

COMPUTAÇÃO 1 – PYTHON

AULA 3 TEÓRICA

SILVIA BENZA

SILVIA@BENZA.COM.PY

UPDATES

Envio de e-mails

- Assunto tem que ter [Turma XXX]
- O nome do arquivo tem que ter o formato
NomeDoAluno_AulaX.py
- Trabalhos: silviabenza@gmail.com
- Duvidas: silvia.benza@gmail.com

UPDATES

Provas:

P1 18 de Setembro

P2 6 de Novembro

P3 13 de Novembro

Calculo das Notas:

$$NF = 30\%P1 + 50\%\max(P2, P3) + 20\%T$$

$$NF = 40\%P2 + 40\%P3 + 20\%T \text{ (se não fizer a P1)}$$

Aprovação é NF >= 5.0

Vocês tem que fazer pelo menos 2 provas, lembrando que a P2 e P3 são a matéria completa!

TIPOS DE DADOS

LEMBRANDO

Dados numéricos

- Números Inteiros: Int/Long
- Ponto Flutuante: Float
- Números Complexos: Complex

Operações com dados de um mesmo tipo tendem a gerar resultados do mesmo tipo dos operandos;

Operações com dados de diferentes tipos geram resultados do tipo mais complexo;

Você não precisa se preocupar quando faz operações com tipos diferentes



Se você já transformou o tipo do seu parâmetro*

* float(<param>), int(<param>), str(<param>)

STRINGS

SEQUÊNCIA DE CARACTERES: STR

- São cadeias de caracteres
- Constituem outro tipo fundamental do python
- Constantes string são escritas usando aspas simples ou duplas
 - Exemplo: "a" ou 'a'
- O operador “+” pode ser usado para concatenar strings
 - Exemplo: "a"+"b" é o mesmo que "ab"
- O operador “*” pode ser usado para repetir strings
 - Exemplo: "a"*10 é o mesmo que "aaaaaaaaaa"

STRINGS

Escreva uma função que receba como parâmetro o nome e a idade de uma pessoa, e que retorne a frase:

“Olá *fulano*, meu nome é Python e eu tenho *x* anos.”

Onde *fulano* e *x* são, respectivamente, o nome e o dobro da idade do usuário.

STRINGS

Escreva uma função que receba como parâmetro o nome e o dobro da idade de uma pessoa, e que retorne a frase:

“Olá *fulano*, meu nome é Python e eu tenho *x* anos.”

Onde *fulano* e *x* são, respectivamente, o nome e o dobro da idade do usuário.

```
def olafulano(nome,idade):
    return "Olá " + nome + ", meu nome é Python, e tenho " +
           str(2*int(idade)) + " anos."
```

STRINGS

- Python usa a tabela de caracteres default do S.O.
 - Ex.: ASCII, UTF-8
- Caracteres não imprimíveis podem ser expressos usando notação “barra-invertida” (\)
 - \n é o mesmo que *new line*
 - \r é o mesmo que *carriage return*
 - \t é o mesmo que *tab*
 - \b é o mesmo que *backspace*
 - \\ é o mesmo que \
 - \x41 é o mesmo que o caractere cujo código hexadecimal é 41 (“A” maiúsculo)

STRINGS

```
>>> "ab\rd"  
'ab\rd'  
>>> print "ab\rd" # print exibe chars não imprimíveis  
db  
>>> print "abc\td"  
abc      d  
>>> print "abc\nd"  
abc  
d  
>>> print "abc\\nd"  
abc\nd  
>>> print "ab\bc"  
ac  
>>> print "\x41\xA1"  
Áí
```

STRINGS

- Constantes string podem ser escritas com várias linhas desde que as aspas não sejam fechadas e que cada linha termine com uma barra invertida
- Ex.:

```
>>> print "abcd\n\"  
... efg\n\"  
... ijk"  
abcd  
efgh  
ijk  
>>> print "abcd\"  
... efg\n\"  
... ijk"  
abcdefghijklm
```

STRINGS

- Também é possível escrever constantes string em várias linhas incluindo as quebras de linha usando três aspas como delimitadores

```
>>> print """  
Um tigre  
dois tigres  
três tigres""""  
  
Um tigre  
dois tigres  
três tigres  
>>> print "abcd  
efgh"  
abcd  
efgh
```



BOOLEANO

BOOLEANO: BOOL

- Também chamadas expressões lógicas
- Resultam em verdadeiro (True) ou falso (False)
- São usadas em comandos condicionais e de repetição
- Servem para analisar o estado de uma computação e permitir escolher o próximo passo
- Operadores mais usados
 - Relacionais: > , < , == (igual), != (diferente), >=, <=
 - Booleanos: and, or, not
- Avaliação feita em “Curto-circuito”
 - Expressão avaliada da esquerda para a direita
 - Se o resultado (verdadeiro ou falso) puder ser determinado sem avaliar o restante, este é retornado imediatamente

BOOLEANO: BOOL

- Também chamados de tipos lógicos
- Resultam em valores booleanos (verdadeiro ou falso)
- São usadas em loops para controlar a repetição
- Servem para armazenar e alterar valores (atribuição) escolher o próprio valor para ATRIBUIR A X O VALOR DE Y
- Operadores mais usados:
 - Relacionais: `>`, `<`, `== (igual)`, `!= (diferente)`, `>=`, `<=`
 - Booleanos: and, or, not
- Avaliação feita em “Curto-circuito”
 - Expressão avaliada da esquerda para a direita
 - Se o resultado (verdadeiro ou falso) puder ser determinado sem avaliar o restante, este é retornado imediatamente

ATENÇÃO

`X == Y` : operador relacional

`X É IGUAL A Y`

repetição

atribuição e permitir

A

`X = Y` : operador de atribuição

ATRIBUIR A X O VALOR DE Y

EXPRESSÕES BOOLEANAS

```
>>> 1==1
```

True

```
>>> 1==2
```

False

```
>>> 1==1 or 1==2
```

True

```
>>> 1==1 and 1==2
```

False

```
>>> 1<2 and 2<3
```

True

```
>>> not 1<2
```

False

```
>>> not 1<2 or 2<3
```

True

```
>>> not (1<2 or 2<3)
```

False

```
>>> "alo" and 1
```

1 (2.7)

```
>>> "alo" or 1
```

'alo' (2.7)

OPERADORES LÓGICOS

- Operadores:
 - **not** (negação),
 - **and** (e),
 - **or** (ou)
- **x and y: verdadeiro se, e somente se x e y forem ambos verdadeiros.**
- **x or y: falso se, e somente se x e y forem ambos falsos.**
- **not x: falso se x for verdadeiro, e verdadeiro se x for falso.**
- **Observe que x e y podem ser variáveis booleanas ou podem ser expressões booleanas compostas de operadores relacionais e operadores lógicos.**

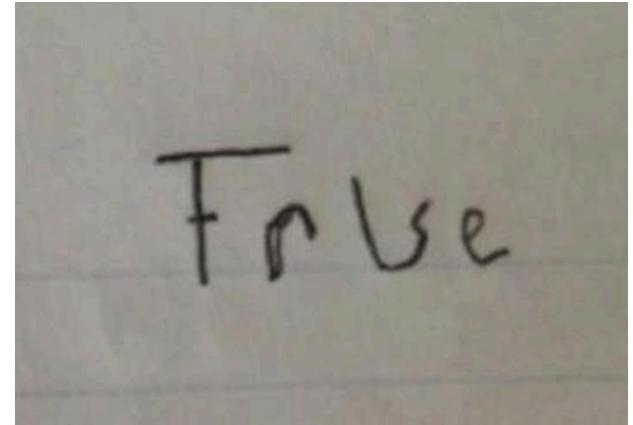
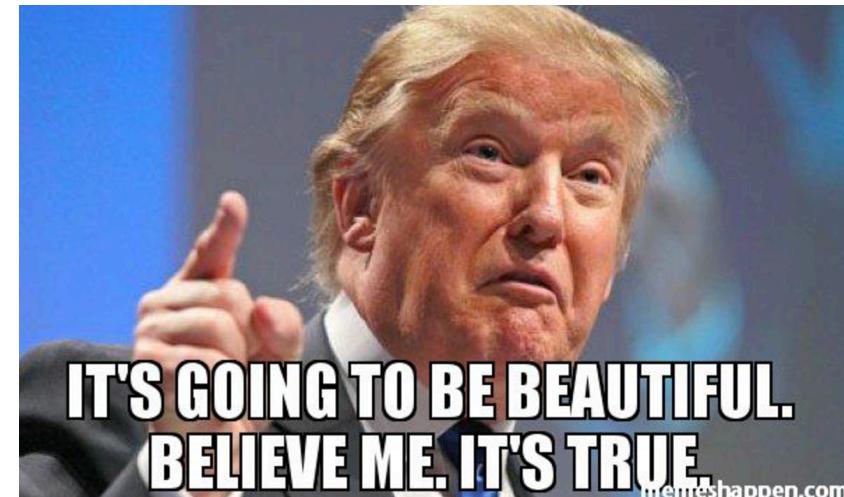


TABELA VERDADE

Exp 1	Exp 2	Exp 1 and Exp 2	Exp 1 or Exp 2	not Exp 1
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V



EXPRESSÕES BOOLEANAS

- As constantes **True** e **False** são apenas símbolos convenientes
- Qualquer valor não nulo é visto como verdadeiro enquanto que **0** (ou **False**) é visto como falso
- O operador **or** retorna o primeiro operando se for vista como *verdadeiro*, caso contrário retorna o segundo
- O operador **and** retorna o primeiro operando se for vista como *falso*, caso contrário retorna o segundo
- Operadores relacionais são avaliados antes de **not**, que é avaliado antes de **and**, que é avaliado antes de **or**

EXPRESSÕES BOOLEANAS

```
>>> 0 or 100  
100  
>>> False or 100  
100  
>>> "abc" or 1  
'abc'  
>>> 1 and 2  
2  
>>> 0 and 3  
0
```

```
>>> False and 3  
False  
>>> 1 and 2 or 3  
2  
>>> 0 or 2 and 3  
3  

```

EXPRESSÕES BOOLEANAS

Exercício - Construa a tabela-verdade para as seguintes expressões:

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != “Maria”) and 1==2

4- (Nome == “Jorge”) and False or (Z <= X +10)

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p	q	Exp 1 (p and q)	Exp 2 (p or q)	Exp 3 not(Exp 2)	Exp 4 Exp 1 and Exp 3

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != “Maria”) and 1==2

4- (Nome == “Jorge”) and False or (Z <= X +10)

p	q	Exp 1 (p and q)	Exp 2 (p or q)	Exp 3 not(Exp 2)	Exp 4 Exp 1 and Exp 3
V	V	V	V	F	F
V	F	F	V	F	F
F	V	F	V	F	F
F	F	F	F	V	F

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p	q	Exp 1 (not q)	Exp 2 (p and Exp 1)	Exp 3 not(Exp 2)	Exp 4 Exp 3 or q

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != “Maria”) and 1==2

4- (Nome == “Jorge”) and False or (Z <= X +10)

p	q	Exp 1 (not q)	Exp 2 (p and Exp 1)	Exp 3 not(Exp 2)	Exp 4 Exp 3 or q
V	V	F	F	V	V
V	F	V	V	F	F
F	V	F	F	V	V
F	F	V	F	V	V

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p (X + Y > Z)	q (Nome != "Maria")	r 1==2	Exp 1 (q and r)	Exp 2 P or Exp 1

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p (X + Y > Z)	q (Nome != "Maria")	r 1==2	Exp 1 (q and r)	Exp 2 P or Exp 1
V	V	F	F	V
V	F	F	F	V
F	V	F	F	F
F	F	F	F	F

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p (Nome == "Jorge")	q False	r (Z <= X +10)	Exp 1 (p and q)	Exp 2 Exp1 or r

EXERCÍCIOS

1- (p and q) and not(p or q)

2- not(p and not q) or q

3- (X + Y > Z) or (Nome != "Maria") and 1==2

4- (Nome == "Jorge") and False or (Z <= X +10)

p (Nome == "Jorge")	q False	r (Z <= X +10)	Exp 1 (p and q)	Exp 2 Exp1 or r
V	F	V	F	V
V	F	F	F	F
F	F	V	F	V
F	F	F	F	F

ESTRUTURA CONDICIONAL

ESTRUTURA CONDICIONAL

Faça uma função que, dado um número inteiro X passado como parâmetro, retorna a string “*positivo*” caso X seja um número positivo, e “*não positivo*” caso contrário.

ESTRUTURA CONDICIONAL

Faça uma função que, dado um número inteiro X passado como parâmetro, retorna a string “*positivo*” caso X seja um número positivo, e “*não positivo*” caso.

```
def positivo(X):
    if X > 0 :
        return 'positivo'
    return 'não positivo'
```

ESTRUTURA CONDICIONAL SIMPLES

```
if expressão :  
    comandos
```

expressão na estrutura condicional é um tipo especial de expressão chamado **expressão booleana**, que pode ser verdadeira ou falsa.

A **expressão** neste problema é: $X > 0$

ESTRUTURA CONDICIONAL SIMPLES

```
1 # Exemplo - Estrutura Condicional Simples
2
3 # Faça uma função que,
4 # dado um número inteiro X
5 # passado como parâmetro,
6 # retorna a string 'positivo'
7 # caso X seja um número positivo,
8 # e 'não positivo' caso.
9
10 def positivo(X):
11     if X > 0 :
12         return 'positivo'
13     return 'não positivo'
14
15 # Caso de Teste 1
16 positivo(3)
17
18 # Caso de Teste 2
19 positivo(-5)
```

[Edit code](#)



<< First

< Back

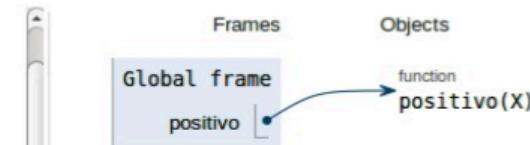
Step 2 of 11

Forward >

Last >>

→ line that has just executed

→ next line to execute



ESTRUTURA CONDICIONAL SIMPLES

```
1 # EXEMPLO - ESTRUTURA CONDICIONAL SIMPLES
2
3 # Faça uma função que,
4 # dado um número inteiro X
5 # passado como parâmetro,
6 # retorna a string 'positivo'
7 # caso X seja um número positivo,
8 # e 'não positivo' caso.
9
10 def positivo(X):
11     if X > 0 :
12         return 'positivo'
13     return 'não positivo'
14
15 # Caso de Teste 1
16 positivo(3)
17
18 # Caso de Teste 2
19 positivo(-5)
```

[Edit code](#)

<< First

< Back

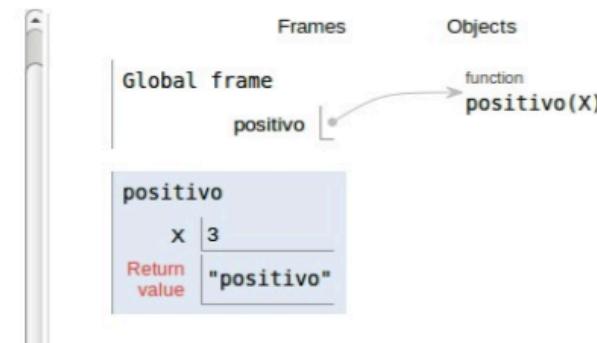
Step 6 of 11

Forward >

Last >>

→ line that has just executed

→ next line to execute



ESTRUTURA CONDICIONAL SIMPLES

```
1 # Exemplo - Estrutura Condicional Simples
2
3 # Faça uma função que,
4 # dado um número inteiro X
5 # passado como parâmetro,
6 # retorna a string 'positivo'
7 # caso X seja um número positivo,
8 # e 'não positivo' caso.
9
10 def positivo(X):
11     if X > 0 :
12         return 'positivo'
13     return 'não positivo'
14
15 # Caso de Teste 1
16 positivo(3)
17
18 # Caso de Teste 2
19 positivo(-5)
```

[Edit code](#)

[**<< First**](#)

[**< Back**](#)

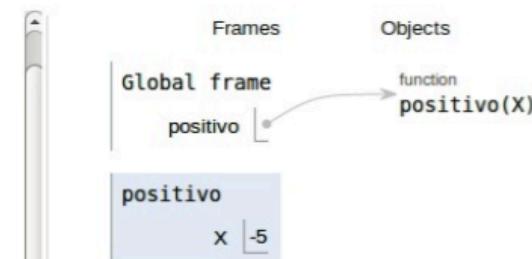
Step 9 of 11

[**Forward >**](#)

[**Last >>**](#)

➡ line that has just executed

→ next line to execute



ESTRUTURA CONDICIONAL SIMPLES

```
1 # EXEMPLO - ESTRUTURA CONDICIONAL SIMPLES
2
3 # Faça uma função que,
4 # dado um número inteiro X
5 # passado como parâmetro,
6 # retorna a string 'positivo'
7 # caso X seja um número positivo,
8 # e 'não positivo' caso.
9
10 def positivo(X):
→ 11     if X > 0 :
12         return 'positivo'
→ 13     return 'não positivo'
14
15 # Caso de Teste 1
16 positivo(3)
17
18 # Caso de Teste 2
19 positivo(-5)
```

[Edit code](#)

<< First

< Back

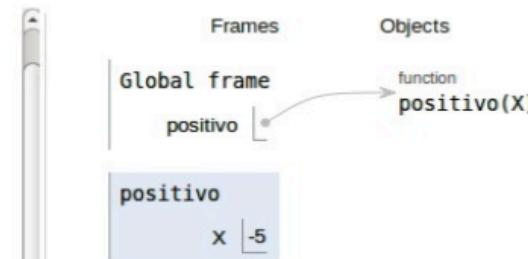
Step 10 of 11

Forward >

Last >>

→ line that has just executed

→ next line to execute



ESTRUTURA CONDICIONAL SIMPLES

```
1 # Exemplo - Estrutura Condicional Simples
2
3 # Faça uma função que,
4 # dado um número inteiro X
5 # passado como parâmetro,
6 # retorna a string 'positivo'
7 # caso X seja um número positivo,
8 # e 'não positivo' caso.
9
10 def positivo(X):
11     if X > 0 :
12         return 'positivo'
13     return 'não positivo'
14
15 # Caso de Teste 1
16 positivo(3)
17
18 # Caso de Teste 2
19 positivo(-5)
```

[Edit code](#)

<< First

< Back

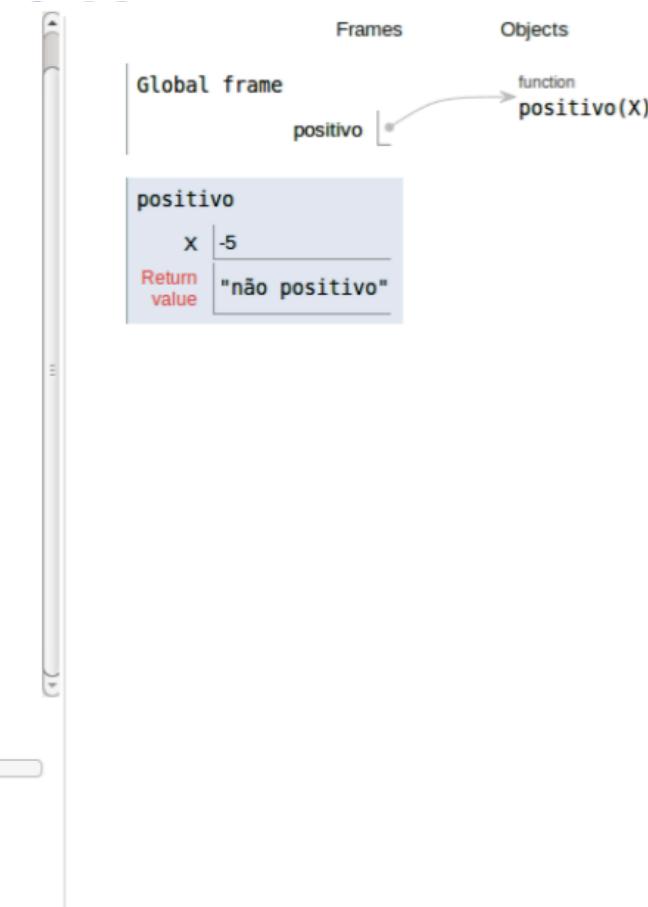
Step 11 of 11

Forward >

Last >>

→ Line that has just executed

→ next line to execute



ESTRUTURA CONDICIONAL

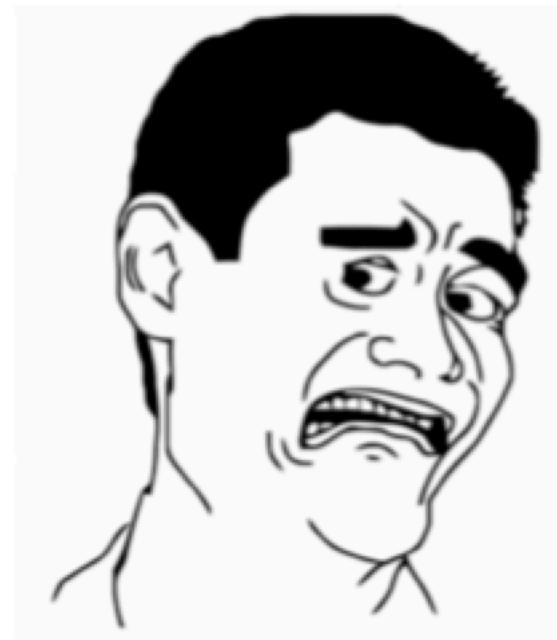
Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

ESTRUTURA CONDICIONAL SIMPLES

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

if expressão :
comandos

```
def PosNegZero(X):
    if X > 0 :
        return "positivo"
    if X < 0 :
        return "negativo"
    if X == 0 :
        return "zero"
```



ESTRUTURA CONDICIONAL SIMPLES

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

```
if expressão :  
    comandos
```

```
def PosNegZero(X):  
    if X > 0 :  
        return "positivo"  
    if X < 0 :  
        return "negativo"  
    if X == 0 :  
        return "zero"
```

Podemos simplificar o código ?
Como ?

ESTRUTURA CONDICIONAL COMPOSTA

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

```
If expressão :  
    Comandos 1  
else:  
    Comandos 2
```

Comandos 1 são executados sempre que a *expressão* for VERDADEIRA

Comandos 2 são executados sempre que a *expressão* for FALSA

ESTRUTURA CONDICIONAL COMPOSTA

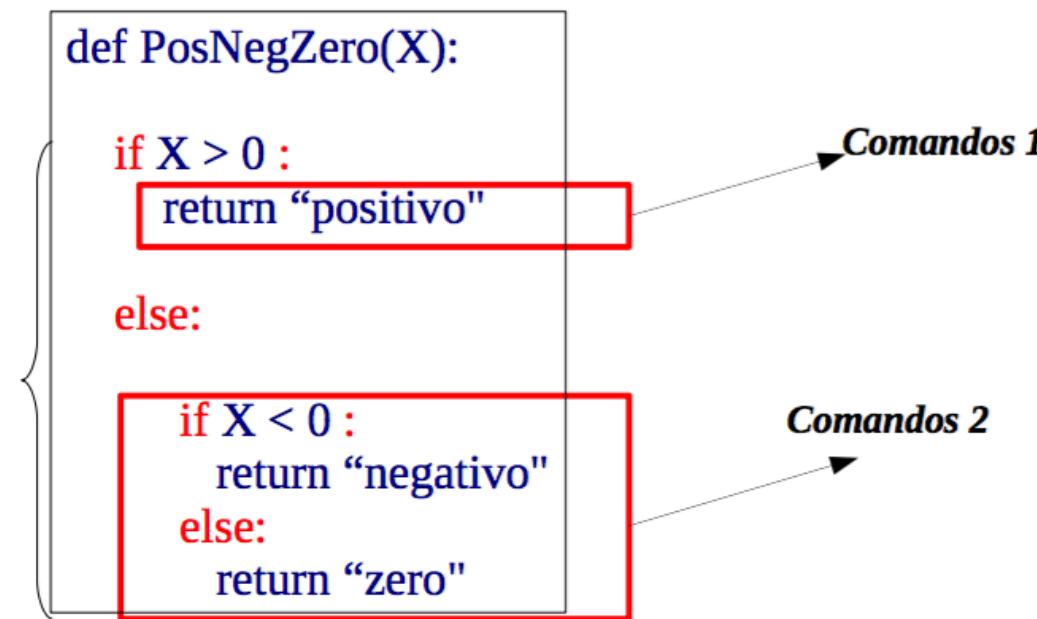
Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

```
If expressão :  
    Comandos 1  
else:  
    Comandos 2
```

```
def PosNegZero(X):  
    if X > 0 :  
        return "positivo"  
    else:  
        if X < 0 :  
            return "negativo"  
        else:  
            return "zero"
```

ESTRUTURA CONDICIONAL COMPOSTA

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.



ESTRUTURA CONDICIONAL COMPOSTA

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

```
def PosNegZero(X):  
  
    if X > 0 :  
        return "positivo"  
  
    else:  
  
        {  
            if X < 0 :  
                return "negativo"  
            else:  
                return "zero"  
        }
```

► Comandos 1

► Comandos 2

ESTRUTURA CONDICIONAL COMPOSTA

Faça uma função que determina se um número inteiro X passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “positivo”, “negativo” ou “zero”.

```
def PosNegZero(X):  
  
    if X > 0 :  
        return "positivo"  
  
    elif X < 0 :  
        return "negativo"  
  
    else:  
        return "zero"
```



```
def PosNegZero(X):  
    if X > 0 :  
        return "positivo"  
    if X < 0 :  
        return "negativo"  
    if X == 0 :  
        return "zero"
```



```
def PosNegZero(X):  
  
    if X > 0 :  
        return "positivo"  
  
    elif X < 0 :  
        return "negativo"  
  
    else:  
        return "zero"
```

COMANDO IF

- É o comando condicional por excelência

- Formatos:

```
if expressao:  
    comandos
```

Executa os comandos
apenas se a 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 expressão seja
verdadeira

Caso contrário, testa
expressão2 e executa
seq de comandos 2 se
verdadeira

COMPUTAÇÃO 1 – PYTHON

AULA 3 TEÓRICA

SILVIA BENZA

SILVIA@BENZA.COM.PY