

Programação Web - Ficha 3 de Python: Funções, ficheiros e classes

OBJECTIVO:

- Nesta ficha exercitará conceitos adquiridos sobre funções e seus módulos, manuseamento e gestão de ficheiros e criação de classes.
- Aprenderá a utilizar módulos para a manipulação de ficheiros CSV e JSON, assim como a criação de gráficos.

PRÉ-REQUISITOS:

- Instale o Pycharm (aconselhado, da JetBrains, versão Profissional para estudante. Use as credenciais da Lusofona como na instalação do IntelliJ) ou o VS Code (instale uma extensão para Python).
- Crie uma pasta `pw-python-03` onde guardará este enunciado. Para cada exercício crie uma pasta.
- Crie um repositório no GitHub onde irá sincronizando gradualmente os exercícios que for fazendo.

PRAZO:

- Finalize e submeta a ficha antes da sua aula prática da semana de 10-14 de maio.

Exercício 1

Manipulação de ficheiros JSON

JSON (JavaScript Object Notation) é um formato de dados popular para representar dados estruturados. Será usado no primeiro exercício desta ficha.

- É comum transmitir e receber dados entre um servidor e uma aplicação web no formato JSON.
- É comum guardar um objeto JSON num ficheiro.
- O módulo `json` permite trabalhar com o formato JSON. Deverá instalá-lo com o comando `python -m pip install json`.
- um dicionário Python pode ser convertido diretamente numa string JSON (com o método `json.dumps()`) ou num ficheiro JSON (com o método `json.dump()`).
- Uma string JSON ou um ficheiro JSON podem ser convertidos diretamente num dicionário Python com os métodos `json.loads()` e `json.load()`, respetivamente.
- Encontra mais detalhes [aqui](https://secure.grupolusofona.pt/ulht/moodle/pluginfile.php/800079/course/section/398731/pw-03-python-03.1-csv-json.pdf) (<https://secure.grupolusofona.pt/ulht/moodle/pluginfile.php/800079/course/section/398731/pw-03-python-03.1-csv-json.pdf>).

Exemplo de utilização:

In [78]:

```
import json

pessoa_dict = {
    "nome": "Pedro",
    "linguas": ["Portugues", "Espanhol"],
    "casado": True,
    "esposa": "Ines",
    "idade": 32,
    "filhos": {
        "Afonso":12,
        "Beatriz":10,
        "Joao":7,
        "Diniz":4
    }
}

with open('pessoa.json', 'w') as json_file:
    json.dump(pessoa_dict, json_file, indent = 4)
```

Enunciado do Exercício 1

Crie uma pasta `exercicio_1`. Nesta, crie o *package* `analisa_ficheiro`, pasta que deverá ter os 3 seguintes módulos:

- `acessorio.py` que contém as seguintes funções:
 - `pede_nome` que pede o nome de um ficheiro de texto (com extensão `.txt`). Deverá verificar se o ficheiro existe. Caso não exista, volte a pedir. Caso exista, retorna numa string o nome do ficheiro.
 - `gera_nome` que recebe o nome do ficheiro e cria o nome onde guardará os resultados da análise do ficheiro, em formato json. Por exemplo, se o ficheiro for `historia.txt`, deverá devolver `historia.json`
- `calculos.py` que contém as seguintes funções:
 - `calcula_linhas` que recebe o nome do ficheiro e retorna o número de linhas do ficheiro
 - `calcula_carateres` que recebe o nome do ficheiro e retorna o número de carateres do ficheiro
 - `calcula_palavra_comprida` que retorna a palavra mais comprida do ficheiro.
 - `calcula_ocorrencia_de_letras` que recebe o nome do ficheiro e cria um dicionário de todas as letras do ficheiro, indicando a quantidade de vezes que cada letra ocorre. Deverá desprezar se é maiúscula ou minúscula. Por exemplo, para o ficheiro com conteúdo "Abracadabra, pura magia!" terá `{"a":8, "b":2, "c":1, ...}`
- `__init__` que indica que a pasta `analisa_ficheiro` é um módulo. Importe os dois módulos do *package*, para que as funções fiquem disponíveis quando importar o *package*.

Crie o módulo `principal.py` que:

- importe o *package* `analisa_ficheiro`.
- importe o módulo `json`
- defina a função `main`, que deverá utilizar todas as funções disponíveis no *package* `analisa_ficheiro` para extrair a informação sobre o ficheiro e guardá-la no ficheiro de resultados no formato json
- avalie se o módulo foi executado como script, e caso positivo, invoque a função `main()`, da seguinte forma:

```
if __name__ == "__main__":  
    main()
```

A estrutura ficará:

```
| principal.py  
|  
└─ analisa_ficheiro  
    acessorio.py  
    calculos.py  
    __init__.py
```

Exercício 2

Manipulação de ficheiros CSV

- Um ficheiro Comma Separated Value (CSV) tem a seguinte sintaxe:
 - os valores são separados por , (mas pode convencionar-se outro delimitador como ;)
 - Não deve conter espaços.
 - Qualquer campo pode estar entre aspas.
 - Item com várias palavras ou vírgulas devem ser envolvidos entre " " .
 - Pode haver campos vazios e quebras de linha.
- O módulo `csv` do Python permite manipular ficheiros CSV. Deverá instalá-lo com o comando `python -m pip install csv`. Encontra mais detalhes [aqui](https://secure.grupolusofona.pt/ulht/moodle/pluginfile.php/800079/course/section/398731/pw-03-python-03.1-csv-json.pdf) (<https://secure.grupolusofona.pt/ulht/moodle/pluginfile.php/800079/course/section/398731/pw-03-python-03.1-csv-json.pdf>).
- Neste exercício escreverá um dicionário num ficheiro CSV. Eis um exemplo de como o poderá fazer:

In [80]:

```
import csv
with open('pessoas.csv', 'w', newline='') as ficheiro:
    campos = ['Nome', 'Idade']
    writer = csv.DictWriter(ficheiro, fieldnames=campos)
    writer.writeheader()
    writer.writerow({'Nome': 'Luis', 'Idade': 27})
    writer.writerow({'Nome': 'Marcelo', 'Idade': 26})
    writer.writerow({'Nome': 'Ana', 'Idade': 20})
```

Criação de gráficos com Matplotlib

O módulo `matplotlib` permite fazer facilmente gráficos de linha, scatter, barras e queijos. Deverá instalá-lo com o comando `python -m pip install matplotlib`.

Usaremos neste exercício grafico de barras e queijos. Apresentam-se em baixo duas funções que criam gráficos e que irá integrar no exercício. Neste caso, recebem um título, e duas listas, uma de chaves e outra de valores.

In [79]:

```
# O modulo matplotlib permite fazer desenho de gráficos
from matplotlib import pyplot as plt

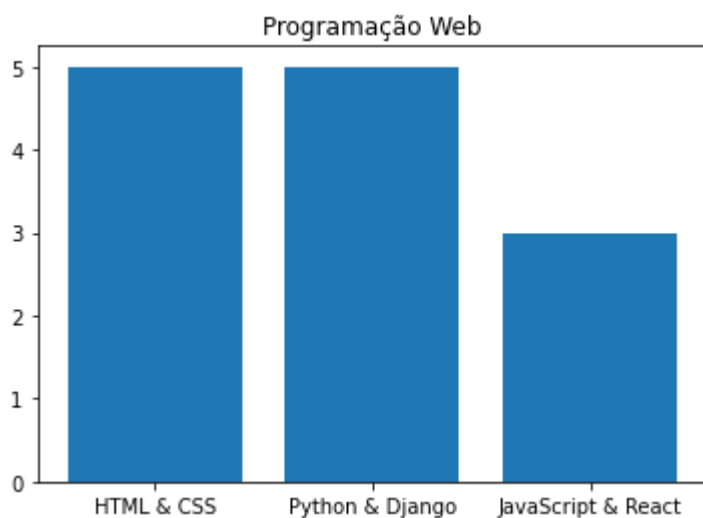
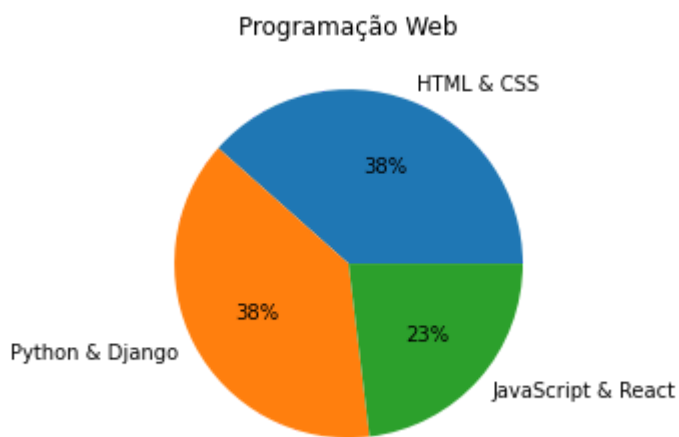
def faz_grafico_queijos(titulo, lista_chaves, lista_valores):

    plt.pie(lista_valores, labels=lista_chaves, autopct='%1.0f%%')
    plt.title(titulo)
    plt.show()

def faz_grafico_barras(titulo, lista_chaves, lista_valores):

    plt.bar(lista_chaves, lista_valores)
    plt.title(titulo)
    plt.show()

lista_chaves = ['HTML & CSS', 'Python & Django', 'JavaScript & React']
lista_valores = [5, 5, 3]
titulo = 'Programação Web'
faz_grafico_queijos(titulo, lista_chaves, lista_valores)
faz_grafico_barras(titulo, lista_chaves, lista_valores)
```



Enunciado do Exercício 2

Crie uma pasta `exercicio_2`. Escreva um programa em Python usando o VS Code que analisa os tipos de ficheiros existentes numa pasta. Despreze as sub-pastas existentes e seus conteúdos.

A. O módulo `os` tem uma série de métodos que serão úteis: `os.getcwd()`, `os.listdir()`, `os.path.isfile()`, `os.path.isdir()`.

B. Defina na pasta `exercicio_2` o módulo `analise_pasta.py` que tem várias funções:

1. `pede_pasta` pede ao utilizador para inserir um caminho para uma pasta.
2. `faz_calculos`, que, para os tipos de ficheiros existentes, contabiliza a quantidade de ficheiros e volume total ocupado em kBytes. Utilize um dicionário que tenha como chaves a extensão dos tipos de ficheiro encontrados. Como valor, utilize um dicionário para guardar informação da quantidade e volume. Exemplo de dicionário:

```
dic_info = {'pdf': {'volume': 4114203, 'quantidade': 5}, 'pptx': {'volume': 891758, 'quantidade': 2}, ...}
```

3. `guarda_resultados` que guarda a informação num ficheiro CSV (com o nome da pasta e extensão csv), e indica na consola o nome do ficheiro com resultados.
4. `faz_grafico_queijos`. Utilize a função em baixo para representar os resultados numa pie chart.
5. `faz_grafico_barras`. Utilize a função em baixo para representar os resultados num gráfico de barras.

C. Crie na pasta `exercicio_2` o módulo `main.py` que importa o módulo e utiliza as funções disponíveis. Este deverá ser executado pela linha de comandos como um script: `python main.py`

Ambos os ficheiros `main.py` e `analise_pasta.py` ficarão na mesma pasta.

Exemplo de interação:

```
Este programa analisa o tipo de ficheiros presente numa pasta. Insira o caminho
para uma pasta: c:\users\lsf\varios
Os resultados foram guardados no ficheiro `varios.csv`
```

Exemplo de conteúdo do ficheiro CSV:

```
Extensao,Quantidade,Tamanho[kByte]
pdf,5,4018
xlsx,4,220
pptx,2,871
png,3,188
```

Exercício 3

Crie uma pasta `exercicio_3`. Este exercício apenas recorrerá a um módulo.

- Crie a função `pede_pasta()` (semelhante a `pede_nome` do módulo do exercício 1), para pedir o caminho (relativo ou absoluto) de uma pasta para analisar. Se a pasta não existir, deverá pedir novamente. Esta função deverá retornar o caminho da pasta a analisar (se estiver na pasta `exercicios_3` basta o nome da pasta, mas pode ser por exemplo `C:\Users\lsf`).
- Crie a função recursiva `calcula_tamanho_pasta(pasta)` que:
 - recebe o caminho completo de uma pasta a analisar.
 - calcula o tamanho total em MBytes dos ficheiros nela contidos, tanto na pasta como nas sub-pastas.
 - Sugestão:
 - defina uma variável `soma` para somar o volume total dos ficheiros da pasta.
 - itere pelos elementos da pasta (com o método `os.listdir(pasta)`)
 - construa o caminho absoluto, com `os.path.join(pasta, elemento)`
 - se for um ficheiro (verifique¹ com `os.path.isfile()`), adicione o seu tamanho a `soma` .
 - Se for uma pasta (verifique¹ com `os.path.isdir()`), adicione a `soma` o resultado de `calcula_tamanho_pasta(pasta)`
 - no final do ciclo retorne `soma` .
- Crie uma função `main()` que utilize as duas funções e imprima na consola o tamanho total da pasta e subpastas.
- Avalie se o módulo foi executado como um script. Caso positivo, invoque a função `main()` , da seguinte forma:

```
if __name__ == "__main__":  
    main()
```

¹: Certos ficheiros não são devidamente reconhecidos como ficheiro ou pasta, pelo que devemos testar ambas as condições, em vez de usar `else`.

Exercício 4

Crie uma pasta `exercicio_4`.

Os automóveis mais recentes mostram a distância que é possível percorrer até ser necessário um reabastecimento. Pretende-se criar esta funcionalidade em Python através da classe `automovel`. Esta classe é construída indicando os seguintes atributos:

- `cap_dep`, a capacidade do depósito.
- `quant_comb`, a quantidade de combustível no depósito.
- `consumo`, o consumo do automóvel em litros aos 100 km.

A classe `automovel` apresenta também os seguintes métodos:

- `combustivel`: devolve a quantidade de combustível no depósito;
- `autonomia`: devolve o numero de Km que é possível percorrer com o combustível no depósito;
- `abastece(n_litros)`: aumenta em `n_litros` o combustível no depósito e retorna a autonomia. Se este abastecimento exceder a capacidade do depósito, gera um erro e não aumenta a quantidade de combustível no depósito;
- `percorre(n_km)` percorre `n_km` Km, desde que a quantidade de combustível no depósito o permita, em caso contrário gera um erro e o trajecto não é efectuado. Retorna a autonomia.

Crie um módulo `carro.py` que tenha:

- a classe declarada
- uma função `main` que apresente na consola um menu de opções para gerir um carro que crie, permitindo utilizar os métodos disponíveis, apresentando informação devidamente formatada.
- avalie se o módulo foi executado como script, e, caso positivo, invoque a função `main()` da seguinte forma:

```
if __name__ == "__main__":  
    main()
```

Exemplo de interação da classe:

```
>>> a1 = automovel(60, 10, 15)  
>>> a1.autonomia()  
66  
>>> a1.abastece(45)  
366  
>>> a1.percorre(150)  
216  
>>> a1.percorre(250)  
-1
```

Deverá criar um interface adequado que mostre esta informação de forma mais "amigável"

In []:

```
def multiplica_lista(lista):  
    produto = 1  
    for numero in lista:  
        produto *= numero  
    return produto  
  
multiplica_lista(lista)
```