IPS Instituto Politécnico de Setúbal Escola Superior de Tecnologia de Setúbal	COMPUTAÇÃO PARALELA E DISTRIBUÍDA 2022/2023
	Laboratório 0: Conceitos Importantes da Linguagem Python

Objetivos do laboratório

Pretende-se que o aluno:

1. Trabalhe com alguns dos tipos de dados mais relevantes da linguagem Python.
2. Utilize módulos em Python.
3. Defina e invoque funções em Python.
4. Utilize o código fornecido para obter informações sobre seu computador

Pré-requisitos:

- Python com versão acima de 3.5 (Para saber a versão do Python: na linha de comandos ou num terminal do IDE, faça ***python -version***)
- O IDE o PyCharm Community Edition (ou outro)

Conceitos Principais e Práticas

Utilize a consola de Python para executar cada um dos exemplos apresentados

Sites de Referência

Operações com *strings*: https://docs.python.org/3/library/string.html

Listas e outras estruturas de dados: https://docs.python.org/3/tutorial/datastructures.html

Funções matemáticas: https://docs.python.org/3/library/math.html

O que é uma string?

As strings em Python são arrays (vetores) de bytes representando caracteres unicode. As strings são envolvidas por aspas duplas ou simples:

```
>>> hello = 'alo'
>>> hello = "alo"
```

Pode-se mostrar o conteúdo da string com a função *print ()*:

```
>>> print('alo')
alo
>>> print(hello)
alo
```

Se for necessário, é possível fazer uma string multi-linha delimitando-a com três aspas (simples ou duplas):

```
>>> b = """Esta string é
... multilinha usando
... três aspas como delimitador"""
>>> print(b)
Esta string é
multilinha usando
três aspas como delimitador

>>> c = "Esta string é\nmultilinha usando\no carater de mudança de linha"
>>> print(c)
Esta string é
multilinha usando
o carater de mudança de linha
```

Pode-se utilizar parênteses retos para aceder aos elementos da string:

```
>>> print(hello[1])
l
```

Como as strings são vetores, pode-se usar ciclos *for* sobre os seus carateres:

```
>>> for x in hello:
>>>     print(x)
a
l
o
```

Operações úteis com strings incluem:

- o método *len (string)*, que retorna o tamanho da string

  ```
  >>> print(len(hello))
  3
  ```

- as palavras-chave *in* e *not in*, para verificar se um carater ou conjunto de carateres está ou não está presente numa string:

  ```
  >>> texto = "Melhores alunos estão no IPS!"
  >>> print('alunos' in texto)
  True
  ```

- obter uma substring da string:

  ```
  >>> texto = "Melhores alunos estão no IPS!"
  >>> print(texto[1:6])   # mostra os carateres do índice 1 ao índice 5
  elhor
  >>> print(texto[:6])   # mostra os carateres do índice 0 ao índice 5
  Melhor
  >>> print(texto[9:])   # mostra os carateres do índice 9 ao final
  alunos estão no IPS!
  ```

- usar o método *replace(string1,string2)* para substituir uma string ou parte de uma string por outra:

  ```
  >>> print(texto.replace('IPS','Politécnico de Setúbal'))
  Melhores alunos estão no Politécnico de Setúbal!
  ```

- usar o método *split(delimitador)* para dividir a string em substrings se encontrar instâncias do delimitador:

```
>>> print(texto.split(' '))
['Melhores', 'alunos', 'estão', 'no', 'IPS!']
```

- usar o operador '+' para concatenar strings:

```
>>> hello = 'alo'
>>> world = "mundo"
>>> frase = hello + ' ' + world
>>> print(frase)
>>> alo mundo
```

- usar o método *format(valor)* para concatenar strings com números. A string tem de ter sido definida com a indicação de onde vai o número ou números (*placeholder*) usando chavetas:

```
>>> idade = 35
>>> dizer = 'Meu nome é João e tenho {} anos'
>>> print(dizer.format(idade))
Meu nome é João e tenho 35 anos
```

  Se houver mais de um placeholder, pode-se colocar um número dentro das chavetas a indicar o posicionamento correto dos argumentos:

```
>>> dizer = 'Meu nome é João e tenho {1} anos e {0} filhos'
>>> print(dizer.format(2,idade))
Meu nome é João e tenho 35 anos e 2 filhos
```

- usar o método *zfill(valor)* para preencher a string com zeros até a string completar o tamanho desejado:

```
>>> print(frase)
alo mundo
>>> zfrase = frase.zfill(15)
>>> print(zfrase)
000000alo mundo
```

Outras operações úteis com strings podem ser encontradas no link referenciado anteriormente.

O que é uma lista?

Lista é uma coleção de itens separados por vírgulas entre parênteses retos. O conteúdo de uma lista podem ser diferentes tipos de itens. As listas são poderosas. Pode-se alterar, adicionar ou remover elementos da lista muito facilmente.

Nesta etapa, verá como armazenar, aceder e manipular dados em listas: o primeiro passo para trabalhar de forma eficiente com grandes quantidades de dados. Vamos praticar usando o repl:

1. Alterar elementos da lista

```
>>> hello = 'alo'       # aspas simples ou duplas, dá no mesmo
>>> minha_lista = [hello, 'mundo', 3.14, 45]
>>> minha_lista
['alo', 'mundo', 3.14, 45]        # mostra com aspas simples
>>> minha_lista[2]
3.14
>>> minha_lista[2] = 33
```

```
>>> minha_lista
['alo', 'mundo', 33, 45]
>>> minha_lista[1:3]      # obtém uma sub-lista
['mundo', 33]             # o elemento no índice 3 não é incluído!!!
```

2. Adicionar elementos na lista

```
>>> nova_lista = minha_lista + ['adicione', 'elemento']
>>> nova_lista
['alo', 'mundo', 33, 45, 'adicione', 'elemento']
```

3. Remover elementos da lista

```
>>> del nova_lista[0]
>>> nova_lista
['mundo', 33, 45, 'adicione', 'elemento']
```

O que é uma matriz?

No Python, tal como em outras linguagens, uma matriz é um vetor ou array multi-dimensional. Podemos tratar uma lista de listas como uma matriz. Por exemplo, uma matriz de 2 linhas e 3 colunas:

```
>>> A = [[1, 4, 5],
...     [-5, 8, 9]]
>>>
>>> A
[[1, 4, 5], [-5, 8, 9]]
```

Os exemplos a seguir ilustram algumas formas de trabalhar com listas encadeadas:

```
>>> M = [[1, 4, 5, 12],
...     [-5, 8, 9, 0],
...     [-6, 7, 11, 19]]
>>>
>>> print("M =", M)
M = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]
>>> print("M[1] =", M[1])      # segunda linha
M[1] = [-5, 8, 9, 0]
>>> print("M[1][2] =", M[1][2])   # terceiro elemento da segunda linha
M[1][2] = 9
>>> print("M[0][-1] =", M[0][-1])   # último element da primeira linha
M[0][-1] = 12
>>>
>>> coluna = [];        # lista vazia
>>> for linha in M:
...   coluna.append(linha[2])
...
>>> print("Terceira coluna =", coluna)
Terceira coluna = [5, 9, 11]
```

Para pequenas tarefas de computação, é suficiente trabalhar com listas. Em tarefas mais abrangentes, vamos preferir utilizar o pacote ***NumPy*** para trabalhar com matrizes:

```
L:\cpd\py> pip install numpy
Collecting numpy
  Downloading numpy-1.20.1-cp39-cp39-win_amd64.whl (13.7 MB)
     |……….. 13.7 MB 3.3 MB/s
Installing collected packages: numpy

L:\cpd\py> python
>>>
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> print(a)              # Output: [1, 2, 3]
[1 2 3]
>>> print(type(a))        # Output: <class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

O que é uma função?

Funções são códigos reutilizáveis. Funções são usadas para resolver uma tarefa específica. O objetivo é reduzir a quantidade de código necessária para resolver problemas desafiadores . Em vez de repetir a escrita do código inteiro, pode-se inserir uma função e chamar essa função sempre que precisar executar o código.

Algumas funções já existentes no Python:

max () : retorna o elemento com o valor máximo na lista
min () : retorna o elemento com o valor mínimo na lista
len () : retorna o comprimento da lista ou uma string
index () : retorna o índice de um elemento da lista. (Este é um método, será abordado após as funções)

Uma função definida pelo utilizador consiste numa sequência de instruções que são executadas quando a própria função é invocada. Por exemplo, o seguinte programa permite-nos fazer print de duas palavras cada vez que se invoca a função *do_hello()*:

```
def do_hello():
    print(“Hello”)
    print(“World”)

do_hello()
```

As funções podem receber parâmetros e podem retornar valores (usando a palavra *return*):

```
>>> def add_one(x):
...     print("Got x=",x)
...     return x + 1
...
>>> value = add_one(1)
Got x= 1
>>> value
2
```

Também pode-se definir funções recursivas, que recorrem a elas próprias de modo a resolver o problema. Tome como exemplo a seguinte função *fatorial* que calcula o fatorial de um número:

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x-1)

Implemente esta função e teste com vários valores.
```

O que é um método?

Os métodos são funções integradas que podem ser chamadas para tipos específicos de elementos. Por exemplo, o objeto de lista tem seus próprios métodos, e o objeto de string tem seus próprios métodos. Vou mostrar alguns exemplos para cada tipo para dar uma ideia de como os métodos funcionam.

Métodos de lista

index (*elemento*): retorna o número do índice de um elemento específico.
count (*elemento*): retorna a quantidade do elemento específico na lista.

Métodos de String

capitalize ( ): retorna o elemento após colocar a primeira letra da string em maiúscula.
replace (*string1,string2*): retorna o elemento após substituir uma string específica por outra string.

Conheça o Seu Computador

Informação sobre o sistema operativo

Salve o código a seguir com o nome *checkSys.py*:

```
# Computação Paralela e Distribuída 2020/2021
# Aluno: (nome do aluno)
# Número: (número do aluno)
# Turma: (turma do aluno)
###############################################
import platform, subprocess

def get_processor_info():
    if platform.system() == "Windows":
        return platform.processor()
    elif platform.system() == "Darwin":
        return subprocess.check_output(['/usr/sbin/sysctl',"-n", "machdep.cpu.brand_string"]).strip()
    elif platform.system() == "Linux":
        command = "cat /proc/cpuinfo | grep name | cut -d ' ' -f 3-8"
        return str(subprocess.check_output(command, shell=True)).strip("b'\\n")
    return ""

if __name__ == '__main__':
    print(get_processor_info())
```

Para entregar a tarefa deste lab, inclua comentários neste código. As seguintes linhas devem ser obrigatórias no início de todos os códigos entregues:

```
# Computação Paralela e Distribuída 2020/2021
# Aluno: (nome do aluno)
# Número: (número do aluno)
# Turma: (turma do aluno)
###############################################
```

Além disso, e para este código, **inclua comentários indicando** (pesquise…):

a. o que faz o *import*
b. quais são algumas funcionalidades dos módulos importados
c. o que faz a função *str ( )*
d. o que faz o método *strip( )*

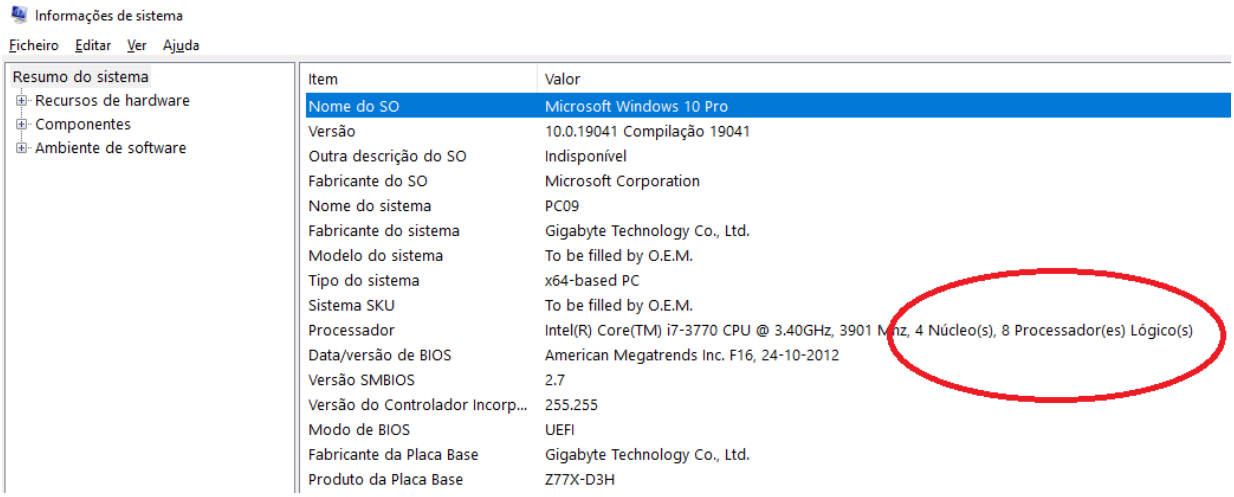
O meu computador pode executar programas em paralelo? Quantos cores físicos tem a CPU do meu computador?

Nem sempre a primeira informação sobre o número de cores para um computador é útil para o conceito de executar código em paralelo. Isto devido ao conceito de *hyperthreading*, que faz o computador "ver" mais cores lógicos do que físicos.

No **Windows**, pode usar o comando abaixo, fora do ambiente Python, para saber quantos cores físicos tem o processador do seu computador

```
msinfo32
```

este comando lança uma interface gráfica onde pode ver a informação desejada, como na figura:



§FIM DO LAB 0§