

## 8 - LINGUAGEM PYTHON

O Python é uma linguagem de programação de alto nível e de uso geral amplamente usada, que permite aos programadores criar aplicações, páginas da web e muitos outros tipos de software.

Python é frequentemente referido como uma linguagem de script, apesar do facto de que o site oficial diz que é uma linguagem de programação. A verdade, é claro, está em algum lugar no meio. O Python pode ser usado como uma linguagem de script ou uma linguagem de programação!

## 8.1 - RAZÕES PARA APRENDER O PYTHON

O Python é conhecido como uma linguagem de programação de “alto nível”. É uma linguagem flexível, mas poderosa, o Python pode ser usado para:

- Desenvolvimento para a Web (Server Side)
- Desenvolvimento de software
- Desenvolvimento de scripts de sistema

É a linguagem perfeita para o ensino de programação, especialmente no nível introdutório, e é amplamente utilizada em computação científica e numérica. O seu estilo de codificação é muito fácil de entender e muito eficiente e corre em diversas plataformas (Windows, Mac, Linux, Raspberry Pi, etc).

## 8.1 - RAZÕES PARA APRENDER O PYTHON (cont..)

Um dos recursos do Python é a capacidade de interagir com o sistema de ficheiros de um computador. O Python pode criar e gravar dados em ficheiros, ou ler o seu conteúdo. Pode também criar diretórios, copiar ficheiros, excluir, renomear ficheiros ou até mesmo alterar os seus atributos. O Python pode executar quase todas as tarefas relacionadas ao sistema de ficheiros, tornando-o adequado até mesmo para tarefas de administração do sistema. Por exemplo, você pode escrever um programa em Python para fazer backup dos seus ficheiros ou um programa que processa ficheiros de texto reformatando seu conteúdo.

Além disso, o Python pode executar comandos do sistema ou qualquer outro programa instalado no seu computador. Assim, programas escritos e compilados em C, C ++, Java ou qualquer outra linguagem de computador podem ser executados em Python e o Python pode fazer uso do seu output.

Isso permite-lhe adicionar funcionalidades aos seus programas Python sem perder tempo reescrevendo os antigos. Existem milhões - provavelmente até bilhões - de linhas de código já escritas em Python e suas possibilidades de reutilização de código são enormes! É por isso que muitas pessoas preferem usar Python a qualquer outra linguagem de programação. Essa também é uma boa razão pela qual você realmente deveria aprender Python!

## 8.2 - COMO FUNCIONA O PYTHON

Os computadores não entendem as linguagens naturais, como inglês ou o português, por isso precisa de uma linguagem de programação como Python para comunicar com o seu computador.

O Python é uma linguagem de computador de alto nível poderosa. O interpretador do Python (ou, na verdade, uma combinação de um compilador e um interpretador) converte a linguagem Python em uma linguagem que os computadores podem realmente entender e que é conhecida como "linguagem de máquina".

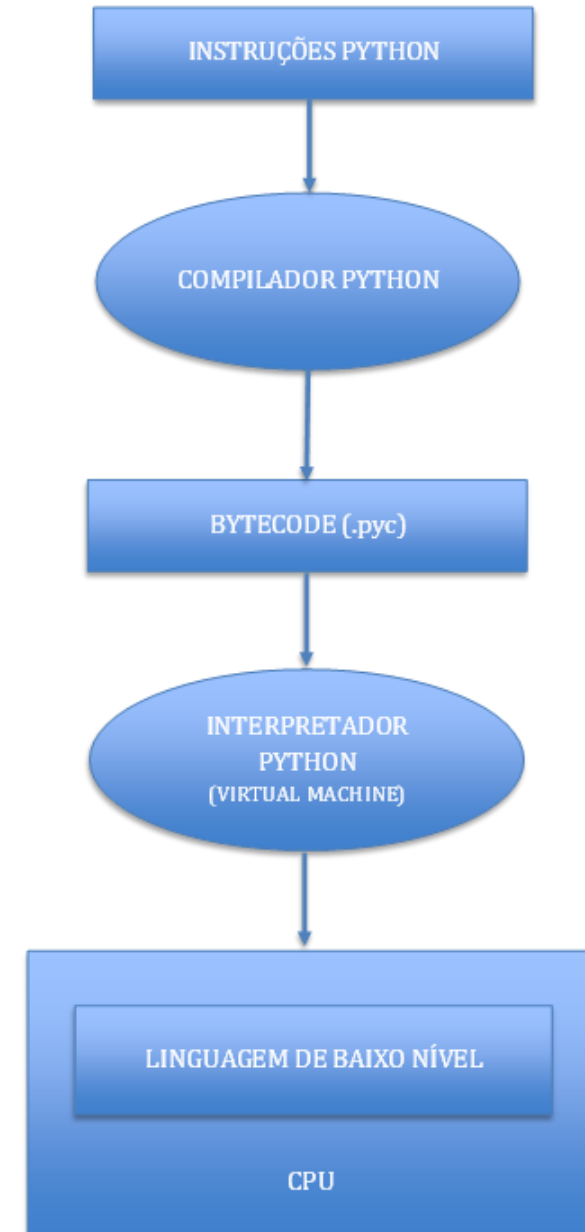
No passado, as linguagens de computador utilizavam um intérprete ou um compilador. Hoje em dia, no entanto, muitas linguagens de computador, incluindo o Python, usam um compilador e um interpretador.

O compilador do Python traduz as instruções Python em instruções bytecode e as salva em um arquivo .pyc e, em seguida, o arquivo .pyc é executado por um interpretador apropriado.

O interpretador é frequentemente chamado de “Python Virtual Machine” (Python VM) e sua missão é converter o bytecode em código de linguagem de máquina de baixo nível para execução direta no hardware.

Na figura abaixo pode observar como as instruções escritas em Python são compiladas em “bytecode” e como o “bytecode” é executado através da sua interpretação.

## 8.2 - COMO FUNCIONA O PYTHON (cont...)



## 8.2 - COMO FUNCIONA O PYTHON (cont..)

Porque razão o Python é traduzido duas vezes? Por que as instruções do Python não são traduzidas diretamente para código máquina? A resposta está no facto de que é tudo uma questão de eficiência.

Hoje em dia, poucos interpretadores realmente interpretam o código diretamente, linha por linha. Quase todos os interpretadores usam algum tipo de representação intermediária por dois motivos:

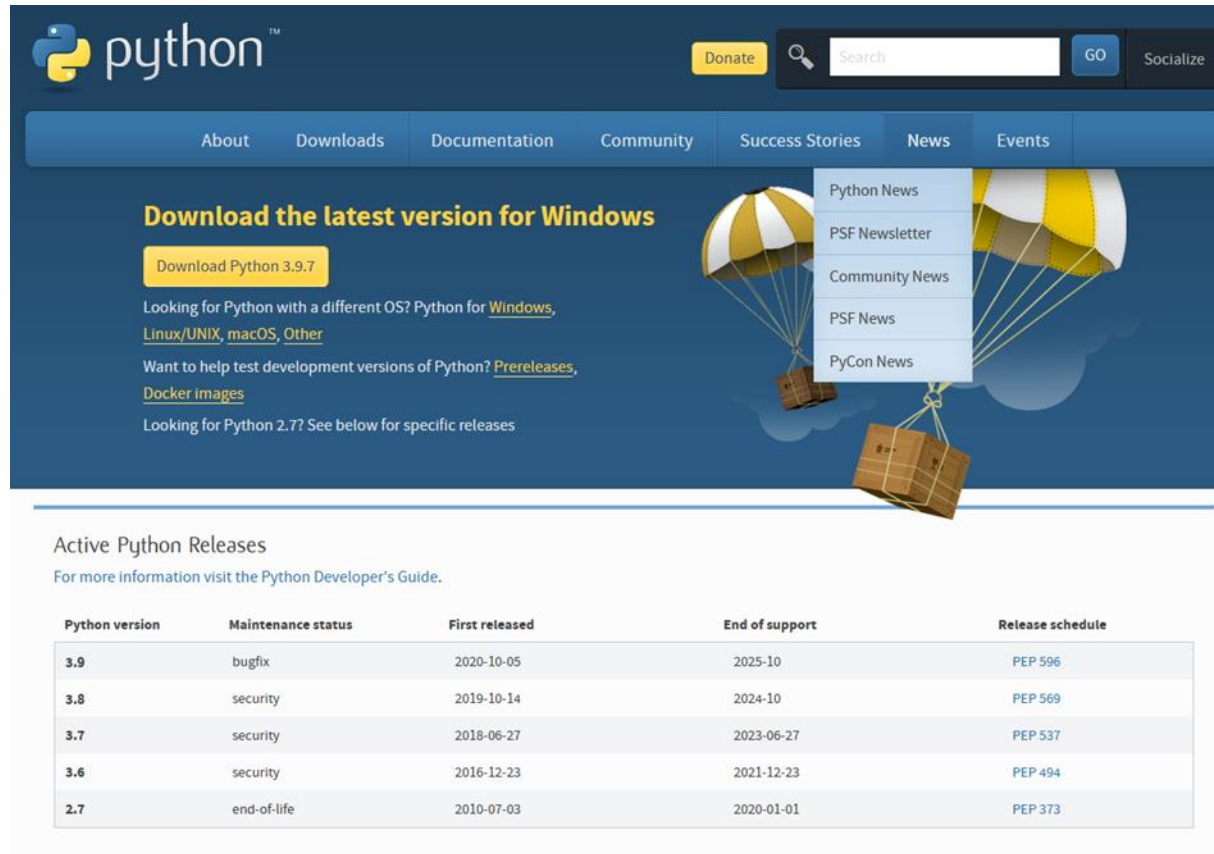
1. Pequenas otimizações podem ser realizadas no código intermediário (bytecode).
2. Se um arquivo .pyc com o mesmo nome do arquivo .py que você invoca estiver presente, o Python executa-o automaticamente. Isso significa que se um arquivo .pyc estiver presente e você não fez nenhuma alteração no código-fonte, o Python pode economizar algum tempo por não ter que recompilar o código-fonte.

### 8.3 - A SINTAXE DO PYTHON COMPARADA COM OUTRAS LINGUAGENS

O Python foi desenhado tendo a legibilidade em mente e apresenta algumas similaridade com o inglês e influências da matemática. O Python utiliza a mudança de linhas “new lines” para indicar o fim de um comando ou instrução, o que difere da maioria das linguagens, que normalmente usam o ponto e vírgula (“;”) e parêntesis. Uma das características do Python é a utilização de espaços em branco para a indentação e para definir o escopo, como por exemplo o escopo dos ciclos, funções e classes. Outras linguagens utilizam chavetas para esse fim.

## 8.4 - INSTALAR O PYTHON

Para instalar o Python, deve descarrega-lo no seguinte endereço: <https://www.python.org/downloads/>



The screenshot shows the Python.org website. The header includes the Python logo, a 'Donate' button, a search bar, and a 'Socialize' button. The navigation menu has links for 'About', 'Downloads', 'Documentation', 'Community', 'Success Stories', 'News', and 'Events'. The main content area features a large banner for downloading the latest version for Windows, with a button for 'Download Python 3.9.7'. Below this, there are links for other operating systems and development versions. A dropdown menu is open over the 'News' link, showing options like 'Python News', 'PSF Newsletter', 'Community News', 'PSF News', and 'PyCon News'. Below the banner, there is a section for 'Active Python Releases' with a table of versions and their support status.

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
3.6	security	2016-12-23	2021-12-23	PEP 494
2.7	end-of-life	2010-07-03	2020-01-01	PEP 373



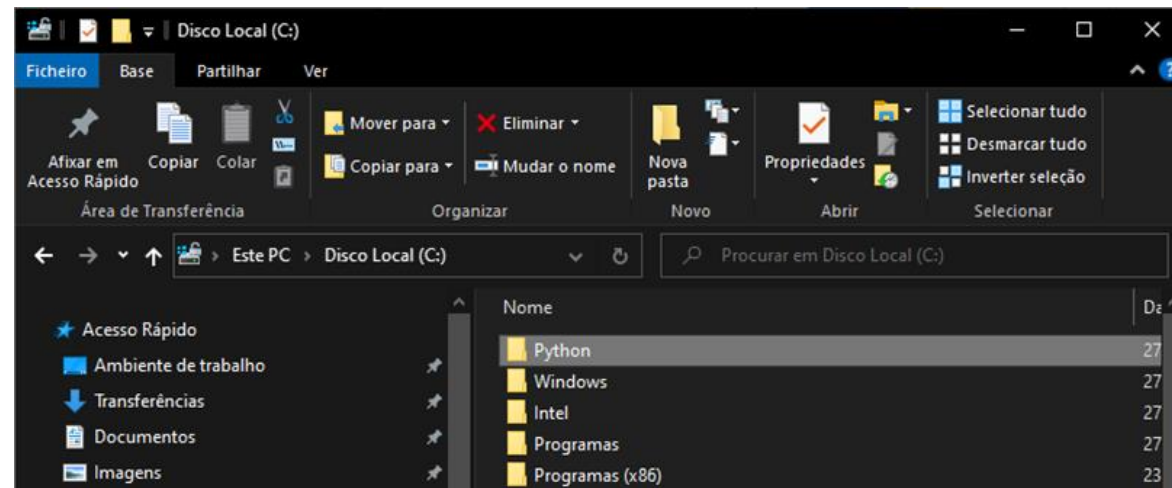
## 8.4 - INSTALAR O PYTHON (cont..)

Escolha a versão adequada para sua plataforma e descarregue a versão mais recente. Este manual mostra o processo de instalação do Python numa plataforma Windows. Quando o download for concluído, execute a instalação.



## 8.5 - CRIAR UMA PASTA PARA OS NOSSOS SCRIPTS

Vamos necessitar de criar uma nova pasta no nosso computador para gravarmos os nossos scripts. Recomendo criar a pasta `c:\python`, embora possa criar a mesma no local que mais lhe convier.



Note que vamos usar a linha de comandos (terminal) para executar os nossos scripts. Se não está familiarizado com a linha de comandos, esta vai ser uma boa altura para isso acontecer. Para lançar uma sessão de terminal, pode executar o programa `CMD.EXE` (ver imagem abaixo).

## 8.5 - CRIAR UMA PASTA PARA OS NOSSOS SCRIPTS

Vamos necessitar de criar uma nova pasta no nosso computador para gravarmos os nossos scripts. Recomendo criar a pasta `c:\python`, embora possa criar a mesma no local que mais lhe convier (imagem 1).

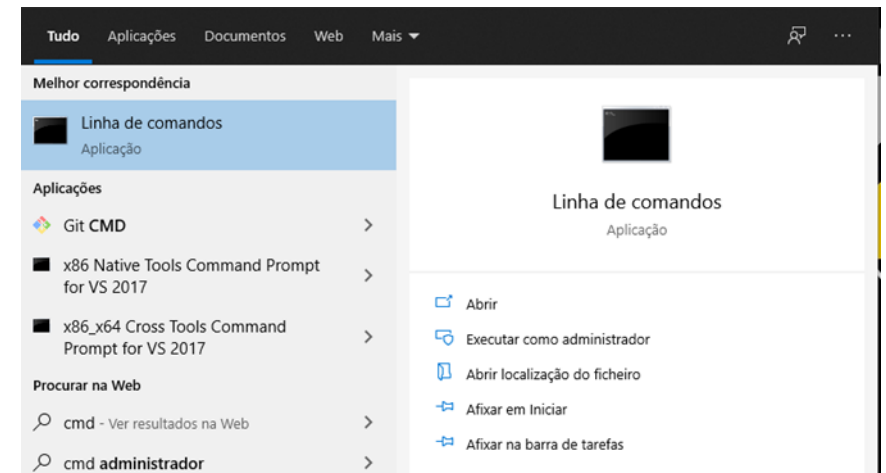
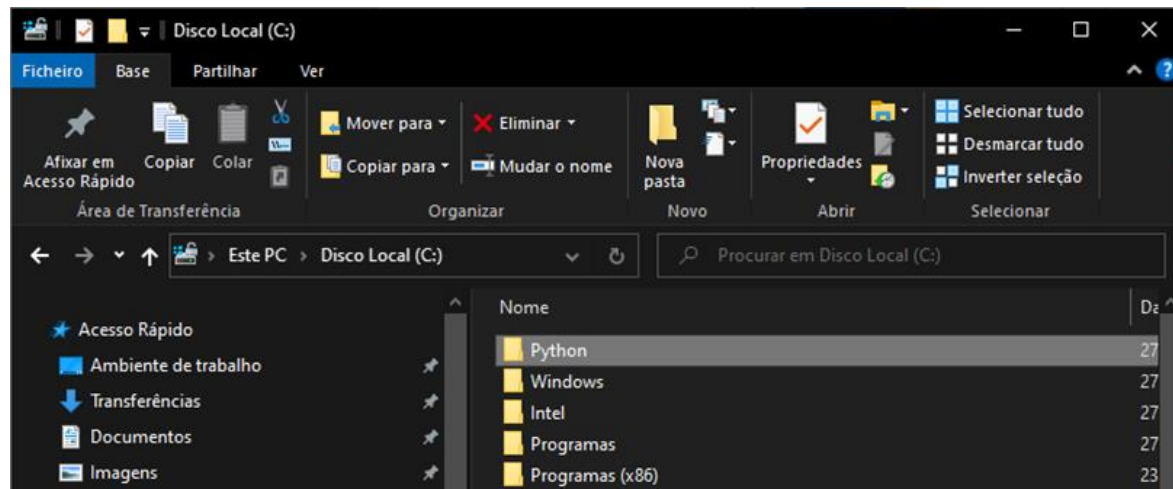
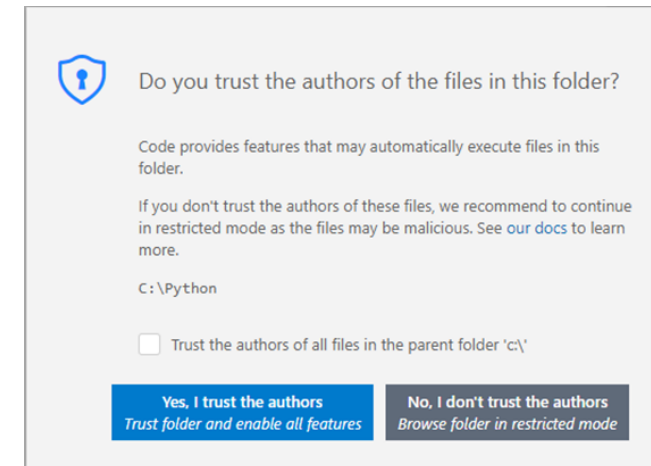
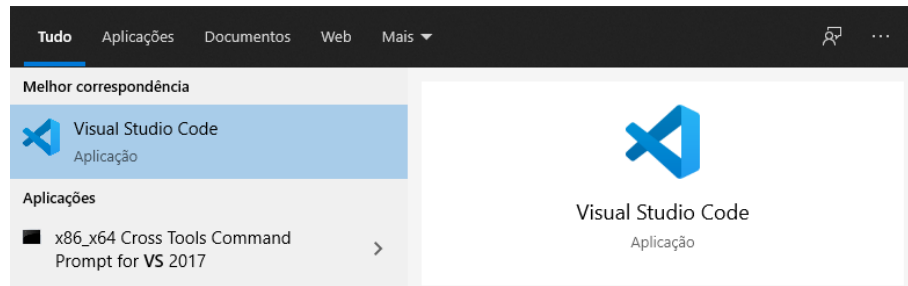


Imagem 2

Note que podemos usar a linha de comandos (terminal) para executar os nossos scripts. Se não está familiarizado com a linha de comandos, esta vai ser uma boa altura para isso acontecer. Para lançar uma sessão de terminal, pode executar o programa `CMD.EXE` (imagem 2)

## 8.5 – Executar os scripts

Outra opção será executar os nossos scripts diretamente na janela do terminal do nosso IDE (Visual Studio Code). Para iniciar o VS Code, deve selecionar o mesmo no menu do seu Windows e de seguida selecionar File -> Open Folder e escolher a pasta c:\Python recentemente criada.



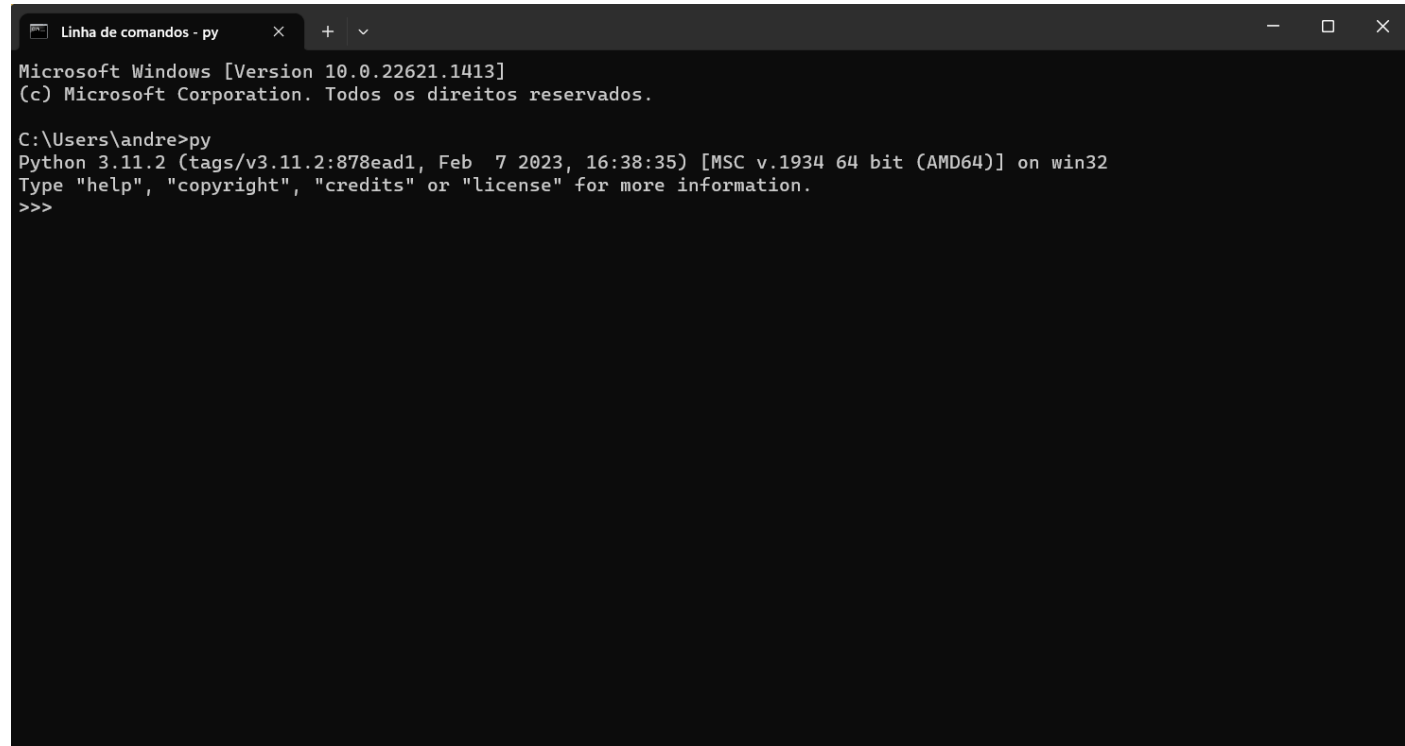
Nota: Pode receber uma mensagem de segurança com a mostrada abaixo. Caso isso aconteça, escolha a opção “Yes, I trust the authors”.

## 8.6 - TESTAR A INSTALAÇÃO DO PYTHON

Após a instalação do Python, é importante testarmos se está a funcionar corretamente. Como vamos trabalhar essencialmente com a linha de comandos, podemos executar o comando abaixo para verificarmos se o Python está instalado e a funcionar corretamente:

`py [enter]`

Se tudo estiver ok, obteremos algo parecido com esta imagem:



```
Linha de comandos - py
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\andre>py
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

## 8.7 – DIFERENÇA ENTRE ERROS DE SINTAXE E ERROS DE LÓGICA

Quando os programadores escrevem código numa linguagem de alto nível, podem cometer dois tipos de erros:

- Erros de sintaxe

Erros de sintaxe são erros como palavras-chave com erros ortográficos, um caracter de pontuação ausente, um parêntesis reto ou parêntese curvo ausente. Felizmente, o VS Code deteta estes erros conforme você digita e sublinha as instruções erradas com uma linha vermelha ondulada. Se tentar executar um programa em Python que inclui erros de sintaxe, receberá mensagens de erro na ecã e o programa não será executado. Deverá por issi corrigir todos os erros e, em seguida, tentar executar o programa novamente.

## 8.7 – DIFERENÇA ENTRE ERROS DE SINTAXE E ERROS DE LÓGICA (cont...)

- Erros de lógica

Erros lógicos são os erros que impedem seu programa de fazer o que é esperado. Nos erros de lógica, você não recebe nenhum aviso. O seu código pode ser compilado e executado, mas o resultado não será o esperado. Os erros lógicos são os erros mais difíceis de detetar. Será necessário analisar o código-fonte do seu programa para determinar onde está o seu erro. Por exemplo, considere um programa em Python que solicita ao utilizador a inserção de três números e, em seguida, calcula e exibe seu valor médio. O programador, entretanto, cometeu um erro tipográfico; no cálculo da média, o programador divide a soma dos três números por 5, e não por 3, como deveria ser. É claro que o programa é executado normalmente, sem nenhuma mensagem de erro, solicitando ao usuário a inserção de três números e exibindo um resultado, mas obviamente não será o correto! É o programador que deve encontrar e corrigir a instrução Python escrita erradamente, porque o computador ou o compilador não serão capazes de descobrir o erro.

## 8.8 - COMO COMENTAR CÓDIGO EM PYTHON

Já alguma vez necessitou de analisar o código escrito por outro programador? E o seu próprio código, já o revisitou mais tarde? Teve dificuldade em perceber cada função, cada bloco de código?

Os comentários servem para ajudar a resolver esses problemas. Os comentários são anotações colocadas pelos programadores que servem para ajudar na compreensão do código ou para colocar qualquer informação relevante, que não seja para interpretação do compilador ou interpretador (por exemplo o nome do programador, a data da criação do programa, etc.). Os comentários podem também servir para desativar a execução de parte do código, o que se torna bastante útil quando testamos os nossos scripts.

No Python, os comentários podem ser precedidos pelo carater “#” (cardinal/hash tag).



## 8.8 - COMO COMENTAR CÓDIGO EM PYTHON (cont...)

Exemplos:

```
#Isto é um comentário numa linha|  
print("Hello, World!")
```

```
print("Hello, World!") #exemplo de comentário|
```

No último exemplo, o comentário foi colocado na mesma linha do código. O Python vai executar o código e ignorar o resto por se um comentário.

Abaixo um exemplo de comentários multilinha:

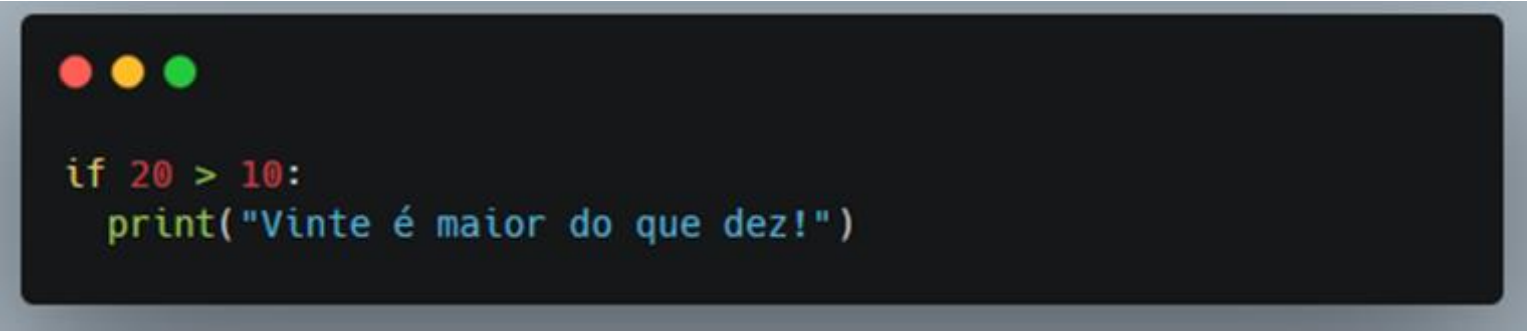
```
#este comentário  
#foi escrito  
#em várias linhas|  
print("Hello, World!")
```

## 8.9 - INDENTAÇÃO

A indentação está relacionada com o espaço em branco no início de cada linha de código. Como referido anteriormente, o Python utiliza espaços em branco para a indentação do código. Vejamos alguns exemplos de indentação.

Nota: Para efeitos de demonstração vamos introduzir algumas instruções e estruturas neste capítulo. Não se preocupe, elas serão devidamente explicadas mais para a frente neste manual.


### 8.9.1 – INDENTAÇÃO CORRETA



```
if 20 > 10:  
    print("Vinte é maior do que dez!")
```

Prática: crie um novo ficheiro no seu IDE chamado “identacao\_correta.py”, execute o código e observe o resultado.

## 8.11 – INDENTAÇÃO INCORRETA



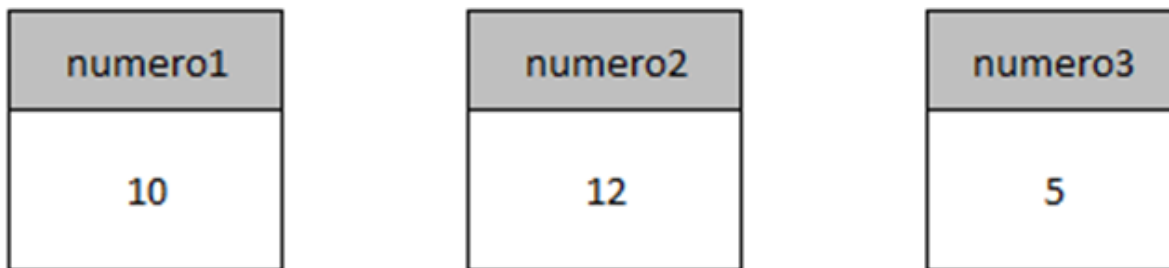
```
if 20 > 10:  
print("Vinte é maior do que dez!")
```

Prática: crie um novo ficheiro no seu IDE chamado “identacao\_incorreta.py”, execute o código e observe o resultado.

## 8.12 - VARIÁVEIS

Em computação, uma variável é um local na memória principal do computador (RAM) onde você pode armazenar um valor e alterá-lo à medida que o programa é executado.

Imagine uma variável como uma caixa transparente na qual você pode inserir e armazenar informação. Como a caixa é transparente, você também pode examinar o que ela contém. Além disso, se você tiver duas ou mais caixas, poderá dar a cada caixa um nome exclusivo. Por exemplo, você pode ter três caixas, cada uma contendo um número diferente, e pode nomear as caixas como `numero1`, `numero2` e `numero3`.



As caixas denominadas `numero1`, `numero2` e `numero3` no exemplo contêm os números 10, 12 e 5, respectivamente. Claro, você pode examinar ou mesmo alterar o valor contido em cada uma dessas caixas a qualquer momento.

## 8.12 – VARIÁVEIS (cont...)

Num fluxograma ou em pseudocódigo, a ação de armazenar um valor numa variável é representado por uma seta para a esquerda

$$\text{numero3} \leftarrow \text{numero1} + \text{numero2}$$

Esta ação é geralmente expressa como “Atribuir um valor, ou o resultado de uma expressão, a uma variável.” A seta para a esquerda é chamada de “operador de atribuição de valor”.

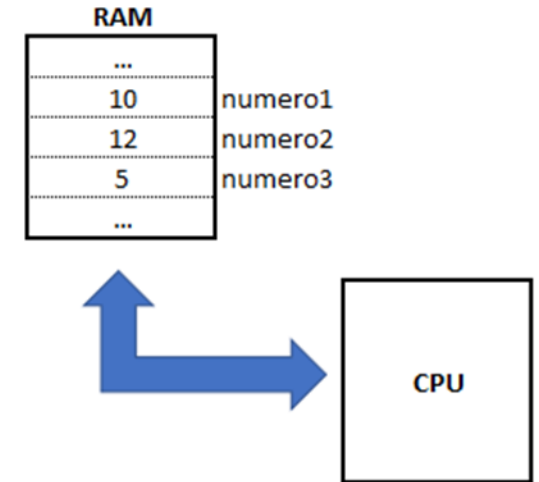
Agora, digamos que alguém lhe pediu para encontrar a soma dos valores das duas primeiras caixas e, em seguida, armazenar o resultado na última caixa.

As etapas que você deve seguir são:

1. Examinar os valores contidos nas duas primeiras caixas.
2. Usar a sua CPU (o seu cérebro) para calcular a soma (o resultado).
3. Inserir o resultado na última caixa. No entanto, como cada caixa pode conter apenas um único valor de cada vez, o valor 5 é substituído pelo número 22.

## 8.12 – VARIÁVEIS (cont...)

Em computação, as três caixas são, na verdade, três regiões individuais na memória principal (RAM), denominadas numero1, numero2 e numero3.



Quando o programa pede ao CPU para executar a instrução  $\text{numero1} \leftarrow \text{numero2} + \text{numero3}$ , o CPU executa os 3 passos referidos anteriormente.

1. O número 10 é transferido da região da RAM chamada numero1 para a CPU e número 12 é transferido da região da RAM chamada numero2 para a CPU.
2. O CPU calcula a soma de  $10 + 12$ .
3. O resultado, 22, é transferido da CPU para a região da RAM chamada numero3, substituindo o número 5 existente.

## 8.12 – VARIÁVEIS (cont...)

Após a execução, a RAM fica assim.

RAM	
...	
10	numero1
12	numero2
22	numero3
...	



## 8.12 – VARIÁVEIS (cont...)

Após a execução, a RAM fica assim.

RAM	
...	
10	numero1
12	numero2
22	numero3
...	

### 8.12.1 - Exemplo da utilização de variáveis:

Passos:

1. Crie um script com o nome “exemplo\_utilizacao\_variaveis.py”.
2. Atribua os valores às variáveis
3. Imprima o valor corrente das variáveis
4. Execute a soma da 1ª e 2ª variável
5. Imprima novamente os valores das variáveis

```
#atribuição de valores às variáveis
numero1 = 10
numero2 = 12
numero3 = 5

#imprimir o valor das variáveis
print("O valor da variável numero1 é: ", numero1)
print("O valor da variável numero2 é: ", numero2)
print("O valor da variável numero3 é: ", numero3)

#soma das variáveis numero1 e numero2
soma = numero1 + numero2

#atribuição do resultado à variável numero3
numero3 = soma

#imprimir o valor das variáveis
print("O valor da variável numero1 é: ", numero1)
print("O valor da variável numero2 é: ", numero2)
print("O valor da variável numero3 é: ", numero3)|
```

### 8.12.1 - Exemplo da utilização de variáveis (cont...)

Da mesma forma que podemos atribuir valores às variáveis, podemos limpar o seu valor atribuindo à variável o valor *None*.

Se quisermos “destruir” a variável, ou seja fazer com que ela deixe de estar referenciada no nosso programa, podemos executar a instrução *del <variável>*.

Se pretender usar a variável mais à frente no seu programa pode ser uma boa ideia usar o *None* em vez da instrução *del*.

Exemplo do *None* e *del*

```
a = "Hello World"
print(a)
a = None
print(a)
del a
print(a)
```

### 8.12.1 - Exemplo da utilização de variáveis (cont...)

Relembrar! Enquanto um programa Python está a ser executado, uma variável pode conter vários valores, mas apenas um valor de cada vez.

Quando você atribui um valor a uma variável, esse valor permanece armazenado até que você atribua um novo valor e quando isso acontece o valor antigo é perdido.

Uma variável é um dos elementos mais importantes em computação porque ajuda a interagir com os dados armazenados na memória principal (RAM).

### 8.13 - REGRAS PARA OS NOMES DAS VARIÁVEIS

Existem regras para definirmos o nome de uma variável. Abaixo pode verificar essas mesmas regras. Note que ao não cumprir estas regras, vai obter um erro e o seu programa não será executado.

- Podem ter um nome curto ou longo, dependendo se pretende classificar uma variável com um nome mais ou menos descritivo. No caso de variáveis multi-nome, devemos utilizar o underscore “\_” para separarmos os nomes
- Devem começar por uma letra ou por um underscore “\_”
- Só podem conter caracteres alfanuméricos ou o underscore
- As variáveis são “case-sensitive”, isto quer dizer que há diferença entre letras maiúsculas e minúsculas (Nif e nif são duas variáveis diferentes)

## 8.13 - REGRAS PARA OS NOMES DAS VARIÁVEIS (cont..)

### 9.13.1 - Exemplos de nomes válidos para variáveis

```
Var1 = "Python"
```

```
var1 = "Phyton"
```

```
_var1 = "Python"
```

```
var_1 = "Python"
```

```
VAR1 = "Python"
```

### 9.13.2 - Exemplos de nomes inválidos para variáveis

```
Var-1 = "Python"
```

```
Var 1 = "Phyton"
```

```
1var = "Python"
```

### 9.13.3 - Exemplo para variáveis descritivas (multi-nome)

```
NOME_ALUNO = "Pedro"
```

```
Data_Nascimento = "14/07/1972"
```

## 8.13 - REGRAS PARA OS NOMES DAS VARIÁVEIS (cont..)

### Técnicas para nomear variáveis

Existem algumas técnicas que ajudam na leitura dos nomes das variáveis, nomeadamente:

Camel Case: nomeDaVariavel

Pascal Case: NomeDaVariavel

Snake Case: nome\_da\_variavel

## 8.14 - CONSTANTES

Por vezes, pode ser necessário usar um valor que não pode ser alterado durante a execução do programa. Esse valor é chamado de "constante".

Em termos simples, pode-se dizer que uma constante é uma variável bloqueada. Isso significa que quando um programa começa a ser executado e um valor é atribuído à constante, nada pode alterar o valor da constante durante a execução do programa. Por exemplo, num programa financeiro, uma taxa de juros pode ser declarada como uma constante.

Um nome descritivo para uma constante também pode melhorar a legibilidade do seu programa e ajudá-lo a evitar erros. Por exemplo, digamos que usa o valor 3,14159265, mas não como uma constante, em muitos locais ao longo do seu programa. Se cometer um erro ao digitar o número, os resultados serão incorretos. Mas, se esse valor receber um nome, qualquer erro tipográfico no nome será detetado pelo compilador e você será notificado com uma mensagem de erro.



## 8.14 – CONSTANTES (cont...)

Atenção: Infelizmente, o Python não oferece suporte para constantes. Podemos usar uma variável no lugar de uma constante, mas temos de ter cuidado para não mudar acidentalmente o seu valor inicial de cada vez que usarmos esta variável no nosso programa.

Podemos representar uma constante utilizando fluxogramas da seguinte forma:

```
CONST PI = 3,14159265
```

## 8.15 - QUANTOS TIPOS DE VARIÁVEIS E CONSTANTES EXISTEM?

Tipos diferentes de variáveis e constantes existem na maioria das linguagens de programação. A razão para essa diversidade são os diferentes tipos de dados que cada variável ou constante pode conter. Na maioria das vezes, as variáveis e constantes contêm os seguintes tipos de dados:

- **Inteiros**

Os valores inteiros são números positivos ou negativos sem nenhuma parte fracionária, como 5, 100, 135, -25 e -5123.

- **Reais**

Os valores reais são números positivos ou negativos que incluem uma parte fracionária, como 5,14, 7,23, 5,0, 3,14 e -23,78976. Os valores reais também são conhecidos como "flutuações".

## 8.15 - QUANTOS TIPOS DE VARIÁVEIS E CONSTANTES EXISTEM?

- **Booleanos**

Variáveis ou constantes booleanas podem conter apenas um de dois valores: True ou False.

- **Caracter**

Os caracteres são valores alfanuméricos (sempre colocados entre aspas simples ou duplas), como "a", "c", "Olá", "Tenho 25 anos" ou "Pedro". Uma sequência de caracteres também é conhecida como “string” .

## 8.16 - DECLARAR VARIÁVEIS?

A declaração é o processo de reservar uma parte da memória principal (RAM) para armazenar o conteúdo de uma variável. Em muitas linguagens de computador de alto nível (incluindo Python), o programador deve escrever uma instrução específica para reservar essa parte na RAM. Na maioria dos casos, eles precisam especificar o tipo de variável para que o compilador ou o interpretador saiba exatamente quanto espaço reservar.

Aqui estão alguns exemplos que mostram como as variáveis são declaradas em diferentes linguagens de computador de alto nível:

Declaração	Linguagem
<code>Dim sum As Integer</code>	Visual Basic
<code>int sum;</code>	C#, C++, Java e outras
<code>sum: Integer;</code>	Pascal, Delphi
<code>var sum;</code>	Javascript

### 8.16.1 - COMO DECLARAR VARIÁVEIS EM PYTHON?

Em Python, não há necessidade de declarar variáveis como em C# ou C++. As variáveis são declaradas quando são usadas pela primeira vez. Por exemplo, a instrução:

```
numero1 = 0
```

declara a variável `numero1` e a inicializa com 0.

Nota: No Python pode representar a ação de atribuir um valor a uma variável usando o sinal de igual (=). Isso é equivalente à seta para a esquerda nos fluxogramas e pseudocódigo.

```
numero1 ← 0
```

### 8.16.2 – CONVERTER VARIÁVEIS

Pode haver alturas em que é necessário especificar um tipo específico numa variável. Isso pode ser feito com os que se chama “casting”.

A conversão em python é, portanto, feita usando as seguintes funções:

- `int ()` - define um número inteiro a partir de um literal inteiro, um literal flutuante (removendo todos os decimais) ou um literal tipo carater (desde que a represente um número inteiro)
- `float ()` - define um número flutuante a partir de um literal inteiro, um literal flutuante ou um literal carater (desde que represente um flutuante ou um inteiro)
- `str ()` - define uma string a partir de uma ampla variedade de tipos de dados, incluindo strings, literais inteiros e literais flutuantes

## 8.16.2 – CONVERTER VARIÁVEIS (cont...)

Alguns exemplos:

Inteiros

```
x = int(10) # x é convertido para o número inteiro 10
y = int(2.8) # y é convertido para o número inteiro 2
z = int("3") # z é convertido para o número inteiro 3
```

Floats

```
x = float(2) # x é convertido para 2.0
y = float(3.5) # y é convertido para 3.5
z = float("5") # z é convertido para 5.0
w = float("5.2") # w é convertido para 5.2
```

Strings

```
x = str("s1") # x é convertido para "s1"
y = str(2) # y é convertido para "2"
z = str(3.0) # z é convertido para "3.0"
```

## 8.17 - CONCATENAÇÃO

O Python, à semelhança de outras linguagens, permite a concatenação de strings, o que é sobretudo útil quando queremos juntar variáveis, texto e tags HTML. Para concatenar usamos o carater “+”.

Prática: crie um novo ficheiro no seu IDE chamado “exemplo\_concatenacao\_variaveis.py”, execute o código e observe o resultado.

Podemos também utilizar o operador de concatenação e atribuição “+=”.



## 8.17 – CONCATENAÇÃO (cont...)

Crie um novo ficheiro no seu IDE chamado “exemplo\_concatenacao\_atribuicao.py”, execute o código e observe o resultado.

```
#exemplo do operador concatenação com atribuição|
a = "Hello" + ", "
a += "World!"
print(a)
```

### 8.17 – CONCATENAÇÃO (cont...)

Crie um script que concatene o seu primeiro e último nome e imprima o resultado no ecrã (concatenar\_primeiro\_ultimo\_nome.py).

```
primeiro_nome = input("Introduza o primeiro nome: ")
ultimo_nome = input("introduza o ultimo nome: ")
nome_completo = primeiro_nome + " " + ultimo_nome
print(nome_completo)
```

### 8.17 – CONCATENAÇÃO (cont...)

Crie um script que concatene o seu primeiro e último nome e imprima o resultado no ecrã (concatenar\_primeiro\_ultimo\_nome.py).

## 8.18 - OPERADORES NO PYTHON

Operadores são as construções, que podem manipular o valor dos operandos. Considere a expressão  $4 + 5 = 9$ . Aqui, 4 e 5 são chamados de operandos e + é chamado de operador.

Tipos de operadores:

- Operadores aritméticos

Assuma duas variáveis a e b, sendo que  $a = 10$  e  $b = 11$

Adição ( + )	$a + b = 21$
Subtração ( - )	$a - b = -1$
Multiplicação ( * )	$a * b = 110$
Divisão ( / )	$a / b = 0,9$
Módulo ( % )	$b \% a = 1$
Expoente ( ** )	$a ** b = 1000000000000$
Divisão inteira ( // )	$a // b = 1$

## 8.18 - OPERADORES NO PYTHON (cont..)

- Operadores de comparação

==	Se os valores dos dois operandos forem iguais, a condição é verdadeira	(a == b) não é verdadeira
!=	Se os valores dos dois operandos não forem iguais, a condição é verdadeira	(a != b) é verdadeira
>	Se o valor do operando esquerdo for maior que o valor do operando direito, a condição é verdadeira	(a > b) não é verdadeira
<	Se o valor do operando esquerdo for menor que o valor do operando direito, a condição é verdadeira	(a < b) é verdadeira
>=	Se o valor do operando esquerdo for maior ou igual ao valor do operando direito, a é verdadeira	(a >= b) não é verdadeira
<=	Se o valor do operando esquerdo for menor ou igual ao valor do operando direito, a condição é verdadeira	(a <= b) é verdadeira

## 8.18 - OPERADORES NO PYTHON (cont..)

- Operadores de atribuição

=	Atribui valores dos operandos do lado direito para o operando do lado esquerdo	$c = a + b$ atribui o valor de $a + b$ a $c$
$+=$ adição AND	Adiciona o operando direito ao operando esquerdo e atribui o resultado ao operando esquerdo	$c += a$ é equivalente a $c = c + a$
$-=$ subtração AND	Subtrai o operando direito do operando esquerdo e atribui o resultado ao operando esquerdo	$c -= a$ é equivalente a $c = c - a$
$*=$ multiplicação AND	Multiplica o operando direito com o operando esquerdo e atribui o resultado ao operando esquerdo	$c *= a$ é equivalente a $c = c * a$
$/=$ divisão AND	Divide o operando esquerdo com o operando direito e atribui o resultado ao operando esquerdo	$c /= a$ é equivalente a $c = c / a$
$\%=$ módulo AND	Aplica o módulo a dois operandos e atribui o resultado ao operando esquerdo	$c \% = a$ é equivalente a $c = c \% a$
$**=$ expoente AND	Executa o cálculo exponencial (potência) nos operandos e atribui valor ao operando esquerdo	$c ** = a$ é equivalente a $c = c ** a$
$//=$ Divisão Inteira AND	Executa a divisão inteira dos operandos e atribui valor ao operando esquerdo	$c //= a$ é equivalente a $c = c // a$

## 8.18 - OPERADORES NO PYTHON (cont..)

- Operadores lógicos

Assuma duas variáveis a e b, sendo que a = True e b = False

and	Se ambos os operandos forem verdadeiros, a condição é verdadeira	(a and b) é <u>falso</u>
or	Se qualquer um dos dois operandos for <u>True</u> , a condição é torna verdadeira.	(a <u>or</u> b) é verdadeiro
not	Usado para reverter o estado lógico do operando.	<u>not</u> (a and b) é verdadeiro

## 8.18 - OPERADORES NO PYTHON (cont..)

- Operadores de associação

in	Avalia como verdadeiro se encontrar uma variável na sequência especificada e como falso caso contrário	x em y, resulta verdadeiro se x for um membro da sequência y
not in	Avalia como verdadeiro se não encontrar uma variável na sequência especificada e como falso caso contrário	x não em y, resulta falso se x não for um membro da sequência y



## 8.18 - OPERADORES NO PYTHON (cont..)

- Operadores de identidade

Assuma duas variáveis  $a$  e  $b$ , sendo que  $a = 5$  e  $b = 5$

<u>is</u>	Verdadeiro se os operandos forem idênticos	$a$ <u>is</u> $b$ retorna verdadeiro
<u>is not</u>	Verdadeiro se os operandos não forem idênticos	$a$ <u>is not</u> $b$ retorna falso

## 8.19 - ESTRUTURAS DE CONTROLO

À semelhança das outras linguagens de programação, no Python é necessário usar estruturas de controlo ou decisão sempre que queremos executar código quando uma determinada condição é satisfeita.

No Python utilizamos a expressão `if...elif...else`. Representado na forma de um fluxograma, temos:

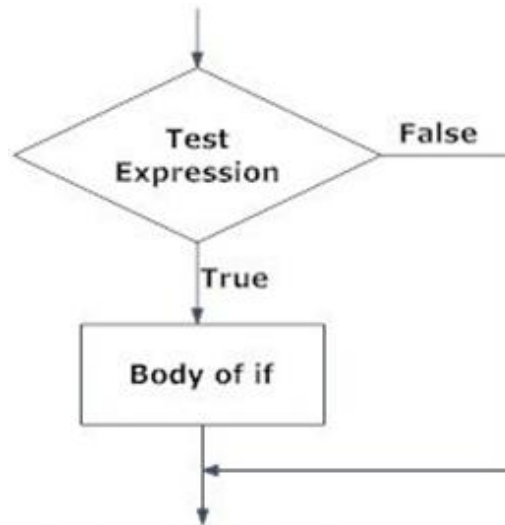


Fig: Operation of if statement

```
# se o número for positivo, imprimimos uma mensagem apropriada

numero = 10
if num > 0:
    print(numero, " é positivo.")

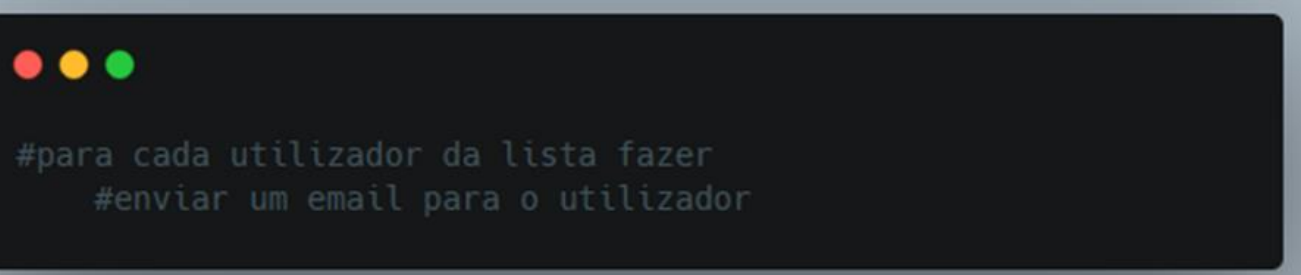
numero = -1
if numero > 0:
    print(numero, " é positivo.")
```

## 8.18 – ESTRUTURAS DE REPETIÇÃO

Por defeito, o fluxo de um programa escrito em Python é sequencial. Por vezes, podemos precisar alterar o fluxo para permitir que um código específico se repita várias vezes.

Para esse propósito, Python fornece vários tipos de estruturas de repetição, que são capazes de repetir blocos de instruções de código. Isto ajuda com tarefas repetitivas e a lidar com grandes quantidades de dados.

Como exemplo, vamos considerar uma tabela com utilizadores numa lista de e-mails. A lista de distribuição contém entradas para um nome e endereço de e-mail. Se quisermos enviar um e-mail para todos, precisamos percorrer todos os utilizadores da tabela. Cada vez que chegamos ao próximo utilizador, enviamos um e-mail.



```
#para cada utilizador da lista fazer  
    #enviar um email para o utilizador
```

## 8.18 – ESTRUTURAS DE REPETIÇÃO (cont...)

O interpretador do Python começa pelo primeiro utilizador e enviará um e-mail para o mesmo. Se houver mais utilizadores na tabela, ele começará novamente do topo do ciclo com o segundo utilizador e enviará para o mesmo também um e-mail.

Isso vai continuar até que tenha percorrido todos os utilizadores da tabela ou se terminarmos manualmente a execução do código.

Python disponibiliza dois tipos de instruções de para ciclos:

- O ciclo “for”
- O ciclo “while”

### 8.18.1 – O CICLO FOR

Usamos um ciclo for para iterar sobre uma coleção ou sequência de itens, como uma string, lista, dicionário, etc. Ao criar um ciclo for, precisamos de duas coisas:

- Algo para fazer um ciclo: uma lista, dicionário, string etc.
- Uma variável temporária que usamos para aceder aos dados.

Vejamos um exemplo de um ciclo for:

```
for x in "hello world":  
    print(x)
```

### 8.18.2 – O CICLO WHILE

Um ciclo while é como uma instrução if. Podemos criar uma condição e se a condição for satisfeita, iniciamos a execução de um bloco de código. O ciclo while continuará a executar o bloco de código indefinidamente até que a condição seja falsa ou o programa seja interrompido.

Vejamos a sintaxe de um ciclo while com o seguinte exemplo:

```
contador = 0

while contador <= 100:
    print(contador)
    contador += 1
```

## 8.19 – LISTAS

O Python não tem suporte para arrays, mas Listas podem ser usadas em seu lugar. As listas são usadas para armazenar vários itens em uma única variável.

Listas são um dos 4 tipos de dados embutidos no Python usados para armazenar coleções de dados (os outros 3 são Tuplos, Conjuntos e Dicionários, todos com diferentes características e utilização).

Os itens da lista são ordenados, alteráveis e permitem valores duplicados. Os itens da lista são indexados, o primeiro item tem o índice [0], o segundo item tem o índice [1], etc.

Quando dizemos que as listas estão ordenadas, significa que os itens têm uma ordem definida e essa ordem não muda. Se adicionar novos itens a uma lista, os novos itens serão colocados no fim da lista. A lista pode ser alterada, o que significa que podemos alterar, adicionar e remover itens de uma lista após sua criação. Como as listas são indexadas, elas podem ter itens com o mesmo valor.

As listas são criadas utilizando parêntesis retos “[ ]”:

## 8.19 – LISTAS (cont...)

Exemplo de implementação de uma lista:


```
disciplinas = ["Matemática", "Português", "Algoritmia"]  
print(disciplinas)
```

Prática: crie um script “exemplo-lista.py” e teste o código acima.



## 8.19 – LISTAS (cont...)

Exemplo de implementação de uma lista com valores duplicados:



```
disciplinas = ["Matemática", "Português", "Algoritmia", "Algoritmia"]  
print(disciplinas)
```

Prática: crie um script “exemplo-lista-duplicados.py” e teste o código acima.

## 8.19 – LISTAS (cont...)

Tamanho da lista:

```
disciplinas = ["Matemática", "Português", "Algoritmia", "Algoritmia"]  
print(len(disciplinas))
```

Prática: crie um script “exemplo-lista-tamanho.py” e teste o código acima.

## 8.19 – LISTAS (cont...)

As listas podem conter qualquer tipo de dado.

```
lista1 = ["Matemática", "Português", "Algoritmia"]
lista2 = [1, 5, 6, 10]
lista3 = [True, True, False]
print(lista1)
print(lista2)
print(lista3)
```

Prática: crie um script “exemplo-lista-tipos-dados.py” e teste o código acima.

## 8.19 – LISTAS (cont...)

A mesma lista pode inclusive conter tipos de dados diferentes.



```
lista = ["Matemática", 1, "Português", True, "Algoritmia", 10.5]  
print(lista)
```

Prática: crie um script “exemplo-tipos-dados-mesma-lista.py” e teste o código acima.

## 8.20 – TUPLES

Um tuplo é constituído por sequência ou conjunto de valores imutáveis. Tuplos são sequências, assim como listas. As diferenças entre tuplos e listas são que os tuplos não podem ser alterados, ao contrário das listas, e os tuplos usam parênteses curvos, enquanto as listas usam parênteses retos.

Criar um tuplo é tão simples quanto colocar diferentes valores separados por vírgula. Opcionalmente, também pode colocar esses valores separados por vírgula entre parênteses.

```
disciplinas = ('algoritmia', 'português', 'matemática', 'programação');  
print(disciplinas)  
numeros = (1, 2, 3, 4, 5, 6, 7, 8, 10 );  
print(numeros)  
letras = "a", "b", "c", "d";  
print(letras|
```

## 8.21 – DICTIONARY

Os “dictionary” são utilizados quando queremos armazenar dados em pares “chave”:"valor". Um dicionário é uma coleção de dados ordenada, mutável e não permite duplicados. Para a sua especificação devemos utilizar chavetas e os dados são especificados utilizando chaves e valores.

```
cidades = {  
    "norte": "Porto",  
    "centro": "Lisboa",  
    "sul": "Faro"  
}  
print(cidades)
```

Exemplo de como iterar num dictionary:

```
for key in cidades:  
    print(key, '->', cidades[key])
```