

# Curso de Iniciação à Programação em Python para Profissionais de Saúde



Luís Vieira e José Ferrão

Instituto Nacional de Saúde Doutor Ricardo Jorge

2-11 Dezembro 2024

# Sessão 2: Expressões condicionais e iteração

- **Conteúdos:**
  - Operações com *strings* (continuação)
  - Entrada de informação do utilizador
  - O editor do *Python*
  - Gravar e executar o código
  - Expressões condicionais (*if-else*)
  - Indentação
  - Ciclo *for*
  - Abrir e ler ficheiros
  - Guardar ficheiros de texto

# Operações com *strings*

- O **comprimento** de uma *string*, ou seja, o número de caracteres e símbolos que a constituem, é obtido com a função *len()*
  - Exemplo: *len*("Python") é igual a 6
- Os caracteres individuais de uma *string* podem ser extraídos usando a **indexação**. A posição do carácter a extrair é colocada dentro de parêntesis rectos após a *string*. No *Python*, a indexação começa com o valor 0 (zero). Por exemplo:

```
>>> "Python"[0] produz a string "P"
```

```
>>> "Python"[6] produz uma mensagem de erro (não existe posição 6)
```

```
>>> "Python"[-1] produz a string "n"
```

# Operações com *strings*

- O **slicing** permite extrair *substrings* de comprimento variável. A expressão `string[início:fim]` extrai a *substring* que começa na posição 'início' e termina na posição 'fim'-1. Por exemplo:

```
>>> "Python"[1:6] produz a string "ython"
```

Nota: A *substring* termina no índice 'fim'-1 em vez de 'fim' para que as expressões como `"Python"[0:len("Python")]` tenham o valor esperado)

```
>>> "Python"[:] produz a string "Python"
```

```
>>> "Python"[::-1] produz a string "nohtyP"
```

# Entrada de informação do utilizador

- O comando *input()* é usado para que o *Python* receba uma resposta e a coloque numa variável:  

```
>>> dia = input ("Que dia do mês é hoje?")
```
- Se a variável *dia* for usada para calcular o número de dias em falta até final do mês, irá produzir um erro (não é possível subtrair um inteiro e uma *string*):  

```
>>> 31-dia
```
- O *TypeCasting* pode ser usado para converter uma *string* num número inteiro através da função *int()*:  

```
>>> type (dia)
>>> 31 - int (dia)
```

# Exercícios

- Execute os seguintes comandos na *Shell* do *Python* e interprete os resultados:

```
>>> nome = input ("A quem deve o nome o Instituto Nacional de Saúde?")
```

```
>>> type (nome)
```

```
>>> casosTB= input ("Quantos casos de tuberculose existem em Portugal por cada 100.000 habitantes/ano?") (resposta: 13.4)
```

```
>>> type (casosTB)
```

```
>>> casosTB = float (input ("Quantos casos de tuberculose existem em Portugal por cada 100.000 habitantes/ano?"))
```

```
>>> type (casosTB)
```

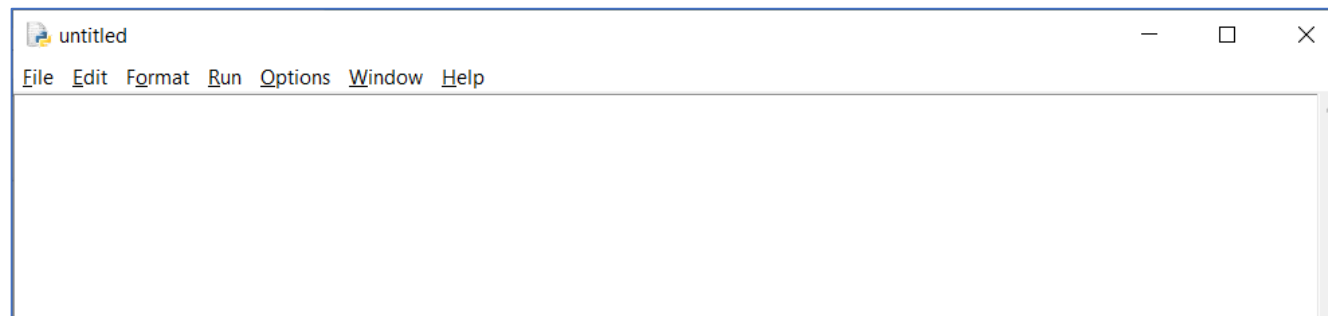
# Palavras-chave do *Python*

- As palavras-chave são as **palavras reservadas** do *Python* que servem para definir a sintaxe e a estrutura da linguagem
- As palavras reservadas não podem ser usadas como nomes de variáveis ou funções e são “case-sensitive”
- Na versão 3.13.0 do *Python* existem 35 palavras reservadas

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

# O editor do *Python*

- O editor do *Python* permite escrever, gravar e executar o código, mesmo que este seja constituído por várias linhas (*scripts*)
- O editor do *Python* é um editor de texto simples que incorpora as características (e.g., indentação) e a codificação de cores do *Python*
- Para aceder ao editor do *Python*:
  - Na caixa de pesquisa do *Windows*, escrever “IDLE”; clicar sobre “IDLE (Python 3.13.0)” para abrir a *Shell* do *Python*
  - Selecionar *File > New File*; a nova janela em branco, com o nome “untitled”, é o editor do *Python*





# Índice de massa corporal

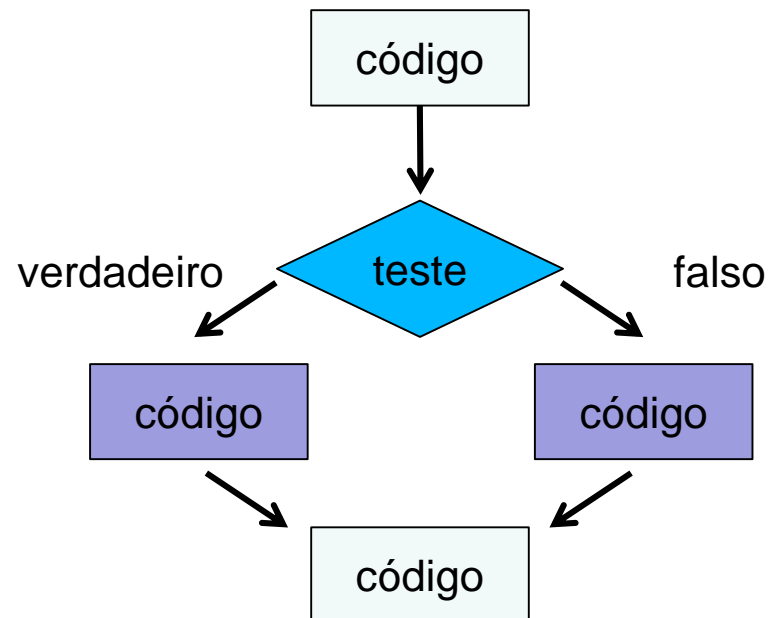
- No editor do *Python*, vamos introduzir o seguinte código para criar um programa que calcula o índice de massa corporal (IMC) a partir da informação pedida ao utilizador:

```
peso = int(input("Qual é o seu peso? "))
altura = float(input("Qual é a sua altura? "))
IMC = peso / altura**2
print("O seu IMC é", round(IMC, 2))
```

- Para executar o código é necessário gravá-lo primeiro. Para tal, pressionar a tecla F5 para aparecer a caixa de diálogo “*Source Must Be Saved. OK to Save?*” e, seguidamente, pressionar o botão OK
- Atribuir um nome ao ficheiro (por ex., ‘imc’) e seleccionar uma pasta onde ficará guardado (os ficheiros do *Python* têm a extensão .py)
- Assim que o ficheiro é gravado, o código é executado e o resultado é apresentado na *Shell* do *Python*

# Programas ramificados

- O programa *imc.py* é um programa em linha recta (“straight-line”), ou seja, executa declaração após declaração na ordem descrita, até que o código termine
- Os programas ramificados (“branching”) não necessitam de executar todas as instruções do código. A declaração mais simples de uma ramificação é uma declaração **condicional**



# Declaração condicional *if-else*

- No *Python*, a declaração condicional *if-else* tem a seguinte forma:

*if* expressão booleana:

bloco de código

(*elif* expressão booleana:

bloco de código)

(*elif* expressão booleana:

bloco de código)

(...)

*else*:

bloco de código

# Declaração condicional *if-else*

- No exemplo do programa *imc.py*, uma declaração condicional pode ser a seguinte:

```
if IMC >= 30:
```

```
    print ("obeso")
```

```
else:
```

```
    print ("não obeso")
```

```
print ("teste condicional")
```

# Indentação

- O *Python* usa a **indentação** como forma de relacionar blocos de código, ou seja, partes do código estão “dentro” de outras partes do código (sendo destacadas usando a tecla de tabulação)
- Uma vantagem da indentação é que a estrutura visual de um programa reflecte a estrutura semântica desse mesmo programa
- No exemplo anterior, se a última declaração estiver indentada, fará parte do bloco **else** em vez do bloco que constitui a declaração condicional;

```
if IMC >= 30:
```

```
    print (“obeso”)
```

```
else:
```

```
    print (“não obeso”)
```

```
    print (“teste condicional”)
```

# Exercício

- Abrir o programa do índice de massa corporal construído anteriormente e editar o código de forma a incluir uma declaração condicional que tenha em conta as seguintes alternativas e use o comando *print()* para apresentar o estado nutricional:
  - Se o IMC for inferior a 18.5, o estado nutricional é “baixo peso”
  - Se o IMC for maior ou igual a 18.5 e menor que 25, o estado é “peso normal”
  - Se o IMC for maior ou igual a 25 e menor que 29.9, o estado é “excesso de peso”
  - Quando o IMC é maior que 30, o estado é “obesidade”

**Nota:** Para cada alternativa, usar a palavra reservada *elif* dentro do bloco *if-else*
- Testar o programa com diferentes valores de peso e altura

# Exercício – solução 1

```
peso = int (input ("Qual é o seu peso? "))
altura = float (input ("Qual é a sua altura? "))
IMC = peso / altura**2
if IMC < 18.5:
    print("baixo peso")
elif 18.5 <= IMC < 25:
    print("peso normal")
elif 25 <= IMC < 29.9:
    print("excesso de peso")
else:
    print("obesidade")
```

# Exercício

- Usar o código do programa do índice de massa corporal (*imc.py*) para construir um programa mais elaborado que permita fazer o seguinte:
  - Guardar o estado nutricional numa variável designada “estado”
  - Apresentar o resultado do programa na seguinte forma escrita:  
Ex: O seu IMC é 25.60 e corresponde a um estado nutricional de excesso de peso
- Testar o programa com diferentes valores de peso e altura



## Exercício – solução 2

```

peso = int (input ("Qual é o seu peso? "))
altura = float (input ("Qual é a sua altura? "))
IMC = peso / altura**2
if IMC < 18.5:
    estado = "baixo peso"
elif 18.5 <= IMC < 25:
    estado = "peso normal"
elif 25 <= IMC < 29.9:
    estado = "excesso de peso"
else:
    estado = "obesidade"
print ("O seu IMC é", round (IMC, 2), "e corresponde a um estado nutricional de", estado)

```

# Ciclo *for*

- O *Python* possui mecanismos de linguagem que permitem iterar sobre uma determinada sequência (por ex., uma *string*). Um destes mecanismos é o ciclo *for*, cuja forma geral de declaração é a seguinte:

*for* variável *in* sequência:

    bloco de código

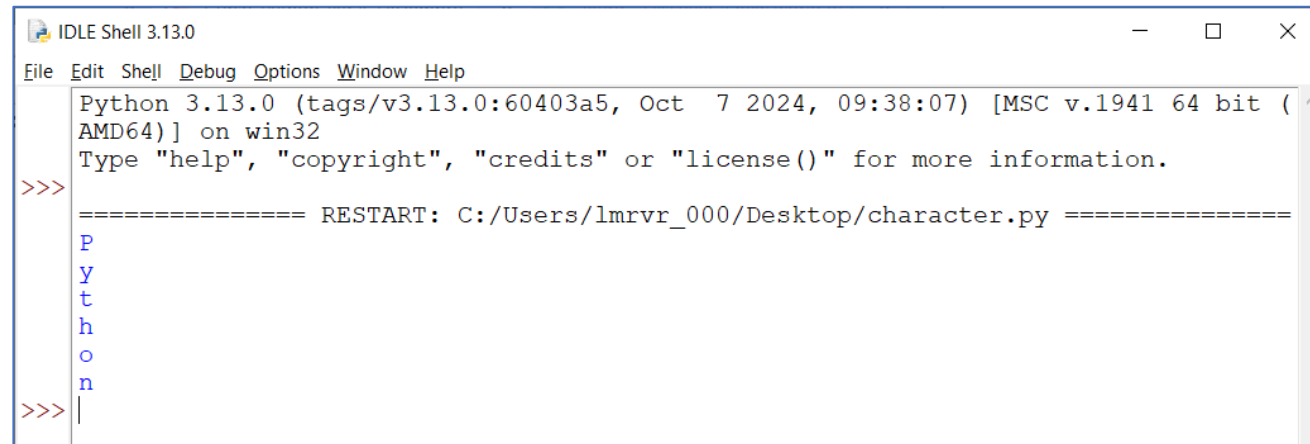
- Da mesma forma que na declaração condicional *if-else*, o bloco de código do ciclo *for* está indentado
- A variável recebe cada valor da sequência por ordem e executa o bloco de código até chegar ao fim da sequência (ou existir uma declaração de *break* na sequência)

# Ciclo *for*

- Exemplo de um ciclo *for* muito simples:

*for* character in "Python":

*print* (character)



```

IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/lmrvr_000/Desktop/character.py =====
P
Y
t
h
o
n
>>>
    
```

# Ciclo *for*

- A sequência de valores atribuídos à variável é normalmente gerada usando a função *built-in range()*, que pode ter 3 argumentos: *início*, *fim* e *passo*. Esta função produz a seguinte progressão aritmética: *início*, *início+passo*, *início+passo\*2*, etc.
- Se o passo é um valor positivo, o último elemento da sequência é o maior inteiro  $\text{início} + \text{passo} * i$  menor do que *fim*
- Se o passo é um valor negativo, o último elemento da sequência é o menor inteiro  $\text{início} + \text{passo} * i$  maior do que *fim*
- Se o primeiro argumento é omitido, o valor por defeito é 0 (zero). Se o último argumento é omitido, o valor por defeito é 1

# Exercícios

- Escrever os seguintes ciclos *for* no editor do *Python*, correr o código e interpretar os resultados obtidos:
  - `for inteiro in range(0, 6, 1):`  
    `print ("Python"[inteiro])`
  - `for inteiro in range(0, 6, 2):`  
    `print ("Python"[inteiro])`
  - `for inteiro in range(5, 0, -1):`  
    `print ("Python"[inteiro])`
  - `for inteiro in range(5, -1, -1):`  
    `print ("Python"[inteiro])`

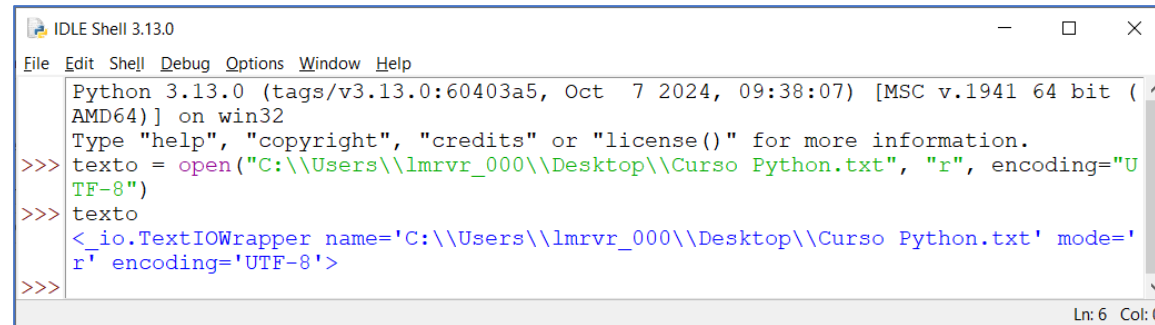
# Abrir e ler ficheiros

- O *Python* permite ler ficheiros em formato de texto ou binário para serem usados nos programas
- Se se tratar de um texto, este deve ser escrito num editor de texto (por ex., *Notepad*) e gravado com extensão *.txt*
- O ficheiro é passado para o *Python* como uma variável usando a função *open()*. Esta função tem normalmente 2 argumentos:
  - nome (e localização) do ficheiro (*string*)
  - modo de utilização do ficheiro (*r* – *read*, *w* – *write*, *a* - *append*, *r+* - *read and write*) (*string* opcional)
- Por exemplo, o seguinte comando é usado para abrir o ficheiro “Curso Python.txt” e passá-lo para a variável ‘texto’ em modo de leitura:

```
>>> texto = open (“Curso Python.txt”, “r”, encoding="UTF-8")
```

# Abrir e ler ficheiros

- Ao escrever-se o nome da variável no interpretador, pode obter-se alguma informação sobre o tipo de ficheiro que se abriu:

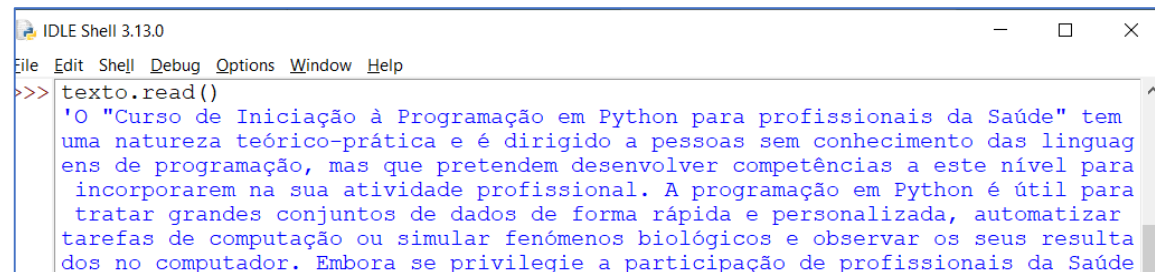


```

Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> texto = open("C:\\Users\\lmrvr_000\\Desktop\\Curso Python.txt", "r", encoding="UTF-8")
>>> texto
<_io.TextIOWrapper name='C:\\Users\\lmrvr_000\\Desktop\\Curso Python.txt' mode='r' encoding='UTF-8'>
>>>
    
```

- Para ler o conteúdo do ficheiro, usa-se a função *read()*:

```
>>> texto.read()
```



```

Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> texto.read()
'O "Curso de Iniciação à Programação em Python para profissionais da Saúde" tem uma natureza teórico-prática e é dirigido a pessoas sem conhecimento das linguagens de programação, mas que pretendem desenvolver competências a este nível para incorporarem na sua atividade profissional. A programação em Python é útil para tratar grandes conjuntos de dados de forma rápida e personalizada, automatizar tarefas de computação ou simular fenómenos biológicos e observar os seus resultados no computador. Embora se privilegie a participação de profissionais da Saúde'
    
```

# Abrir e ler ficheiros

- Assim como nas *strings*, é possível indexar caracteres individuais do texto. Por exemplo, o seguinte comando lê as primeiras 10 posições do texto\*:

```
>>> texto.read(10)
```

```
>>> texto = open("C:\\Users\\lmrvr_000\\Desktop\\Curso Python.txt", "r", encoding="UTF-8")
>>> texto
<_io.TextIOWrapper name='C:\\Users\\lmrvr_000\\Desktop\\Curso Python.txt' mode='r' encoding='UTF-8'>
>>> texto.read(10)
'O Curso de'
>>> |
```

Ln: 18 Col: 0

- A função *readline()* mostra a primeira linha de texto\*:  

```
>>> texto.readline()
```
- O ficheiro pode ser fechado usando a seguinte declaração:  

```
>>> texto.close()
```

\*Nota: O ficheiro tem de ser aberto novamente porque o Python não volta a ler do início.



# Guardar ficheiros de texto

- O *Python* também permite guardar texto em ficheiros. Por exemplo, para criar um ficheiro designado “Curso Python.txt” pode ser usada a seguinte declaração\*:

```
>>> ficheiro = open("Notas Python.txt", "w")
```

- Usando a função *write()* pode escrever-se um texto no ficheiro “Notas Python.txt”:

```
>>> ficheiro.write("Este curso de Python é o máximo!")
```

- O ficheiro é fechado usando a função *close()*:

```
>>> ficheiro.close()
```

```
>>> ficheiro = open("C:\\Users\\lmrvr_000\\Desktop\\Notas Python.txt", "w")
>>> ficheiro.write("Este curso de Python é o máximo!")
32
>>> ficheiro.close()
>>> |
```

Ln: 11 Col: 0

\* Nota: Se já existir no mesmo diretório um ficheiro com o mesmo nome, este será substituído pelo novo ficheiro.

# Exercício

- As sequências de DNA que se encontram depositadas em bases de dados são muito utilizadas em estudos de biologia molecular. Usando como exemplo a sequência de DNA do gene da beta-globina humana (nº de acesso: AH001475.2), vamos tentar obter várias estatísticas relacionadas com este gene\*:
  - a) Comprimento total da sequência do gene da beta-globina
  - b) Número total de nucleótidos A (adenina)
  - c) Concatenar as sequências dos 4 exões da beta-globina:
    - Exão 1: posições 1612..1703
    - Exão 2: posições 1834..2056
    - Exão 3: posições 2907..3015
    - Exão 4: posições 3972..4072

\* Nota: A sequência do gene da beta-globina encontra-se no ficheiro 'Homo\_sapiens\_beta-globin\_gene\_complete\_sequence'.

# Exercícios

- d) Determinar o comprimento total da sequência codificante
- e) Determinar o número de aminoácidos da proteína
- f) Identificar a localização (posições de início e fim) da sequência génica reguladora 'TATA\_box' (motivo "ATAAAA")
- g) Obter a sequência inversa e complementar do gene da beta-globina

# Fim da sessão 2

