Fundamentos da Programação
LEIC/LETI

Aula 6

Funções

Visualização e execução de programas. Depuração. Erros. Módulos. Mais exemplos.

Alberto Abad, Tagus Park, IST, 2021-22

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 (https://docs.python.org/3/library/exceptions.html#AttributeError), IndexError (https://docs.python.org/3/library/exceptions.html#IndexError), KeyError (https://docs.python.org/3/library/exceptions.html#KeyError), NameError (https://docs.python.org/3/library/exceptions.html#NameError), SyntaxError (https://docs.python.org/3/library/exceptions.html#SyntaxError), ValueError (https://docs.python.org/3/library/exceptions.html#ValueError) e ZeroDivisionError (https://docs.python.org/3/library/exceptions.html#ZeroDivisionError).

Funções

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 [64]:
```
## 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 ValueError("divisao por 0")

    return 1/n


#invocação/chamada

inverte(-0.5)
```

```
---------------------------------------------------------------------
---------
ValueError                                Traceback (most recent c
all last)
<ipython-input-64-cb9ac1e40b7a> in <module>
     11 #invocação/chamada
     12
---> 13 inverte(-0.5)
     14

<ipython-input-64-cb9ac1e40b7a> in inverte(n)
      4         raise ValueError("erro nao e numero")
      5     if -1 <= n <= 1:
----> 6         raise ValueError("divisao por 0")
      7
      8     return 1/n

ValueError: 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 (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>
```

Funções

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: soma.py)
- Importar utilizando o nome do ficheiro/módulo (sem extensão):

```
>>> import soma
>>> soma.soma(100)
5050
```

In [73]:

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 [ ]:
```
def dividir(num, den = 1):
    return num/den
print("Ex1:", dividir(10,2))
print("Ex2:", dividir(10))
print("Ex3:", dividir(den=2, num=10))
```


Visualização e execução de programas

- http://pythontutor.com/visualize.html#mode=edit (http://pythontutor.com/visualize.html#mode=edit)
- IDEs como o PyCharm e WingIDE

A treinar mais!!!!

Funções

Exemplo 4, Máximo divisor comum (Algoritmo de Euclides)

1. O máximo divisor comum entre um número e zero é o próprio número: `mdc(m,0) = m`
2. Quando dividimos um número *m* por *n*, o máximo divisor comum entre o resto da divisão e o divisor é o mesmo que o máximo divisor comum entre o dividendo e o divisor: `mdc(m, n) = mdc(n, m%n)`

- Exemplo algorimo para mdc(24, 16):

m	n	$m \,\%\, n$
24	16	8
16	8	0
8	0	8

Funções

Exemplo 4, Máximo divisor comum (Algoritmo de Euclides)

In [107]:
```
# Máximo divisor comum (mdc)
# Euclidian algorithm

def mdc(m, n):
    pass

x = eval(input("Da-me valor x:"))
y = eval(input("Da-me valor y:"))
print(mdc(x, y))
```

```
Da-me valor x:24
Da-me valor y:16
8
```

Exemplo 5, Raiz quadrada (Algoritmo da Babilónia)

- Em cada iteração, partindo do valor aproximado, p_i, para a raiz quadrada de x, podemos calcular uma aproximação ao melhor, p_{i+1}, através da seguinte fórmula:

$$p_{i+1} = \frac{p_i + \frac{x}{p_i}}{2}.$$

- Exemplo algoritmo para $\sqrt{2}$

Número da tentativa	Aproximação para $\sqrt{2}$	Nova aproximação
0	1	$\frac{1+\frac{2}{1}}{2} = 1.5$
1	1.5	$\frac{1.5+\frac{2}{1.5}}{2} = 1.4167$
2	1.4167	$\frac{1.4167+\frac{2}{1.4167}}{2} = 1.4142$
3	1.4142	...

Funções

Exemplo 5, Raiz quadrada (Algoritmo da Babilónia)

```
def calcula_raiz(x, palpite):
    while not bom_palpite(x, palpite):
        palpite = novo_palpite(x, palpite)
    return palpite

def raiz(x):
    if x < 0:
        raise ValueError("raiz definida só para números positivos")
    return calcula_raiz(x, 1)
```

- **Exercício**: Definir as funções *bom_palpite* e *novo_palpite*

In [110]:
```
def calcula_raiz(x, palpite):
    while not bom_palpite(x, palpite):
        palpite = novo_palpite(x, palpite)
    return palpite

def raiz(x):
    if x < 0:
        raise ValueError("raiz definida só para números positivos")
    return calcula_raiz(x, 1)

def bom_palpite(x, palpite):
    pass

def novo_palpite(x, palpite):
    pass

raiz(9)
import math


x = 7
print("Aprox",raiz(x))
print("Exacto",math.sqrt(x))
```

```
Aprox 2.6457513111113693
Exacto 2.6457513110645907
```

Funções

Exemplo 6, Séries de Taylor

- Definição:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

- Exemplos dalgumas aproximações:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}x^{2n+1} \quad = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}x^{2n} \quad = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

Exemplo 6, Séries de Taylor

```
def proximo_termo(x, n):
    pass #completar (diferente dependendo da função a aproximar)

def funcao_aproximada(x, delta):
    n = 0
    termo = proximo_termo(x, n)
    resultado = termo
    while termo > delta:
        n = n + 1
        termo = proximo_termo(x, n)
        resultado = resultado + termo
    return resultado
```

- **Exercício:** Definir a série de Taylor para as funçoes *e(x)*, *sin(x)* e *cos(x)*
- **Exercício:** Alterar para que o cômputo de termo seja função do anterior termo, `termo = proximo_termo(x, n, termo)`

Exemplo 6, Séries de Taylor: Exponencial

In [54]:
```
def proximo_termo(x, n):
    pass


def exp_aproximada(x, delta):
    n = 0
    termo = proximo_termo(x, n)
    resultado = termo

    while termo > delta:
        n = n + 1
        termo = proximo_termo(x, n)
        resultado = resultado + termo

    return resultado

import math

print("Aprox",exp_aproximada(4,0.0001))
print("Exacto",math.exp(4))
```

```
Aprox 54.598136483106295
Exacto 54.598150033144236
```

Funções

Exemplo 6, Séries de Taylor: Seno

In [130]:
```
def proximo_termo(x, n):
    pass

def sin_aproximada(x, delta):

    n = 0
    termo = proximo_termo(x, n)
    resultado = termo

    while termo > delta:
        n = n + 1
        termo = proximo_termo(x, n)
        resultado = resultado + termo

    return resultado

import math

print("Aprox",sin_aproximada(math.pi/6,0.001))
print("Exacto",math.sin(math.pi/6))
```

```
Aprox 0.49967417939436376
Exacto 0.49999999999999994
```

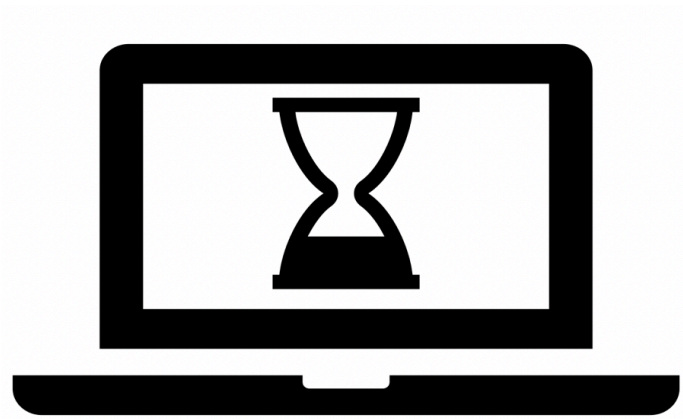
Exemplo 6, Séries de Taylor: Cosseno

In [56]:
```
def proximo_termo(x, n):
    pass

def cos_aproximada(x, delta):
    n = 0
    termo = proximo_termo(x, n)
    resultado = termo

    while termo > delta:
        n = n + 1
        termo = proximo_termo(x, n)
        resultado = resultado + termo

    return resultado

import math

print("Aprox",cos_aproximada(math.pi/6,0.0001))
print("Exacto",math.cos(math.pi/6))
```

```
Aprox 0.8629221610959812
Exacto 0.8660254037844387
```

Funções - Tarefas para as próxima aulas

- Trabalhar matéria apresentada até hoje --> Fazer todos os programas!
- Ler capítulo 4 do livro da UC: Tuplos, ciclos contados e cadeias de carateres
- Na próxima **aula laboratorial (L03)**: funções, verificação de argumentos, exepções



In [ ]: