Fundamentos da Programação

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

Aula 6

José Monteiro

(slides adaptados do Prof. Alberto Abad)

Erros / Excepções

- Nas aulas anteriores falámos 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 domínio.
  - As excepções interrompem o fluxo de execução
- Para isso podemos utilizar a instrução *raise* que gera um erro de execução, em BNF:

```
<instrução raise> ::= raise <nome>(<mensagem>)
```

```
<mensagem> ::= <cadeia de carateres>
```

- *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.

In [ ]:

```

```

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.

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

Erros / Excepções

Exemplo:

In [ ]:

```
def inverte(n):
    if not (type(n) == int) and not (type(n) == float):
        # raise ValueError("erro: não é número")
        print("erro: não é número")
    elif n == 0:
        raise ValueError("erro: igual a 0")
    print("Início corpo")
    return 1/n

inverte("a")
```

Módulos: Importar

- Não é preciso reiventar a roda, o 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 a Funções de um Módulo

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

```
<nome composto> ::= <nome simples>.<nome simples>
```

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
```

In [ ]:

Módulos: Construir

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

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

Parâmetros de Funções

- Python permite maior flexibilidade na definição e passagem dos parâmetros duma função:
  - **Default parameters**
  - **Keyword arguments**
  - Número variável de parâmetros

In [11]:

```python
def dividir(num, den = 10, i = 5):
    return num / den + i
print("Ex1:", dividir(10, 2))
print("Ex2:", dividir(10))
print("Ex3:", dividir(den=2, num=10))
```

```
Ex1: 10.0
Ex2: 6.0
Ex3: 10.0
```

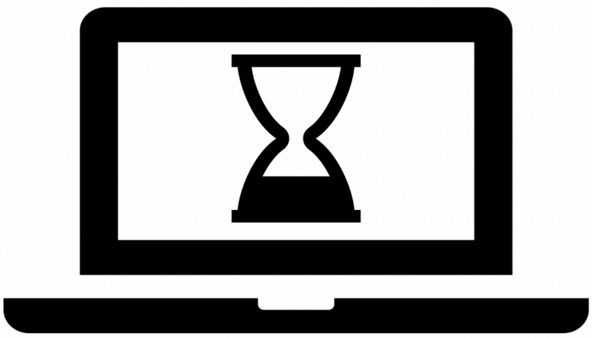
Visualização e execução de programas

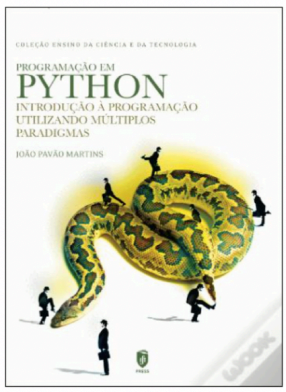
- http://pythontutor.com/visualize.html#mode=edit

- IDEs como o PyCharm e WingIDE

Tarefas para as Próximas Aulas

- Trabalhar matéria apresentada esta semana --> Fazer todos os programas!
- Ler o Capítulo 4 do livro da UC
- Nas aulas de problemas: funções, verificação de argumentos, exepções





Para treinar mais!!!!

Máximo Divisor Comum (Algoritmo de Euclides)

Exemplo 4:

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

Raiz Quadrada (Algoritmo da Babilónia)

Exemplo 5:

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

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

- Exemplo 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	...

Séries de Taylor

Exemplo 6:

- Definição:

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

- Exemplos dalgumas aproximações:

$$\begin{aligned} e^x &= \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \\ \sin(x) &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \\ \cos(x) &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \end{aligned}$$

Funções

Exemplo 4, Máximo divisor comum (Algoritmo de Euclides)

In [21]:

```
# Máximo divisor comum (mdc)
# Euclidian algorithm

def mdc(m, n):
    if type(n) != int or type(m) != int or n < 1 or m < 1:
        raise ValueError('euclides: argumentos negativos!')

    while n > 0:
        m, n = n, m % n

    return m

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

```
Dá-me o valor x:1
Dá-me o valor y:1
1
```

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 [53]:
```
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):
    delta = 0.0001
    return abs(palpite*palpite - x) < delta

def novo_palpite(x, palpite):
    return (palpite + x/palpite)/2

raiz(9)
import math


print("Aprox",raiz(4))
print("Exacto",math.sqrt(4))
```

```
Aprox 2.0000000929222947
Exacto 2.0
```


Exemplo 6, Séries de Taylor: Exponencial

def exp_aproximada(x, delta):

```
def proximo_termo(x, n):
        return x**n/factorial(n)

    def factorial(n):
        prod = 1
        while n > 0:
            prod = prod*n
            n = n -1
        return prod

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

    while abs(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))
```

Funções

Exemplo 6, Séries de Taylor: Seno

In [22]:

```python
def sin_aproximada(x, delta):
    def factorial(n):
        prod = 1
        while n > 0:
            prod = prod*n
            n = n -1
        return prod

    def proximo_termo(x, n):
        return (-1)**n*x**(2*n+1)/factorial(2*n+1)

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

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

    return resultado

import math

print("Aprox",sin_aproximada(math.pi/6,0.0001))
print("Exacto",math.sin(math.pi/6))
```

```
Aprox 0.49999999918690232
Exacto 0.49999999999999994
```


Exemplo 6, Séries de Taylor: Cosseno

In [23]:

```python
def cos_aproximada(x, delta):
    def factorial(n):
        prod = 1
        while n > 0:
            prod = prod*n
            n = n -1
        return prod

    def proximo_termo(x, n):
        return (-1)**n*x**(2*n)/factorial(2*n)

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

    while abs(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 0.8660252641005711
Exacto 0.8660254037844387
```