

# Técnicas de Programação e Algoritmos

- Douglas Baptista de Godoy

 [/in/douglasbgodoy](https://www.linkedin.com/in/douglasbgodoy)

 [github.com/douglasbgodoy](https://github.com/douglasbgodoy)

# Python Essentials 1: Module 3.

conditional execution

---



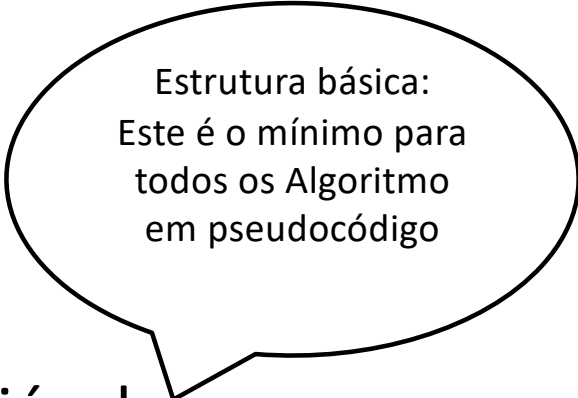
# Index of Contents



Comparison operators and conditional execution



- Comandos da linguagem de programação
- Algoritmo em pseudocódigo



Estrutura básica:  
Este é o mínimo para  
todos os Algoritmo  
em pseudocódigo

ALGORITMO

DECLARE nome\_da\_variável tipo\_da\_variável

bloco\_de\_comandos

FIM\_ALGORITMO.

# • Comandos da linguagem de programação

- Algoritmo em pseudocódigo
- Declaração de variáveis em algoritmos
- As variáveis são declaradas após a palavra DECLARE e os tipos mais utilizados são: NUMÉRICO (para variáveis que receberão números), LITERAL (para variáveis que receberão caracteres) e LÓGICO (para variáveis que receberão apenas dois valores: verdadeiro ou falso).
- Exemplo:

DECLARE X NUMÉRICO  
Y, Z LITERAL  
TESTE LÓGICO

- Comandos da linguagem de programação

- Algoritmo em pseudocódigo
- Comando de atribuição em algoritmos
- O comando de atribuição é utilizado para conceder valores ou operações a variáveis, sendo representado pelo símbolo  $\leftarrow$ . (=)

Exemplo:

$x \leftarrow 4$   
 $x \leftarrow x + 2$   
 $y \leftarrow \text{"aula"}$   
 $\text{teste} \leftarrow \text{falso}$

# • Comandos da linguagem de programação

- Algoritmo em pseudocódigo
- Comando de entrada em algoritmos
- O comando de entrada é utilizado para receber dados digitados pelo usuário, que serão armazenados em variáveis. Esse comando é representado pela palavra LEIA.

Exemplo:

LEIA X

Um valor digitado pelo usuário será armazenado na variável X.

LEIA Y

Um ou vários caracteres digitados pelo usuário serão armazenados na variável Y.

- Comandos da linguagem de programação

- Algoritmo em pseudocódigo
- Comando de saída em algoritmos
- O comando de saída é utilizado para mostrar dados na tela ou na impressora. Esse comando é representado pela palavra ESCREVA, e os dados podem ser conteúdos de variáveis ou mensagens.

Exemplo:

ESCREVA X

Mostra o valor armazenado na variável X.

ESCREVA "Conteúdo de Y = ",Y

Mostra a mensagem "Conteúdo de Y = " e, em seguida, o valor armazenado na variável Y.



# • Programação estruturada

- **Estrutura condicional em algoritmos** - A estrutura condicional em algoritmos pode ser simples ou composta.

Exemplo:

## **Estrutura condicional simples**

SE condição *ENTÃO*  
comando

- O comando só será executado se a condição for verdadeira. Uma condição é uma comparação que possui dois valores possíveis: verdadeiro ou falso.

Exemplo:

SE condição *ENTÃO*  
INÍCIO  
comando1  
comando2  
comando3  
FIM

- Os comandos 1, 2 e 3 só serão executados se a condição for verdadeira. As palavras INÍCIO e FIM serão necessárias apenas quando dois ou mais comandos forem executados.

# • Programação estruturada

- Estrutura condicional composta

- Exemplo1:

```
SE condição ENTÃO
    comando1
SENÃO
    comando2
```

Se a condição for verdadeira, será executado o comando1; caso contrário, será executado o comando2.

Exemplo2:

```
SE condição
    ENTÃO INÍCIO
        comando1
        comando2
    FIM
    SENÃO INÍCIO
        comando3
        comando4
    FIM
```

Se a condição for verdadeira, o comando1 e o comando2 serão executados; caso contrário, o comando3 e o comando4 serão executados.

# Algoritmo em pseudocódigo

- Exemplo: Faca um algoritmo para mostrar o resultado da divisão de dois números

ALGORITMO

DECLARE N1, N2, D NUMÉRICO

ESCREVA "Digite dois Números"

LEIA N1, N2

SE N2 = 0 ENTÃO

    ESCREVA "Impossível dividir"

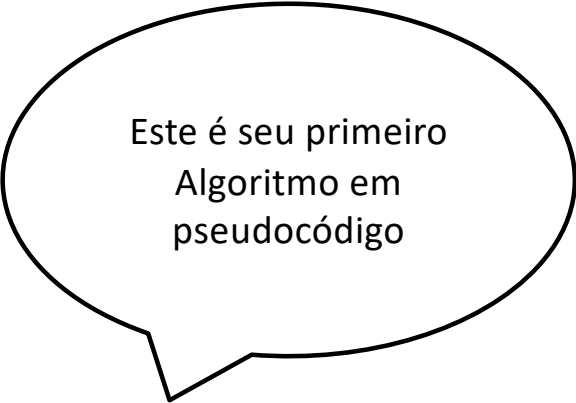
SENÃO INÍCIO

$D = N1/N2$

    ESCREVA "Divisão = ", D

    FIM

FIM\_ALGORITMO



Este é seu primeiro  
Algoritmo em  
pseudocódigo

Fonte: Fundamentos da Programação de Computadores, Pearson Editora, 3ª edição





# Questions & Answers.

---

A programmer writes a program and the program asks questions. A computer executes the program and provides the answers. The program must be able to react according to the received answers.

Fortunately, computers know only two kinds of answers:

-  yes, this is true;
-  no, this is false.

You will never get a response like "Let me think...., I don't know", or "Probably yes, but I don't know for sure".

**To ask questions, Python uses a set of very special operators.**

## Equality: the equal to operator (==)

The `=` (equal to) operator compares the values of two operands. If they are equal, the result of the comparison is `True`. If they are not equal, the result of the comparison is `False`.

```
1 print(2==2)
2
```

True

## Inequality: the not equal to operator (!=)

The `!=` (not equal to) operator compares the values of two operands, too. Here is the difference: if they are equal, the result of the comparison is `False`. If they are not equal, the result of the comparison is `True`.

ponto de exclamação (!)



## Comparison operators: greater than ( > )

You can also ask a comparison question using the > (greater than) operator.

```
n = int(input("Digite Numero: "))  
print(n>100)
```

120  
True

## greater than or equal to ( >= )

The greater than operator has another special, non-strict variant, but it's denoted differently than in classical arithmetic notation: >= (greater than or equal to).

Both of these operators, are binary operators with left-sided binding, and their **priority** is greater than that shown by == and !=



## Comparison operators: Less than ( < )

You can also ask a comparison question using the < (Less than) operator.

```
n = int(input("Digite Numero: "))  
print(n<100)
```

3  
True

## Less than or equal to ( <= )

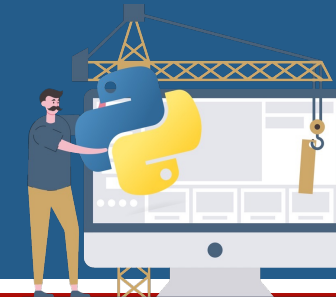
The less than operator has another special, non-strict variant, but it's denoted differently than in classical arithmetic notation: <= (less than or equal to).

Both of these operators, are binary operators with left-sided binding,

and their **priority** is greater than that shown by == and !=

Now we need to update our priority table, and put all the new operators into it. It now looks as follows:

Priority	Operator	
1	+ , -	unary
2	**	
3	* , / , // , %	
4	+ , -	binary
5	< , <= , > , >=	
6	== , !=	



# Comparison operators:

1. Os operadores de **comparação** (também conhecidos como *relacionais*) são usados para comparar valores. A tabela abaixo ilustra como os operadores de comparação funcionam, supondo que `x = 0`, `y = 1`, e `z = 0`:

Operador	Descrição	Exemplo
<code>==</code>	retorna <code>True</code> se os valores dos operandos forem iguais e <code>False</code> caso contrário	<code>x == y # False</code> <code>x == z # True</code>
<code>!=</code>	retorna <code>True</code> se os valores dos operandos não forem iguais e <code>False</code> caso contrário	<code>x != y # True</code> <code>x != z # False</code>
<code>&gt;</code>	<code>True</code> se o valor do operando esquerdo for maior que o valor do operando direito e <code>False</code> caso contrário	<code>x &gt; y # False</code> <code>y &gt; z # True</code>
<code>&lt;</code>	<code>True</code> se o valor do operando esquerdo for menor que o valor do operando direito e <code>False</code> caso contrário	<code>x &lt; y # True</code> <code>y &lt; z # False</code>
<code>&gt;=</code>	<code>True</code> se o valor do operando esquerdo for maior ou igual ao valor do operando da direita e <code>False</code> caso contrário	<code>x &gt;= y # False</code> <code>x &gt;= z # True</code> <code>y &gt;= z # True</code>
<code>&lt;=</code>	<code>True</code> se o valor do operando esquerdo for menor ou igual ao valor do operando à direita e <code>False</code> caso contrário	<code>x &lt;= y # True</code> <code>x &lt;= z # True</code> <code>y &lt;= z # False</code>



# Comparison operators:

The comparison (or the so-called relational) operators are used to compare values. The table below illustrates how the comparison operators work, assuming that  $x = 0$ ,  $y = 1$ , and  $z = 0$ :

Operator	Description	Example
<code>==</code>	returns if operands' values are equal, and <code>False</code> otherwise	<pre>x == y # False x == z # True</pre>
<code>!=</code>	returns <code>True</code> if operands' values are not equal, and <code>False</code> otherwise	<pre>x != y # True x != z # False</pre>
<code>&gt;</code>	<code>True</code> if the left operand's value is greater than the right operand's value, and <code>False</code> otherwise	<pre>x &gt; y # False y &gt; z # True</pre>
<code>&lt;</code>	<code>True</code> if the left operand's value is less than the right operand's value, and <code>False</code> otherwise	<pre>x &lt; y # True y &lt; z # False</pre>
<code>&gt;=</code>	<code>True</code> if the left operand's value is greater than or equal to the right operand's value, and <code>False</code> otherwise	<pre>x &gt;= y # False x &gt;= z # True y &gt;= z # True</pre>
<code>&lt;=</code>	<code>True</code> if the left operand's value is less than or equal to the right operand's value, and <code>False</code> otherwise	<pre>x &lt;= y # True x &lt;= z # True y &lt;= z # False</pre>

# Conditions and conditional execution

---

You already know how to ask Python questions, but you still don't know how to make reasonable use of the answers. You have to have a mechanism which will allow you to do something if a condition is met, and not do it if it isn't.

To make such decisions, Python offers a special instruction. Due to its nature and its application, it's called a conditional instruction (or conditional statement).

The first form of a conditional statement, which you can see below is written very informally but figuratively:

```
if true_or_not:  
    do_this_if_true
```

# Conditions and conditional execution

---

This conditional statement consists of the following, strictly necessary, elements in this and this order only:



the `if` keyword;



one or more white spaces;



an expression (a question or an answer) whose value will be interpreted solely in terms of `True` (when its value is non-zero) and `False` (when it is equal to zero);



a colon followed by a newline;



an indented instruction or set of instructions.

If the `true_or_not` expression represents the truth, the indented statement(s) will be executed;

If the `true_or_not` expression does not represent the truth the indented statement(s) will be omitted (ignored), and the next executed instruction will be the one after the original indentation level.

# The `if` statement

---

You already know how to ask Python questions, but you still don't know how to make reasonable use of the answers. You have to have a mechanism which will allow you to do something if a condition is met, and not do it if it isn't.

```
x = 10

if x == 10: # condition
    print("x is equal to 10") # Executed if the condition is True.
```

Now we know what we'll do if the conditions are met, and we know what we'll do if not everything goes our way. In other words, we have a "Plan B".

```
x = 10

if x < 10: # Condition
    print("x is less than 10") # Executed if the condition is True.
else:
    print("x is greater than or equal to 10") # Executed if the condition is False.
```

# The `if` statement

---



a series of `if` statements, e.g.:

```
x = 10

if x > 5: # condition one
    print("x is greater than 5") # Executed if condition one is True.

if x < 10: # condition two
    print("x is less than 10") # Executed if condition two is True.

if x == 10: # condition three
    print("x is equal to 10") # Executed if condition three is True.
```

# The `if` statement



a series of `if` statements followed by an `else`, e.g.:

```
x = 10

if x > 5:  # True
    print("x > 5")

if x > 8:  # True
    print("x > 8")

if x > 10: # False
    print("x > 10")

else:
    print("else will be executed")
```

Each `if` is tested separately. The body of `else` is executed if the last `if` is False.

# The `elif` statement

---

The second special case introduces another new Python keyword: `elif`.

`elif` is used to check more than just one condition, and to stop when the first statement which is true is found.

```
x = 10

if x == 10: # True
    print("x == 10")

if x > 15: # False
    print("x > 15")

elif x > 10: # False
    print("x > 10")

elif x > 5: # True
    print("x > 5")

else:
    print("else will not be executed")
```

# The `elif` statement

---

If the condition for `if` is false, the program checks the conditions of the subsequent `elif` blocks – the first `elif` block that is `True` is executed. If all the conditions are `False`, the `else` block will be executed.

Nested conditional statements, e.g.:

```
x = 10

if x > 5: # True
    if x == 6: # False
        print("nested: x == 6")
    elif x == 10: # True
        print("nested: x == 10")
    else:
        print("nested: else")
else:
    print("else")
```



# Exercise 1

What is the output of the following snippet?

```
x, y, z = 5, 10, 8
x, y, z = z, y, x

print(x > z)
print((y - 5) == x)
```

True  
False

What is the output of the following snippet?

```
x = 10

if x == 10:
    print(x == 10)
if x > 5:
    print(x > 5)
if x < 10:
    print(x < 10)
else:
    print("else")
```

True  
True  
else



# Exercise 2

What is the output of the following snippet?



```
x = "1"

if x == 1:
    print("one")
elif x == "1":
    if int(x) > 1:
        print("two")
    elif int(x) < 1:
        print("three")
    else:
        print("four")
if int(x) == 1:
    print("five")
else:
    print("six")
```

four  
five

# Estruturas de Controle

## If-Else

```
if exp:  
    #comandos  
else:  
    #comandos
```

## If-Else-If-Else

```
if exp:  
    #comandos  
elif exp:  
    #comandos  
else:  
    #comandos
```

# Computer logic

Let's look at this sentence:

If we have some free time, and the weather is good, we will go for a walk.

We've used the conjunction `and`, which means that going for a walk depends on the simultaneous fulfilment of these two conditions. In the language of logic, such a connection of conditions is called a conjunction.

And now another example:

If you are in the mall or I am in the mall, one of us will buy a gift for Mom.

The appearance of the word `or` means that the purchase depends on at least one of these conditions. In logic, such a compound is called a disjunction.



# and (conjunction)

One logical conjunction operator in Python is the word `and`. It's a binary operator with a priority that is lower than the one expressed by the comparison operators. It allows us to code complex conditions without the use of parentheses like this one:

```
counter > 0 and value == 100
```

The result provided by the `and` operator can be determined on the basis of the truth table.

Argument A	Argument B	A and B
False	False	False
False	True	False
True	False	False
True	True	True



# or (disjunction)

The word `or` is a disjunction operator. It's a binary operator with a lower priority than `and` (just like `+` compared to `*`). Its truth table is as follows:

Argument A	Argument B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

## not

In addition, there's another operator that can be applied for constructing conditions. It's a unary operator performing a logical negation. Its operation is simple: it turns truth into falsehood and falsehood into truth.

Argument	not Argument
False	True
True	False



# Operadores Relacionais

- Exemplos

- `x=3`
- `num1 = 50`
- `num2 = 8`

- `if( x == 3):`
- `print("Número igual a 3")`
- 
- `if (num1 > num2):`
- `print("\nO maior numero :",num1)`
- `if (x > 0):`
- `print("O numero digitado e positivo")`

# Operadores Lógicos e Expressões Lógicas

TABELA E (and)	TABELA OU (or)	TABELA NÃO (not)
$V \text{ e } V = V$	$V \text{ ou } V = V$	$\text{Não } V = F$
$V \text{ e } F = F$	$V \text{ ou } F = V$	$\text{Não } F = V$
$F \text{ e } V = F$	$F \text{ ou } V = V$	
$F \text{ e } F = F$	$F \text{ ou } F = F$	

Conjunção and  
Disjunção or



# Operadores Lógicos e Expressões Lógicas

p	q	p && q	p    q
falso	falso	falso	falso
falso	verdadeiro	falso	verdadeiro
verdadeiro	falso	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro

Exemplo 1:  $72 > 30$  &&  $32 \leq 10$   
Verdadeiro && Falso  
Falso

Exemplo 2:  $9 == 3$  ||  $10 \leq 10$   
Falso || Verdadeiro  
Verdadeiro

# Operadores Lógicos e Expressões Lógicas

- Exemplos
- if  $x > 5$  and  $x < 10$ :
- `print("\n Numero entre 5 e 10 ")`
- if  $((x == 5 \text{ and } y == 2) \text{ or } (y == 3))$  :
- `print(" x é igual a 5 e y é igual a 2, ou y é igual a 3 ")`
- if  $(x == 5 \text{ and } (y == 2 \text{ or } y == 3))$  :
- `print("x é igual a 5, e y é igual a 2 ou y é igual a 3 ")`

# Referências Bibliográficas

- MENEZES, Nilo Ney Coutinho, **Introdução à Programação com Python**, Novatec Editora, 4ª edição.
- CISCO NETWORKING ACADEMY - Python Essentials 1, Disponível em: <https://skillsforall.com/pt/course/python-essentials-1?courseLang=en-US>. Acesso em: 21 fevereiro.2024.
- [www.pythoninstitute.org](http://www.pythoninstitute.org)

