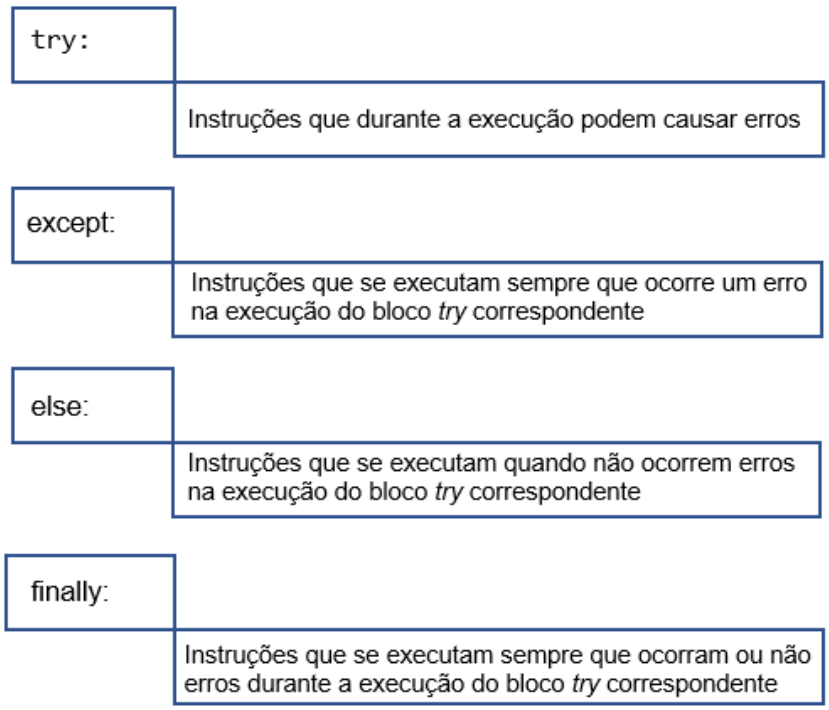
Erros, validação e correção:

A introdução e o processamento de dados podem gerar erros que alteram a execução dos programas. Esses `erros são causados pelo ambiente de programação` ou por `erros de semântica` (como sejam: acesso a uma posição inválida de um vetor, atribuição de uma variável do tipo inteira para uma variável do tipo string) dos programas que executamos.

Um programa robusto `deteta estes erros e trata-os`, isto é, contém procedimentos de deteção e resposta aos erros - `mecanismos de error handling` - para que não seja interrompido sem, pelo menos, informar o utilizador sobre as caraterísticas dos erros que ocorreram.

Um programa é tão mais robusto quantos mais erros detetar, permitir a sua correção e continuar a sua execução.

Estes erros são designados por exceções, uma vez que resultam de situações extraordinárias que alteram a execução geral do programa, e procura dar-lhe resposta, apresentando a seguinte estrutura de tratamento de exceções:



A cada bloco `_try_` podem corresponder `vários blocos _except_`, um para cada tipo de erro. As estruturas de erros podem encaixar-se umas nas outras, sendo resolvidas do exterior para o interior.

A estrutura de tratamento de erros, para além de capturar os erros resultantes da introdução ou tratamento de dados, pode também auxiliar o programador na validação de dados.

Levantar uma exceção

A palavra-chave `raise` é usada para gerar uma exceção. Pode-se definir que tipo de erro gerar e o texto a ser impresso para o utilizador (simula o que é efetuado pelo sistema).

Exemplo 1:

```
x = "Olá"
if not type(x) is int:
  raise TypeError("Apenas são admitidos inteiros!")
```

`Raise` também pode servir para relançar a exceção (neste caso de uma função e a ser capturado no programa principal):

Exemplo 2:

```
def divide(n1, n2):
    if n2 == 0:
        raise ValueError("n2 não pode ser 0.")
    return n1 / n2

try:
    print(divide(2,0))
except ValueError as error:
    print(error)
```

Exercícios: ¶

1. Elabora um programa que calcule:

$$\frac{x}{y-6}$$

   Mas que imprima uma mensagem sempre que o denominador for igual a 0, isto é, sempre que o utilizador digite 6 para y. Considera x e y números reais.

```
x= float(input("Valor de x? "))
y= float(input("Valor de y? "))
try:
    Z=x/(y-6)
except ZeroDivisionError:
    print("Divisão por zero. Y tem de ser diferente de 6")

else:
    print(f"z= {Z:.2f}")
finally:
  print("'try except' completo")
```

2. Define uma função que receba dois parâmetros e os multiplique. Se pelo menos um dos parâmetros for alfanumérico deve ser impressa uma mensagem de erro A função deve imprimir também o número de erros detetados.

```
def Multiplica(x,y):
    global TErros   #declaração de variável global
    try:
        if type(x)==str or type(y)==str:
            raise ValueError
    except:
        print("Erro: as variáveis têm de ser numéricas")
        TErros+=1
    else:
        print(f"{x} x {y} = {x*y}")
    finally:
        print("Nº de erros detetados: ",TErros)
TErros=0
print("Insira 2 valores numéricos para parar insira 0: ")
z=1
while z!='0':
    x= input("Valor de x? ")
    y= input("Valor de y? ")
    Multiplica(x, y)
    z=input("Continuar?")
```

3. Define uma função que receba um inteiro compreendido entre 0 e 200 (excluídos) e calcule o seu inverso. Utiliza as exceções `TypeError` e `ValueError` para validades os dados. Não é necessário usares a cláusula `finally`. Deves criar um programa principal apropriado.

Possível output:

```
Erro. X tem de ser inteiro.
Erro. X deve ser maior que 0 e menor que 200
1/5=0.2
```

4. Elabora uma função que leia e valide a categoria profissional de um funcionário, aceitando apenas as letras C, D ou E.
   Parâmetros e variável categoria:

Parâmetro	Tipo	Significado
Cat	alfanumérico	Categoria profissional

Variável	Tipo	Significado
Valida	lógica	Categoria profissional

Algoritmo:	Output:
Valida=Falso Enquanto Valida=Falso Ler(Cat) Valida =(Cat>=’C’ e Cat<=’E’) Se VALIDA=FALSO ENTÃO ERRo Fim se Fim enquanto	Digita a categoria -> A Digita C, D ou E Digita a categoria -> C Categoria validada!

5. Escreve a função **LerMes**, que dado como parâmetro o nº do mês de nascimento, de 1 a 12, e que caso o valor esteja fora desse intervalo ou não seja inteiro que seja enviada au utilizador uma mensagem de erro, até que o valor esteja correto e a devolver ao programa principal. No programa principal deverá ser definida uma lista com os nomes dos meses e após a invocação da função LerMes faça a correspondência entre o nº lido e o mês de nascimento.

Output:

```
Digita o mês do teu nascimento (de 1 a 12): ago
tem de ser um inteiro entre 1 e 12
Digita o mês do teu nascimento (de 1 a 12):-3
tem de ser um inteiro entre 1 e 12
Digita o mês do teu nascimento (de 1 a 12):12
O teu mês de nascimento foi dezembro!
```

Bibliografia:

https://www.w3schools.com/python/python_try_except.asp
https://docs.python.org/3/library/exceptions.html
Carvalho. Adelaide. (2021). Práticas de Python - Algoritmia e programação. Lisboa: FCA
Vasconcelos, J. (2015). Python - Algoritmia e Programação Web. Lisboa: FCA
https://docs.python.org/3/library/exceptions.html