

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 3: Objectos estruturados

- **Conteúdos:**
 - Estruturas de dados
 - Tuplos
 - Listas
 - Dicionários
 - Comentários do código
 - Exercícios

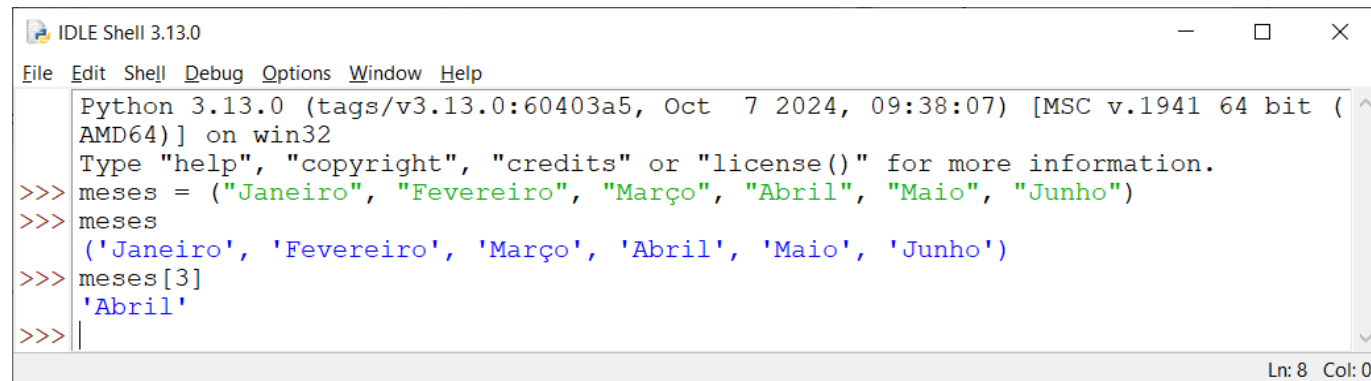
Tuplos

- Os **tuplos** são estruturas de dados **imutáveis**, ou seja, são apropriados para armazenar dados fixos
- Os tuplos são criados colocando os itens de dados separados por vírgulas dentro de **parêntesis curvos**. Por exemplo:

```
>>> meses = ("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho")
```

- Os itens de um tuplo podem ser indexados. Por exemplo:

```
>>> meses[3]
```



```

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.
>>> meses = ("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho")
>>> meses
('Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho')
>>> meses[3]
'Abril'
>>>
    
```

Tuplos

- É possível criar tuplos dentro de tuplos. Por exemplo:

```
>>> investigadores = (("John Snow", "Inglaterra"), ("Louis Pasteur", "França"), ("Ricardo Jorge", "Portugal"))
```

- Para referenciar os elementos deste tipo de tuplo, a indexação é diferente do habitual. Assim, o primeiro tuplo interno é acedido usando:

```
>>> investigadores[0]
```

- O primeiro elemento do segundo tuplo interno é acedido da seguinte forma:

```
>>> investigadores[1][0]
```

```
>>> investigadores = (("John Snow", "Inglaterra"), ("Louis Pasteur", "França"), ("Ricardo Jorge", "Portugal"))
>>> investigadores[0]
('John Snow', 'Inglaterra')
>>> investigadores[1][0]
'Louis Pasteur'
>>> |
```

Ln: 13 Col: 0

Tuplos

- Os tuplos também podem ser concatenados como as *strings*. Por exemplo:

```
>>> meses_1a6 = ("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho")
>>> meses_7a12 = ("Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro")
>>> meses = meses_1a6 + meses_7a12
```

```
>>> meses_1a6 = ("Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho")
>>> meses_7a12 = ("Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro")
>>> meses = meses_1a6 + meses_7a12
>>> meses
('Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho', 'Julho', 'Agosto', 'Setembro', 'Outubro', 'Novembro', 'Dezembro')
>>>
```

Ln: 20 Col: 0

Tuplos

- Depois de criados, não é possível adicionar ou retirar itens dos tuplos. No entanto, é possível descompactar os tuplos atribuindo os itens de dados a variáveis. Por exemplo:

```
>>> médico= ("Denis Mukwege", "Congolês")
```

```
>>> (nome, nacionalidade) = médico
```

```
>>> nome
```

```
>>> nacionalidade
```

```
>>> médico = ("Denis Mukwege", "Congolês")
>>> (nome, nacionalidade) = médico
>>> nome
'Denis Mukwege'
>>> nacionalidade
'Congolês'
>>> |
```

Ln: 29 Col: 0

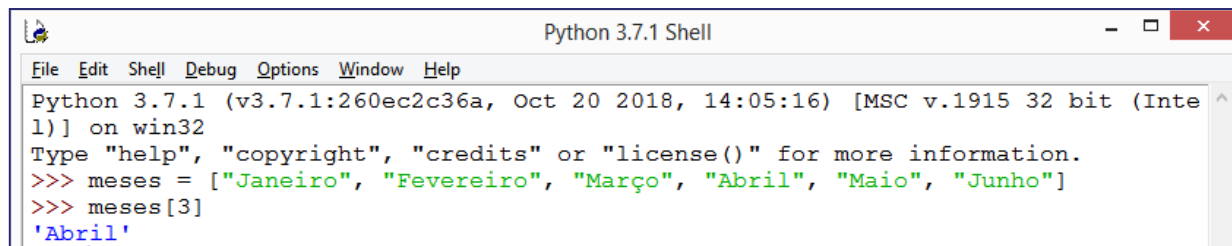
Listas

- As listas são um tipo de estrutura de dados **mutável** que é normalmente usada para armazenar colecções de itens homogêneos
- As listas são criadas colocando os itens separados por vírgulas dentro de **parêntesis rectos**. Por exemplo:

```
>>> meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho"]
```

- Os itens de uma lista podem ser indexados. Por exemplo:

```
>>> meses[3]
```



```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho"]
>>> meses[3]
'Abril'
```

Listas

- É possível fazer o *slicing* de uma lista para obter apenas alguns itens específicos. Por exemplo:
`>>> meses[0:3]`
- As listas permitem actualizar, inserir, remover ou adicionar novos itens usando diferentes operações. Por exemplo, para adicionar o mês *Julho* à lista de meses usa-se o seguinte comando:

`>>> meses.append("Julho")`

```
>>> meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho"]
>>> meses[0:3]
['Janeiro', 'Fevereiro', 'Março']
>>> meses.append("Julho")
>>> meses
['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho', 'Julho']
>>>
```

Ln: 50 Col: 0

Listas

- As listas contêm um método adicional (*sort*) que permite ordenar os itens de uma lista, usando somente comparações. Por exemplo, a ordenação da lista de meses dá o seguinte resultado:

```
>>> meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho"]
```

```
>>> meses.sort()
```

```
>>> meses
```

```
>>> meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho"]
>>> meses.sort()
>>> meses
['Abril', 'Fevereiro', 'Janeiro', 'Junho', 'Maio', 'Março']
>>> |
```

Ln: 55 Col: 0

Dicionários

- Os dicionários são estruturas de dados **mutáveis** em que cada item do dicionário é um par composto por uma **Chave** e por um **Valor**
- Num dado dicionário, a Chave tem de ser única (número ou *string*), enquanto o Valor pode ser qualquer tipo de dado
- Cada chave e respetivo valor do dicionário são introduzidos separados por dois pontos (:) dentro de **chavetas**. Por exemplo:

```
>>> investigadores = {"John Snow": "Inglaterra", "Louis Pasteur": "França", "Ricardo Jorge": "Portugal"}
```

Dicionários

- Uma vantagem do dicionário relativamente às listas é que os valores podem ser acedidos através das respetivas chaves e não da posição que ocupam no dicionário. Por exemplo:

```
>>> investigadores["John Snow"]
```

- Um item pode ser adicionado ao dicionário da seguinte forma:

```
>>> investigadores["James Watson"] = "EUA"
```

- Um item pode ser removido do dicionário usando o comando `del`:

```
>>> del investigadores["Ricardo Jorge"]
```

```
>>> investigadores = {"John Snow": "inglaterra", "Louis Pasteur": "França", "Ricardo J
orge": "Portugal"}
...
>>> investigadores["John Snow"]
...
'inglaterra'
>>> investigadores["James Watson"] = "EUA"
...
>>> del investigadores["Ricardo Jorge"]
...
>>> investigadores
...
{'John Snow': 'inglaterra', 'Louis Pasteur': 'França', 'James Watson': 'EUA'}
>>> |
```

Ln: 35 Col: 0

Operações suportadas por tipos de sequências mutáveis

Operation	Result	Notes
<code>s[i] = x</code>	item <i>i</i> of <i>s</i> is replaced by <i>x</i>	
<code>s[i:j] = t</code>	slice of <i>s</i> from <i>i</i> to <i>j</i> is replaced by the contents of the iterable <i>t</i>	
<code>del s[i:j]</code>	same as <code>s[i:j] = []</code>	
<code>s[i:j:k] = t</code>	the elements of <code>s[i:j:k]</code> are replaced by those of <i>t</i>	(1)
<code>del s[i:j:k]</code>	removes the elements of <code>s[i:j:k]</code> from the list	
<code>s.append(x)</code>	appends <i>x</i> to the end of the sequence (same as <code>s[len(s):len(s)] = [x]</code>)	
<code>s.clear()</code>	removes all items from <i>s</i> (same as <code>del s[:]</code>)	(5)
<code>s.copy()</code>	creates a shallow copy of <i>s</i> (same as <code>s[:]</code>)	(5)
<code>s.extend(t)</code> or <code>s += t</code>	extends <i>s</i> with the contents of <i>t</i> (for the most part the same as <code>s[len(s):len(s)] = t</code>)	
<code>s *= n</code>	updates <i>s</i> with its contents repeated <i>n</i> times	(6)
<code>s.insert(i, x)</code>	inserts <i>x</i> into <i>s</i> at the index given by <i>i</i> (same as <code>s[i:i] = [x]</code>)	
<code>s.pop()</code> or <code>s.pop(i)</code>	retrieves the item at <i>i</i> and also removes it from <i>s</i>	(2)
<code>s.remove(x)</code>	removes the first item from <i>s</i> where <code>s[i]</code> is equal to <i>x</i>	(3)
<code>s.reverse()</code>	reverses the items of <i>s</i> in place	(4)

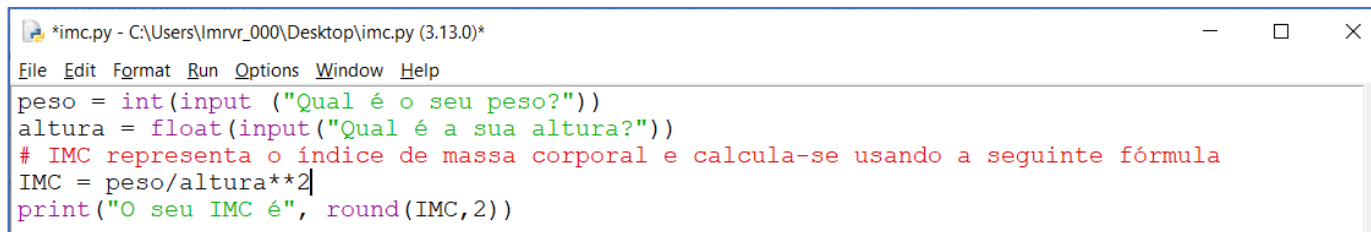
Comentários do código

- Quando alguém escreve o código de um programa, toda a informação que está na base do mesmo reside somente no programador
- Se outro programador ler o código poderá, por exemplo, ter dúvidas sobre o que representa uma dada variável ou o que faz uma dada secção do código
- Frequentemente, até o programador que escreveu o código poderá ter-se esquecido do papel de alguns comandos ou secções, se não usar o programa durante um longo período de tempo
- Para evitar estas e outras situações, os programadores devem ter a boa prática de tornar o seu código compreensível, usando **comentários** em determinados comandos ou secções.

Comentários do código

- O símbolo # permite ao programador introduzir uma linha de texto para explicar o que representa uma secção particular do código
- O símbolo # e o texto à sua direita apresentam uma cor própria que permite distinguir da linguagem do código
- Quando o código é executado, os comentários não são visíveis para o utilizador
- Por exemplo, poderá ser incluído o seguinte comentário no programa que calcula o índice de massa corporal:

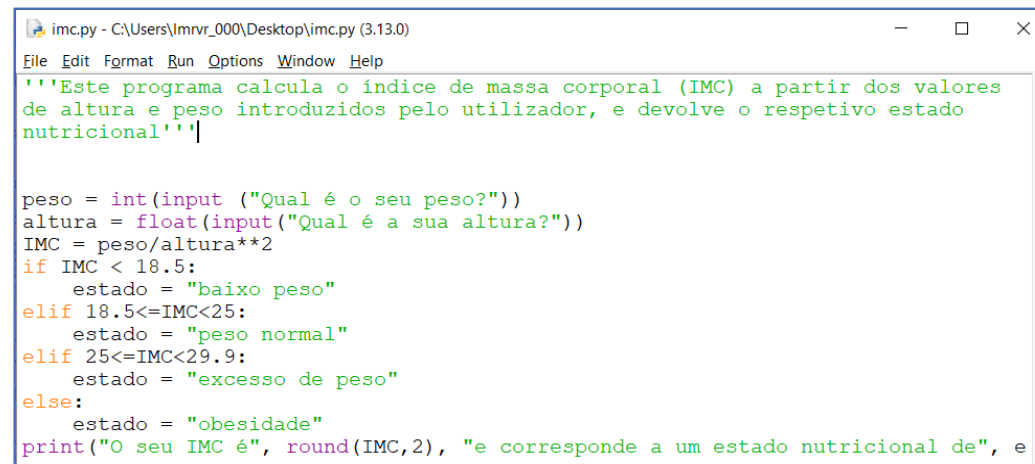
IMC representa o índice de massa corporal e calcula-se usando a seguinte fórmula



```
*imc.py - C:\Users\Imrvr_000\Desktop\imc.py (3.13.0)*
File Edit Format Run Options Window Help
peso = int(input("Qual é o seu peso?"))
altura = float(input("Qual é a sua altura?"))
# IMC representa o índice de massa corporal e calcula-se usando a seguinte fórmula
IMC = peso/altura**2
print("O seu IMC é", round(IMC,2))
```

Comentários do código

- Os comentários também podem ser incluídos dentro de aspas triplas ("""). Esta alternativa é normalmente usada para comentários extensos, nomeadamente para informar outro utilizador sobre a função do programa
- Por exemplo, no caso do programa que calcula o índice de massa corporal, poderia ser incluído o seguinte comentário inicial:



```
imc.py - C:\Users\Imrvr_000\Desktop\imc.py (3.13.0)
File Edit Format Run Options Window Help
'''Este programa calcula o índice de massa corporal (IMC) a partir dos valores
de altura e peso introduzidos pelo utilizador, e devolve o respetivo estado
nutricional'''

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", e
```

Exercícios

- No dia 5 de novembro de 2024 a OMS definiu um conjunto de 17 agentes patogénicos com necessidade urgente de vacinas. Escreva um pequeno *script* que percorra o seguinte tuplo usando um ciclo *for* e imprima na *Shell* do *Python* todas as bactérias:

```
agentes = (("bactéria", "Streptococcus tipo A"), ("vírus", "hepatite C"), ("vírus", "VIH-1"), ("bactéria", "Klebsiella pneumoniae"),
("vírus", "citomegalovírus"), ("vírus", "influenza"), ("parasita", "Leishmania"), ("bactéria", "Salmonella"), ("vírus", "norovírus"),
("parasita", "Plasmodium falciparum"), ("bactéria", "Shigella"), ("bactéria", "Staphylococcus aureus"), ("vírus", "dengue"),
("bactéria", "Streptococcus tipo B"), ("bactéria", "Escherichia coli"), ("bactéria", "Mycobacterium tuberculosis"), ("vírus", "vírus sincicial respiratório"))
```

Nota: O tuplo 'agentes' pode ser copiado do ficheiro "agentes.txt" para o editor do *Python*.

Exercícios

- Vamos supor que o conjunto de agentes patogénicos definido anteriormente pela OMS será alvo de uma revisão em 2025, após o surgimento de novos dados epidemiológicos. Assim, efetue uma atualização do tuplo do conjunto de agentes de acordo com o seguinte:
 - Inclua a bactéria *Yersinia*
 - Elimine o vírus dengue
 - Substitua a "Escherichia coli" pela "Escherichia coli extraintestinal"

Exercícios

- As sequências de DNA codificantes podem ser usadas para derivar as sequências das respetivas proteínas. Tendo como base a sequência codificante do gene da beta-globina humana, conseguem criar um programa que faça a tradução proteica e apresente a sequência da proteína beta-globina?

Nota: Copiar o dicionário *cod_gen* (ficheiro 'código_genético') para o editor do *Python*.

Fim da sessão 3

