FICHA TRABALHO 5

MODALIDADE:	Aprendizagem +	Escolha um item.	
CURSO:	Técnico/a Especialista em Aplicações Informáticas de Gestão		
UC:	Desenvolver projeto de tecnologias e aplicações informáticas de gestão	CÓDIGO UC:	00891
FORMADOR/A:	Bruno Silva	DATA:	

OBJETIVOS

- Programação defensiva e tratamento de exceções com Python

Tratamento de exceções

Quando estamos a aprender Python, um dos maiores desafios é entender como lidar com erros que ocorrem durante a execução do programa. No entanto, a linguagem Python oferece mecanismos para capturar e tratar esses erros, utilizando a programação defensiva dos blocos conhecidos por **try** e **except** (e opcionalmente **finally**)

(**Nota Importante**: em outras linguagens de programação são as exceções try-catch).

O que são exceções (Exceptions) em Python?

Exceções (Exceptions) em Python são erros que ocorrem durante a execução de um programa, sendo utilizadas para sinalizar situações excecionais que podem interromper o fluxo normal do código. A principal forma de tratar esses erros é utilizando os **blocos try-except**.

```
try:
    #código para executar no bloco try
except:
    #se ocorrer algum erro, o bloco except
    #vai capturar o erro e podem tratar do mesmo
```

No **bloco try**, colocamos o código que é para ser executado na aplicação. Desta forma, estamos a dizer ao programa "tenta correr este código de uma forma mais segura, mas poderá ocorrer uma exceção". Se ocorrer alguma exceção/erro no código do try, o programa irá saltar para o bloco do except.

Para capturar as exceções, utilizamos a instrução **exception** e opcionalmente, podemos mostrar os detalhes do erro no programa (quando necessário), caso contrário estamos dependentes da consola. Este passo é ótimo quando queremos guardar os erros gerados num ficheiro, base de dados, entre outros suportes de logs/monitorizações de software.

Para mostrar detalhes específicos do erro no programa, podemos usar a sintaxe:

***except Exception* as ...**

onde nas reticências, podemos dar um nome sugestivo ao erro a ser capturado (neste caso, foi colocado o nome erro como forma sugestiva). Esta instrução permite capturar a exceção para uma variável e exibir informações detalhadas sobre o erro, como por exemplo:

```
1  try:
2      num1 = int(input("Insira um número: "))
3      num2 = int(input("Insira outro número: "))
4      resultado = num1 / num2
5      print(f"O resultado da divisão é: {resultado}")
6
7  except Exception as erro:
8      print(f"Ocorreu o seguinte erro: {erro}")
```

Neste caso, estamos a utilizar a classe **Exception** que permite **capturar os vários tipos de exceções que podem ocorrer na aplicação**, embora que mais à frente vamos ver como apanhar exceções de forma mais específica.

Exemplos de Exceptions mais comuns

ValueError: ocorre quando uma função recebe um argumento com a conversão do tipo certo, mas valor completamente inapropriado:

```
numero = int("ola")
```

Ao correr o programa será dada a seguinte informação:

```
Traceback (most recent call last):
  File "c:\Users\Silva\Documents\projeto\projeto\testes.py", line 1, in <module>
    numero = int("ola")
ValueError: invalid literal for int() with base 10: 'ola'
```

Esta exceção **ValueError** é acionada quando tentamos trabalhar com valores inteiros/decimais, mas, recebemos outra informação como texto, caracteres especiais, entre outros elementos;

ZeroDivisionError: ocorre quando tentamos dividir um número por zero:

```
resultado = 10 / 0
```

Ao correr o programa será dada a seguinte informação:

```
Traceback (most recent call last):
  File "c:\Users\Silva\Documents\projeto\projeto\testes.py", line 1, in <module>
    resultado = 10 / 0
                ~~~^~~
ZeroDivisionError: division by zero
```

SyntaxError: ocorre quando esquecemos de fechar alguma operação, como por exemplo, as aspas, parênteses, nome de um comando/função mal escrita:

```
print("Hello"
```

Ao correr o programa será dada a seguinte informação:

```
  File "c:\Users\Silva\Documents\projeto\projeto\testes.py", line 1
    print("Hello"
          ^
SyntaxError: '(' was never closed
```

Neste exemplo e como o erro indica, esquecemos de fechar o parenteses do print.

IndentationError: Neste caso não utilizamos a indentação de forma correta (pois o python obriga a utilização da indentação para uma melhor organização do código) e a função print está a dar erro:

```
if True:
print("Indentação incorreta")
```

Ao correr o programa será dada a seguinte informação:

```
  File "c:\Users\Silva\Documents\projeto\projeto\testes.py", line 2
    print("Indentação incorreta")
    ^^^^^
IndentationError: expected an indented block after 'if' statement on line 1
```

Neste exemplo e como o erro indica, esquecemos de dar a indentação depois de escrever a condição do if.

ModuleNotFoundError: ocorre quando tentamos importar uma biblioteca que não está instalada e o Python lança um ModuleNotFoundError. Então, podemos usar try-except para tratar esse erro e fornecer uma mensagem mais clara

```
try:
    import pyspark
except ModuleNotFoundError:
    print("Erro: Biblioteca não encontrada!")
```

Exemplo de Bloco Try-Except (com tratamento de mais do que uma exceção)

Lidar com erros é uma parte essencial do desenvolvimento de software. No Python, uma das formas mais eficazes de tratar esses erros é utilizando o bloco try-except.

Resumindo, o bloco try-except permite que o programa "tente" executar um bloco de código e, caso ocorra um erro, o mesmo vai "capturar" essa exceção e execute um código alternativo para que o programa não pare completamente (a não ser que seja uma exceção muito grave).

Exemplo:

```
try:
    num1 = int(input("Insira um número: "))
    num2 = int(input("Insira outro número: "))
    resultado = num1 / num2
    print(f"O resultado da divisão é: {resultado}")

except ZeroDivisionError:
    print("Erro Divisão: Não é possível dividir por zero!")
except ValueError:
    print("Erro Valor: Deve inserir apenas números.")
```

Neste exemplo, temos o seguinte código:

- O bloco **try** tenta executar a divisão de dois números que vão ser inseridos na consola;

```
1   try:
2       num1 = int(input("Insira um número: "))
3       num2 = int(input("Insira outro número: "))
4       resultado = num1 / num2
5       print(f"O resultado da divisão é: {resultado}")
```

- **No entanto pode acontecer duas situações:**

  Se for feita a operação da divisão por zero, será acionada a exceção **ZeroDivisionError**. Então nesse caso, se a exceção ocorrer, abaixo do código do bloco try, vamos capturar a exceção em causa com bloco except e exibir uma mensagem de erro.

```
except ZeroDivisionError:
    print("Erro Divisão: Não é possível dividir por zero!")
```

  Mas, pode tratar do que uma exceção ao mesmo tempo.

  Logo a seguir podemos criar outro bloco **except** para ser acionado para quando a função **input()** recebe um argumento com a conversão do tipo inteiro, mas valor completamente inapropriado. Nesse caso, vamos capturar a exceção **ValueError** para lidar esse tipo de exceções:

```
 9    except ValueError:
10        print("Erro Valor: Deve inserir apenas números.")
```

Bloco Finally (operação de finalização)

Como garantir a execução de um código, independentemente de erros?

O bloco finally é executado independentemente de qualquer exceção que ocorra. Isso é útil para libertar recursos ou executar ações de limpeza. Exemplo:

```
1   try:
2       num1 = int(input("Insira um número: "))
3       num2 = int(input("Insira outro número: "))
4       resultado = num1 / num2
5       print(f"O resultado da divisão é: {resultado}")
6
7   except Exception as erro:
8       print(f"Ocorreu o seguinte erro: {erro}")
9   finally:
10      print(f"Operação finalizada")
```

Veja que o bloco finally contém o código que será executado independentemente de qualquer exceção. Isso garante que a mensagem “Operação finalizada.” seja exibida, mesmo que ocorra um erro.

Considerações finais

Existem mais tipos de tratamentos de exceções para as mais diversas situações

Para capturar as exceções, utilizamos a instrução **exception** e opcionalmente, podemos mostrar os detalhes do erro no programa (quando necessário), caso contrário estamos dependentes da consola. Este passo é ótimo quando queremos guardar os erros gerados num ficheiro, base de dados, entre outros suportes de logs/monitorizações de software.

Pode consultar exceções mais especificas e para vários casos na seguinte hiperligação: https://realpython.com/ref/builtin-exceptions/

Exercícios

1. Construa um programa para fazer os seguintes passos:
    a. Criar o **bloco try** com as seguintes instruções:
        i. pedir dois números inteiros ao utilizador com a função input();
        ii. criar uma variável e guardar o resultado da divisão;
        iii. mostrar o resultado;
    b. Criar o **bloco except** que vai capturar qualquer exceção com a instrução **Exception** e dar um nome sugestivo ao erro (para ser mais fácil capturar o mesmo);
    c. Criar o **bloco finally** para correr independentemente se executa todo o try ou except (como operação de finalização), em que:
        i. Deve mostrar a mensagem “Programa finalizado”

Quando executar o programa, teste da seguinte forma:

Caso de Teste 1:

1. Introduza dois números inteiros. O programa deve conseguir correr apenas o bloco try e finnaly;
2. Corra novamente o programa e quando pedir um dos valores coloque um número decimal.
3. Se tudo correr bem, o programa deve ter saltado imediatamente para o bloco do Except (devido a exceção) e mostrar apenas o erro “invalid literal for int() ...” e depois disso a mensagem do bloco finally;

Caso de Teste 2:

1. Introduza dois números inteiros. O programa deve conseguir correr apenas o bloco try e finnaly;
2. Corra novamente o programa e no segundo valor coloque o valor 0.
3. Se tudo correr bem, o programa deve ter saltado imediatamente para o bloco do Except (devido a exceção) e mostrar apenas o erro "division by zero" e depois disso a mensagem do bloco finally;

Tratamento de exceções

Nos casos de teste podem acontecer duas situações. Como tal, vamos tratar das exceções de forma mais específica:

Se for feita a operação da divisão por zero, será acionada a exceção **ZeroDivisionError**. Então nesse caso, se a exceção ocorrer, abaixo do código do bloco try, vamos capturar a exceção em causa com bloco except e exibir uma mensagem de erro.

```
except ZeroDivisionError:
    print("Erro Divisão: Não é possível dividir por zero!")
```

Mas, pode tratar do que uma exceção <u>ao mesmo tempo</u>.

Logo a seguir podemos criar outro bloco **except** para ser acionado para quando a função **input()** recebe um argumento com a conversão do tipo inteiro, mas valor completamente inapropriado. Nesse caso, vamos capturar a exceção **ValueError** para lidar esse tipo de exceções:

```
 9  except ValueError:
10      print("Erro Valor: Deve inserir apenas números.")
```

2. Construa um programa para pedir ao utilizador para introduzir a idade. Se o utilizador inserir um valor não numérico (Exceção **ValueError**), o programa deve capturar a exceção e exibir uma mensagem de erro "Erro: Digite um número inteiro válido.";