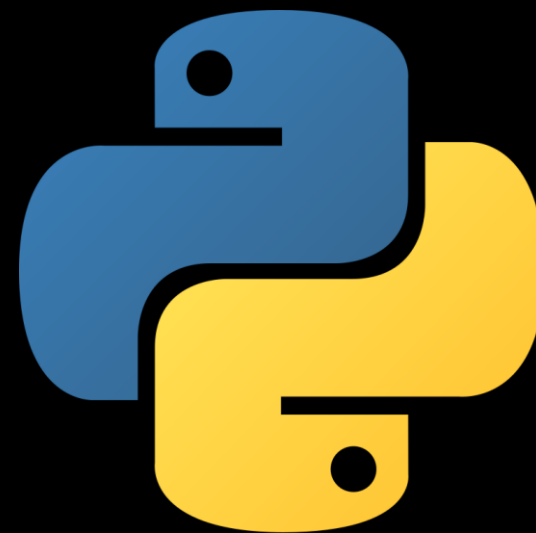


PYTHON – NOTAS DE ESTUDO

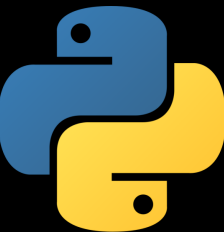


Por Thiago Barros

Github - <https://github.com/Tbarros1996>

PYTHON – NOTAS DE ESTUDO

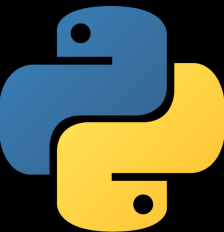
VISÃO GERAL DA LINGUAGEM PYTHON



AMBIENTE DE DESENVOLVIMENTO INTEGRADO

Software que contém um conjunto de ferramentas para o desenvolvimento de Script em uma linguagem de programação específica ou várias.

Com uma IDE é possível construir o código fonte de um software e melhorá-lo. Algumas IDE permitem o processo de correção de erros (bugs) entre outras coisas.



AMBIENTE DE DESENVOLVIMENTO INTEGRADO

IDE

USO GERAL

PYTHON

VSCODE

VSCODE

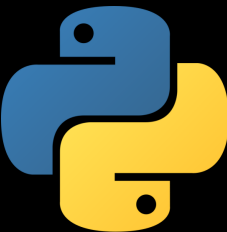
NOTEPAD++

PYCHARM

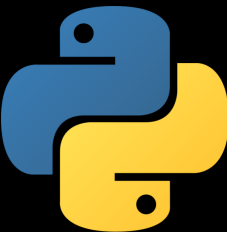
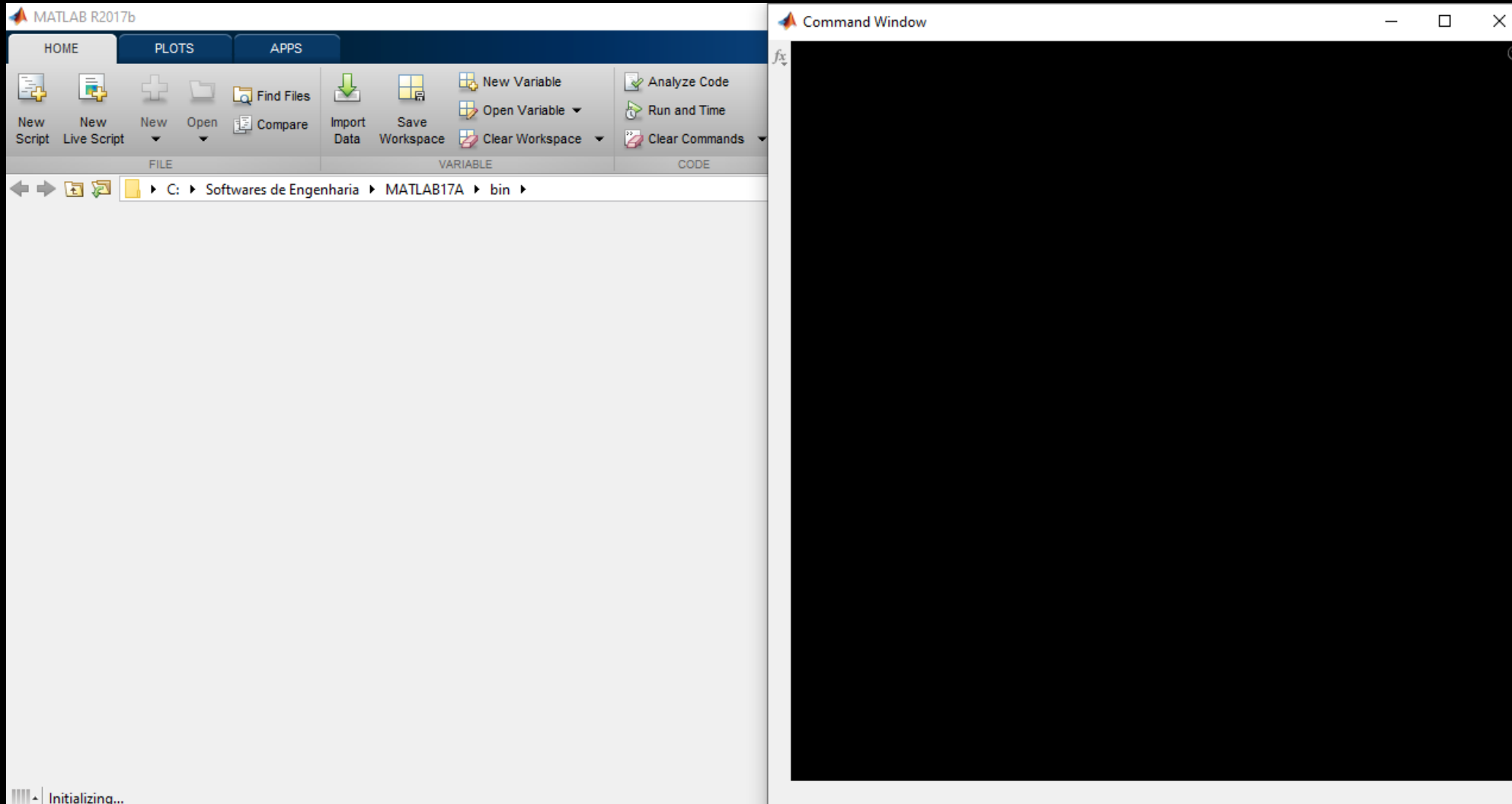
Editor de
Texto(???)

JUPYTER

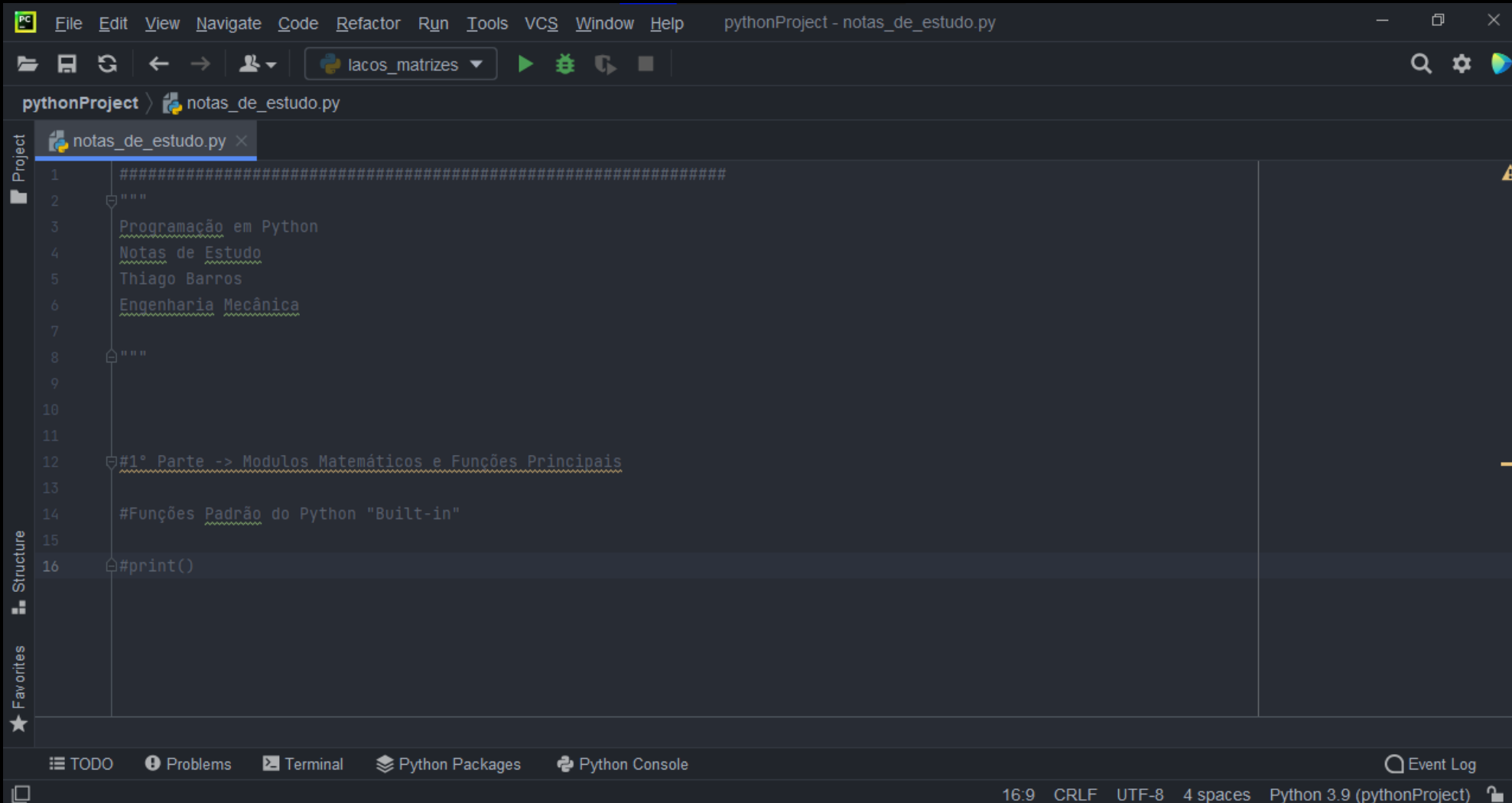
SPYDER



AMBIENTE DE DESENVOLVIMENTO INTEGRADO – EX MATLAB



PYCHARM - PYTHON



INTRODUÇÃO A LINGUAGEM PYTHON

CRIADOR: GUIDO VAN ROSSUM

ANO DE CRIAÇÃO: 1992

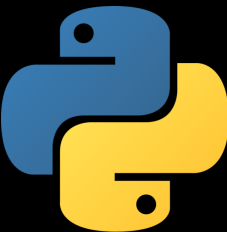
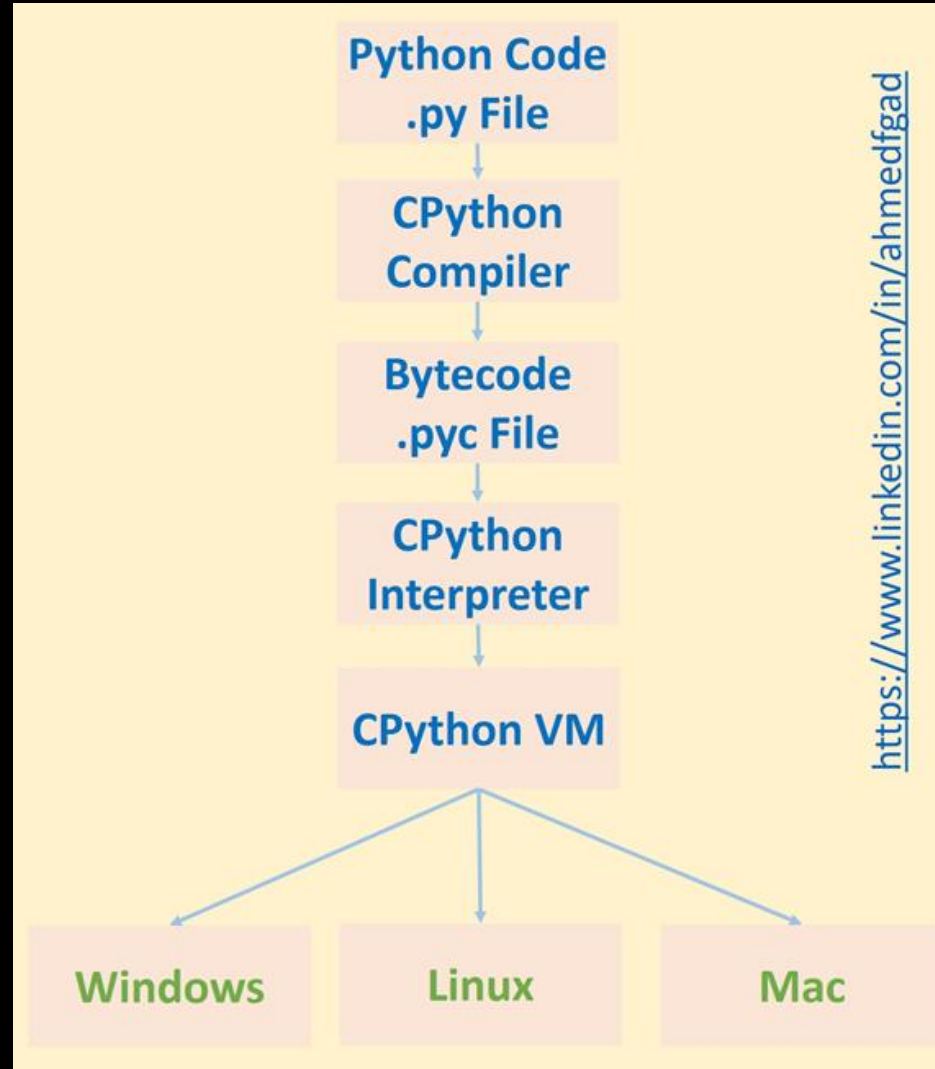
LINGUAGEM COMPILADA, ORIENTADA A OBJETOS, DE ALTO NÍVEL

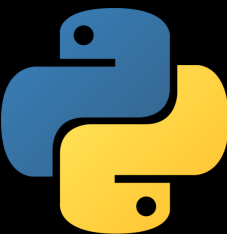
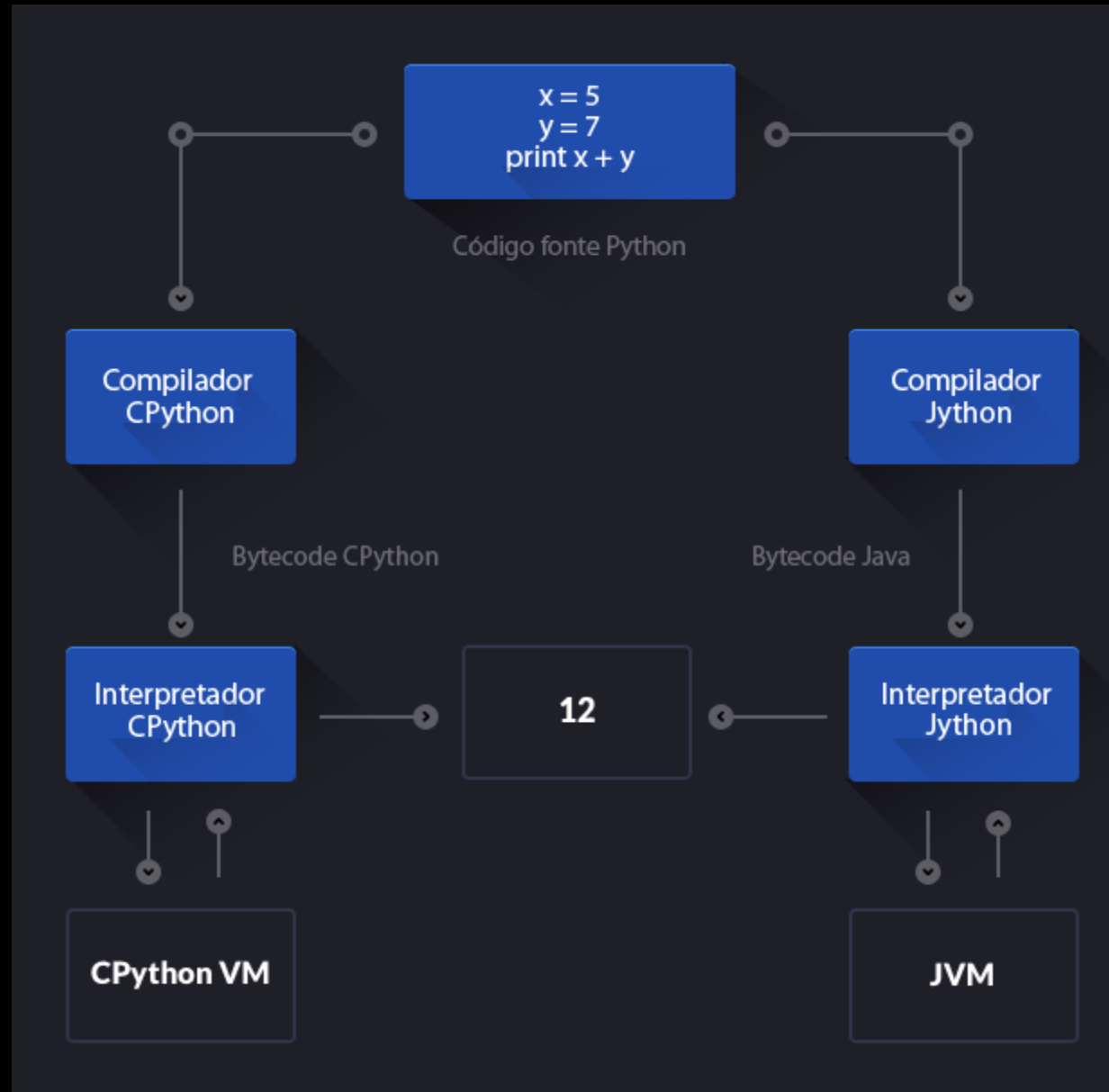
CANVETE SUÇO

USO GERAL



FUNCIONAMENTO DA LINGUAGEM PYTHON





CÓDIGO FONTE EM LINGUAGEM
PYTHON

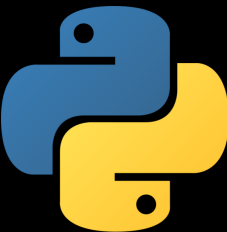
IMPLEMENTAÇÃO DA
LINGUAGEM PYTHON EM OUTRA
LINGUAGEM COPIADA (C-
PYTHON/JAVA-PYTHON)

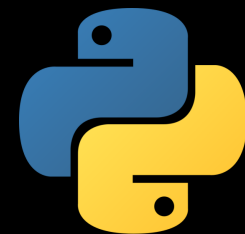
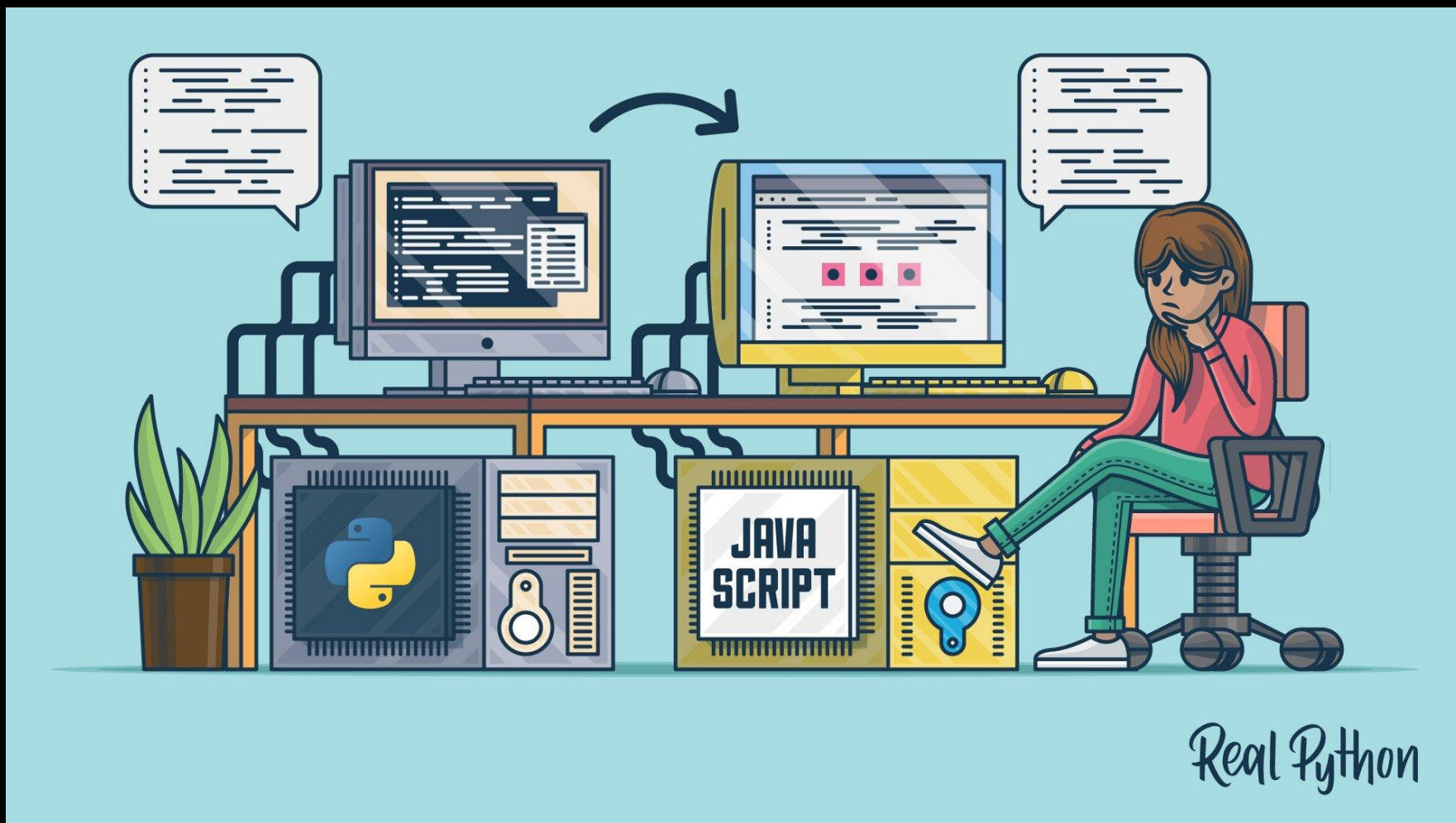
CÓDIGO BINÁRIO

PROCESSAMENTO – MÁQUINA
VIRTUAL

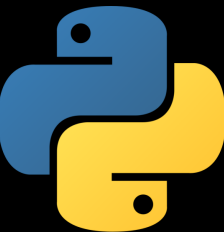
GERAÇÃO DE RESULTADO

INTERPRETAÇÃO DE RESULTADO
NA LINGUAGEM HUMANA





Funções Padrão do Python – Parte 1

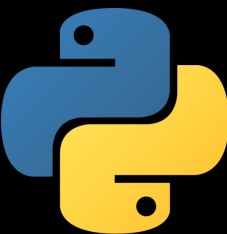


Função `print()`

Objetivo – Escreve um resultado, ou um conjunto de resultados na tela. (similar a `fprintf` do MATLAB)

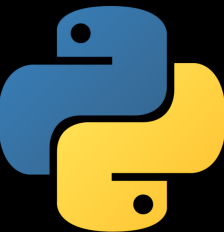
```
>>> print("batata")  
batata
```

Sempre que for imprimir um texto, esse deve está em aspas retas duplas, para que o Python Interprete aquilo como texto.



```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

`end = ' '` ---→ Inserção de caracteres no final de cada texto
para tabulação ou quebra de texto



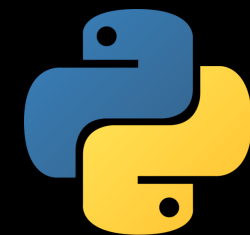
```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

*Objects--→ O que deve ser impresso em tela, pode ser uma ou várias variáveis

Sep=' '---→ Um símbolo numérico para ser impresso em tela para separa os textos



Sequência	Descrição
\n	Insere uma quebra de linha.
\t	Insere tabulação horizontal.
\v	Insere tabulação vertical.
\r	Equivalente ao efeito da tecla Enter.
\'	Aspas simples.
\"	Aspas duplas.
\\	Insere uma barra invertida (<i>backslash</i>).
\a	Chamado de ASC bell ou <i>beep</i> do sistema. Se houver suporte, aciona um bipe.
\b	Aciona o <i>backspace</i> , ou seja, apaga o caractere anterior.
\f	Insere uma quebra de página.
\u	Insere um caractere UNICODE. Deve acompanhar um código com 4 números.

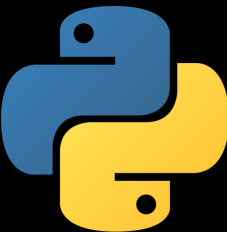


Função `input()`

Objetivo – inserção de informação pelo usuário em uma variável

```
>? 12
>>> x = input(print("Entre com Valor",sep="---"))
Entre com Valor
None>? 21
>>> print(x)
21

>>>
```



Função `help()`

Objetivo – Abre um conjunto de informações explicando uma determinada função ou classe



```
>>> help()
```

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <https://docs.python.org/3.9/tutorial/>.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

```
help> version
```

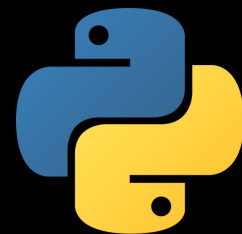
No Python documentation found for 'version'.

Use help() to get the interactive help utility.

Use help(str) for help on the str class.

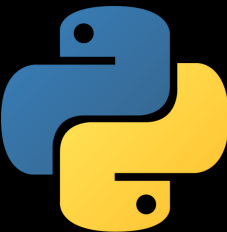
```
help> string
```

Squeezed text (143 lines).



Função `help()`

Objetivo – Abre um conjunto de informações explicando uma determinada função ou classe



```
>>> #Função Help no Power Shell
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

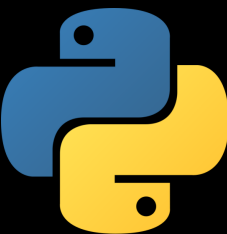
help> for
The "for" statement
*****

The "for" statement is used to iterate over the elements of a sequence
(such as a string, tuple or list) or other iterable object:

    for_stmt ::= "for" target_list "in" expression_list ":" suite
               ["else" ":" suite]

The expression list is evaluated once; it should yield an iterable
object.  An iterator is created for the result of the
"expression_list".  The suite is then executed once for each item
provided by the iterator, in the order returned by the iterator.  Each
item in turn is assigned to the target list using the standard rules
for assignments (see Assignment statements), and then the suite is
executed.  When the items are exhausted (which is immediately when the
sequence is empty or an iterator raises a "StopIteration" exception),
the suite in the "else" clause, if present, is executed, and the loop
terminates.

A "break" statement executed in the first suite terminates the loop
```



Função `type()`

Função que Identifica o tipo de classe do objeto(Ou o tipo de dado, exemplo `FLOAT`,`TRING`,`BOOL`)

```
>>> x = True
>>> type(x)
<class 'bool'>
```



Função len()

Função que calcula o número de objetos em uma classe

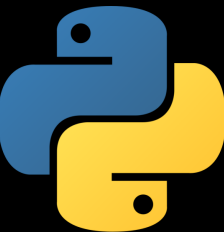
```
>>> nome = "Batata"
>>> y = len(nome)
>>> print(y)
6
```



Função `max()`, `min()`

Retornamos o máximo e o mínimo em uma estrutura de dados (Iterable) como uma lista

```
>>> lista = [1,2,3,4,5,6,7,8,9,0]
>>> max(lista)
9
>>> min(lista)
0
```



1) Escreva um algoritmo que armazene o valor 10 em uma variável A e o valor 20 em uma variável B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final, escrever os valores que ficaram armazenados nas variáveis.

```
main()
#Algoritmo
declare a,b,a2,b2 inter
write "Entre com valor de a"
input a
write "Entre com valor de b"
input b

print "Os valores de a e b são: "
print a, b

a2<---a
b2<---b

a<----b2
b<----a2

print "Os valores de a e b invertidos são:"
print a, b

end algorithm()
```

```
a = input(print("Entre com valor de A: "))
b = input(print("Entre com o valor de B:"))

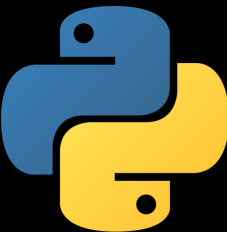
a = int(a)
b = int(b)

print("Os valores de a e b são:", a, b, "respectivamente")

a2 = a
b2 = b

a = b2
b = a2

print("Os valores de a e b invertidos são", a, b, "Respectivamente")
```



```
F:\Projetos_Python\venv\Scripts\python.exe F:/Projetos_Pythor
```

```
Entre com valor de A:
```

```
None10
```

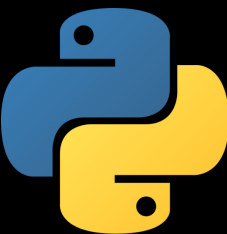
```
Entre com o valor de B:
```

```
None20
```

```
Os valores de a e b são: 10 20 respectivamente
```

```
Os valores de a e b invertidos são 20 10 Respectivamente
```

```
Process finished with exit code 0
```



```
a = input(print("Entre com valor de A: "))
b = input(print("Entre com o valor de B:"))

a = int(a)
b = int(b)

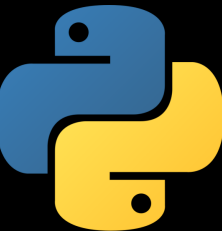
print("Os valores de a e b são:", a, b, "respectivamente")

arranjo = [a, b]

a = arranjo[1]
b = arranjo[0]

print("Os valores de a e b, invertidos, são:", a, b, "respectivamente")
```

|



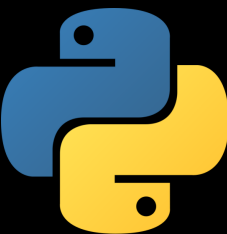
2) Analise os algoritmos abaixo e diga o que será impresso na tela ao serem executados:

a)

$A \leftarrow 10$
$B \leftarrow 20$
Escrever B
$B \leftarrow 5$
Escrever A, B

```
main()
declare a,b inter
a<---10
b<---20
write b
b<---5
write a,b
end algorithm()
```

a vai receber 10
b vai receber 20
vai mostrar na tela 20
b vai ser reescrito com 5
vai mostrar na tela 10 e 5



2) Analise os algoritmos abaixo e diga o que será impresso na tela ao serem executados:

```
F:\Projetos_Python\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2021
```

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['F:\\Projetos_Python', 'F:/Projetos_Python'])
```

```
Python Console>>> a = 10
```

```
>>> b = 20
```

```
>>> print(a,b)
```

```
10 20
```

```
>>> b = 5
```

```
>>> print(a,b)
```

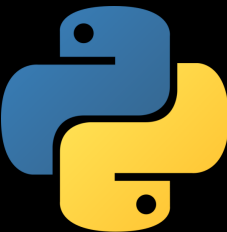
```
10 5
```

```
>>>
```

```
01 a = {int} 10
```

```
01 b = {int} 5
```

```
> ■ Special Variables
```



2) Analise os algoritmos abaixo e diga o que será impresso na tela ao serem executados:

b)

$A \leftarrow 30$

$B \leftarrow 20$

$C \leftarrow A + B$

Escrever C

$B \leftarrow 10$

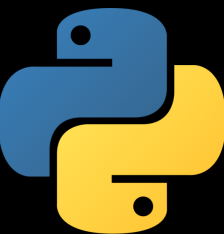
Escrever B, C

$C \leftarrow A + B$

Escrever A, B, C

A a e b vai ser atribuido 30 e 20
c é a soma de a e b
escreve na tela b
b é atualizado com 10
b é escrito na tela com c mas c ainda possui o valor anterior
c é atualizado com o novo valor de b e a soma a e b ocorre
agora temos a e b e c com os valores finais

```
main algorithm ()  
  
declare a,b,c inter  
  
a<---30  
b<---20  
  
c<---(a+b)  
  
write c  
  
b<-----10  
  
write b,c  
  
c<-----(a+b)  
  
write a,b,c  
  
end algorithm()
```



2) Analise os algoritmos abaixo e diga o que será impresso na tela ao serem executados:

$A \leftarrow 6*(3+2)$	$F \leftarrow (6/3)+(8/2)$
$B \leftarrow 2+(6*(3+2))$	$G \leftarrow ((3+(8/2))*4)+(3*2)$
$C \leftarrow 2+(3*6)/(2+4)$	$H \leftarrow (6*(3*3)+6)-10$
$D \leftarrow 2*(8/(3+1))$	$I \leftarrow (((10*8)+3)*9)$
$E \leftarrow 3+(16-2)/(2*(9-2))$	$J \leftarrow ((-12)*(-4))+(3*(-4))$

```
>>> 6*(3+2)
30
```

```
>>> 2+(3*6)/(2+4)
5.0
```

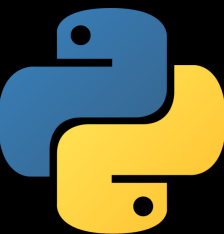
```
>>> (((10*8)+3)*9)
747
```

```
>>> 3+(16-2)/(2*(9-2))
4.0
```

```
>>> 2+(6*(3+2))
32
```

```
>>> ((-12)*(-4))+(3*(-4))
36
```

```
>>> ((3+(8/2))*4)+(3*2)
34.0
```



5) Escreva um algoritmo para ler um valor (do teclado) e escrever (na tela) o seu antecessor.

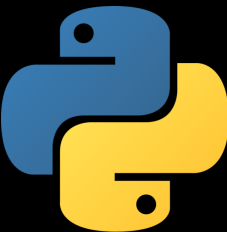
```
"""
Programa que o usuario executa o e insere um número e ele mostra o número
"""

numero = input(print("Insira o número: "))

numero = float(numero)

antecessor = numero - 1

print("O antecessor de: ", numero, "é", antecessor)
```



Projeto Python > Problemas_Matemáticos > Problemas_Gerais > area_triângulo.py

area_triângulo.py

```
1 """
2     Esse algoritmo calcula a área do triângulo com a fórmula  $at = b*a/2$ 
3
4
5 """
6
7 base = input("Entre com o valor da base, em METROS")
8 altura = input("Entre com valor da altura em METROS")
9
10 base = float(base)
11 altura = float(altura)
12
13 area = (base*altura)/2
14
15 print("A área do triângulo é: ", area)
16
17
```

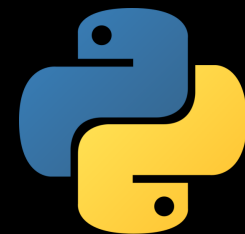
Problemas: Current File Project Errors

No problems in area_triângulo.py

7:1 CRLF UTF-8 4 spaces Python 3.9 (Projeto Python)

F:\Projetos_Python\venv\Scripts\python.exe F:/Projetos_Python/Problemas
Entre com o valor da base, em METROS2
Entre com valor da altura em METROS2
A área do triângulo é: 2.0

Process finished with exit code 0



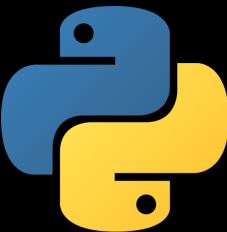
7) Faça um algoritmo que leia a idade de uma pessoa expressa em anos, meses e dias e escreva a idade dessa pessoa expressa apenas em dias. Considerar ano com 365 dias e mês com 30 dias.

```
Você nasceu a 181440 dias
```

```
Voce tem: 504 anos
```

```
É IMPORTANTE SALIENTAR QUE ESSE PROGRAMA NÃO CONSIDERA ANOS BISSEXTOS E TODOS OS MESES TEM MESMA QUANTIDADE DE DIAS:30
```

```
Process finished with exit code 0
```



```
"""
```

```
Zeus Computação Científica
```

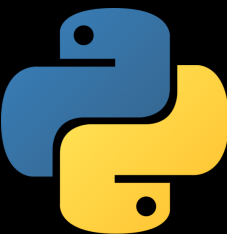
```
Thiago Barros
```

```
Programa: Algoritmo que calcula as raízes de uma equação de segundo grau com base nos coeficiente da equação (a,b,c)
```

$$ax^2 + bx + c = 0$$

```
Programa baseado no Algoritmo em Matlab do Livro, Programação em Matlab para Cientistas e Engenheiros do Stephen Chapman
```

```
"""
```



```
# Definição das Variaveis
```

```
"""
```

```
a -----> Coeficiente do primeiro termo da equação de segundo grau  $x^2$ 
```

```
b -----> Coeficiente do segundo termo da equação de segundo grau  $x$ 
```

```
c -----> Coeficiente do terceiro termo da equação de segundo grau
```

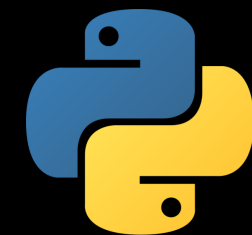
```
raizes_complexas-----> Soluções Complexas para a discriminante
```

```
raizes_reais -----> Soluções reais para a discriminante
```

```
x1-----> Primeira solução possível para a equação
```

```
x2 -----> Segunda solução Possível para a equação
```

```
"""
```



```
import math
```

```
import cmath
```

```
# Prompt para entrada de dados do usuario
```

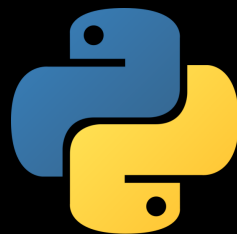
```
a = float(input(print("Entre com o valor do coeficiente 'a' da equação : ")))
```

```
b = float(input(print("Entre com o valor do coeficiente 'b' da equação : ")))
```

```
c = float(input(print("Entre com o valor do coeficiente 'c' da equação : ")))
```

```
# Calculo da discriminante
```

```
""" Calculando a equação de 2º grau """
```



```
# Cálculo da discriminante
```

```
discriminante = b ** 2 - 4 * a * c
```

```
if discriminante > 0:
```

```
    print("A discriminante é maior que zero, logo, a equação possui valores reais e diferentes")
```

```
    x1 = (((-1) * b) + math.sqrt(discriminante)) / (2 * a)
```

```
    x2 = (((-1) * b) - math.sqrt(discriminante)) / (2 * a)
```

```
    print("O valor da primeira raiz é: \n ", x1)
```

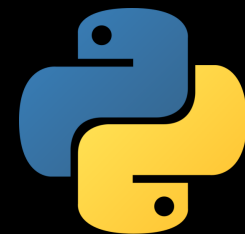
```
    print("O valor da segunda raiz é: \n ", x2)
```

```
if discriminante == 0:
```

```
    print("A função tem apenas uma raiz real")
```

```
    x1 = ((-1) * b) / (2 * a)
```

```
    print("A raiz da função é : \n", x1)
```



```
if discriminante > 0:

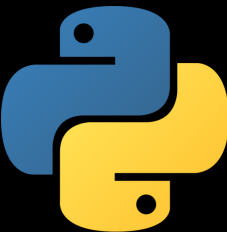
    print("A discriminante é maior que zero, logo, a equação possui valores reais e diferentes")
    x1 = (((-1) * b) + math.sqrt(discriminante)) / (2 * a)
    x2 = (((-1) * b) - math.sqrt(discriminante)) / (2 * a)
    print("O valor da primeira raiz é: \n ", x1)
    print("O valor da segunda raiz é: \n ", x2)

if discriminante == 0:

    print("A função tem apenas uma raiz real")
    x1 = ((-1) * b) / (2 * a)
    print("A raiz da função é : \n", x1)

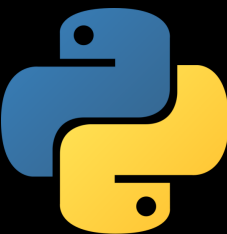
elif discriminante < 0:

    print("A equação tem discriminante menor que zero e possui raízes complexas")
    x1 = (((-1) * b) + cmath.sqrt(discriminante)) / (2 * a)
    x2 = (((-1) * b) - cmath.sqrt(discriminante)) / (2 * a)
    print("O valor da primeira raiz é: \n ", x1)
    print("O valor da segunda raiz é: \n ", x2)
```



Criar um algoritmo que leia o valor de um depósito e o valor da taxa de juros. Calcular e imprimir o valor do rendimento e o valor total depois do rendimento.

```
def juros(taxa, tempo, capital):  
    juros_acrescido = capital * taxa * tempo  
    montante = capital + juros_acrescido  
    print("O juros é de: ", juros_acrescido)  
    print("O montante foi de: ", montante)  
  
juros(0.04, 3, 500)
```

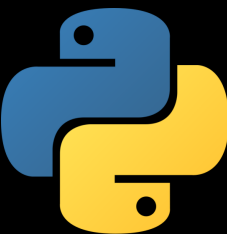


Criar um algoritmo que receba um número real, calcular e imprimir:

- *a parte inteira do número*
- *a parte fracionária do número*
- *o número arredondado*

```
# Retorna a parte inteira, fracionaria e real e arredondado

x = float(input(print("Entre com o Número Real")))
x_fracionario = x - int(x)
print(round(x_fracionario, ndigits=6))
x_inteiro = int(abs(x))
print(x_inteiro)
x_arredondado = round(x)
print(x_arredondado)
```



VÍDEO

entre com um numero com parte
fracionaria:7.1

parte inteira:7
parte fracionária:0.100
numero arredondado:7

entre com um numero com parte
fracionaria:8.5

parte inteira:8
parte fracionária:0.500
numero arredondado:9

entre com um numero com parte
fracionaria:7.49

parte inteira:7
parte fracionária:0.490
numero arredondado:7

entre com um numero com parte
fracionaria:8.4999

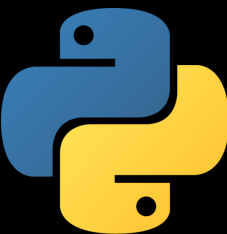
parte inteira:8
parte fracionária:0.499
numero arredondado:8

entre com um numero com parte
fracionaria:7.4999

parte inteira:7
parte fracionária:0.499
numero arredondado:7

entre com um numero com parte
fracionária:8.0

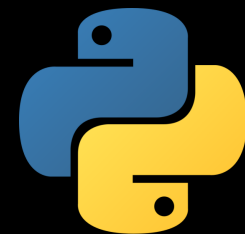
parte inteira:8
parte fracionária:1.000
numero arredondado:8



Instruções lógicas

Uma instrução isolada não permite realizar o processo completo. Para seguir uma sequência lógica é necessário um **conjunto de instruções** colocadas em **ordem sequencial lógica**.

É evidente que temos de seguir as instruções em uma **ordem adequada**, para realizar uma determinada tarefa. Imagine a tarefa de beber um refrigerante em lata, por exemplo:

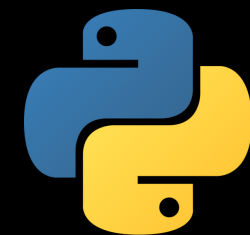


Introdução à lógica de programação

Conforme vimos no exemplo anterior (de beber o refrigerante em lata), para chegarmos a um resultado é necessário uma **ordem sequencial lógica**. A mesma coisa acontece com a lógica de programação.

Quando pensamos em iniciar uma programação para computador, temos de ter em mente que a máquina desconhece totalmente alguns conceitos que para nós são muito óbvios.

Por isto, devemos descrever cada passo de forma detalhada, por mais simples que possa parecer, para que tenha uma **sequência lógica na programação** e, assim, o computador possa executar todas as instruções necessárias para uma determinada tarefa.

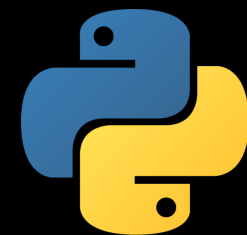


Por que a lógica de programação é tão importante?

A lógica para programação é o processo de procurar pensar na mesma **sequência** em que o computador executará as tarefas.

Seguindo a lógica de programação, procuramos imaginar como as **ações serão executadas**, partindo-se do estudo de um problema até chegar à solução dele, por meio da construção de um **algoritmo**.

Por isso, a lógica de programação é tão importante!



Ilustrando a sequência lógica

Já vimos que as **instruções lógicas** devem ser executadas em uma ordem adequada para o sucesso de uma programação. Uma instrução descrita errada, perde o sentido e influencia no resultado. Precisamos colocar em prática o conjunto de todas as instruções, na ordem correta. Para tal, podemos utilizar os **algoritmos**.

Algoritmos

Sequência finita de passos que levam à execução de uma tarefa.

Podemos pensar em um algoritmo como uma receita, ou seja, apresenta uma sequência de instruções para um fim específico.

Essas instruções devem ser descritas de maneira simples e objetiva.

Programas de Computadores

Os programas de computadores são os algoritmos escritos em uma linguagem computacional, como Java, C#, Python, entre outras.

Essas instruções são interpretadas e executadas por um computador.



Características básicas do algoritmo

Todo algoritmo, por padrão, tem de apresentar algumas características básicas.

Vamos conferir quais são?



Ponto inicial e Ponto final

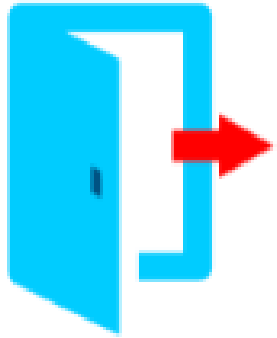
Todo algoritmo deve ter um ponto inicial e chegar a um ponto final.



Não ser ambíguo

A leitura de um algoritmo tem de ser clara, não pode ter dupla interpretação.





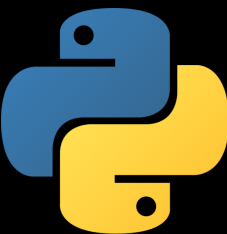
Tratar dados externos

O algoritmo tem de receber dados externos e ser capaz de retornar resultados (saída de dados).



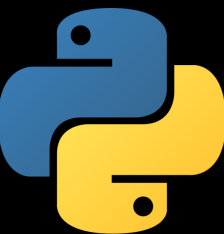
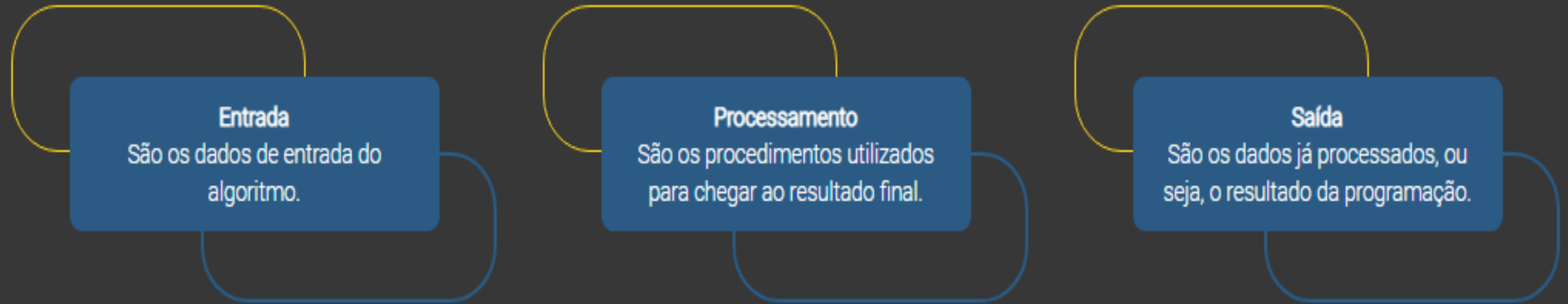
Etapas alcançáveis

O algoritmo deve ter suas etapas alcançáveis em algum momento da programação.



Composição de um algoritmo

Ao iniciar um algoritmo, é necessário dividir a situação-problema em três fases fundamentais:



Formas de representação

Podemos representar algoritmos estruturados de três formas:

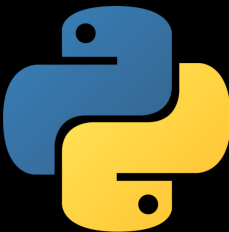
Descrição Narrativa



Fluxograma



Pseudocódigo



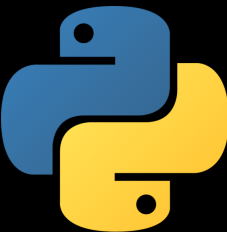
Forma de representação utilizada para descrever um algoritmo de forma que o receptor da informação entenda e interprete o assunto, mesmo não conhecendo sobre algoritmos.

Forma de representação que utiliza símbolos gráficos para representar os algoritmos.

Existem símbolos padronizados para cada tipo de instrução, como início, entrada de dados, processamento, entre outros.

Também conhecido como Portugol ou Português Estruturado, é a principal porta de entrada para a linguagem de programação.

Esse formato consiste na definição de uma pseudolinguagem de programação, cujos comandos são em português, para representar algoritmos.



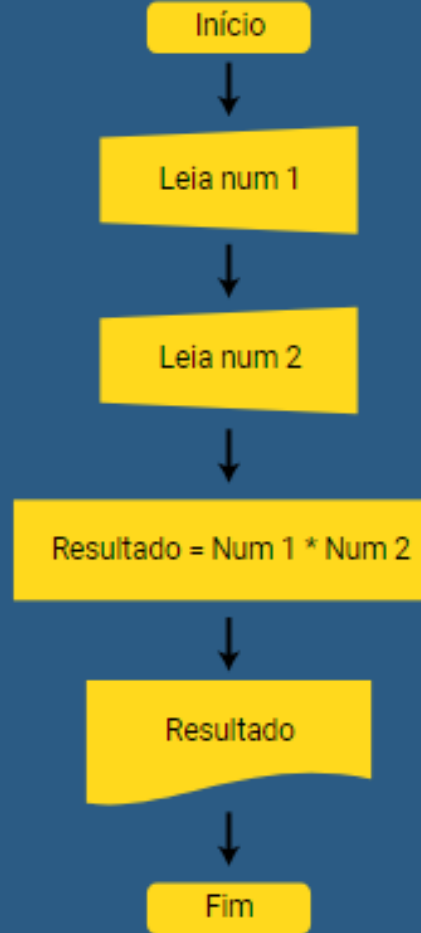
Descrição Narrativa

Passo 1: Receber os números que serão multiplicados;

Passo 2: Multiplicar os números;

Passo 3: Exibir o resultado da multiplicação.

Fluxograma



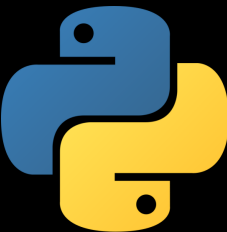
Pseudocódigo

Declare num1, num2, resultado;

Leia num1, num2;

Resultado = num1+num2;

Escreva Resultado.



Variáveis e Constantes

Quando iniciamos a programação temos de nos preocupar onde vamos armazenar as informações.

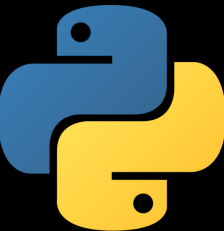
Para isso, temos as **variáveis e constantes**. Vamos conhecer um pouco mais sobre elas?

>>Variáveis

Variável é um recurso utilizado na programação para armazenar e recuperar dados, ou seja, é um espaço que reservamos na memória atribuindo um nome e organizando os dados manipulados no programa.

Por exemplo, podemos criar uma variável chamada "nome", que armazenará o nome de uma pessoa. Fazendo uma analogia, seria como se você criasse uma gaveta em seu escritório com várias divisórias. A gaveta em si, seria a memória e cada divisão, uma variável para armazenar algum objeto.

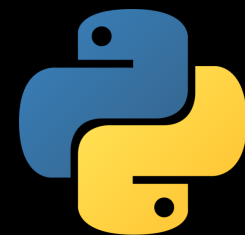
Chamamos o espaço alocado na memória de variável, porque o valor armazenado neste espaço pode ser alterado ao longo do tempo, ou seja, o valor ali alocado é "variável" ao longo da execução do programa.



>>Constantes

Uma constante é responsável por armazenar um valor **fixo** em um espaço da memória. Esse valor **não se altera** durante a execução do programa.

Um exemplo clássico é o valor de **PI**. Suponha que você precise trabalhar com o número **PI**, que é um valor fixo de, aproximadamente, 3,14. Você pode simplesmente declará-lo e utilizá-lo em todo o seu programa.



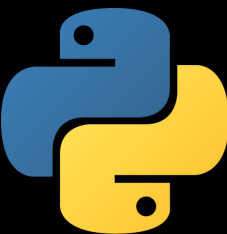
Tipos de dados



Ao criarmos uma **variável** em um programa temos de especificar o tipo de dado que será armazenado. Por exemplo, uma variável que vai **armazenar** o nome de uma pessoa será do tipo **"caracter"**, pois armazenará letras. Uma variável que armazenará a idade de uma pessoa, poderá receber somente "números inteiros", portanto, deverá ser declarada como sendo do tipo **"inteiro"**.

Já uma **variável "valor"**, como está representando um espaço para armazenar número com casas decimais, deverá ser declarada como sendo do tipo **"real"**.

Os tipos de dados influenciam na forma como o programa irá trabalhar, no desempenho do programa e no seu consumo de memória.



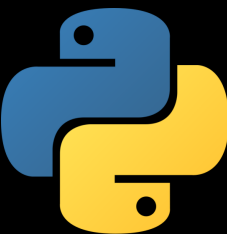
CARACTER	Qualquer conjunto de caracteres alfanuméricos. Exemplo: "Fabio", "São Paulo", "VZ32", "123"...
INTEIRO	Qualquer número inteiro, negativo, nulo ou positivo. Exemplo: -21, 0, 10, 5025...
REAL	Qualquer número real no formato decimal, negativo, nulo ou positivo. Exemplo: -0.5, 0, 5, 9.5...
LÓGICO	Conjunto de valores (FALSO ou VERDADEIRO).



Tipos de Variáveis

A declaração do tipo da variável vai depender da linguagem de programação que está sendo utilizada. Por exemplo, na Linguagem C, C# e Java deve ser especificado o tipo de dados. Já em PHP, ASP e JavaScript não é necessário.

Cada tipo de linguagem também tem seus tipos e formatos de declaração próprios.

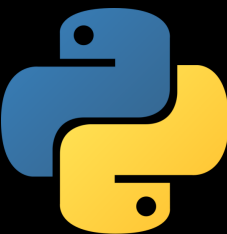
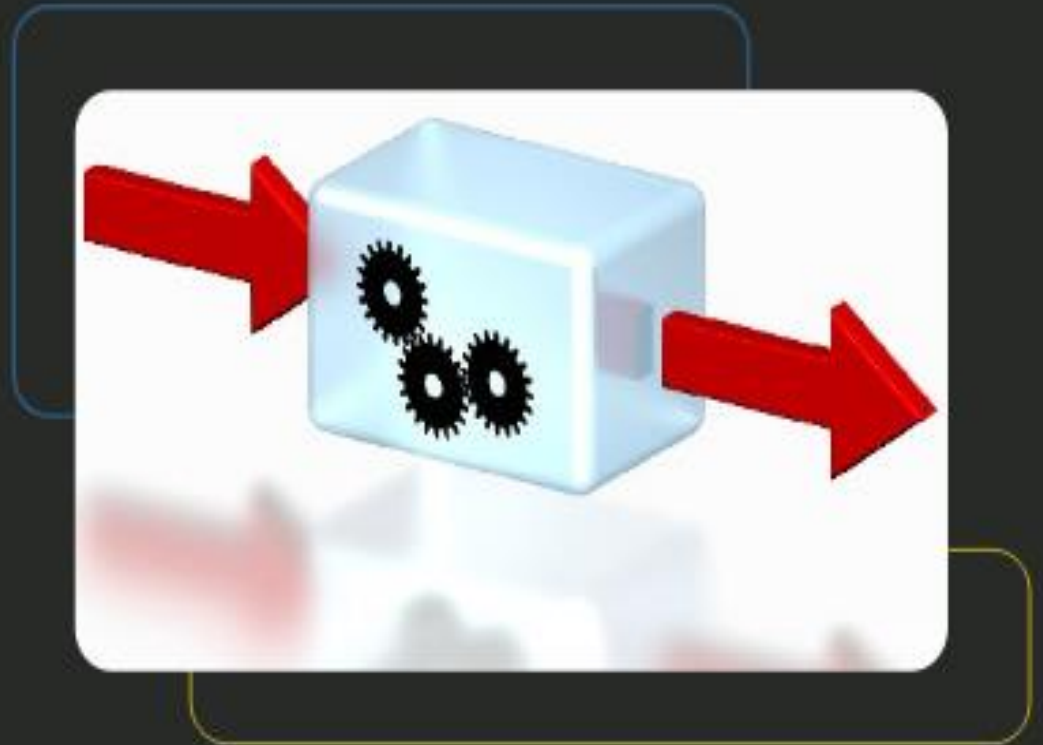


Comandos de entrada e saída

Na maioria das vezes, os sistemas são construídos a partir de dados adquiridos por meio da interação humana e os resultados também devem ser apresentados.

Para interação, as linguagens de programação fornecem comandos para inserção e visualização de informações.

Os **comandos de entrada e saída** são os que permitem a interação com o usuário por meio dos **dispositivos de saída**. Exemplo: teclado, mouse, leitor, entre outros.



Algoritmos – Comandos de entrada e saída

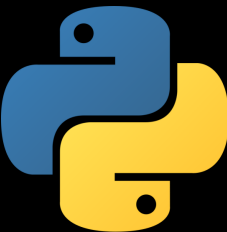
Assim como na declaração de variáveis, cada linguagem de programação tem seus comandos próprios para entrada e saída de dados. Na descrição de algoritmo utilizamos os seguintes comandos:

Clique nos cartões para acessar o conteúdo.

ENTRADA



SAÍDA



Algoritmos – Comandos de entrada e saída

Assim como na declaração de variáveis, cada linguagem de programação tem seus comandos próprios para entrada e saída de dados. Na descrição de algoritmo utilizamos os seguintes comandos:

ENTRADA

O comando LEIA é utilizado para que o usuário informe um valor a ser atribuído em uma variável do sistema.

SAÍDA

Para escrita de dados ou mensagens utilizamos o comando ESCRIVA, para escrita em uma linha e ESCRIVAL, para a escrita com quebra de linha ao final.

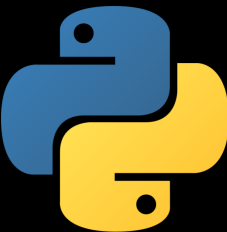


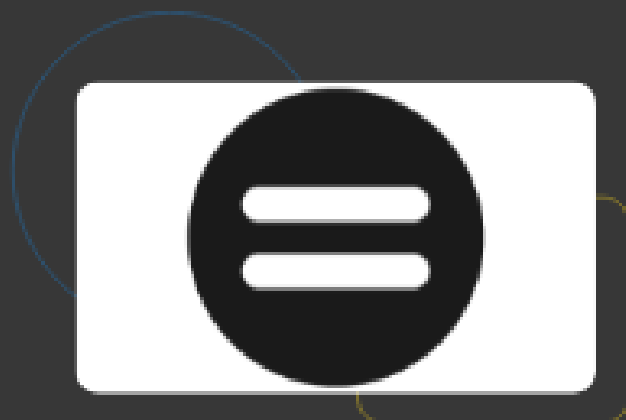
Operadores

Os operadores são meios de **incrementar, decrementar, comparar e avaliar dados** durante a execução do programa.

Por padrão, temos quatro tipos de operadores:

- | Operador de Atribuição
- | Operadores Aritméticos
- | Operadores Relacionais
- | Operadores Lógicos





Operador de Atribuição

O **operador de atribuição** em algoritmos é representado pela seta \leftarrow , indicando que algum valor será atribuído em alguma variável. Exemplo: multiplica $\leftarrow 2 * 2$.

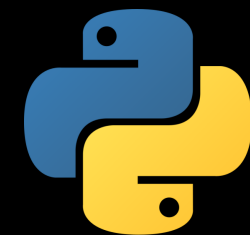
Na maioria das linguagens de programação o símbolo de atribuição é o operador igual (=).

Operadores Aritméticos

Os **operadores aritméticos** permitem a realização de operações matemáticas com dados do tipo numérico.

Além dos símbolos já conhecidos para adição, subtração, multiplicação e divisão, podemos utilizar, também, alguns outros símbolos, como operadores.

Os símbolos para os operadores aritméticos são:



OPERAÇÃO	SÍMBOLO
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**



Hierarquia das Operações Aritméticas

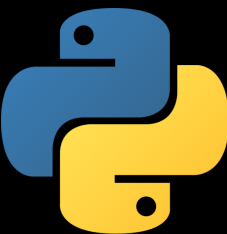
Ao realizar as operações matemáticas temos uma **sequência das operações a serem realizadas**, que são:

Cálculos que estão entre parênteses: ()

Exponenciação: * *

Multiplicação, divisão (o que estiver primeiro): * /

Adição ou Subtração (o que estiver primeiro): + ou -

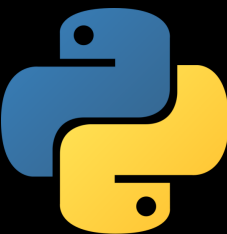


Operadores Relacionais

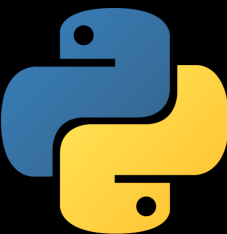
Os **operadores relacionais** são utilizados para comparar caracteres, números, variáveis e expressões.

Estes operadores sempre retornam valores lógicos (verdadeiro/falso, *true/false* ou 0/1), podendo variar a simbologia, dependendo da linguagem de programação utilizada.

Os operadores relacionais utilizados nos algoritmos são:



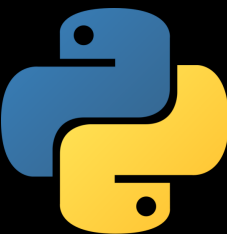
OPERAÇÃO	SÍMBOLO
Comparação	=
Diferente de	<>
Maior que	>
Menor que	<
Maior ou igual	>=
Menor ou igual	<=



Operadores lógicos

Os **operadores lógicos** servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso.

E	AND
OU	OR
NÃO	NOT



E/AND

OU/OR

NÃO/NOT

O resultado de uma expressão lógica usando o operador lógico AND é verdadeira, **somente se todas as condições forem verdadeiras.**

EXEMPLO

$(3 > 5) \text{ AND } (8 > 1) = \text{FALSO}$

$(A > B) \text{ AND } (A > C) = \text{VERDADEIRO}$

E/AND

OU/OR

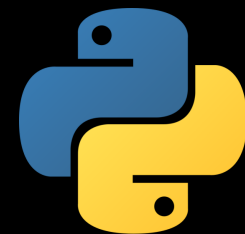
NÃO/NOT

O resultado de uma expressão lógica usando o operador lógico OR é falsa, **somente se todas as condições forem falsas.**

EXEMPLO

$(3 > 5) \text{ OR } (8 > 1) = \text{VERDADEIRO}$

$(A > B) \text{ OR } (A > C) = \text{VERDADEIRO}$



E/AND

OU/OR

NÃO/NOT

O resultado de uma expressão lógica usando o operador lógico NOT, inverte o valor da expressão ou condição, ou seja, **se verdadeira inverte para falsa e vice-versa**.

EXEMPLO

NOT (3 > 5) = VERDADEIRO

NOT (A > B) = FALSO

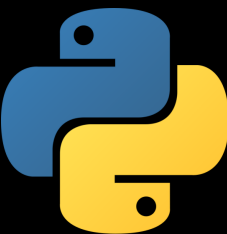


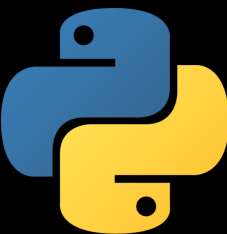
Tabela verdade

A tabela verdade apresenta os valores possíveis gerados pelos três operadores lógicos (E/AND, OU/OR e NÃO/NOT), envolvendo dois valores lógicos:

1º VALOR	OPERADOR	2º VALOR	RESULTADO
VERDADEIRO	AND	VERDADEIRO	VERDADEIRO
VERDADEIRO	AND	FALSO	FALSO
FALSO	AND	VERDADEIRO	FALSO
FALSO	AND	FALSO	FALSO

1º VALOR	OPERADOR	2º VALOR	RESULTADO
VERDADEIRO	OR	VERDADEIRO	VERDADEIRO
VERDADEIRO	OR	FALSO	VERDADEIRO
FALSO	OR	VERDADEIRO	VERDADEIRO
FALSO	OR	FALSO	FALSO

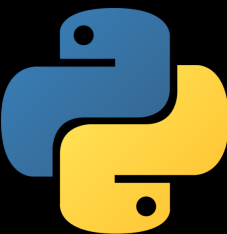
1º VALOR	OPERADOR	2º VALOR	RESULTADO
VERDADEIRO	NOT	—	FALSO
FALSO	NOT	—	VERDADEIRO



Sequência	Descrição
\n	Insere uma quebra de linha.
\t	Insere tabulação horizontal.
\v	Insere tabulação vertical.
\r	Equivalente ao efeito da tecla Enter.
\'	Aspas simples.
\"	Aspas duplas.
\\	Insere uma barra invertida (<i>backslash</i>).
\a	Chamado de ASC bell ou <i>beep</i> do sistema. Se houver suporte, aciona um bipe.
\b	Aciona o <i>backspace</i> , ou seja, apaga o caractere anterior.
\f	Insere uma quebra de página.
\u	Insere um caractere UNICODE. Deve acompanhar um código com 4 números.



Máscara	Tipo de Dado	Descrição
%d ou %i	Int (inteiro)	Exibe um valor inteiro.
%f	Float ou double	Exibe um valor decimal.
%ld	Long Int	Exibe um número inteiro longo.
%e ou %E	Float e double	Exibe um número exponencial (número científico).
%C	Char (caractere)	Exibe um caractere.
%O	Int	Exibe um número inteiro em formato octal.
%x ou %X	Int	Exibe um número inteiro no formato hexadecimal.
%s	Char	Exibe uma cadeia de caracteres (string).
%r	Boolean	Exibe true ou false (verdadeiro ou falso).



Indentação

Uma das características mais importantes da linguagem Python é a...

INDENTAÇÃO

Indentação é uma forma de arrumar o código, fazendo com que algumas linhas fiquem mais à direita que outras, à medida que adicionamos espaços em seu início.

A indentação é uma característica importante no Python, pois além de promover a legibilidade, é essencial para o bom funcionamento do código.

Enquanto na maioria das linguagens, como C, Java e PHP, os blocos de código são delimitados por chaves ({ }) ou comandos, em Python, os blocos são delimitados por espaços ou tabulações, formando uma indentação visual.

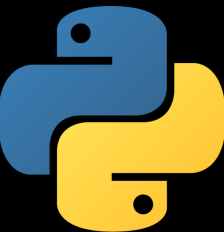
Não existem símbolos de "abre" e "fecha".



Importante

A linguagem Python requer uma **indentação padronizada**.

Em outras linguagens, a indentação é apenas uma boa prática, mas não é obrigatória, por causa da delimitação dos blocos, utilizando comandos ou símbolos.



Exemplo de Indentação

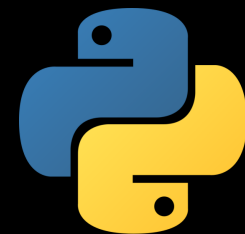
Note que há algumas linhas que ficam mais à direita que outras, à medida que adicionamos espaços em seu início. Isto é o que chamamos de Indentação.

```
idade = int(input("Digite a idade da pessoa: "))  
  
if idade >= 18:  
    → print("maior idade")  
else:  
    → print("menor idade")
```

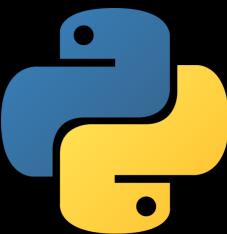
Note que não há espaços no início de cada linha. Neste exemplo não há Indentação.

```
idade = int (input ("Digite a idade da pessoa: "))  
if idade >= 18:  
print("maior idade")  
else:  
print("menor idade")
```

Caso esteja utilizando uma IDE específica para Python, a indentação é feita, de forma automática, ao pressionar a tecla **Enter**. Por padrão, são usados quatro espaços em branco para definir a indentação.



Graphic character symbol		Hexadecimal character value									
0020	0	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
! 0021	1	A 0041	Q 0051	a 0061	q 0071	i 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2	B 0042	R 0052	b 0062	r 0072	ç 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
\$ 0024	4	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	:	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	;	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	<	L 004C	\ 005C	l 006C	007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	=	M 004D] 005D	m 006D	}	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	>	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	?	O 004F	_ 005F	o 006F		¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF



```

print("hello", "world", sep="--")
print("hello", "world", end="\n")
print("hello", "world", sep="--", end="\t")
print("hello", "world", end="\v")
print("hello", "world", end='\n')
print("hello", "world", end='\\')
print("hello", "world", end='\a')
print("hello", "world", end='\b')
print("hello", "world", end='\f')
print("hello", "world", end='\u000F')






```

+

```

>>> print("batata")
batata
>>> print("batata", "feijao", sep="--")
batata--feijao
>>> print("batata", "feijao", sep="vai a bosta")
batatavai a bostafeijao
>>> print("batata", "feijao", sep="----")
batata----feijao
>>> print("hello", "world", sep="--")
hello--world
>>> print("hello", "world", end="\n")
hello world
>>> print("hello", "world", sep="--", end="\t")
hello--world
>>> print("hello", "world", end="\n")
hello world
>>> print("hello", "world", end="\u000F")
hello worldB
>>>

```

 TODO
  Problems
  Terminal
  Python Packages
  Python Console

