

# LINGUAGENS DE PROGRAMAÇÃO

Engenharia Informática

# Exceções

- Evento que resulta de uma quebra (erro) do fluxo normal de execução.
  - *Normalmente resulta de uma anomalia.*
- Quando o Python não consegue lidar com uma situação e o Código não pode prosseguir levanta-se (raise) uma exceção.
  - *Ex:*

```
>>> 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

# Lidar com a Excepção

- O código que se pensa onde poderá ocorrer uma excepção coloca-se dentro de um bloco try/except.

```
try:
    x=3/0
except ZeroDivisionError:
    print("Excepção detetada.")
Opcional: { else:
              print("Executou sem problema.")
Opcional: { finally:
              print("E finalmente... Código de Limpeza")
```

- Mais tipos de excepções:
  - <https://docs.python.org/3/library/exceptions.html>

```
try:
    x=3/0
    x=int("letras")
except ZeroDivisionError:
    print("ZeroDivisionError detetada.")
except ValueError:
    print("ValueError detetada.")
except:
    print("Detetada outra exceção.")
```

```
try:
    x=3/0
    x=int("letras")
except ZeroDivisionError as erro:
    print("Detetada.", erro)
except ValueError as erro:
    print("Detetada.", erro)
except:
    print("Detetada outra exceção.")
```

```
try:
    x=3/0
    x=int("letras")
except Exception as erro:
    print("Detetada.", erro)
```

# Mais um exemplo:

```
def getNumero():  
    while True:  
        try:  
            x = int(input("Insira um número:"))  
        except ValueError:  
            print("Tente novamente...")  
        else:  
            return x
```

# Sinalizar uma exceção

- As exceções são "capturadas", mas são geradas nalgum sítio.
- O **raise** permite gerar essas exceções em código.

```
def funcao( valor ):  
    if valor < 1:  
        raise Exception("Valor inserido inválido! " + str(valor))
```

```
    print("Este código já não é executado se valor < 1.")
```

```
funcao(-1)
```

```
try:  
    x = funcao(-1)  
except:  
    x = funcao(10)
```

# Ficheiros I/O

```
f = open('ficheiro.txt', 'r')  
read_data = f.read()  
f.close()
```

Exepção:  
FileNotFoundError

```
with open('ficheiro.txt') as f:  
    read_data = f.read()
```

Abre e fecha o ficheiro no final

```
if f.closed:  
    print("Ficheiro está fechado")
```

# Open modes

- 'r' default – Abre para leitura.
  - 'w' Abre para escrita. Se não existe tenta criar. Se existir escreve por cima.
  - 'x' Cria novo ficheiro. Aborta se ficheiro já existir.
  - 'a' Como 'w' mas com cursor no fim – append.
- 
- 't' Modo de texto.
  - 'b' Modo binário.



# Ler e escrever

`f.read()` *#Lê o ficheiro todo*

`f.readline()` *#Lê uma linha e avança para a seguinte*

`f.write("Texto para o ficheiro")`

`f.write("Linha1\nLinha2")`

# Armazenar/Comunicar dados

- Formatos de ficheiros para guardar conteúdos (human-readable).

*Multi-plataforma e Multi-Linguagem*

–CSV (*Comma-Separated Values*)

- `import csv`

–XML (*Extensible Markup Language (XML)*)

- `from lxml import etree`

–JSON (*JavaScript Object Notation*)


- `import json`

–...

# CSV

**'fx.csv'**

|                  |         |         |
|------------------|---------|---------|
| Coluna1          | Coluna2 | Coluna3 |
| Text1 na coluna1 | texto2  | texto3  |



```
import csv
with open('fx.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print (' <-> '.join(row))
```

# JSON - JavaScript Object Notation

**'fx.json'**

```
{"nome": "Manuel", "idade": 48, "isMan": true}
```

Está armazenado como um dicionário (chave-valor). Mas podia ser uma lista de dicionários.

**import** json

**with** open('fx.json') **as** f:  
    data = json.load(f)  
    print (data['nome'])

data['idade'] = 20

**with** open('fx\_2.json', 'w') **as** outfile:  
    json.dump(data, outfile)

# Serializar

- Imagine que pretende enviar/armazenar o conteúdo de um objeto.  
De forma a poder reconstruir o objeto.
- Transforma o objeto numa String (pode ser binária). E vice-versa.
- Python
  - *marshal* – mais rápido
  - *pickle* - completo

# Serializar – a ter em atenção

- Quando o conteúdo serializado é armazenado ou transferido
  - *Versões da classe (do objeto) devem ser idênticas*
  - *Versões do serializador (ex: pickle)*
  - *Só funciona de Python para Python*
- Alternativas?
  - *XML – JSON - etc*

# Serializar - Exemplo

```
class Fruta:  
    def __init__(self, cor, valor):  
        self.cor = cor  
        self.valor = valor
```

```
banana = Fruta( 'amarelo', 30 )
```

```
import pickle
```

```
with open("Frutas.obj","wb") as f:  
    pickle.dump(banana, f)
```

```
with open("Frutas.obj",'rb') as f:  
    banana_nova = pickle.load(f)
```

```
print(banana_nova.cor, banana_nova.valor)
```

# Fim

- Questões?