.py).

Funções Sem Retorno

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Pyt
2  # -*- coding: utf-8 -*-
3  # Ex001-d - 
4
5
6  def calcular_Produto(fator1, fator2):
7      print(fator1, 'x', fator2, '=', fator1*fator2)
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     calcular_Produto(ft1, ft2)
14
```

- As funções definem-se a partir da palavra “def” do Python (linha 6).
- Neste caso a função não retorna valor nenhum (O ‘print’ não é ‘transportado’ pela função, é apenas uma instrução dentro da função. Por isso diz-se também que esta ‘função’ é um procedimento (linha6).
- Este procedimento recebe dois valores (fator1 e fator2). Estes valores chamam-se os argumentos da função / procedimento. Os argumentos só têm significado dentro da função.
- Igualmente à instrução “if” a função é delimitada no seu domínio através da indentação.

Funções Sem Retorno

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Pyt
2  # -*- coding: utf-8 -*-
3  # Ex001-d - 
4
5
6  def calcular_Produto(fator1, fator2):
7      print(fator1, 'x', fator2, '=', fator1*fator2)
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     calcular_Produto(ft1, ft2)
14
```

- A estrutura de uma função é definida por um nome, pode ter ou não parâmetros, um processamento e pode ter ou não um retorno (linha 6)
- Para chamar uma função em Python, invocar o nome da função (se for o caso com os respectivos parâmetros) (linha 13).
- Repare no pormenor das linhas 11 e 12, utilizaram uma conversão explícita para garantir que o valor dos “input” sejam do tipo inteiro, caso contrário poderia afetar (erro) a linha 7.
- Neste caso a função só vai ser executada se a execução pertencer ao próprio programa (__main__) (Linha 10).

Funções Sem Retorno



10 ‘

Exercício

(Ex001d.py)

Experimente usar o procedimento ('calcular_Produto') no seu terminal Python (não confundir o terminal Python com o console do Python).

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Pyt
2  # -*- coding: utf-8 -*-
3  # Ex001-d - 
4
5
6  def calcular_Produto(fator1, fator2):
7      print(fator1, 'x', fator2, '=', fator1*fator2)
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     calcular_Produto(ft1, ft2)
14
```

Funções Sem Retorno

Exercício

(Ex001d.py)

Experimente usar o procedimento ('calcular_Produto') no seu terminal Python (não confundir o terminal Python com o console do Python).

```
>>> import Ex001d
>>> Ex001d.calcular_Produto(40,20)
40 x 20 = 800
>>> 
```

Será que se pode invocar a função apenas pelo nome da função ? (Ex001d.calcular_Produto(x,y)). Se sim, como ?

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Pyt
2  # -*- coding: utf-8 -*-
3  # Ex001-d - 
4
5
6  def calcular_Produto(fator1, fator2):
7      print(fator1, 'x', fator2, '=', fator1*fator2)
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     calcular_Produto(ft1, ft2)
14
```

Funções Sem Retorno

Exercício

(Ex001d.py)

Experimente usar o procedimento ('calcular_Produto') no seu terminal Python (não confundir o terminal Python com o console do Python).

Será que se pode invocar a função apenas pelo nome da função ? (Ex001d.calcular_Produto(x,y)). Se sim, como ?

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Pyt
2  # -*- coding: utf-8 -*-
3  # Ex001-d - 
4
5
6  def calcular_Produto(fator1, fator2):
7      print(fator1, 'x', fator2, '=', fator1*fator2)
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     calcular_Produto(ft1, ft2)
14
```

```
>>> from Ex001d import calcular_Produto
>>> calcular_Produto(30,40)
30 x 40 = 1200
>>> █
```

- Sim, pode-se. Desde que o Import seja referido ao nome do módulo e especificamente à função que se pretende utilizar (calcular_Produto()).

Funções Com Retorno

```
6  def calcular_Produto(fator1, fator2):
7      return fator1*fator2
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     produto = calcular_Produto(ft1, ft2)
14     print(ft1, 'x', ft2, '=', produto)
15
```

- Funções com Retorno são iguais aos procedimentos, mas utilizam a palavra 'return' para retornar um determinado valor (linha 7).
- Neste caso temos a função 'calcular_Produto' que recebe dois parâmetros (fator1 e fator2) (linha 6) e devolve uma resultado (fator1 * fator2) para o exterior (linha 7).

Funções Com Retorno

```
6 def calcular_Produto(fator1, fator2):
7     return fator1*fator2
8
9
10 if __name__ == '__main__':
11     ft1 = int(input('Indique o fator 1: '))
12     ft2 = int(input('Indique o fator 2: '))
13     produto = calcular_Produto(ft1, ft2)
14     print(ft1, 'x', ft2, '=', produto)
15
```

```
>>> import Ex001e as myMath
>>> produto = myMath.calcular_Produto(20,30)
>>> print (produto * 3)
1800
>>> 
```

- Ao utilizar a função 'calcular_Produto' do módulo 'Ex001e' no terminal Python repare que já é possível igualar uma variável (por exemplo) ao resultado devolvido da função => 'produto = myMath.calcular_Produto(20,30).'
- De notar também que podemos usar a palavra 'as' para utilizar outro nome módulo a utilizar : 'import Ex001e as myMath' . MyMath foi um nome inventado e a partir desse momento o programa irá conhecer apenas o nome 'MyMath' e não o Original.



Enviar parâmetros a partir da execução do programa

```
5  import sys
6
7
8  def calcular_Produto(fator1, fator2):
9      return fator1*fator2
10
11
12 if __name__ == '__main__':
13     # ft1 = int(input('Indique o fator 1: '))
14     # ft2 = int(input('Indique o fator 2: '))
15     # produto = calcular_Produto(ft1, ft2)
16     # print(ft1, 'x', ft2, '=', produto)
17     ft1 = int(sys.argv[1])
18     ft2 = int(sys.argv[2])
19     produto = calcular_Produto(ft1, ft2)
20     print(ft1, 'x', ft2, '=', produto)
```

- Para utilizar parâmetros “vindos do exterior” do programa utiliza-se a função ‘argv[]’ que se encontra no módulo ‘sys’. Por isso é necessário importar o esse modulo (linha 5).
- Neste exemplo, as linhas 17 e 18 recebem os argumentos 1 e 2 (sys.argv[1] ; sys.argv[2])
- Repare que não se utilizaram as linhas 13 e 14 (input’s) para receber os argumentos para a função ‘calcular_Produto()’.
- Repare que as linhas 17 e 18 usam a conversão explícita para converter os argumentos em números inteiros, mas poderia ser para qualquer outro tipo de dados.
- Nota: O argumento 0 é o nome do script

Controlar / Validar o Envio de Parâmetros

```
4 import sys
5
6
7 def calcular_Produto(fator1, fator2):
8     return fator1*fator2
9
10
11 if __name__ == '__main__':
12
13     if len(sys.argv) < 3:
14         print("""São necessários dois argumentos\
15 Sintaxe: "" + sys.argv[0] + " Arg1 Arg2")
16     else:
17         ft1 = int(sys.argv[1])
18         ft2 = int(sys.argv[2])
19         produto = calcular_Produto(ft1, ft2)
20         print(ft1, 'x', ft2, '=', produto)
```

- Já foi referido que para usar o controlo dos argumentos externos, devemos utilizar o módulo 'sys' (linha 4)
- Neste exemplo verifica-se que o controlo do número de argumentos é controlado pela função 'len' (Length) que determina o número de elementos de um array, e que neste caso devolve o número total de argumentos inseridos pelo utilizador.
- De notar que o argumento exterior de índice '0' é o nome do script.
- Portanto neste caso para controlar / validar o número de argumentos introduzidos pelo utilizador deve-se usar o comando 'if' / 'else'. (linhas 13 e 16)



HELP

```
4 import sys
5
6
7 def calcular_Produto(fator1, fator2):
8     return fator1*fator2
9
10
11 if __name__ == '__main__':
12
13     if len(sys.argv) < 3:
14         print("""São necessários dois argumentos\
15 Sintaxe: "" + sys.argv[0] + " Arg1 Arg2")
16     else:
17         ft1 = int(sys.argv[1])
18         ft2 = int(sys.argv[2])
19         produto = calcular_Produto(ft1, ft2)
20         print(ft1, 'x', ft2, '=', produto)
```

- Como fazer com que a linha 13,14 e 15 sejam substituídas por um procedimento ?



```
4 import sys
5
6
7 def calcular_Produto(fator1, fator2):
8     return fator1*fator2
9
10
11 def ajuda_Utilizador():
12     print("""São necessários dois argumentos\
13     Sintaxe: """ + sys.argv[0] + " Arg1 Arg2")
14
15
```

```
16 if __name__ == '__main__':
17
18     if len(sys.argv) < 3:
19         ajuda_Utilizador()
20         sys.exit(1)
21         print("ESTE PRINT NUNCA ACONTECE...")
22     else:
23         ft1 = int(sys.argv[1])
24         ft2 = int(sys.argv[2])
25         produto = calcular_Produto(ft1, ft2)
26         print(ft1, 'x', ft2, '=', produto)
```

Como fazer com que a linha 13,14 e 15 sejam substituídas por um procedimento ?

- Cria-se uma função (procedimento) chamada por exemplo “ajuda_Utilizador” (linha11) e transfere-se para dentro dela a mensagem de erro que estavam nas linhas 14 e 15 do script anterior.
- Repare na utilização da instrução “if ... else...” do Python, em que cada grupo de código está indentado.



```
4 import sys
5
6
7 def calcular_Produto(fator1, fator2):
8     return fator1*fator2
9
10
11 def ajuda_Utilizador():
12     print("""São necessários dois argumentos\
13     Sintaxe: """ + sys.argv[0] + " Arg1 Arg2")
14
15
```

```
16 if __name__ == '__main__':
17
18     if len(sys.argv) < 3:
19         ajuda_Utilizador()
20         sys.exit(1)
21         print("ESTE PRINT NUNCA ACONTECE...")
22     else:
23         ft1 = int(sys.argv[1])
24         ft2 = int(sys.argv[2])
25         produto = calcular_Produto(ft1, ft2)
26         print(ft1, 'x', ft2, '=', produto)
```

- Claro que dentro do “if”, deve-se chamar essa função que se criou (ajuda_Utilizador) (Linha 19)
- Nota: Para chamar uma função que não tenha argumentos, é necessário também, colocar os parenteses.
- Se desejarmos mesmo sair do script quando existe uma mensagem de erro, devemos utilizar o método ‘exit’ so módulo ‘sys’ (linha 20)

Tratamento de Exceções

O Script ao lado representa uma função que verifica se o argumento 'numero' se trata efetivamente de um número, retornando apenas um valor "True" ou "False".

```
4 import sys
5
6
7 def eUmNumero(numero):
8     try:
9         val = int(numero)
10        return True
11    except ValueError:
12        try:
13            val = float(numero)
14            return True
15        except ValueError:
16            return False
```

- x

```
17
18
19 def calcular_Produto(fator1, fator2):
20     return fator1 * fator2
21
22
23 def ajuda_Utilizador():
24     print("""São necessários dois argumentos\
25     Sintaxe: """ + sys.argv[0] + " Arg1 Arg2")
26
27
```

- x



ISI A

```
28 if __name__ == '__main__':
29
30     if len(sys.argv) < 3:
31         ajuda_Utilizador()
32         sys.exit(1)
33         print("ESTE PRINT NUNCA ACONTECE...")
34     elif not eUmNumero(sys.argv[1]) or not eUmNumero(sys.argv[2]):
35         print("Todos os Argumentos devem ser numéricos.")
36         sys.exit(1)
37     else:
38         ft1 = float(sys.argv[1])
39         ft2 = float(sys.argv[2])
40         produto = calcular_Produto(ft1, ft2)
41         print(ft1, 'x', ft2, '=', produto)
42
```

- x



```
4 import sys
5
6
7 def eUmNumero(numero):
8     try:
9         val = int(numero)
10        return True
11    except ValueError:
12        try:
13            val = float(numero)
14            return True
15        except ValueError:
16            return False
```

```
17
18
19 def calcular_Produto(fator1, fator2):
20     return fator1 * fator2
21
22
23 def ajuda_Utilizador():
24     print("""São necessários dois argumentos\
25     Sintaxe: "" + sys.argv[0] + " Arg1 Arg2")
26
27
```

```
28 if __name__ == '__main__':
29
30     if len(sys.argv) < 3:
31         ajuda_Utilizador()
32         sys.exit(1)
33         print("ESTE PRINT NUNCA ACONTECE...")
34     elif not eUmNumero(sys.argv[1]) or not eUmNumero(sys.argv[2]):
35         print("Todos os Argumentos devem ser numéricos.")
36         sys.exit(1)
37     else:
38         ft1 = float(sys.argv[1])
39         ft2 = float(sys.argv[2])
40         produto = calcular_Produto(ft1, ft2)
41         print(ft1, 'x', ft2, '=', produto)
42
```


Estruturas de Controlo (Condicional)

(IF ... ELIF... ELSE)

Executar uma programa em Python para transformar uma nota quantitativa em nota qualitativa de acordo com as equivalências da tabela em baixo: (utilizar apenas números inteiros)

Elevado	→	[19 a 20]
Muito Bom	→	[17 a 19[
Bom	→	[14 a 17[
Suficiente	→	[10 a 14[
Insuficiente	→	[07 a 10[
Mau	→	[03 a 07[
Muito Mau	→	[00 a 03[
Nota Incorreta	→	menor que 0 ou > 20

O Software só deverá solicitar a nota ao utilizador se o programa se encontrar no módulo principal, caso contrário o software de ser utilizado como módulo. Utilize uma função para a conversão da nota.



```
1  #!C:\Users\fbent\AppData\Local\Programs
2  # Conversor de Notas Quantitativas em Q
3
4
5  def converter_Nota_Quantitativa(nota):
6      nota = float(nota)
7      if nota > 20 or nota < 0:
8          return 'Nota Incorreta.'
9      elif nota >= 19:
10         return 'Elevado'
11     elif nota >= 17:
12         return 'Muito Bom'
13     elif nota >= 14:
14         return 'Bom'
15     elif nota >= 10:
```

As linhas 5 a 22 representam a função que converte a nota quantitativa em qualitativa. A função “converter_Nota_Quantitativa(x)” recebe um valor (nota) que será convertido através do encadeamento de “if” e “else” (elif), e no fim devolve a conversão respetiva (“return”).



```
16         return 'Suficiente'
17     elif nota >= 7:
18         return 'Insuficiente'
19     elif nota >= 3:
20         return 'Mau'
21     else:
22         return 'Muito Mau'
23
24
25 if __name__ == '__main__':
26     nota = input('Nota :')
27     nota_Convertida = converter_Nota_Quantitativa(nota)
28     print(nota_Convertida)
29
```

```
Nota :30
Nota Incorreta.
```

```
Nota :16
Bom
```

Da linha 25 para a frente, o programa usa um “if” para se certificar que se encontra ou não um módulo “main” (linha 25), em caso afirmativo e só nesse caso é solicitado ao utilizador que introduza uma nota (linha 26), a qual é utilizada na chamada da função de conversão (linha 27), finalmente é impresso o resultado.



```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # Utilização do Conversor de Notas Quantitativas em Qualitativas
3
4  from Ex002a import converter_Nota_Quantitativa
5
6  print(converter_Nota_Quantitativa(15))
7  |
```

Bom

[Done] exited with code=0 in 0.084 seconds

Neste caso o programa ‘Ex002a’ (Conversor de notas) é utilizado como módulo (linha 4) num outro programa (Ex002a1) de onde utiliza apenas a função “converter_Nota_Quantitativa(x)” diretamente num “print” (linha 6).

Se a linha 4 fosse substituída pelo comando “import Ex002a” apenas, o programa teria o mesmo comportamento, uma vez que o teste ao “__main__” garantia que o programa “Ex002a” não solicitaria o valor ao utilizador, uma vez que não se encontrava como “__main__”.



Outras Formas:

```
1  #!C:\Users\fbent\AppData\Local\Program
2  # Outras Utilizações do IF
3
4
5  def conversor_Nota_Quantitativa(nota):
6
7      if 0 <= nota < 3:
8          return "Muito Mau"
9      elif nota in range(3, 7):
10         return "Mau"
11     elif nota in range(7, 10):
12         return "Insuficiente"
13     elif nota in range(10, 14):
```

```
14         return "Suficiente"
15     elif nota in range(14, 17):
16         return "Bom"
17     elif nota in range(17, 19):
18         return "Muito Bom"
19     elif nota in range(19, 21):
20         return "Elevado"
21     else:
22         return "Nota incorreta."
23
24
25 print(conversor_Nota_Quantitativa(21))
26
```

Neste exemplo faz-se a mesma coisa com outras técnicas. Repare-se na linha 7, em que se testa se a nota é maior ou igual que 0 e também menor que 3.

Outras Formas:

```
1  #!C:\Users\fbent\AppData\Local\Program
2  # Outras Utilizações do IF
3
4
5  def conversor_Nota_Quantitativa(nota):
6
7      if 0 <= nota < 3:
8          return "Muito Mau"
9      elif nota in range(3, 7):
10         return "Mau"
11     elif nota in range(7, 10):
12         return "Insuficiente"
13     elif nota in range(10, 14):
```

```
14         return "Suficiente"
15     elif nota in range(14, 17):
16         return "Bom"
17     elif nota in range(17, 19):
18         return "Muito Bom"
19     elif nota in range(19, 21):
20         return "Elevado"
21     else:
22         return "Nota incorreta."
23
24
25 print(conversor_Nota_Quantitativa(21))
26
```

Também na linha 9 começa-se a utilizar um novo conceito (range) que serve para definir um intervalo (range(x,y)), em que x pertence ao intervalo e y não pertence ao intervalo ([x,y[). Por fim de pois de tudo testado se o valor continuar a não ser encontrado, então é dado como valor incorreto (linha 21 e 22).

Estruturas de Controlo (Repetição)

(WHILE)



Estrutura Base do While :

```
while (condição):  
    <indentação> instrução 1  
    <indentação> instrução 2  
    <indentação> instrução ...n
```

Nota: enquanto as instruções se mantiverem na mesma “indentação”, o Python considera essas instruções como pertencentes a um bloco único (tal como acontece na instrução “IF”)



Exercício (explicado):

Estrutura Base do While :

```
while (condição):  
    <indentação> instrução 1  
    <indentação> instrução 2  
    <indentação> instrução ...n
```

À semelhança de um mini trabalho já antes solicitado, pretende-se um programa em Python que simule a extração de bolas de um jogo de bingo.

Parta do principio que o jogo de bingo tem 90 bolas. As bolas devem ser impressas uma por linha aleatoriamente no ecrã, e não podem naturalmente sair bolas repetidas.

O jogo termina imediatamente após a nonagésima bola ter saído, isto é, após todas as 90 bolas extraídas.

Deve privilegiar a utilização de estruturas de repetição “WHILE”

Informação técnica adicional: Para extrair um número inteiro aleatório entre um intervalo numérico, deverá utilizar uma função chamada “randint” de um módulo de nome “random”. A sintaxe é **randint(x,y)**, em que ‘x’ é o limite inferior e ‘y’ o limite superior. Esta função devolverá um numero aleatório entre os referidos limites.

Pode consultar toda a informação que é disponibilizada na moodle e o tempo para este exercício é de 30 minutos.



```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # WHILE
3
4  from random import randint
5  ''' Poderá utilizar, mas o pretendido
6     neste exercicio é privilegiar o "while"...
7  bolas = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
8           16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
9           31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
10          46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
11          61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
12          76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]
13  '''
```

Tal como exercício sugere, deve-se utilizar o módulo “random” e especificamente uma das suas funções “randint” (linha 4). Esta função serve para extração de números inteiros aleatórios.

As linhas 5 a 13 estão comentadas, apenas para informar que também se poderia criar uma estrutura de dados da forma que se apresenta na linha 7, mas o exercício obriga a privilegiar as estruturas “while”...



```
14  bolas = []
15  y = 0
16  while y < 90:
17      y = y+1
18      bolas.append(y)
19
20  while len(bolas) > 0:
21      x = randint(1, 90)
22      while (x in bolas):
23          print(bolas.pop(bolas.index(x)))
24  print("\nTerminado!!")
25
```

A linha 14, declara uma estrutura do tipo “list”, para se poder utilizar o método “pop”. (linhas 14 e 23). A linha 14 criar uma lista vazia.

A linhas 15 a 18 inicializa uma variável (y) para preencher a lista com os números do jogo sequencialmente (de 1 a 90). Por isso inicializa-se com 0 e incrementa-se a variável (linha 17) através de um ciclo “while” (linha 16 a 18) enquanto o y for menor que 90 (linha 16). Este ciclo garante que quando começa o jogo a lista contém todas as bolas necessárias, disponíveis para serem extraídas.



```
14  bolas = []
15  y = 0
16  while y < 90:
17      y = y+1
18      bolas.append(y)
19
20  while len(bolas) > 0:
21      x = randint(1, 90)
22      while (x in bolas):
23          print(bolas.pop(bolas.index(x)))
24  print("\nTerminado!!")
25
```

A linha 20 inicia um novo ciclo enquanto o numero de elementos da lista (bolas) for maior que zero, o que significa que ainda existem bolas por extrair.

O referido bloco começa por extrair um numero entre 1 e 90(linha 21) e guarda-o na variável x.

Há que levar em conta que só se deve considerar o numero aleatório da linha 21 se e só se o mesmo constar na lista (linha 22). O que significa que o número não vai sair de forma repetida, pois caso o numero não exista na lista (bolas) então o ciclo deve escolher aleatoriamente outra bola (linha 21).

Se o numero x realmente estiver contido na lista “bolas” então significa que ainda não saiu e então pode ser “impresso” no ecrã (linha23), é também automaticamente removido da lista bolas (linha 21).



```
14  bolas = []
15  y = 0
16  while y < 90:
17      y = y+1
18      bolas.append(y)
19
20  while len(bolas) > 0:
21      x = randint(1, 90)
22      while (x in bolas):
23          print(bolas.pop(bolas.index(x)))
24  print("\nTerminado!!")
25
```

Enquanto existirem elementos disponíveis na lista “bolas” o ciclo while (linha 20) fica ativo, saindo apenas do ciclo quando a lista “bolas” ficar vazia. Neste caso é executada a linha 24, encerrando o programa, ao imprimir a mensagem “Terminado” depois de um enter (linha 24 “\n”).

Nota: a linha 22 poderia ser substituída por um ‘IF’, mas não esquecer que o exercício solicita para privilegiar o “while”...

Este é um exercício meramente académico para explicar a instrução while. No entanto, neste caso se já tivéssemos já estudado a instrução “FOR”, poderia também ser utilizada. (tomar nota da explicação verbal).

Estruturas de Controlo (Repetição) (FOR)



```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # FOR
3  from random import randint
4  bolas = []
5  for n in range(1, 91):
6      bolas.append(n)
7  while len(bolas) > 0:
8      x = randint(1, 90)
9      while (x in bolas):
10         print(bolas.pop(bolas.index(x)))
11  print("\nTerminado!!")
12
```

A instrução 'for' tem uma utilização muito simples. Repare na linha 5, que reduziu linhas de código do exercício anterior.

A lógica desta instrução é "iterar", para isso necessita-se de uma variável (variável iteradora), que neste caso é representada por 'n'.



```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # FOR
3  from random import randint
4  bolas = []
5  for n in range(1, 91):
6      bolas.append(n)
7  while len(bolas) > 0:
8      x = randint(1, 90)
9      while (x in bolas):
10         print(bolas.pop(bolas.index(x)))
11  print("\nTerminado!!")
12
```

Utiliza-se também a função 'range' já estudada anteriormente. O 'range' estabelece um domínio (range (a,b)). Neste caso representado por 'range(1,91)'.

Nota: Não esquecer que a função range tem os intervalos '[...[' , o que quer dizer que o limite superior não é considerado no intervalo, neste caso vai de 1 a 90, isto é, o '91' não é considerado ([1,91[).



A função 'range' pode ainda ser utilizada apenas com o limite superior (range(90)) e neste caso a iteração irá ser feita entre '0' e '89'. Analise o seguinte código baseado no código do exercício anterior...

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # FOR
3  from random import randint
4  bolas = []
5  for n in range(90):
6      bolas.append(n)
7  while len(bolas) > 0:
8      x = randint(1, 90)
9      while (x in bolas):
10         print(bolas.pop(bolas.index(x)))
11 print("\nTerminado!!")
12
```

Porque será que o programa tal como é apresentado em cima, entra em 'loop' ? ... e onde entra em 'loop' ?



A função 'range' pode ainda ser utilizada apenas com o limite superior (range(90)) e neste caso a iteração irá ser feita entre '0' e '89'. Analise o seguinte código baseado no código do exercício anterior...

```
1  #!C:\Users\fbent\AppData\Local\Programs\Python\Python37\python
2  # FOR
3  from random import randint
4  bolas = []
5  for n in range(90):
6      bolas.append(n)
7  while len(bolas) > 0:
8      x = randint(1, 90)
9      while (x in bolas):
10         print(bolas.pop(bolas.index(x)))
11 print("\nTerminado!!")
12
```

Porque será que o programa tal como é apresentado em cima, entra em 'loop' ? ... e onde entra em 'loop' ?

Porque a linha 5, carrega a lista com 90 números, mas que vão de 0 a 89. O programa entra em 'loop' na linha 7, porque o numero '0' nunca é extraído, uma vez que a linha 8, apenas escolhe números entre 1 e 90. Então o '0' ficará sempre na lista, e a linha 7 é testada incondicionalmente, porque a condição será sempre maior que 0.



A função 'range' pode ainda também determinar um “salto” entre cada elemento do seu domínio. 'range(X,Y,Z)', em que, 'X' é o limite inferior e pertence ao intervalo, o 'Y' é o limite superior e não pertence ao intervalo, e 'Z' é o 'salto' (passo) que se dá entre cada um dos elementos entre 'X' e 'Y'.

Faça um programa em Python que produza o resultado mostrado na imagem ao lado.

```
Numero: 0
0
Numero: 2
0 1 2
Numero: 4
0 1 2 3 4
Numero: 6
0 1 2 3 4 5 6
Numero: 8
0 1 2 3 4 5 6 7 8
Numero: 10
0 1 2 3 4 5 6 7 8 9 10
```



Faça um programa em Python que produza o resultado mostrado na imagem ao lado.

```
4  for n in range(0, 11, 2):
5      print(f'Numero: {n}')
6      for m in range(0, n+1):
7          print(m, ' ', end='')
8      print()
9
```

Neste exercício, recorreu-se a um for encadeado (dois 'for', um dentro de outro), linhas 4 e 6.

A linha 4, inicia um ciclo de iterador 'n' de '0' a 10, sendo que cada elemento deve dar um passo '2' (neste caso, '0,2,4,6,8,10').

```
Numero: 0
0
Numero: 2
0 1 2
Numero: 4
0 1 2 3 4
Numero: 6
0 1 2 3 4 5 6
Numero: 8
0 1 2 3 4 5 6 7 8
Numero: 10
0 1 2 3 4 5 6 7 8 9 10
```

Como a ideia seria sequenciar uma lista de números (de 0 até ao limite do próprio número) por cada elemento do primeiro ciclo. Então cria-se outro ciclo (linha 6) de iterador 'm', de '0' até ao número que estiver em causa do primeiro ciclo (n), mas + '1' porque já se sabe que o limite máximo do range não pertence ao intervalo.

Assim para o primeiro número, '0', é impresso o próprio número (linha 5) com uma formatação, e imediatamente a seguir, inicia-se um novo ciclo que de 0 até $0+1 = 1$, uma que o $(0+1)$ é o limite máximo o programa imprime apenas o '0' (linha 7), neste caso precave a situação que todos os números impressos neste ciclo devem ser seguidos ('end="").



Faça um programa em Python que produza o resultado mostrado na imagem ao lado.

```
4  for n in range(0, 11, 2):
5      print(f'Numero: {n}')
6      for m in range(0, n+1):
7          print(m, ' ', end='')
8      print()
9
```

Para o numero $n=0$, o m é só iterado uma vez, por isso sai do segundo ciclo (m) e executa a linha 8, que imprime apenas um enter para separar o próximo número (n), que será '2', uma vez que o ciclo ' n ' tem um passo '2' (de dois a dois) (linha 4).

```
Numero: 0
0
Numero: 2
0 1 2
Numero: 4
0 1 2 3 4
Numero: 6
0 1 2 3 4 5 6
Numero: 8
0 1 2 3 4 5 6 7 8
Numero: 10
0 1 2 3 4 5 6 7 8 9 10
```

Então o próximo n será igual a 2 (linha 4, segunda iteração de ' n '), então com $n=2$, a linha 5 imprime 'Numero: 2'. A linha 6 irá agora iterar ' m ' de 0 até 3 ($n+1$), sendo que o 3, como já se sabe não pertence ao intervalo. Isto quer dizer que irá existir a iteração para os números 0,1,2. Precisamente a linha 7 irá ser utilizada 3 vezes (com $m= 0,1$ e 2 respetivamente.)

No fim da iteração ' m ', a linha 8 fará novamente um print para se seguir a próxima iteração de ' n '... e assim sucessivamente.



Faça um programa em Python que solicite ao utilizador os limites de um intervalo e de seguida imprima todos os números ímpares desse intervalo incluindo os seus limites. O utilizador deve introduzir primeiro o valor mínimo do intervalo e de seguida o valor máximo.

```
Minimo : 10  
Maximo : 21  
11 13 15 17 19 21
```




Faça um programa em Python que solicite ao utilizador os limites de um intervalo e de seguida imprima todos os números ímpares desse intervalo incluindo os seus limites. O utilizador deve introduzir primeiro o valor mínimo do intervalo e de seguida o valor máximo.

```
4  print("Minimo : ", end="")
5  imin = int(input())
6  print("Maximo : ", end="")
7  imax = int(input())
8  if imin % 2 == 0:
9      imin = imin + 1
10
11  for i in range(imin, imax + 1, 2):
12      print(i, " ", end="")
13
```

As linhas 4 e 6, solicitam ao utilizador, para introduzir os valores mínimo e máximo respetivamente. O 'end=""' significa que esta instrução print não deve fazer enter, isto é, o próximo output será imediatamente a seguir e em linha com o print em causa.

As linhas 5 e 7 atribuem às variáveis 'imin' e 'imax' respetivamente, os valores introduzidos pelo utilizador.



Faça um programa em Python que solicite ao utilizador os limites de um intervalo e de seguida imprima todos os números ímpares desse intervalo incluindo os seus limites. O utilizador deve introduzir primeiro o valor mínimo do intervalo e de seguida o valor máximo.

```
4  print("Minimo : ", end="")
5  imin = int(input())
6  print("Maximo : ", end="")
7  imax = int(input())
8  if imin % 2 == 0:
9      imin = imin + 1
10
11 for i in range(imin, imax + 1, 2):
12     print(i, " ", end="")
13
```

A linha 8, testa se o módulo do numero mínimo por 2 é 0. Se for então trata-se de um número par, por isso deve-se incrementar 1 ao 'imin' para não ser incluído na primeira incrementação do ciclo 'i' (linha 11). Por exemplo se o número mínimo introduzido for '10', como se pretende só números impares adiciona-se mais 1 ao 10 (linha 9) , e posiciona-se logo no primeiro numero impar da iteração, resolvendo o assunto, começando no '11'.



Faça um programa em Python que solicite ao utilizador os limites de um intervalo e de seguida imprima todos os números ímpares desse intervalo incluindo os seus limites. O utilizador deve introduzir primeiro o valor mínimo do intervalo e de seguida o valor máximo.

```
4  print("Minimo : ", end="")
5  imin = int(input())
6  print("Maximo : ", end="")
7  imax = int(input())
8  if imin % 2 == 0:
9      imin = imin + 1
10
11  for i in range(imin, imax + 1, 2):
12      print(i, " ", end="")
13
```

De seguida a linha 11 inicia o ciclo (i) a partir do primeiro número impar até ao 'imax' +1, para incluir o ultimo limite (imax +1). É também visível nesta linha, o iterador tem um passo de '2'.

Finalmente a linha 12, imprime cada um dos elementos do intervalo gerido pelo for 'i', neste caso também sem entre no fim de cada print.



Outras Formas de Utilizar a Instrução 'FOR'...

Desafio 1: Percorrer todas as letras de uma string e imprimir no ecrã cada uma delas...



Outras Formas de Utilizar a Instrução 'FOR'...

Desafio 1: Percorrer todas as letras de uma string e imprimir no ecrã cada uma delas...

A linha 4, atribui um nome à variável 'nome'.

A linha 5, inicia uma iteração a todos os elementos da String (letras).

A linha 6, imprime por cada iteração a variável 'n' (que neste caso assume o caracter da string).

Não esquecer que também no 'for' o bloco do mesmo é definido pela indentação (print(n)).

Desafio 2: Como apresentar o resultado não de uma forma vertical, mas de uma forma horizontal ?

```
4 nome = 'Manuel Joaquim Gomes Pardal'
5 for n in nome:
6     print(n)
7
```

M
a
n
u
e
l

J
o
a
q
u
i
m

G
o
m
e
s

P
a
r
d
a
l



Outras Formas de Utilizar a Instrução 'FOR'...

Desafio 2: Como apresentar o resultado não de uma forma vertical, mas de uma forma horizontal ?

```
4 nome = 'Manuel Joaquim Gomes Pardal'
5 for n in nome:
6     print(n, end='')
7
```

Manuel Joaquim Gomes Pardal

Neste caso usa-se o "end=""

O 'end' estabelece qual o 'terminador' da string. Por defeito o 'print' assume ser o 'enter', mas poderá ser um outro qualquer caracter. Naturalmente se indicarmos o terminador como vazio (linha 6), então o "print" não fará o 'enter' que assume por defeito.

Desafio 3: Como apresentar de uma forma horizontal, mas com cada letra separada por '-' e a última sem nenhum 'terminador' ?



Outras Formas de Utilizar a Instrução 'FOR'...

Desafio 3: Como apresentar de uma forma horizontal, mas com cada letra separada por '-' e a última sem nenhum 'terminador' ?

```
3
4 nome = 'Manuel Joaquim Gomes Pardal'
5 posicao = -1
6 for n in nome:
7     posicao += 1
8     if posicao != len(nome)-1:
9         print(n, end='-')
10    else:
11        print(n)
12
```

Neste caso utiliza-se um contador (linha 5), para controlar o índice da String.

É necessário incrementar a variável que representa o contador (linha 7) por cada iteração, fazendo assim acompanhar o valor dessa variável ao índice atual da string.

Dentro do 'loop' é testada uma condição: se a posição (índice) é diferente do ultimo índice da string. (Linha 8)

M-a-n-u-e-l- -J-o-a-q-u-i-m- -G-o-m-e-s- -P-a-r-d-a-l



Outras Formas de Utilizar a Instrução 'FOR'...

Desafio 3: Como apresentar de uma forma horizontal, mas com cada letra separada por '-' e a última sem nenhum 'terminador' ?

```
3
4 nome = 'Manuel Joaquim Gomes Pardal'
5 posicao = -1
6 for n in nome:
7     posicao += 1
8     if posicao != len(nome)-1:
9         print(n, end='-')
10    else:
11        print(n)
12
```

Se for diferente, então imprime o elemento da string e o terminador é um '-'. (Linha 9)

Caso contrário, significa que se encontra no último elemento da String e imprime naturalmente a respetiva letra com um enter no fim (terminador por defeito do print) (linha 11)

```
M-a-n-u-e-l- -J-o-a-q-u-i-m- -G-o-m-e-s- -P-a-r-d-a-l
```



Outras Formas de Utilizar a Instrução 'FOR'...

ENUMERATE

É possível criar um índice explícito, quando se utiliza um “for...in”, através da função ‘enumerate’.
Esta questão fará com que não seja necessário incrementar uma variável para ‘alimentar’ um índice no loop, porque o ‘enumerate’ fará isso automaticamente.

Vamos utilizar o exemplo anterior para demonstrar a funcionalidade do ‘enumerate’. Tome atenção à diferença de código (entre o exercício Ex003e2.py e Ex003e3.py) e verifique também o tempo de execução de cada um dos scripts.



Outras Formas de Utilizar a Instrução 'FOR'...

ENUMERATE

```
1 nome = 'Manuel Joaquim Gomes Pardal'
2 # posicao = -1
3 # for n in (nome):
4 for posicao, n in enumerate(nome):
5     # posicao += 1
6     if posicao != len(nome)-1:
7         print(n, end='-')
8     else:
9         print(n)
10
```

M-a-n-u-e-l- -J-o-a-q-u-i-m- -G-o-m-e-s- -P-a-r-d-a-l

Relativamente ao exercício anterior, retiraram-se 3 linhas de código, e alterou-se apenas a linha 4.

Repare que o exercício anterior, necessitava de mais 4 linhas para fazer o mesmo que este Script. E este script com menos linhas ainda o faz mais rápido.

Na linha 4, verifica-se que a iteração de 'n' é antecipada por uma variável (posicao) cuja função é representar um índice para o ciclo de 'n'. Isto é, por cada letra da string 'nome' a variável posição representa o índice dos elementos da string e 'n' representa cada um dos elementos da string (letra). Tudo de uma forma automática, sem termos que nos preocupar-mos com a incrementação de 'posição'. Naturalmente que se a 'posicao' representa um índice, então começa do '0'.



**UTILIZAÇÃO DO FOR COM TIPOS DE DADOS COMPLEXOS
(LISTA, TUPLA, SET, DICIONARIO)**

Outras Formas:

Agora vai se utilizar o exemplo anterior para mostrar uma tupla com os resultados da conversão dos respetivos valores.

Valores para a tupla : (-1, 0.1, 0, 3, 9.5, 17, 20, 2, 14, 16, 20.1, 21, 20)

Resultado pretendido :

```
-1 : Nota incorreta.  
0.1 : Muito Mau  
0 : Muito Mau  
3 : Mau  
9.5 : Nota incorreta.  
17 : Muito Bom  
20 : Elevado  
2 : Muito Mau  
14 : Bom  
16 : Bom  
20.1 : Nota incorreta.  
21 : Nota incorreta.  
20 : Elevado
```

Nota: Usar o comando “FOR”

Outras Formas:

```
1  # Ex002b1
2  # Range
3
4
5  def conversor_Nota_Quantitativa(nota):
6
7      if nota in range(0, 3):
8          return 'Muito Mau'
9      elif nota in range(3, 7):
10         return 'Mau'
11     elif nota in range(7, 10):
12         return 'Insuficiente'
13     elif nota in range(10, 14):
14         return 'Suficiente'
```

Neste exemplo utiliza-se a mesma função, mas para testar uma tupla.

```
-1      : Nota Incorreta
0.1     : Nota Incorreta
0       : Muito Mau
3       : Mau
9.5     : Nota Incorreta
17      : Muito Bom
20      : Elevado
2       : Muito Mau
14      : Bom
16      : Bom
20.1    : Nota Incorreta
21      : Nota Incorreta
20      : Elevado
```

Outras Formas:

```
15     elif nota in range(14, 17):
16         return 'Bom'
17     elif nota in range(17, 19):
18         return 'Muito Bom'
19     elif nota in range(19, 21):
20         return 'Elevado'
21     else:
22         return 'Nota Incorreta'
23
24
25 if __name__ == '__main__':
26     for nota in (-1, 0.1, 0, 3, 9.5, 17, 20, 2, 14, 16, 20.1, 21, 20):
27         print(f'{nota}\t: {conversor_Nota_Quantitativa(nota)}')
28
```

A tupla está representada na linha 26 (valores entre parênteses).

O ciclo percorre todos os elementos da tupla e transporta cada um deles iterativamente para a variável “notas” (linha 26).

A linha 27 utiliza um print formatado (f’) para testar cada um dos elementos através da variável ‘notas’ como o parâmetro da função, recebendo assim o respetivo resultado.

Desafio:

Elabore um programa em Python, que através dos operadores relacionais e operadores lógicos, produza o seguinte efeito:

Elevado	→	19 a 20
Muito Bom	→	17 a 18
Bom	→	14 a 16
Suficiente	→	9.5 a 13
Insuficiente	→	07 a 9.4
Mau	→	03 a 06
Muito Mau	→	00 a 02
Nota Incorreta	→	menor que 0

Deve poder usar números reais, e a nota 9.5 deverá contar suficiente

Valores a testar:

“-1, 0.1, 0, 3, 9.5, 17, 20, 2, 14, 16, 20.1, 21, 20”



```
1  #!C:\Users\fbent\AppData\Local\Programs
2  # Outras Utilizações do IF
3
4
5  def conversor_Nota_Quantitativa(nota):
6
7      if nota >= 0 and nota < 3:
8          return "Muito Mau"
9      elif nota >= 3 and nota < 7:
10         return "Mau"
11     elif nota >= 7 and nota < 9.5:
12         return "Insuficiente"
13     elif nota >= 9.5 and nota < 14:
14         return "Suficiente"
15     elif nota >= 14 and nota < 17:
16         return "Bom"
```

Neste caso usamos os operadores relacionais, e lógicos para determinar os limites (corretos e incorretos): (exemplo linhas 7 e 22 respetivamente)

Note-se que neste exemplo (na ausência do 'range') pode-se fazer a comparação diretamente com o numero float (exemplo: linha 11), comparando com '9.5' sem problema.



```
17 elif nota >= 17 and nota < 19:
18     return "Muito Bom"
19 elif nota >= 19 and nota <= 20:
20     return "Elevado"
21 else:
22     return "Nota incorreta."
23
24
25 if __name__ == '__main__':
26     for notas in (-1, 0.1, 0, 3, 9.5, 17, 20, 2, 14, 16, 20.1, 21, 20):
27         print(f'{notas}\t: {conversor_Nota_Quantitativa(notas)}')
28
```

```
-1 : Nota incorreta.
0.1 : Muito Mau
0 : Muito Mau
3 : Mau
9.5 : Suficiente
17 : Muito Bom
20 : Elevado
2 : Muito Mau
14 : Bom
16 : Bom
20.1 : Nota incorreta.
21 : Nota incorreta.
20 : Elevado
```

Destacada nota para a linha 21 que depois de todos os testes validados, resta senão a última hipótese possível de informar o utilizar que a nota avaliada está incorreta (linha 22) .



FOR e LIST...

```
1  # For e Lista
2  racas_Caes = ['Pincher', 'Buldog Ingles',
3               'Buldog Frances', 'Rottweiler']
4  for raca in racas_Caes:
5      print(raca)
6
```

```
Pincher
Buldog Ingles
Buldog Frances
Rottweiler
```

A linha 1, cria uma lista ([]) com 4 raças de cães.

A linha 4, tal como já demonstrado nos exemplos anteriores fazem a iteração dos elementos da lista através neste caso da variável 'raca'.

A linha 5, escreve no ecrã o elemento atual da lista.

➔ A partir deste exemplo faça um outro script que mostre também o índice de cada um dos elementos tal como na figura em baixo:

```
Indice: 0, Raca: Pincher
Indice: 1, Raca: Buldog Ingles
Indice: 2, Raca: Buldog Frances
Indice: 3, Raca: Rottweiler
```



FOR e LIST...

```
1  # For e Lista
2  racas_Caes = ['Pincher', 'Buldog Ingles',
3               'Buldog Frances', 'Rottweiler']
4  for n, raca in enumerate(racas_Caes):
5      print(f'Indice: {n}, Raca: {raca}')
6
```

```
Indice: 0, Raca: Pincher
Indice: 1, Raca: Buldog Ingles
Indice: 2, Raca: Buldog Frances
Indice: 3, Raca: Rottweiler
```

Note-se na linha 4 a presença de um 'enumerate', situação que permitiu a utilização de um print do 'índice', representado neste exemplo pela variável 'n'.

A linha 5 usa uma formatação para representar as duas variáveis ('n' e 'raca') com a string 'Indice e Raca' respetivamente.



FOR e TUPLA...

```
1  # For e Tupla
2  racas_Caes = ('Pincher', 'Buldog Ingles',
3               'Buldog Frances', 'Rottweiler')
4  for n, raca in enumerate(racas_Caes):
5      print(f'Indice: {n}, Raca: {raca}')
6
```

```
Indice: 0, Raca: Pincher
Indice: 1, Raca: Buldog Ingles
Indice: 2, Raca: Buldog Frances
Indice: 3, Raca: Rottweiler
```

Trabalhar com a instrução 'FOR' com uma tupla é exatamente da mesma forma que se trabalha com uma list (exercício anterior), levando em conta as características de cada uma destes tipos de dados complexos.



FOR e SET...

```
1  # For e SET
2  racas_Caes = {'Pincher', 'Buldog Ingles',
3               'Buldog Frances', 'Rottweiler'}
4  for n, raca in enumerate(racas_Caes):
5      print(f'Indice: {n}, Raca: {raca}')
6
```

```
Indice: 0, Raca: Pincher
Indice: 1, Raca: Rottweiler
Indice: 2, Raca: Buldog Frances
Indice: 3, Raca: Buldog Ingles
```

```
Indice: 0, Raca: Buldog Frances
Indice: 1, Raca: Pincher
Indice: 2, Raca: Buldog Ingles
Indice: 3, Raca: Rottweiler
```

No caso do SET, a utilização do 'FOR' também é idêntica à dos exercícios anteriores.

Verificar que neste caso (SET), a ordem dos elementos, pode não ser respeitada pela ordem 'original' (linhas 2 e 3), uma vez que esta é uma das características deste tipo de dados.



FOR e DICIONÁRIO...

```
1 # For e Dicionário
2 caes = {'Raca': 'Pincher', 'Porte': 'Pequeno', 'Classificacao': '**'}
3 for campo, conteudo in caes.items():
4     print(f'{campo}: {conteudo}')
5 print()
6 for campo in caes.keys():
7     print(f'Campo: {campo}')
8 print()
9 for conteudo in caes.values():
10    print(f'Campo: {conteudo}')
11
```

```
Raca: Pincher
Porte: Pequeno
Classificacao: **

Campo: Raca
Campo: Porte
Campo: Classificacao

Campo: Pincher
Campo: Pequeno
Campo: **
```

Neste exemplo percorrem-se todos os elementos do dicionário ('Key' e 'Value') (linhas 2 a 4), percorrem-se apenas as chaves (keys) do dicionário (linhas 6 a 7) e finalmente percorrem-se todos os valores do dicionário (linhas 9 a 10).

Nas três situações visualizam-se os respetivos outputs a partir de um print formatado.



FOR e DICIONÁRIO...

```
Indice: 0, Raca: Pincher  
Indice: 0, Porte: Pequeno  
Indice: 0, Classificacao: ***  
  
Indice: 1, Raca: Buldog Ingles  
Indice: 1, Porte: Grande  
Indice: 1, Classificacao: *  
  
Indice: 2, Raca: Buldog Frances  
Indice: 2, Porte: Medio  
Indice: 2, Classificacao: **  
  
Indice: 3, Raca: Rottweiler  
Indice: 3, Porte: Grande  
Indice: 3, Classificacao: **
```

Exercício:

Utilize tipos de dados complexos para fazer um programa que produza um output similar ao da imagem ao lado..



FOR e LISTA e DICIONÁRIO...

```
1  # For , Lista e Dicionário
2  caes = [{'Raca': 'Pincher', 'Porte': 'Pequeno', 'Classificacao': '***'},
3          {'Raca': 'Buldog Ingles', 'Porte': 'Grande', 'Classificacao': '*'},
4          {'Raca': 'Buldog Frances', 'Porte': 'Medio', 'Classificacao': '**'},
5          {'Raca': 'Rottweiler', 'Porte': 'Grande', 'Classificacao': '**'}]
6
7  for n, dicionario in enumerate(caes):
8      for campo, conteudo in dicionario.items():
9          print(f'Indice: {n}, {campo}: {conteudo}')
10     print()
11
```

O exercício proposto sugere a combinação de tipos de dados complexos. Neste cado optou-se por utilizar uma lista cujos os elementos são dicionários. Assim no sentido do objetivo do exercido proposto, bastou utilizar 2 ciclos 'FOR'. Um para percorrer os elementos (listas) da lista (caes) e um outro para percorrer os elementos de cada um dos dicionários ('dicionário'). Desta forma percorreram-se todos os elementos da lista 'caes' com todo seu conteúdo organizado. Neste exemplo optou-se também por utilizar um índice explícito na lista para melhor organização dos registos (output).



FOR e LISTA e DICIONÁRIO...

```
1  # For , Lista e Dicionário
2  caes = [{'Raca': 'Pincher', 'Porte': 'Pequeno', 'Classificacao': '***'},
3          {'Raca': 'Buldog Ingles', 'Porte': 'Grande', 'Classificacao': '*'},
4          {'Raca': 'Buldog Frances', 'Porte': 'Medio', 'Classificacao': '**'},
5          {'Raca': 'Rottweiler', 'Porte': 'Grande', 'Classificacao': '**'}]
6
7  for n, dicionario in enumerate(caes):
8      for campo, conteudo in dicionario.items():
9          print(f'Indice: {n}, {campo}: {conteudo}')
10     print()
11
```

As linha 2 a 5 criam uma lista ([caes]) de dicionários({dicionário}, linha 7).

A linha 7 inicia um ciclo que percorre todos os dicionários da lista, e utiliza um índice explícito (n) através do 'enumerate'.

A linha 8 inicia um ciclo dentro do ciclo da linha 7 para percorrer todas as chaves e valores (campo e conteudo) de cada dicionário da lista.

A linha 9, encontra-se dentro do ciclo da linha 8 (dos dicionários) e imprime o Índice, a chave e o valor do dicionário em causa.

Integração de Conhecimentos :

- ✓ **Estruturas de Dados Simples**
- ✓ **Estruturas de Dados Complexas**
- ✓ **Conversões de dados**
- ✓ **“Import”**
- ✓ **Estruturas de Controlo :**
 - ✓ **Condicionais**
 - ✓ **Repetição**
- ✓ **Funções**
- ✓ **Input**
- ✓ **Tratamento de Exceções**

Integração de novos conhecimentos :

- ✓ **Tratamento de Ficheiros**
- ✓ **Utilização da Biblioteca ‘OS’ (Operating System).**

EXERCICIO EXPLICADO

```
=====
=                      MENU                      =
=====
= 1 - Novo Registo                                =
= 2 - Editar Registo                             =
= 3 - Eliminar Registo                           =
= 4 - Consultar Registo                          =
= 5 - Listar todos os Registos                   =
= 6 - Eliminar Base de Dados                     =
= Outra opção - Sair                             =
=====
Qual a sua opção ?█
```

Pretende-se criar um programa em Python que crie uma estrutura de dados não volátil, no sentido de persistir dados relativamente ao registo de alunos. Os atributos do ficheiro são : N° Aluno, Nome do Aluno, E-mail e Telefone.

EXERCICIO EXPLICADO

```
=====
=                      MENU                      =
=====
= 1 - Novo Registo                               =
= 2 - Editar Registo                             =
= 3 - Eliminar Registo                           =
= 4 - Consultar Registo                          =
= 5 - Listar todos os Registos                    =
= 6 - Eliminar Base de Dados                      =
= Outra opção - Sair                              =
=====
Qual a sua opção ?█
```

O número de aluno não pode ser repetido. Se o ficheiro não existir o programa deverá criá-lo na primeira tentativa de se criar um novo registo. Devem existir validações naturais a cada tipo de entrada do menu apresentado (por exemplo avisar que o aluno que se pretende 'editar' não existe e como tal não é possível a operação, etc...)

EXERCICIO EXPLICADO

```
=====
=                MENU                =
=====
= 1 - Novo Registo                    =
= 2 - Editar Registo                  =
= 3 - Eliminar Registo                =
= 4 - Consultar Registo               =
= 5 - Listar todos os Registos        =
= 6 - Eliminar Base de Dados          =
= Outra opção - Sair                  =
=====
Qual a sua opção ?█
```

- ✓ Menu
- ✓ Definir o Nome da 'Base de Dados' (Ficheiro)
- ✓ Verificar o ficheiro já existe
- ✓ Criar o Ficheiro
- ✓ Verificar se determinado já existe no Ficheiro
- ✓ Inserir novos Registos no Ficheiro
- ✓ Editar um determinado registo (Alterar)
- ✓ Consultar um determinado registo
- ✓ Listar todos os registos do Ficheiro
- ✓ Eliminar o Ficheiro.
- ✓ Controlar as validações naturais de cada uma das opções do menu.

Nota: Neste programa não é necessário validar tipos de dados (caso exista uma introdução de dados não coerente o programa apresenta erro).

```
1  import os
2
3
4  def BD():
5      return 'alunos.txt'
6
7
8  def limpar():
9      os.system('cls' if os.name == 'nt' else 'clear')
10
11
```

- A biblioteca 'os' (linha 1) permite aceder a método para tratamento funcionalidades relacionadas com o sistema operativo.
- No Python, não existe definidamente algo para declarar uma constante. No entanto pode-se utilizar uma prática, que consiste na criação de uma função (neste caso 'BD()' (linha 4-5)) e retornar um valor (como se de uma constante se trata-se).
- É necessário também prever uma função que limpe a consola, por isso foi necessário o import de 'os', pois permite-nos aceder aos comandos do sistema operativo (linhas 8-9). No Windows o comando utilizado para limpar o ecrã é 'cls' tipicamente noutros sistemas operativos utiliza-se o 'clear'.



```
12 def ecran():
13     limpar()
14     # Construção do Menu
15     print('=====')
16     print('=          MENU          =')
17     print('=====')
18     print('= 1 - Novo Registo          =')
19     print('= 2 - Editar Registo        =')
20     print('= 3 - Eliminar Registo      =')
21     print('= 4 - Consultar Registo     =')
22     print('= 5 - Listar todos os Registos =')
23     print('= 6 - Eliminar Base de Dados =')
24     print('= Outra opção - Sair        =')
25     print('=====')
26
27
```

- Naturalmente existirá uma função para chamar o ecran. (linhas 12-25)
- É necessário antes de apresentar o ecran, limpar (linha 13) o que existe, para garantir que em cada opção este ecran é novamente chamado até o utilizador desejar sair, sem que o ecran se repita continuamente.

```
28 def verifica_Base_Dados(nomeBd):
29     try:
30         baseDados = open(nomeBd, 'r+')
31         baseDados.close
32         return True
33     except FileNotFoundError:
34         baseDados = open(nomeBd, 'w')
35         baseDados.close
36         print('A Base de Dados Não Existia e Foi Criada...')
37         return False
38
```

- Como referido na base dos requisitos do programa é necessário verificar se o ficheiro (base de dados existe), então utiliza-se o comando 'open' (linha 30)
- O comando 'open' pode ser utilizado de várias formas; para escrever ('w') para ler ('r+')
- Nesta função é utilizador o tratamento de exceções; try... except...
A função começa por tentar ler uma base de dados, caso seja acusado uma exceção (erro) então o 'except' é ativado para o tipo de erro "FileNotFoundError", caso se verifique (linhas 34-37) então significa que a base de dados não está criada e como tal é criada através do comando "open...w" (linha 34).
- É visível também que a função retorna um valor booleano em qualquer dos casos (encontre a base de dados ou não encontre a base de dados).



```
39 # Ver outra forma de fazer esta função
40
41
42 def verifica_Numero_Aluno(n):
43     ficheiro = open(BD(), 'r')
44     linha = ficheiro.read()
45     ficheiro.close
46     contador = -1
47     found = -1
48     for registo in linha.splitlines():
49         x = registo.split(',')
50         contador += 1
51         if int(x[0]) == int(n):
52             found = contador
53             return found
54     return found
55
```

- Após ter a garantia que o ficheiro está criado (base de dados) já é possível utilizar funções para manuseamento de registos. Uma das funções necessárias para este efeito é poder verificar se um determinado aluno existe ou não.
- Admitindo que os registos dos ficheiro são 'listas' (List), então uma das formas de verificar se o registo existe ou não na base de dados é percorrer todos os registos da mesma e verificar se o numero de aluno pretendido (n) existe nesses registos ou não.
- Então o numero de aluno é assumido por ser o primeiro elemento das listas (elemento 0) (linha 51)
- Assim basta receber o aluno (n) (linha 42), abrir a base de dados para leitura (linha 43). Colocar todos os registos numa string (linha 44) e fechar a base de dados.



```
39 # Ver outra forma de fazer esta função
40
41
42 def verifica_Numero_Aluno(n):
43     ficheiro = open(BD(), 'r')
44     linha = ficheiro.read()
45     ficheiro.close
46     contador = -1
47     found = -1
48     for registo in linha.splitlines():
49         x = registo.split(',')
50         contador += 1
51         if int(x[0]) == int(n):
52             found = contador
53             return found
54     return found
55
```

- De Seguida, inicia-se um loop no sentido de transformar cada uma das linhas (registo) numa lista (linha 49). A função `splitlines` garante a separação da string 'linha' em linhas isoladas, e a função `split` já explicada anteriormente, transforma essa string num tipo de dados 'list' para poder ser interpretado registo a registo do ficheiro como uma lista.
- De seguida é efetuado um teste de comparação entre o aluno que se pretende verificar se existe e o aluno corrente da lista referida (registo do ficheiro) (linha 51)
- Caso se verifique uma igualdade (linhas 52-53), significa que o aluno existe e portanto devolve a posição do elemento do registo a uma variável 'contador' (linha 52) e retorna esse valor à entidade que chamou a função. Caso a igualdade da linha 51 nunca seja verificada, então significa que o aluno não existe nos registos do ficheiro e por isso mantem a variável contador a '-1' e retorna essa valor à entidade que chamou a função. Portanto se a função retornar o valor de -1 significa que o alunos não foi encontrado nos registos do ficheiro.



```
56 # Esta Função é outra forma de fazer...
57
58
59 def verifica_Numero_Aluno1(n):
60     ficheiro = open(BD())
61     contador = -1
62     found = -1
63     for registo in ficheiro:
64         x = registo.split(',')
65         contador += 1
66         if int(x[0]) == int(n):
67             found = contador
68             ficheiro.close
69             return found
70     ficheiro.close
71     return found
72
73
```

- Esta é outra forma de poder verificar se um determinado registo existe num ficheiro.
- A diferença é que desta maneira o ficheiro não é transferido para uma string mas é utilizado em tempo. Repare que o close do ficheiro (método que fecha o ficheiro) é apenas efetuado no fim do ficheiro, o que significa que o ficheiro está em aberto enquanto o loop está ativo. (linha 70) ou até encontrar o registo. (linha 69).
- Não esquecer da importância das 'indentações' que definem os diferentes blocos de código.
- Esta é uma forma mais correta de fazer, pois o ficheiro é utilizado em 'real time'.



```
74  def continuar():  
75      |      input('Enter Para Continuar...')  
76  
77
```

- Uma das formas para poder fazer parar o programa e esperar que o utilizador faça enter, é utilizar uma função sem retorno que a única que coisa que faz é um input com um texto '...continuar...'.

```
78 def novo_Registo(wNumeroaluno=None, wNome=None, wEmail=None, wTelefone=None):
79     verifica_Base_Dados(BD())
80
81     if wNumeroaluno is None:
82         print('===== NOVO REGISTO =====')
83         numero = int(input('número de Aluno: '))
84     else:
85         numero = wNumeroaluno
86     if verifica_Numero_Aluno1(numero) == -1 and numero > 0:
87         if wNumeroaluno is None:
88             nome = input('nome do Aluno: ')
89             email = input('e-mail do Aluno: ')
90             telefone = input('telefone do Aluno: ')
91         else:
92             nome = wNome
93             email = wEmail
94             telefone = wTelefone
95
96     registo = str(numero) + ","
97     registo += str(nome) + ","
98     registo += str(email) + ","
99     registo += str(telefone)
100
101     baseDados = open(BD(), 'a')
102     baseDados.write(registo + '\n')
103     baseDados.close
104     if wTelefone is None:
105         print('Registo Adicionado.')
106
107     else:
108         print('Este número de Aluno já existe, ou está incorreto.')
109
```

- Esta é a função que escreve os registos no ficheiro, pode no entanto escrever os registos através da introdução de dados do utilizador, ou através de receber os dados a partir de parâmetros enviados por outras entidades (programa, outras funções).
- Para se utilizarem parâmetros opcionais numa função, basta nos parâmetros que se pretende ter como opcionais igualar à palavra reservada 'None'. Esta situação fará com que quando se chama a função caso não se preencha os parâmetros que estão igualados a 'None', então o programa assume o parâmetro igual a esse valor 'None', mas não apresenta erro.
- Desta forma é possível fazer com que a função esteja preparada para receber valores pelo próprio utilizador ou através de parâmetros associados à função. (linha 78)

```
78 def novo_Registo(wNumeroaluno=None, wNome=None, wEmail=None, wTelefone=None):
79     verifica_Base_Dados(BD())
80
81     if wNumeroaluno is None:
82         print('===== NOVO REGISTO =====')
83         numero = int(input('número de Aluno: '))
84     else:
85         numero = wNumeroaluno
86     if verifica_Numero_Aluno1(numero) == -1 and numero > 0:
87         if wNumeroaluno is None:
88             nome = input('nome do Aluno: ')
89             email = input('e-mail do Aluno: ')
90             telefone = input('telefone do Aluno: ')
91         else:
92             nome = wNome
93             email = wEmail
94             telefone = wTelefone
95
96     registo = str(numero) + ","
97     registo += str(nome) + ","
98     registo += str(email) + ","
99     registo += str(telefone)
100
101     baseDados = open(BD(), 'a')
102     baseDados.write(registo + '\n')
103     baseDados.close
104     if wTelefone is None:
105         print('Registo Adicionado.')
106
107     else:
108         print('Este número de Aluno já existe, ou está incorreto.')
109
```

- A lógica desta função é se o aluno que se pretende criar não existir então a função servirá para criar esse aluno.
- Tal como previsto nos requisitos, o programa deve na altura da intenção de criação de registos verificar se a base de dados existe ou não, em caso negativo cria o ficheiro (alunos.txt) através da função já criada "verifica_Base_Dados()". (Linha 79)
- De seguida o programa interroga-se se realmente é o utilizador que introduzir os dados ou se os dados vêm de fora (de outras funções ou programa) (linha 81). Se o parâmetro 'wNumeroaluno' = None, então significa que o aluno vai ser criado a partir do utilizador (linha 81) caso contrário (linha 84) então o aluno vai ser criado a partir dos dados que vêm dos parâmetros.
- A verificação do aluno é efetuada na linha 86, através da função verifica aluno (já explicada atrás).

```
78 def novo_Registo(wNumeroaluno=None, wNome=None, wEmail=None, wTelefone=None):
79     verifica_Base_Dados(BD())
80
81     if wNumeroaluno is None:
82         print('===== NOVO REGISTO =====')
83         numero = int(input('número de Aluno: '))
84     else:
85         numero = wNumeroaluno
86     if verifica_Numero_Aluno1(numero) == -1 and numero > 0:
87         if wNumeroaluno is None:
88             nome = input('nome do Aluno: ')
89             email = input('e-mail do Aluno: ')
90             telefone = input('telefone do Aluno: ')
91         else:
92             nome = wNome
93             email = wEmail
94             telefone = wTelefone
95
96     registo = str(numero) + ","
97     registo += str(nome) + ","
98     registo += str(email) + ","
99     registo += str(telefone)
100
101     baseDados = open(BD(), 'a')
102     baseDados.write(registo + '\n')
103     baseDados.close
104     if wTelefone is None:
105         print('Registo Adicionado.')
106
107     else:
108         print('Este número de Aluno já existe, ou está incorreto.')
109
```

- Após a introdução ou associação dos dados do aluno for confirmada (linhas 87-94), forma-se o registo para escrever no ficheiro (linhas 96-99)
- Neste caso o ficheiro é aberto em modo 'append', isto é, em modo de adicionar registos ('open...'a')(linha 101)
- De seguida o registo é gravado através da função 'write' (linha 102) e com um enter no fim.
- A base de dados é encerrada (linha 103)
- E finalmente verifica-se novamente se os dados do aluno foram introduzidos pelo utilizador ou se foram obtidos a partir dos parâmetros da função, caso tenham vindo do utilizador, então é enviada uma notificação ao utilizador em como o registo foi criado e é encerrado o processo (linhas 104-105).
- Caso a verificação do aluno (linha 86) seja -1 isto é, o aluno não existe, então tá uma mensagem a informar dessa situação e o processo é igualmente encerrado.


```
110 def eliminar_Registo(perguntarAluno=None):
111     if not verifica_Base_Dados(BD()):
112         return
113     if perguntarAluno is not None:
114         n_Aluno = perguntarAluno
115     else:
116         n_Aluno = int(input('Numero de Aluno? '))
117         x = input('Deseja mesmo eliminar este aluno?(S/N)')
118         if x != 's' and x != 'S':
119             print('Operação Cancelada.')
120             return
121     existe_aluno = verifica_Numero_Aluno(n_Aluno)
122     if existe_aluno == -1:
123         print('Aluno não Encontrado')
124     else:
125         ficheiro = open(BD(), "r")
126         linha = ficheiro.read()
127         ficheiro.close
128         ficheiro = open(BD(), 'w')
129         for registo in linha.splitlines():
130             x = registo.split(',')
131             if int(x[0]) == int(n_Aluno):
132                 pass
133                 if perguntarAluno is None:
134                     print("Registo eliminado, Aluno Nº", n_Aluno)
135             else:
136                 ficheiro.write(registo + '\n')
137     ficheiro.close
```

- Uma vez solucionada a questão da inserção de novos registos (novos alunos), prepara-se agora uma função para poder eliminar registos (apagar um aluno).
- A ideia é criar uma função (elimina_Registo()) preparada para receber um argumento (parâmetro) 'perguntarAluno' que servirá para indicar se o programa deve receber o aluno pelo utilizador ou via outra função o programa. No caso desse argumento for diferente de 'None' então o numero do aluno (n__Aluno) assume o valor do argumento (perguntarAluno) (linha 114), caso contrário a função pergunta ao utilizar qual é o numero de aluno que se pretende eliminar do ficheiro (linhas 116).
- De seguida, se a operação não foi cancelada, então a função verifica se o aluno existe para poder ser eliminado (Linha 121) . Em caso negativo, informa o utilizador que o Aluno não existe, caso contrário prossegue com código (Linha 124).

```
110 def eliminar_Registo(perguntarAluno=None):
111     if not verifica_Base_Dados(BD()):
112         return
113     if perguntarAluno is not None:
114         n_Aluno = perguntarAluno
115     else:
116         n_Aluno = int(input('Numero de Aluno? '))
117         x = input('Deseja mesmo eliminar este aluno?(S/N)')
118         if x != 's' and x != 'S':
119             print('Operação Cancelada.')
120             return
121     existe_aluno = verifica_Numero_Aluno(n_Aluno)
122     if existe_aluno == -1:
123         print('Aluno não Encontrado')
124     else:
125         ficheiro = open(BD(), "r")
126         linha = ficheiro.read()
127         ficheiro.close()
128         ficheiro = open(BD(), 'w')
129         for registo in linha.splitlines():
130             x = registo.split(',')
131             if int(x[0]) == int(n_Aluno):
132                 pass
133                 if perguntarAluno is None:
134                     print("Registo eliminado, Aluno Nº", n_Aluno)
135             else:
136                 ficheiro.write(registo + '\n')
137         ficheiro.close()
```

- Executadas todas as validação para que o registo seja eliminado com segurança, é aberto o ficheiro alunos.txt em modo de leitura (linha 125).
- O conteúdo desse ficheiro (ficheiro) é passado para m string (linha) (linha 126) e o ficheito é encerrado, no entanto o seu conteúdo já está na string 'linha'.
- Assim abre-se o ficheiro em modo escrita (linha 128), o o que significa tratar-se de um novo ficheiro, porque se pretende-se adicionar ao ficheiro seria em modo 'a' (append) como anteriormente referido.
- Neste momento o ficheiro encontra-se vazio (linha 128) e agora transforma-se a string num conjunto de linhas separadas por enter (linha 129) e por cada uma dessas linhas transforma-se em lista (linha 130).


```
110 def eliminar_Registo(perguntarAluno=None):
111     if not verifica_Base_Dados(BD()):
112         return
113     if perguntarAluno is not None:
114         n_Aluno = perguntarAluno
115     else:
116         n_Aluno = int(input('Numero de Aluno? '))
117         x = input('Deseja mesmo eliminar este aluno?(S/N)')
118         if x != 's' and x != 'S':
119             print('Operação Cancelada.')
120             return
121     existe_aluno = verifica_Numero_Aluno(n_Aluno)
122     if existe_aluno == -1:
123         print('Aluno não Encontrado')
124     else:
125         ficheiro = open(BD(), "r")
126         linha = ficheiro.read()
127         ficheiro.close()
128         ficheiro = open(BD(), 'w')
129         for registo in linha.splitlines():
130             x = registo.split(',')
131             if int(x[0]) == int(n_Aluno):
132                 pass
133                 if perguntarAluno is None:
134                     print("Registo eliminado, Aluno Nº", n_Aluno)
135             else:
136                 ficheiro.write(registo + '\n')
137     ficheiro.close()
```

- Ao longo deste loop cada lista (x) cujo numero de aluno (elemento 0 de 'x') seja diferente do aluno a eliminar (n_aluno) é escrito no novo ficheiro (linha 136), ficando de fora deste registo o aluno a eliminar (linha 132).
- De notar que o Python tem uma palavra reservada que significa “passar”, isto é, não fazer nada, essa palavra reservada é “pass” (linha 132). Serve basicamente neste caso, para que se a verificação anterior (linha 131) se verificar, não faz nada. No entanto se o aluno a eliminar foi introduzido pelo utilizador, então envia uma mensagem a informar que o aluno do eliminado (linhas 133-134). Esta situação é muito utilizada em controlo condicional quando apenas uma das partes do bloco de código tem código.
- Este é um caso académico, para demonstrar o manuseamento de ficheiros físicos, aproveitando as estruturas de dados complexas, pelo que existe outras técnicas mais eficientes para produzir o mesmo efeito.



```
138
139
140 def consultar_Registo(altera=None): # altera = True (Para alterar Registo)
141     if not verifica_Base_Dados(BD()):
142         return
143     else:
144         n_Aluno = int(input('Numero de Aluno? '))
145         existe_aluno = verifica_Numero_Aluno1(n_Aluno)
146         if existe_aluno == -1:
147             print('Aluno não Encontrado...')
148             return
149         else:
150             ficheiro = open(BD(), 'r')
151             linha = ficheiro.read()
152             ficheiro.close
153             for registo in linha.splitlines():
154                 ll = registo.split(',')
155                 if int(ll[0]) == int(n_Aluno):
156                     print('Aluno Nº : ' + str(ll[0]))
157                     print('Nome      : ' + str(ll[1]))
158                     print('e-Mail   : ' + str(ll[2]))
159                     print('Telefone : ' + str(ll[3]))
160                     break
161             if altera is True:
162                 print('Alterar Registo:\n')
163                 ll[1] = input('Nome      :')
164                 ll[2] = input('e-Mail   :')
165                 ll[3] = input('Telefone :')
166                 eliminar_Registo(n_Aluno)
167                 novo_Registo(n_Aluno, ll[1], ll[2], ll[3])
168
169                 print('Registo do Aluno N.', n_Aluno, ' alterado.')
170
```

- A função para consultar um registo (aluno), serve dois propósitos; alterar o aluno e consultar o aluno.
- Para isso, recorrer a um argumento (altera) que define se a operação é para alterar o registo ou se é para consultar um registo. Caso o argumento assuma o valor 'True' então deve-se tratar de uma alteração (linhas 162-160). Isto é, a função faz sempre uma consulta do aluno, no entanto se o argumento 'altera' for igual a True, pede ao utilizador para introduzir os dados para alterar o registo do aluno solicitado.
- No caso de se tratar de uma alteração e após o utilizador introduzir os dados do aluno (linhas 162-165), a função executa outras duas funções já criadas; a função 'eliminar_Registo()' para eliminar o aluno e a função 'novo_Registo()' para criar um novo aluno com os dados introduzidos... Os dados guardados nas variáveis (linhas 163-165) mais estas duas funções acabam por funcionar como alteração do registo.



```
171
172 def listar_Registos():
173     if not verifica_Base_Dados(BD()):
174         return
175     else:
176         print('Listar todos os registos.')
177         ficheiro = open(BD(), 'r')
178         linhas = ficheiro.readlines()
179         print('=====Lista de Alunos=====')
180         print('NºAluno \t\tAluno\t\t\tte-Mail\t\t\t\tTelefone')
181         print('=====')
182         for line in linhas:
183             ll = line.split(',')
184             print(f'{ll[0]}\t\t\t{ll[1]}\t\t\t{ll[2]}\t\t\t{ll[3]}\t')
185         ficheiro.close
186
187
```

- A opção Listar Registos nada mais é que um loop que percorre todos os registos do ficheiro.
- Para o efeito utiliza-se um loop (linhas 182-184) que percorrer um ficheiro aberto em modo leitura (linha 177) e escreve no ecrã cada um dos elementos da lista 'll' (linha 183-184)
- O ficheiro é fechado na linha 185, que neste caso poderia estar também a seguir à linha 178.



```
188 def eliminar_Base_Dados():
189     if not verifica_Base_Dados(BD()):
190         return
191     else:
192         x = input('Atenção: Continuar esta operação significa perder todos '
193                 + 'os dados, deseja continuar? (S/N)')
194         if x != 's' and x != 'S':
195             print('Operação Cancelada.')
196             return
197         os.remove(BD())
198         print('Ficheiro', BD(), 'removido.')
199
```

- A eliminação da base de dados (alunos.txt) é efetuada a partir de uma função 'eliminar_Base_Dados()'.
- Depois de verificar que a base de dados existe (linha 189), é perguntado ao utilizado se tem a certeza de que quer mesmo eliminar o ficheiro (linhas 192-193). Em caso negativo, a operação é cancelada (linhas 195 e 196), caso contrário a função utiliza o método 'remove' da biblioteca 'os' para remover o ficheiro (linha 197) e informa que o ficheiro foi removido (linha 198).



```
200 # Programa Principal
201
202
203 opcaoMenu = 1
204
205 while opcaoMenu in range(1, 7):
206     ecran()
207     opcaoMenu = int(input('Qual a sua opção ?'))
208     if opcaoMenu == 1:
209         novo_Registo()
210     elif opcaoMenu == 2:
211         consultar_Registo(True)
212     elif opcaoMenu == 3:
213         eliminar_Registo()
214     elif opcaoMenu == 4:
215         consultar_Registo()
216     elif opcaoMenu == 5:
217         listar_Registos()
218     elif opcaoMenu == 6:
219         eliminar_Base_Dados()
220     else:
221         print('Sair')
222     continuar()
223
```

- Resta apenas agora apresentar o “programa principal”, pois até agora foram só as funções que necessitávamos para colocar o programa a correr...
- Assim, criou-se uma variável com a opção do menu igual a 1 (linha 203)
- ...e de seguida desenvolve-se um loop controlado por um while que determina o seu fim, desde que a variável ‘opcaoMenu’ se encontre fora do valores ente 1 e 6 (linha 205). Portanto se a opção introduzida pelo utilizador for fora desse intervalo o programa termina.
- Pelo contrário é então apresentado o menu (linha 206) e de seguida solicita-se ao utilizador, qual a opção desejada (linha 207).
- Cada uma destas opções executará uma função (linhas 208-219) caso nenhuma se verifique o programa sai (linha 220-221)
- Por qualquer uma das opções introduzidas e após o retorno da função invocada, o programa pede ao utilizador para clicar ‘enter’ para continuar (linha 222).