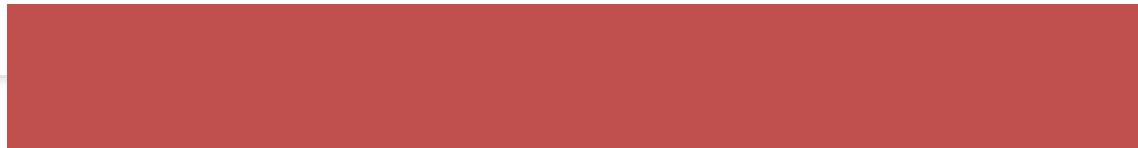


PPT 02 – Fundamentos de Python

Sandra Liliana Meira de Oliveira

Conceitos Básicos - Parte I




A Reter...



- Um programa em Python pode ser um único ficheiro com a extensão `*.py` ou então, uma pasta que contém várias subpastas com diversos ficheiros contendo código Python `*.py` e muitos outros contendo informações relevantes ao programa.

Ainda que seja permitido a utilização do caractere de finalização ponto e vírgula, não é obrigatório finalizar as instruções com o mesmo.



Até podemos colocá-lo se desejarmos, porém, o deve somente ser utilizado quando precisamos colocar mais de uma instrução numa mesma linha.

Em Python,
não se
finalizam as
instruções
com ponto
e vírgula.

COMENTÁRIOS



A utilização de comentários é uma prática comum na programação



O objetivo é poder adicionar descrições em partes específicas do código, seja para documentá-lo, seja para adicionar uma descrição, ou mesmo, para marcar que uma determinada linha, ou um conjunto de linhas, não devem ser executados.



Para adicionarmos comentários, utilizamos uma notação especial, de modo a indicar ao interpretador que não deve interpretar os caracteres contidos na notação que demarca blocos de comentários.

COMENTÁRIOS

Utilizamos o caractere cardinal **#** para demarcarmos que tudo que estiver a frente desse caractere, deve ser ignorado pelo interpretador do Python.

```
1  #resto da divisão inteira entre 5 e 2
2
3  numero = 5
4
5  numero %= 2
6  print(numero)  # Resultado será
7
8  |
```

```
'''resto da divisão
| inteira entre 5 e 2'''
```

```
numero = 5
```

```
numero %= 2
```

```
print(numero)  # Resultado será
```

A utilização de **3 aspas simples**, ou **3 aspas duplas**, permite delimitar um bloco de informação a ignorar e que não tem que estar, obrigatoriamente, na mesma linha

Indentação

No Python, a indentação é usada para indicar o nível de código aninhado. Cada nível de indentação é representado por uma certa quantidade de espaços ou tabulações no início de uma linha. **A convenção padrão é usar quatro espaços para cada nível de indentação ou uma tabulação.**

Indentar é o recuo do texto em relação a sua margem, ou seja, se antes de escrevermos uma instrução, utilizamos 4 espaçamentos da margem esquerda até a instrução propriamente dita, podemos dizer que a indentação utilizada possui 4 espaços.

Em Python, a indentação possui função bastante especial, até porque, os blocos de instrução são delimitados pela profundidade da indentação, isto é,:

O código que estiverem rente à margem esquerda, fará parte do primeiro nível hierárquico;

O código que estiver a 4 espaços da margem esquerda, estará no segundo nível hierárquico

Aquele que estiver a 8 espaços, estará no terceiro nível e assim sucessivamente.

```
valor = 1
if valor > 1:
    if valor > 7:
        print("valor alto ")
else:
    print("valor igual a ", valor)
```

print(nível 1)#primeiro nível hierárquico

if(True):

print(nível 2)#segundo nível hierárquico

Indentação



Todos os blocos de código são delimitados pela profundidade da indentação e por isso, a sua importância, é vital para um programa em Python.



O mau uso, isto é, utilizar 4 espaçamentos, por exemplo, enquanto deveríamos estar a utilizar 8, acarretará na não execução, ou então, no mal funcionamento em geral.

Blocos de Código

Todos os blocos de código são delimitados pela profundidade da indentação e por isso, a sua importância, é vital para um programa em Python.

O mau uso, isto é, utilizar 4 espaçamentos enquanto deveríamos estar utilizando 8, acarretará na não execução, ou então, no mal funcionamento em geral.

Variáveis, Constantes & Tipos de Dados



Como é que um programa de computador consegue receber valores, processar esses valores e retornar um resultado? Utilizado variáveis.



uma variável é “algo” onde armazenamos alguma informação



Pode ser entendidas como uma caixa, onde os dados são armazenados temporariamente ou em definitivo e que são manipuladas durante a execução do programa



uma **variável** é um dado que pode sofrer alterações de valor ao longo do programa



Entende-se por **constante** um dado que permanece inalterável do início ao fim do algoritmo

Variáveis, Constantes & Tipos de Dados



Devemos pensar nas variáveis como sendo um espaço físico. Este espaço é administrado pelo hardware, pela motherboard, processador e sistema operativo.



Cada pedaço do espaço físico possui um número que o identifica (número identificador) e assim, ao declararmos uma variável, reservamos um espaço físico para guardar informações temporariamente, vinculando esse espaço ao nome da variável (**referência**).



As informações que serão armazenadas são temporárias.



Não existem limites pré-definidos da quantidade de variáveis, ou quantidade de informações que é possível utilizar. Essas quantidades sempre serão definidas pela quantidade de memória física que houver no sistema.



A quantidade de variáveis que podemos declarar, também estará, diretamente relacionada com a quantidade de espaço físico disponível, isto é, a quantidade de memória RAM existente

Variáveis, Constantes & Tipos de Dados

- O conceito de variável inclui quatro características: o **nome** usado pelo programa para a variável, o **endereço de memória** para o qual aquele nome aponta, o **valor armazenado naquele endereço de memória** e o **tipo de dado** armazenado.
- O **tipo** é uma forma de classificar as informação (rótulo atribuído aos dados que vão ser armazenados).

A lista dos principais tipos built-ins (tipos de dados básicos) da linguagem Python:

int - para números inteiros
str - para conjunto de caracteres
bool - armazena True ou False
list - para agrupar um conjunto de elementos
tuple - semelhante ao tipo list
float – números decimais
dic - para agrupar elementos que serão recuperados por uma chave

Cada tipo citado, possui um conjunto de funções e métodos que permitem manipularmos as informações, contidas na variável, de maneira bastante eficiente

Nome - dado tipo String
Idade - dado tipo Integer

Variáveis, Constantes & Tipos de Dados

- Python é uma linguagem de tipos dinâmicos
- O conceito de variável é uma associação entre um nome e um valor, mas não é necessário declarar o tipo da variável, portanto, o tipo relacionado à variável pode variar durante a execução do programa-

O tipo de variável é definido na altura da atribuição:

```
#A variável é criada no momento da atribuição do valor  
x = 5  
y = "Olá Mundo!"  
#Imprimir os valores para o ecrã  
print(x)  
print(y)
```

Nota: As variáveis podem mesmo mudar de tipo depois de feita uma primeira atribuição

Variáveis, Constantes & Tipos de Dados

- Especificar o tipo de dados de uma variável

```
x = str(3) #texto  
y = int(3) #valor inteiro  
z = float(3) #valor real
```

- As *strings* podem ser declaradas com aspas ou pelicas

```
x = "John"  
x = 'John'
```

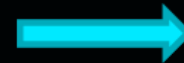
Variáveis, Constantes & Tipos de Dados

- Outras formas de atribuição

```
x, y, z = "Verde", 4, "Amarelo"  
x = y = z = "Preto"
```

- Saída de variáveis

```
x = "fantástico"  
print("O Python é " + x)  
  
valor1 = 5  
valor2 = 10  
print(valor1 + valor2)
```



```
O Python é fantástico  
15
```

Variáveis, Constantes & Tipos de Dados

- Na verdade não é possível criar constantes em Python, mas podemos criar uma variável com sintaxe de constante.

Padrão recomendado da documentação Python:

- Todas as letras da variável que servirá como constante deve ficar em maiúsculas
- Se houver mais de uma palavra elas devem ser separadas por underline

`PI=3.14`

Variáveis, Constantes & Tipos de Dados

Mesmo não sendo declarados explicitamente os valores associados à variável vão assumir sempre um tipo de dados

As operações sobre os valores atribuídos às variáveis depende do tipo de dados.

Quando utilizamos o operador (adição) entre 2 variáveis que contenham números inteiros, o interpretador, somará os valores. Porém, se utilizarmos o sinal de adição com 2 variáveis do tipo *String*, o interpretador fará uma concatenação

A função **type** permite-nos verificar qual o tipo de dados atribuído.

```
>>> a = 1
>>> type(a)
<type 'int'>
```

```
>>> a = 1.0
>>> type(a)
<type 'float'>
```

```
>>> a = True
>>> type(a)
<type 'bool'>
```

```
>>> a = 4+3j
>>> type(a)
<type 'complex'>
```

O tipo de dados pode ser alterado dinamicamente.

```
>>> a = 1
>>> type(a)
<type 'int'>
>>> a = 1.0
>>> type(a)
<type 'float'>
>>>
```

Nomenclatura

O nome de uma variável deve seguir algumas regras:

- Pode-se utilizar, quaisquer letras, sejam elas maiúsculas ou minúsculas.
- Variáveis não devem utilizar nomes de classes ou pacotes, ou seja, não devemos declarar uma variável de nome "str", ou então, "int".
- Não se podem utilizar caracteres especiais (exceto o caracter _).
- Pode conter números, desde que este não seja o primeiro caractere. Assim, referências como **por exemplo, 9num, ou então, 1var**, não são permitidas, mas do tipo **a2b3** sim.
- Para definirmos o nome de variáveis, temos que iniciar utilizando um caractere que esteja no intervalo de a à z ou A à Z. A única exceção a esta regra é o caractere underline_.

```
num_int = 5
```

```
num_dec = 7.3
```

```
val_str = "qualquer texto"
```

```
vvar = 5
```

```
_vVar = ""
```

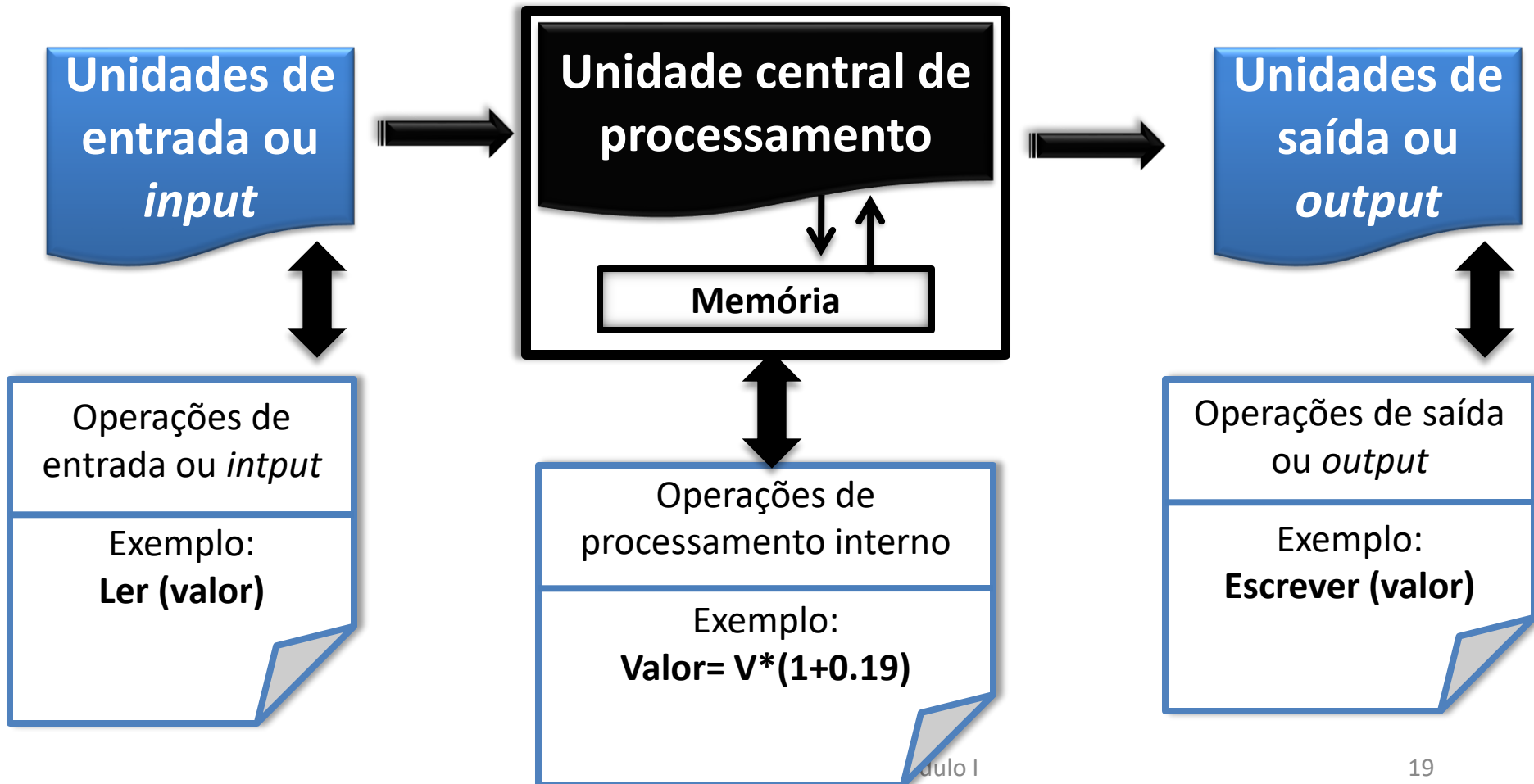
```
_____idade = 19
```

```
_minhaVar = 55
```

```
_____texto = ""
```

Operações elementares

Operações básicas e fundamentais de um sistema informático:



Operações elementares

As operações básicas e fundamentais de um sistema informático são dos seguintes tipos:

- **Operações de input (entrada de dados);**
- **Operações de processamento interno - estas são essencialmente operações de cálculos aritméticos e lógicos;**
- **Operações de output (saída de dados).**

Operações elementares

Operadores aritméticos:

Esses operadores são utilizados para criarmos expressões matemáticas comuns, como soma, subtração, multiplicação e divisão.

Operadores	Descrição	Exemplo
+	Realiza a soma entre operandos	- 10 + 7 - +4
-	Realiza a subtração entre operandos	- 10 - 7 - -4
*	Realiza a multiplicação entre operandos	3 * 4
/	Realiza a divisão entre operandos	10 / 5
//	Realiza a divisão inteira entre operandos	10 // 6
%	Retorna o resto da divisão inteira entre operandos	4 % 2
**	Retorna um número elevado a potência de outro	4 ** 2

Operações elementares

Operadores aritméticos:

Esses operadores são utilizados para criarmos expressões matemáticas comuns, como soma, subtração, multiplicação e divisão.

```
quatro = 4
dois = 2

soma = quatro + dois
print(soma) # Resultado: 6

subtracao = quatro - dois
print(subtracao) # Resultado: 2

multiplicacao = quatro * dois
print(multiplicacao) # Resultado: 8

divisao = quatro / dois
print(divisao) # Resultado: 2.0

divisao_inteira = quatro // dois
print(divisao_inteira) # Resultado: 2

modulo = quatro % dois
print(modulo) # Resultado: 0

exponenciacao = quatro ** dois
print(exponenciacao) # Resultado: 16
```

Operações elementares

Operadores lógicos:

Operadores usados para **comparar** dois valores:

Operador	Nome	Função
==	Igual a	Verifica se um valor é igual ao outro
!=	Diferente de	Verifica se um valor é diferente ao outro
>	Maior que	Verifica se um valor é maior que outro
>=	Maior ou igual	Verifica se um valor é maior ou igual ao outro
<	Menor que	Verifica se um valor é menor que outro
<=	Menor ou igual	Verifica se um valor é menor ou igual ao outro

Operações elementares

Operadores de comparação:

Operadores usados
para **comparar** dois valores:

```
var = 5

if var == 5:
    print('Os valores são iguais')

if var != 7:
    print('O valor não é igual a 7')

if var > 2:
    print('O valor da variável é maior de 2')

if var >= 5:
    print('O valor da variável é maior ou igual a 5')

if var < 7:
    print('O valor da variável é menor que 7')

if var <= 5:
    print('O valor da variável é menor ou igual a 5')
```

Resultado

```
Os valores são iguais
O valor não é igual a 5
O valor da variável é maior de 5
O valor da variável é maior ou igual a 5
O valor da variável é menor que 7
O valor da variável é menor ou igual a 5
```


Operações elementares

Operadores de atribuição:

Esses Operadores são utilizados no momento da **atribuição** de valores às variáveis e controlam como a atribuição será realizada.

Operador	Equivalente a
=	$x = 1$
+=	$x = x + 1$
-=	$x = x - 1$
*=	$x = x * 1$
/=	$x = x / 1$
%=	$x = x \% 1$

Operações elementares

Operadores de atribuição – exemplos de utilização

```
numero = 5

numero += 7
print(numero) # Resultado será 12
```

```
numero = 5

numero /= 4
print(numero) # Resultado será 1.25
```

```
numero = 5

numero -= 3
print(numero) # Resultado será 2
```

```
numero = 5

numero %= 2
print(numero) # Resultado será 1
```

```
numero = 5

numero *= 2
print(numero) # Resultado será 10
```

Operações elementares

Operadores lógicos:

Possibilitam construir **testes lógicos**.

Operador	Definição
and	Retorna True se ambas as afirmações forem verdadeiras
or	Retorna True se uma das afirmações for verdadeira
not	retorna Falso se o resultado for verdadeiro

```
1  num1 = 7
2  num2 = 4
3
4  # Exemplo and
5  if num1 > 3 and num2 < 8:
6      print("As Duas condições são verdadeiras")
7
8  # Exemplo or
9  if num1 > 4 or num2 ≤ 8:
10     print("Uma ou duas das condições são verdadeiras")
11
12 # Exemplo not
13 if not (num1 < 30 and num2 < 8):
14     print('Inverte o resultado da condição entre os parênteses')
```

Operações elementares

Operadores de identidade:

Estes Operadores são utilizados para *comparar* objetos, verificando se os objetos testados referenciam o mesmo objeto (is) ou não (is not).

Operador	Definição
<code>is</code>	Retorna <code>True</code> se ambas as variáveis são o mesmo objeto
<code>is not</code>	Retorna <code>True</code> se ambas as variáveis não forem o mesmo objeto

```
1 lista = [1, 2, 3]
2 outra_lista = [1, 2, 3]
3 recebe_lista = lista
4
5 # Recebe True, pois são o mesmo objeto
6 print(f"São o mesmo objeto? {lista is recebe_lista}")
7
8 # Retorna False, pois são objetos diferentes
9 print(f"São o mesmo objeto? {lista is outra_lista}")
```

```
1 São o mesmo objeto? True
2 São o mesmo objeto? False
```

Operações elementares

Operadores de Associação:

Servem para verificar se determinado objeto está **associado** ou **pertence** a determinada estrutura de dados

Operador	Função
<code>in</code>	Retorna <code>True</code> caso o valor seja encontrado na sequência
<code>not in</code>	Retorna <code>True</code> caso o valor não seja encontrado na sequência

Exemplo da utilização de cada operador de associação mencionado acima:

```
1 lista = ["Python", 'Academy', "Operadores", 'Condições']
2
3 # Verifica se existe a string dentro da lista
4 print('Python' in lista) # Saída: True
5
6 # Verifica se não existe a string dentro da lista
7 print('SQL' not in lista) # Saída: True
```

Prioridade dos Operadores

Prioridade dos Operadores (da maior para a menor prioridade):

`()` ->Parêntesis

`**`, `not`

`*`, `/`, `%`, `//`, and

`+`, `-`, or

`<=`, `<`, `>`, `>=`

`==`, `!=`

`=`, `%=`, `/=`, `//=`, `-=`, `+=`, `*=`

Palavras Reservadas

```
import keyword  
print(keyword.kwlist)
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue',  
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',  
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Bibliotecas padrão Python

O ambiente Python tem por defeito um conjunto de módulos predefinidos que contém as funções normalmente aplicadas em diferentes domínios, tais como os módulos `math`, `time`, `random`, `string`, entre outras que definem bibliotecas de software e de funções que são aplicadas no desenvolvimento e implementação de programas.

Exemplos de bibliotecas padrão do Python:

In []:

```
import os
#interação com o sistema operativo (os)

import sys
#interação com o sistema Python

import string
#string: biblioteca de funções de manipulação de strings (cadeia de caracteres)
import math
#math: biblioteca de funções matemáticas
print(math.sqrt(25))
import time
?time
#time: biblioteca de funções data e hora
print(time.asctime())
```


Casting – conversão de tipos

O Python disponibiliza também várias funcionalidades que permitem converter valores entre diferentes tipos.

```
print (int("523")+1) #será exibido 524  
y = int(2.8) # y será 2  
z = int("3") # z será 3  
print (y, "\t", z)
```

```
int(2.9)
```

```
float(2)
```

```
str(4.78)
```

Exercícios – Parte I

Exercício 1

Converte as seguintes expressões matemáticas para que possam ser calculadas usando o interpretador Python.

$$10 + 20 \times 30$$

$$42 \div 30$$

$$(94 + 2) \times 6 - 1$$

Exercício 2

Considere a seguinte expressão:

$$10 \% 3 * 10 ** 2 + 1 - 10 * 4 / 2$$

Tente resolver o mesmo cálculo, usando apenas lápis e papel. Observe como a prioridade das operações é importante.

Agora digite a mesma expressão no interpretador. Obteve o mesmo resultado?

$$\begin{array}{lcl} 0 \rightarrow & 10 \% 3 * 10 ** 2 + 1 - 10 * 4 / 2 \\ 1 \rightarrow & 10 \% 3 * 100 & + 1 - 10 * 4 / 2 \\ 2 \rightarrow & 1 & * 100 & + 1 - 10 * 4 / 2 \\ 3 \rightarrow & 100 & + 1 - 10 * 4 / 2 \\ 4 \rightarrow & 100 & + 1 - 40 & / 2 \\ 5 \rightarrow & 100 & + 1 - 20 \\ 6 \rightarrow & 101 & - 20 \\ 7 \rightarrow & & 81 \end{array}$$

Exercício 3

- Cria um programa que imprima o teu nome completo no ecrã.

Exercício 4

Escreve um programa que exiba o resultado de $2a \times 3b$, em que **a** vale 3 e **b** vale 5.

Exercício 5

Indica qual o tipo de dados de cada elemento da lista seguir.

5

5.0

4.3

-2

100

1.333

Exercício 6

Considera

$a=4$, $b=10$, $c=5$, $d=1$, $f=5$

Indica se cada uma das expressões devolve **True** ou **False**

$a == c$

$a < b$

$d > b$

$c != f$

$a == b$

$c < d$

$b > a$

$c >= f$

$f >= c$

$c <= c$

$c <= f$

Exercício 7

Testa as seguintes operações:

```
print(5 + 10)
```

```
print(3 * 7, (17 - 2) * 8)
```

```
print(2 ** 16)
```

```
print(37 / 3)
```

```
print(37 // 3)
```

```
print(37 % 3)
```

Conceitos Básicos - Parte II

Entrada de Dados

Uma aplicação pode, a qualquer momento, ter a necessidade de pedir ao utilizador algum tipo de informação, para isso basta invocar a função `input()`

```
#coding: utf-8  
num = input("Digite um número:")  
print(num)
```

Saída de Dados

- O operador **%s** é colocado onde a *string* deve ser especificada
- O número de valores que se deseja anexar à *string* deve ser equivalente ao número especificado dentro dos parênteses após o operador **%** no final do conteúdo da *string*
- Também aceita valores numéricos, fazendo a sua conversão automaticamente.

```
nome="Pedro"  
idade = 10  
print("A idade da pessoa é %s." %idade)  
print("O %s gosta de Python." %nome)  
print("O %s tem %s anos de idade."%(nome,idade))
```

Saída de Dados

- O operador `%d` é usado como um espaço reservado para valores inteiros e o `%f` para decimais.
- Permitem imprimir números dentro de *strings*.
- Com o operador `%d` os valores decimais são convertidos automaticamente em valores inteiros.

```
numero = 10.9785
print("Valor inteiro:%d, Valor decimal:%f"%(numero,numero))
print("Valor arredondado:%2.2f"%numero)
```

Saída e Entrada de Dados

valor da função input é uma string. Para o utilizarmos como outro tipo de dados temos que efetuar conversão

```
v=input("valor? ")  
print("valor =",v," , quadrado =",int(v)**2)
```

Saída e Entrada de Dados

Carateres de escape e seu significado:

\\ Barra ao contrário ()

\' Plica (')

\\" Aspas (")

\a Toque de campainha

\b Retrocesso de um espaço

\f Salto de página

\n Salto de linha

\r "Return"

\t Tabulação horizontal

\v Tabulação vertical

```
v=input("Insira um número\n-> ")
```

```
print("valor =",v,"\tquadrado =",int(v)**2)
```

```
help('FORMATTING')
```

Estrutura e Controlo Sequencial

O mais simples dos mecanismos de composição de instruções é a denominada *sequenciação*. Consiste, essencialmente, na execução consecutiva de várias instruções, na ordem especificada no programa.

```
x=5;y=x+1;media=(x+y)/2;print("A média é: ", media)
```

(;) termina a execução sequencial das várias atribuições, da esquerda para a direita.

Apesar de suportada, e muito comum noutros ambientes, esta sintaxe é altamente desaconselhada em Python, por razões que promovem a legibilidade do código, bem como os princípios essenciais da programação estruturada. Deve-se, portanto, expressar a sequência das instruções com a mudança de linha.

```
x=5
y=x+1
media=(x+y)/2
print("A média é: ", media)
```


FT 01