Aula 2 - PARI

Sumário

```
Editores (pycharm e sua configuração)
Primeiros programas em Python (continuação)
Ainda o cálculo de números primos
Ciclos (for, while)
Uso de funções
Introdução aos comentários em pydoc
Separação em ficheiros distintos
Tipos de dados em python
```

Algumas notas em resumo da aula anterior

Informação sobre os ficheiros SumarioA1.adoc, SumarioA2.adoc, SumarioA3.adoc, etc.

Tal como referido na aula anterior, estes ficheiros de Sumários são legíveis no formato ASCII, mas podem ser convertidos em HTML para uma melhor apresentação usando o comando `asciidoc`. Para usufruir desta facilidade, intalar o `asciidoc` e o `source-highlight`:

```
sudo apt install asciidoc
sudo apt install source-highlight
```

Depois basta fazer `asciidoc SumarioA2.adoc` na linha de comando e gera-se o ficheiro `SumarioA2.html` que pode ser visualizado num qualquer browser, como o firefox. Se se usar a opção `-a data-uri` o ficheiro HTML gerado inclui todas as imagens e outros elementos multimédia no próprio ficheiro HTML, o que pode ser prático para efeitos de portabilidade do documento: `asciidoc -a data-uri SumarioA2.adoc`.

Pode também visualizar diretamente os ficheiros adoc no github em

https://github.com/miguelriemoliveira/pari_20-21

ou usar um IDE (por exemplo o pycharm) para o fazer.

Editor

A ferramenta principal para criar e modificar ficheiros é o editor, muitas vezes integrado num ambiente de desenvolvimento (IDE). Há inúmeras opções desde simples editores (`gedit`, `kate`, `kwrite`, etc.) até ambientes de desenvolvimento muito sofisticados (`codeblocks`, `eclipse`, `vscode`, etc.).

Além das propriedades fundamentais dos editores, hoje em dia são excelentes *add-ons* a "automated completion" (preenchimento automático de palavras e estruturas) , o "syntax highlight"

(realce da sintaxe da linguagem), o "intellissense" (apresentação de todas as opções de preenchimento automático de campos e estruturas em variáveis, funções, etc.), ou a inserção automática de fragmentos de código padrão ("code snippets").

O editor com mais tradição por excelência é o "vim" (ou "vi" improuved) mas a sua utilização eficaz pode requerer anos de prática continuada e permite todas as facilidades indicadas acima, mas a sua configuração, por ser praticamente ilimitada, pode-se tornar complexa e, por isso, contraproducente em utilizadores iniciados.

Recomenda-se como IDE o pycharm. Pode-se utilizar a licença profissional ou a licença base. Os estudantes podem solicitar acesso à licença profissional.

Outra alternativa uma solução disponibilizada pela Microsoft para muitas plataformas, incluindo linux. Trata-se do *Visual Studio Code* ou *vscode*. Pode ser instalado diretamente do gestor de aplicações do Ubuntu (*Ubuntu Software*) ou por outras vias (https://askubuntu.com/questions/616075/how-do-i-install-visual-studio-code).

Diretivas gerais para uma melhor organização do trabalho nas aulas

- Cada aula deve estar numa pasta (diretório): Aula1, Aula2, etc.
- Os exercício 1 e 2 da aula de hoje devem ser feitos cada um no seu diretório (criar um para cada caso), mas a partir do Exercício 3 a metodologia consiste em manter-se no mesmo diretório e cada novo exercíco corresponderá a uma função nova criada, como se explica adiante.
- Múltiplas alíneas de um exercício fazem parte desse exercício.
- Em alguns casos poderá ser útil duplicar um diretório existente (para criar novo exercício ou uma nova aula). Para criar o diretório e todo o seu conteúdo recursivamente para o novo pode-se usar o seguinte comando que copia toda a estrutura a começar em `ex1` para o diretório `ex2` que é criado se não existir.

```
cp -rp ex1 ex2
```

Exercício 1

Copiar o exercício dos números perfeitos da aula anterior. Introduzir comentários no início do ficheiro `perfeito.py` e antes de cada função para a descrever. Os comentários devem acrescentar informação relevante ao código. Ver tutorial sobre como comentar código python.

perfeitos.py

```
#!/usr/bin/env python
# -------------------------------------------------
# A simple python script to print hello world!
# Miguel Riem Oliveira.
# PARI, Setember 2020.
# -------------------------------------------------

maximum_number = 100  # maximum number to test.


def getDividers(value):
    """
    Return a list of dividers for the number value
    :param value: the number to test
    :return: a list of dividers.
    """
    # <Add stuff here>
    return []


def isPerfect(value):
    """
    Checks whether the number value is perfect.
    :param value: the number to test.
    :return: True or False
    """

    # <Add stuff here>
    return False


def main():
    print("Starting to compute perfect numbers up to " + str(maximum_number))

    for i in range(0, maximum_number):
        if isPerfect(i):
            print('Number ' + str(i) + ' is perfect.')


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

Como executar o pydoc para gerar a documentação.

O pydoc permite gerar um html com documentação sobre um módulo python partindo dos comentários inseridos em formato docstring. Para instalar:

```
sudo apt-get install python-doc
```

e depois, na pasta de um qualquer exercício:

```
pydoc -w ./
```

Mais informação sobre o pydoc em https://packagecontrol.io/packages/PyDOC

Exercício 2

Separar o ficheiro primo.py em dois ficheiros distintos.

```
main.py -> inclui a função principal (main)
my_functions.py -> contém a função de teste dos números.
```

Manter/reforçar os comentários apropriados.

Executar o script e gerar documentação com o pydoc.

Exercício 3

Criar uma cópia do Exercício anterior e modificar o programa nos seguintes pontos:

```
ficheiro main.py:
    Para interpretar um parâmetro de entrada (que será um limite
    máximo de até onde procurar números primos) e:
```

```
ficheiro my_functions.py:
    Para o teste de divisores se fazer até à raiz quadrada do número
    e não até ao próprio número. Aumenta a eficiência do cálculo.
```

Usar um argumento de linha de comandos para definir o número máximo a testar (usar o package argparse). No final deve-se executar com:

```
main --max_numnber 1000
```

Exercício 4: Ciclos "for", "while" e input do terminal

4 a)

Escrever uma função `printAllPreviousChars()` que não tem parâmetros e não retorna nada, mas que quando executada lê um caractere do teclado usando a função readchar e imprime todos os caracteres desde o espaço ' ' até esse caractere, considerando a numeração de caracteres da tabela asccii.

Completar o código fornecido com o código que falta.

main.py

```
def printAllCharsUpTo(stop_char):
    # <to complete>

def main():
    # <to complete>

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

4 b)

Adicionar a função *readAllUpTo(stop_char)* para ler caracteres de forma contínua e terminar quando chegar o caractere *'X'*.

4 c)

Criar a função *countNumbersUpto(stop_char)* para ler caracteres continuamente e terminar quando chegar o caractere *'X'*, e nessa altura indicar quantos caracteres são algarismos e quantos não são algarismos. Usar a função `isnumeric()`.

main.py

```
def countNumbersUpTo(stop_char):
    total_numbers = 0
    total_others = 0
    while True:
        # add code here ...

    print('You entered ' + str(total_numbers) + ' numbers.')
    print('You entered ' + str(total_others) + ' others.')
```

Exercício 5 - Tipos de dados em python

O python é uma linguagem denominada dynamically typed, por oposição às linguagens typed como o c em que é preciso indicar explicitamente o tipo de cada variável, ou outras em que as variáveis não têm tipo associado, como no caso do javascript.

Na prática, não é obrigatóriamente necessário indicar o tipo de uma variável e quando o tipo não é indicado, este é deduzido a partir de regras estabelecidas.

Mais informação:

https://www.tutorialspoint.com/What-are-the-differences-between-untyped-and-dynamically-typed-programming-languages

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

5 a)

Usando como ponto de partida o exercício 4, Alterar a função anterior para criar uma lista dos inputs realizados e processar essa lista (para calcular o número de digitos/outros) a posteriori. Esta forma é mais **pythonic**!

main.py

```
def countNumbersUpTo(stop_char):

    while True:
        # add code here to create a list of inputs

    total_numbers = 0
    total_others = 0
    for input in inputs:
        # process each input in the list

    print('You entered ' + str(total_numbers) + ' numbers.')
    print('You entered ' + str(total_others) + ' others.')
```

5 b)

Crie uma lista que contenha apenas os inputs numéricos que foram inseridos (pela ordem em que foram inseridos).

5 c)

Crie um dicionario apenas com os inputs *other* em que as chaves são a ordem dos inputs inseridos e o valor são os inputs.

5 d)

Reordene a lista da alínea 5 b) de modo a que esteja por ordem crescente do valor dos inputs.

5 e)

O python tem uma functionalidade chamada list comprehension que permite gerar a lista de números numa só linha de código. Veja o link e tente refazer a alínea 5 b) usando uma list comprehension.