Fundamentos da Programação
LEIC/LEGM

Aula 9

Funções

Erros. Módulos. Exemplos.

Alberto IST, 2022-23

Erros/Excepções

- Nas aulas anteriores falamos dos tipos de erros: sintaxe, semântica e *runtime*
- As funções podem *lançar* erros quando os argumentos utilizados são de tipo inválido e/ou estão fora do dominio.
    - As excepções interrompen o fluxo de execução, o que não acontece se fizermos um simples *print*
- Para isso podemos utilizar a instrução *raise* que gera um erro de execução, em BNF:

```
<instrução raise> ::= raise <nome>(<mensagem>)
<mensagem> ::= <cadeia de caracteres>
```

- *nome* corresponde à identificação de um dos tipos de erros (ou excepções) conhecidos pelo Python (ou a novos tipos de erros definidos pelo programador):

AttributeError, **IndexError**, **KeyError**, **NameError**, **SyntaxError**, **ValueError** e **ZeroDivisionError**.

Erros/Excepções

Nome	Situação correspondente ao erro
`AttributeError`	Referência a um atributo não existente num objeto.
`ImportError`	Importação de uma biblioteca não existente.
`IndexError`	Erro gerado pela referência a um índice fora da gama de um tuplo ou de uma lista.
`KeyError`	Referência a uma chave inexistente num dicionário.
`NameError`	Referência a um nome que não existe.
`SyntaxError`	Erro gerado quando uma das funções `eval` ou `input` encontram uma expressão com a sintaxe incorreta.
`ValueError`	Erro gerado quando uma função recebe um argumento de tipo correto mas cujo valor não é apropriado.
`ZeroDivisionError`	Erro gerado pela divisão por zero.

Tabela 3.3: Alguns dos identificadores de erros em Python.

- Python (como outras linguagens) fornecem um *protocol* para tratar das excepções (*try*/*except*) que veremos nas próximas semanas

Erros/Excepções, Exemplo:

```
In [50]: ## Definição da função
         def inverte(n):
    if not type(n) == int and not type(n) == float:
         raise ValueError("erro nao e numero")
    if n == 0:
         raise ZeroDivisionError("divisao por 0")

    return 1/n


#invocação/chamada

inverte(0)
```

```
---------------------------------------------
-----------------------------
ZeroDivisionError
Traceback (most recent call last)
<ipython-input-50-a1db5bd29ac6> in <module>
     11 #invocação/chamada
     12
---> 13 inverte(0)
     14

<ipython-input-50-a1db5bd29ac6> in inverte(n)
      4         raise ValueError("erro nao e
numero")
      5     if n == 0:
----> 6         raise ZeroDivisionError("divi
sao por 0")
      7
      8     return 1/n

ZeroDivisionError: divisao por 0
```

Módulos: Importar

- Não é preciso reinventar a roda, Python fornece um grande número de bibliotecas (*libraries*) ou módulos com funções que podemos importar:
- Lista de módulos disponíveis por omissão: https://docs.python.org/3/py-modindex.html

```
<instrução import> ::=
    import <módulo> {as <nome>} NEWLINE |
    from <módulo> import <nomes a importar> NEWLINE

<módulo> ::= <nome>

<nomes a importar> ::= * | <nomes>

<nomes> ::= <nome> | <nome>, <nomes>
```


Módulos: Aceder funções dum módulo

- Necessário no caso de *import* (sem *from*):

```
<composed name> ::= <simple name>.<simple name>
```

Exemplos:

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sin(math.pi/2)
1.0
```

```
>>> from math import pi, sin
>>> pi
3.141592653589793
>>> sin(pi/2)
1.0
```

Módulos: Construir módulos

- Colocar funções num ficheiro *.py* (ex: fp_utils.py)
- Importar utilizando o nome do ficheiro/módulo (sem extensão):

```
>>> import fp_utils
>>> fp_utils.soma_prog_arit(100)
5050
```

```
In [ ]:
```


Funções e parâmetros em Python ++ (opcional)

- Python permite maior flexibilidade na definição e passagem dos parâmetros duma função:
  - **Default parameters**
  - **Keyword arguments**
  - Número variável de parâmetros posicionais e keyword (não nesta disciplina)

```
In [1]: def dividir(num=1, den = 1):
            return num/den

print("Ex1:", dividir(10,2))
print("Ex2:", dividir(10))
print("Ex3:", dividir())
print("Ex3:", dividir(den=4))
```

```
Ex1: 5.0
Ex2: 10.0
Ex3: 1.0
Ex3: 0.25
```

A treinar mais!!!!

Exemplo 2, Potência de dois números inteiros

- Solução iterativa para `k` positivos
- E para `k` negativos?
- E para qualquer `k`?

```
In [2]: def potencia(x, n):
            pass

# Power of two numbers inteiros
x = eval(input("Escreva a base da potencia: "))
n = eval(input("Escreva a potencia: "))


print(potencia(x, n))
```

```
Escreva a base da potencia: 2
Escreva a potencia: 4
None
```


Exemplo 3, Factorial

- Para números inteiros não negativos (`fact(0) = 1`) iterativo

```
In [3]: def factorial(x):
            pass


x = eval(input('Inteiro: '))
f = factorial(x)
print(f)
```

```
Inteiro: 34
None
```

Funções - Tarefas próxima aula

- Trabalhar matéria apresentada hoje
- Fazer todos os exercícios/programas