

# **Python: Comandos Básicos**

Douglas Duarte

# Primeiros passos em programação

- Até agora só vimos como computar algumas expressões simples
  - Expressões são escritas e computadas imediatamente
  - Variáveis podem ser usadas para valores temporários
- Um programa típico entretanto usa vários tipos de construções tais como:
  - Comandos condicionais
  - Comandos de repetição
  - Definição e uso de procedimentos (subprogramas)
  - Definição e uso de classes e objetos (programação OO)

# Primeiros passos em programação

```
>>> # Série de Fibonacci
```

Comentário

```
... a,b=0,1
```

Atribuição dupla

```
>>> while b < 10:
```

Comando de repetição

```
...     print b
```

```
...     a,b=b,a+b
```

Expressão booleana

```
1
```

```
1
```

```
2
```

```
3
```

```
5
```

```
8
```

Indentação

Bloco do  
comando de repetição

Resultado

# Programas armazenados

- À medida que os programas vão se tornando mais complicados, é mais interessante guardá-los em arquivos e executá-los quando necessário
- Arquivo fibo.py (use um editor de textos como o do IDLE):  
# Série de Fibonacci:  
a, b = 0, 1  
while b < 10:  
 print (b)  
 a, b = b, a+b

# Formas de Executar um Programa

- Digite *python fibo.py* no seu shell, ou
- Clique no ícone do arquivo, ou
- De dentro do editor IDLE, selecione *Run Module (F5)*, ou
- De dentro do interpretador *python*:

```
>>> execfile("fibo.py")
```

```
Entre com um numero 5
```

```
1 1 2 3
```

```
>>>
```

# print

- Forma geral: **print** *expr, expr, ...*
- Os valores das expressões são escritos um após o outro sem pular de linha:

```
>>> print ("1.001 ao quadrado é ", 1.001**2)
1.001 ao quadrado é  1.002001
```

- Se o comando terminar com vírgula, o próximo print escreverá na mesma linha. Por exemplo:

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print (b, end=" ")
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

# input

- O programa que computa elementos da série de Fibonacci termina quando atinge um elemento com valor superior a uma constante
- Podemos tornar o programa mais flexível se ao usuário for permitido estipular o valor máximo
- O comando input permite perguntar ao usuário um valor (normalmente é atribuído a uma variável)
  - Formato: **input**(*pergunta*)
  - onde *pergunta* é uma string opcional que será exibida para indicar o valor que se espera (i.e., *prompt*)

- Exemplo:

```
>>> a = input("Entre com um numero: ")
Entre com um numero: 19
>>> print (a)
19
```



Usuário digita o número

# Input

- O comando `input` espera que se digite algo, e retorna uma string, sem nenhum tipo de interpretação (`raw_input` do python 2).
  - O resultado é simplesmente uma string com o texto digitado

- Ex.:

```
>>> nome = input ("Entre seu nome: ")
Entre seu nome: Douglas Duarte
>>> print (nome)
```

Douglas Duarte

```
>>> nome
'Douglas Duarte'
```



# while

- Repete uma seqüência de comandos enquanto uma dada expressão booleana é avaliada como verdadeira

- Formato:

```
while expressão:  
    comando  
    ...  
    comando
```

- Exemplo:

```
>>> a = 10  
>>> while a > 8:  
...     print (a, end=" ")  
...     a = a-1  
...  
10 9
```

# Laços Infinitos

- Como em todo comando de repetição, é importante evitar os chamados “laços infinitos”

- Ex.:

```
>>> a = 10
>>> while a > 8:
...     print (a, end=" ")
...     a = a+1
...
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 ...
```

# if

- É o comando condicional por excelência

- Formatos:

- **if** *expressao:*  
*comandos*

Executa comandos apenas se expressão for verdadeira

- **if** *expressao:*  
*comandos1*  
**else:**  
*comandos2*

Executa seq de comandos 1 caso expressão seja verdadeira.

Caso contrário, executa seq de comandos 2

- **if** *expressao1:*  
*comandos1*  
**elif** *expressao2:*  
*comandos2*  
**else:**  
*comandos(N)*

Executa seq de comandos 1 caso expressão1 seja verdadeira.

Caso contrário, testa expressao2 e executa seq de comandos 2 se verdadeira

Vários desses... }

Caso contrário, executa seq de comandos N

# if

## ■ Exemplo 1

```
■ a = int(input("Entre com um numero: "))  
  if a < 0:  
      print (a, " é negativo")  
  print "Obrigado!"
```

### ■ Execução 1:

```
Entre com um numero: 2  
Obrigado!
```

### ■ Execução 2:

```
Entre com um numero: -2  
-2 é negativo  
Obrigado!
```

# if

## ■ Exemplo 2

```
■ a = int(input("Entre com um numero: "))  
  if a < 0:  
      print (a, " é negativo")  
  else:  
      print (a, " é zero ou positivo")  
  print ("Obrigado! ")
```

### ■ Execução 1:

```
Entre com um numero: 2  
2 é zero ou positivo  
Obrigado!
```

### ■ Execução 2:

```
Entre com um numero: -2  
-2 é negativo  
Obrigado!
```

# if

## ■ Exemplo 3

```
■ a = int(input("Entre com um numero: "))  
  if a < 0:  
      print (a, " é negativo")  
  elif a==0:  
      print (a, " é zero")  
  else:  
      print (a, " é positivo")  
  print ("Obrigado! ")
```

### ■ Execução 1:

```
Entre com um numero: 0  
0 é zero  
Obrigado!
```

### ■ Execução 2:

```
Entre com um numero: 2  
2 é positivo  
Obrigado!
```

# Exercício: algarismos romanos

- Fazer um programa que escreva a representação em algarismos romanos de um número inteiro positivo
  - O usuário deve entrar com um número (*input*)
  - O resultado deve ser impresso no console (*print*)
- Exemplo de execução:  
Entre com um numero positivo: 1985  
Em algarismos romanos: MCMLXXXV

# Exercício: algarismos romanos

## ■ Algoritmo

- A representação em romanos é uma string à qual é acrescentada uma letra por vez
  - Inicialmente, uma string vazia
- Examinar as sucessivas potências de 10
  - Por exemplo, a letra 'M' corresponde à casa dos milhares
  - Se o número é 2200, sabemos que teremos dois M's na representação em romanos
  - Sabemos que há M's se o número é maior ou igual a 1000
    - Sempre que um milhar for computado, subtrair 1000 do número
- Um processamento semelhante é feito para outros algarismos romanos, por exemplo:
  - Se o número é maior ou igual que 500, acrescentar 'D'
  - Se o número é maior que 900, acrescentar 'CM'



# Exercício: algarismos romanos

- DICA: processando um número entre 1 e 9

```
num = int(input("Digite um inteiro entre 1 e 9: "))
```

```
romano = " "
```

```
if num >= 9:
```

```
    romano = romano + "IX"
```

```
    num = num - 9
```

```
if num >= 5:
```

```
    romano = romano + "V"
```

```
    num = num - 5
```

```
if num >= 4:
```

```
    romano = romano + "IV"
```

```
    num = num - 4
```

```
while num >= 1:
```

```
    romano = romano + "I"
```

```
    num = num - 1
```

# Exercício: números primos

- Fazer um programa que decida se um número positivo dado é primo ou não
  - Entrada: número inteiro positivo
  - Saída: diagnóstico de primalidade do número
- Exemplos de execução:
  - Entre com um número inteiro positivo: 169  
169 é múltiplo de 13
  - Entre com um número inteiro positivo: 983  
983 é primo

# Exercício: números primos

- Um número natural é primo se é divisível apenas por si mesmo ou pela unidade
- Isto sugere o seguinte algoritmo:
  - Se o número é 1, então **não** é primo
  - Se o número é 2, então é primo
  - Caso contrário,
    - Seja  $d$  um possível divisor, cujo valor é inicialmente 2
    - Repetir
      - Se o resto da divisão do número por  $d$  é zero, então o número não é primo
      - Caso contrário, incrementar  $d$
      - Se  $d$  é igual ou maior que o número, então terminar repetição diagnosticando o número como primo