

CURSO DE PROGRAMAÇÃO

SOUL **CODE** + GRAN
PYTHON



SUMÁRIO

Apresentação	3
Python - Parte I	4
1. Introdução e Conceituação.....	4
2. Sintaxe	10
3. Variáveis	13
4. Tipos de Dados.....	16
5. Expressões, Instruções e Blocos.....	19
Resumo.....	21
Mapa Mental	23
Referências	27

APRESENTAÇÃO

APRESENTAÇÃO DO PROFESSOR

Fala, aluno(a)! Tudo beleza com você?

Sou o professor Rogério Araújo. Sou formado em Bacharelado em Ciência da Computação pela Universidade Estadual do Piauí (UESPI), especialista em Governança em TI pela Unieuro e em Desenvolvimento de Sistemas Baseados em Software Livre pela Universidade da Amazônia (UNAMA). Atualmente, estou cursando o MBA *Data Science e Analytics* pela USP/Esalq. Possuo as certificações *Certified ScrumMaster*, *COBIT 4.1 Foundation Certified* e *Sun Certified Associate for J2SE (SCJA)*. Sou autor de artigos no site www.rogeraoaraujo.com.br e no blog do Gran Cursos Online (<https://blog.grancursosonline.com.br/author/rogerio-araujo/>) e tenho meu canal no Youtube (www.youtube.com/rgildoaraujo). Sou professor de cursos na área de Tecnologia da Informação.

Quanto à minha carreira como servidor público, atualmente trabalho na Secretaria do Tesouro Nacional, vinculada ao Ministério da Economia, exercendo o cargo de Auditor Federal de Finanças e Controle, na área de Governança de TI. Também já passei pelo Tribunal Regional Federal (TRF) 1ª Região, exercendo o cargo de Analista Judiciário, na especialidade de Analista de Sistemas, e pelo Ministério Público Federal (MPF), como Técnico de Informática.

APRESENTAÇÃO DO CURSO

Depois da minha apresentação, vou apresentar o nosso curso. Eu e você, meu (minha) consagrado(a), iremos estudar a linguagem de programação Python, usada em desenvolvimento de aplicações *web*, aplicações *desktop* (baseadas em GUI) e aplicações para dispositivos móveis. E hoje, muito requisitada para Ciência de dados e Inteligência Artificial.

Python, como é uma linguagem de sintaxe mais enxuta (em comparação com linguagens como Java), ela está sendo bem adotada por quem trabalha com análise de dados.

Meu (minha) prezado(a), como iremos aprender a sintaxe do Python? Irei destrinchar a linguagem com teoria, exemplos práticos, esquemas, e mapas mentais.

Teremos uma excelente jornada, meu consagrado. Tentarei passar cada tópico de forma prática, com muitos exemplos.

Então vamos iniciar nossa jornada? Simbora comigo!

Professor Rogerão Araújo

@profRogerãoAraújo

PYTHON – PARTE I

1. INTRODUÇÃO E CONCEITUAÇÃO

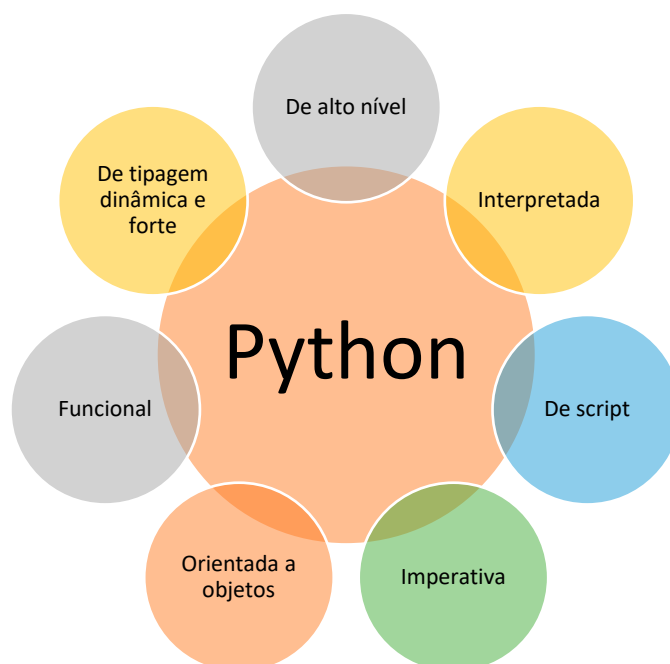
1.1. INTRODUÇÃO E CONCEITUAÇÃO

1.1.1. CONCEITUAÇÃO DA LINGUAGEM PYTHON

Python é uma **linguagem de programação**:

- De **alto nível**;
 - É uma linguagem possui a sintaxe que se aproxima da linguagem humana;
- **Interpretada**;
 - Cada linha do código em Python é lida e executada por um interpretador;
- De **script**;
 - Em um script, por exemplo, código em Python, descreve-se uma sequência de comandos e tarefas que um interpretador deve executar;
- **Imperativa**;
 - É uma linguagem orientada a ações, onde a computação é vista como uma sequência de instruções que manipulam valores de variáveis;
- **Orientada a objetos**;
 - Suporta os conceitos da orientação a objetos;
- **Funcional**;
 - É um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e evita estados ou dados mutáveis;
- De **tipagem**:
 - **Dinâmica**; e
 - **Forte**.

FIGURA 1 | características da linguagem Python.



Irei destacar a seguir os pontos do Python ser de alto nível e interpretada. O restante vamos aprendendo aos poucos.

1.1.2. LINGUAGEM DE ALTO NÍVEL E INTERPRETADA

Para entendermos o que significa uma linguagem de programação ser de alto nível, precisamos fazer uma introdução.

No nosso dia a dia, utilizamos programas de computadores para os mais diversos fins, seja para nos auxiliar em nosso trabalho, seja para usos pessoais.

Um **programa de computador** é um **conjunto de instruções** que:

- Possui um determinado fim;
 - Por exemplo, o Photoshop para edição de imagens; e
- É **executado** por um **processador**.

Mas para haja essa execução pelo processador, o programa precisa estar em uma linguagem que o processador possa entender: a **linguagem de máquina**. Ela é a linguagem que um processador é capaz de compreender e é composta de apenas de números 0 e 1.

O conjunto de instruções que forma um programa é escrito em linguagem de máquina. Dessa forma, o processador reconhecerá o programa e irá executá-lo.

Professor, mas como vamos programar escrevendo apenas 0 e 1?

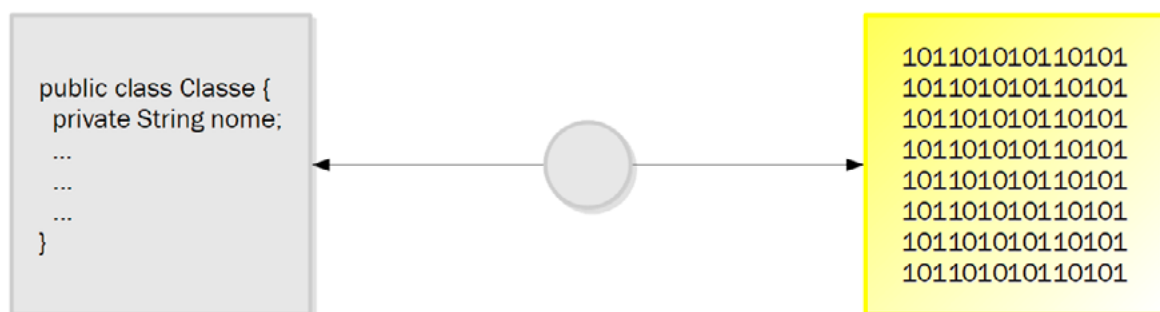
Meu (minha) caro(a) aluno(a), não precisamos ter conhecimento de linguagem de máquina para escrever nossos programas. Para nosso trabalho como desenvolvedores de software, utilizamos linguagens de alto nível, como, por exemplo, a própria linguagem Python.

Programar já não é uma tarefa tão fácil e programar escovando bits, é extremamente complexo e trabalhoso, o que elevaria muito o custo de desenvolvimento e da manutenção de softwares.

Aproveitando, temos que ter em mente que (figura 1.2):

- Quanto mais semelhante uma linguagem for da de máquina:
 - Mais baixo é o nível dessa linguagem; e
 - Menos legível é para o ser humano;
- Quanto mais “distante” uma linguagem for da de máquina:
 - Mais alto é o nível dessa linguagem; e
 - Mais legível é para o ser humano.

FIGURA 2 | **Classificações de linguagens.**



A linguagem Python, por exemplo, é uma linguagem de alto nível, pois possui a sintaxe que se aproxima da linguagem humana. Veremos vários exemplos para mostrar que a sintaxe é tranquila de entendimento.

Professor, se o processador apenas reconhece a linguagem de máquina, como ele irá executar um programa escrito em uma linguagem de alto nível?

Meu(minha) prezado(a), neste momento, entram em ação dois brothers:

- **Compilador;** e
- **Editor de ligação.**

O **compilador** traduz um programa escrito em uma linguagem de alto nível em um **programa-objeto não executável** (também chamado de **módulo-objeto**).

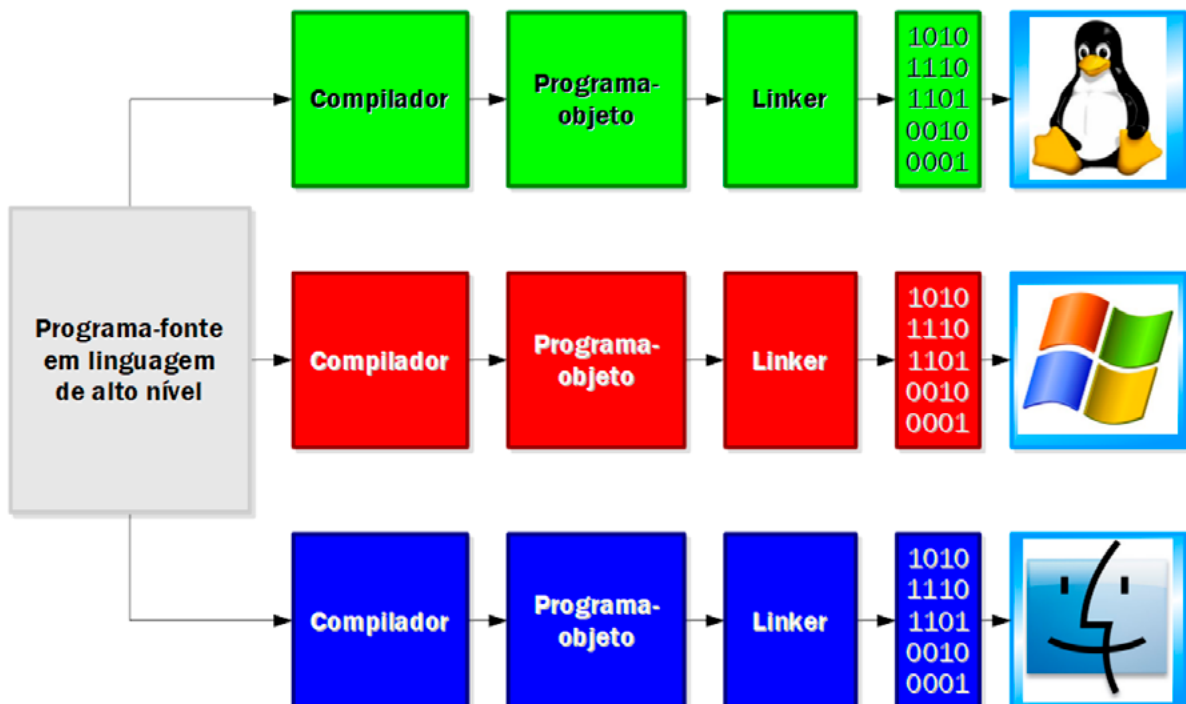
Apesar desse módulo-objeto ser em linguagem de máquina, ele não é executável ainda.

O **editor de ligação**, também chamado de **linker**, gera um **programa executável** a partir de um ou mais módulos-objetos.

Apenas visualizando o que eu falei (figura 1.3):

- Criamos um **programa-fonte** em **linguagem de alto nível**;
- Ele é **compilado**;
 - Gerando um **programa-objeto** em linguagem de máquina, porém ainda não executável;
- O **editor de ligação**:
 - Recebe esse **programa-objeto**; e
 - Gera um **programa em linguagem de máquina executável** para uma **plataforma específica**;
- Como uma linguagem de máquina de uma plataforma às vezes difere de outra, precisamos de um compilador e de um linker específicos para que um programa-fonte possa ser traduzido em na linguagem de máquina própria de uma plataforma.

FIGURA 3 | **processo de compilação de um programa.**



Esse processo é feito uma vez para cada plataforma. Se houver alguma alteração do programa-fonte, precisaremos compilar novamente o código.

As linguagens C e C++ são exemplos de linguagens compiladas.

Expliquei que Python é uma linguagem de alto nível, mas ela não passar pelo processo de compilação/ligação. Usamos outro processo: a **interpretação**. Nesse processo, temos a participação do **interpretador**. Ele é uma **instância** de **hardware** ou **software** que **lê** e **executa** diretamente as **instruções apresentadas**. Ou seja, cada linha de um código é lida e executada por um interpretador.

O interpretador não produz módulo-objeto. Apenas executa as instruções contidas no programa-fonte, não necessariamente em linguagem de alto nível.

Durante uma execução, o interpretador lê cada instrução a partir de um programa-fonte e executa-a imediatamente.

A cada vez que o código for executado, o processo é feito, portanto, a interpretação é mais lenta que a compilação.

EXEMPLO

Como exemplos de linguagens interpretadas: BASIC, Perl, PHP, JavaScript, Lisp, Ruby, etc. Além delas, temos a nossa linguagem **Python**.

Java passa por um processo híbrido de compilação e interpretação.

VOCÊ SABIA?

A **linguagem Python** é uma **linguagem interpretada**.

1.1.3. COMO PYTHON É DISTRIBUÍDO

Python é **software livre** e é **distribuído** através da licença **Python Software Foundation License** (compatível com a **GNU GPL**). Isso torna a linguagem **gratuita**, **reutilizável** e **distribuível** até mesmo para **software comercial**.

1.1.4. POR QUE USAR PYTHON

Por que usar a linguagem Python?

Primeiro, por conta de sua simplicidade, o que reduz o tempo criação e manutenção de um programa. Outro ponto é que Python suporta módulos e pacotes, o que encoraja a programação modularizada e reuso de códigos.

Há outros motivos, por exemplo, Python é uma das linguagens que mais tem crescido devido a sua compatibilidade (roda na maioria dos sistemas operacionais) e a sua capacidade de auxiliar outras linguagens.

Por último, Python tem se tornado popular para análise de dados e conquistou a comunidade científica.

1.1.5. OBJETIVOS DA LINGUAGEM

Python possui alguns objetivos, tais como:

- É uma linguagem de propósito geral para:
 - Buscar dados em um banco de dados
 - Ler uma página na internet
 - Exibir graficamente os resultados
 - Criar planilhas
 - Entre outras ações
- Possui vários módulos, prontos para realizar essas tarefas.

1.1.6. ONDE PODE SER USADO PYTHON

A linguagem em questão é usada em:

- Desenvolvimento de:
 - Aplicações web;
 - Aplicações desktop;
 - Baseadas em GUI;
 - Aplicações para dispositivos móveis.
- Ciência de dados; e
- Inteligência Artificial.

1.1.7. EXEMPLOS DE USO

Como exemplo de aplicações feitas em Python, temos: Dropbox, Reddit e Instagram.

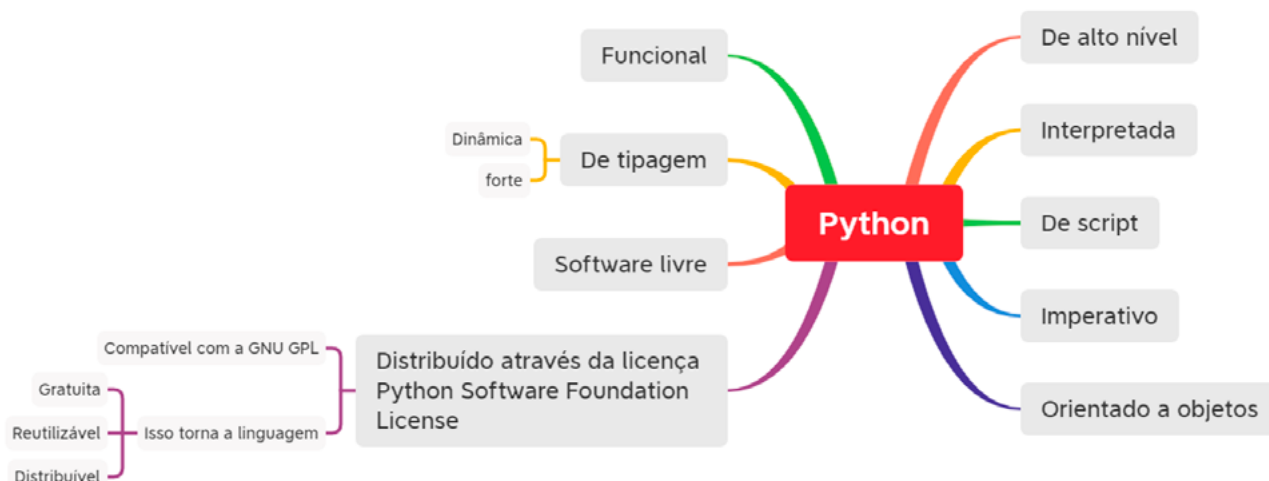
Outros exemplos de projetos usando a linguagem são:

- BitTorrent começou como um programa Python;

- NSA (Agência de Segurança Nacional) aplica o Python para análise e criptografia de inteligência;
- Youtube foi desenvolvido com Python e outras linguagens; e
- Google baseou seu sistema de busca em Python.

1.1.8. MAPA MENTAL SOBRE INTRODUÇÃO E CONCEITUAÇÃO DO PYTHON

FIGURA 4 | **Resumo da conceituação da linguagem Python.**



2. SINTAXE

2.1. SINTAXE EM PYTHON

A indentação refere-se aos espaços no início de uma linha de código. É aplicada ao código fonte de um programa para ressaltar ou definir a estrutura do algoritmo.

EXEMPLO

1:

main.py	Run	Shell
<pre>1 x = 15 2 y = 10 3 if x > y: 4 print("x é maior que y") # Código indentado.</pre>		<pre>x é maior que y > </pre>

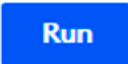
**VOCÊ SABIA?**

Para muitas linguagens de programação, a indentação é usada apenas para uma boa legibilidade, mas, para o Python, é de muita importância: Python usa a indentação para indicar (delimitar) um bloco de código. Se a indentação não for feita de forma correta em um código nem Python, ele não será executado.

Veremos alguns exemplos corretos e outros errados, para entendermos o uso da indentação em Python.

EXEMPLO


2:

main.py		Shell
<pre>1 x = 15 2 y = 10 3 if x > y: 4 print("x é maior que y")</pre>		<pre>x é maior que y > </pre>

No exemplo 2, temos uma estrutura de condição if com bloco de uma linha apenas. O interpretador irá executar o código tranquilamente,

EXEMPLO

3:

main.py		Shell
<pre>1 x = 15 2 y = 10 3 if x > y: 4 print("x é maior que y")</pre>		<pre>File "<string>", line 4 print("x é maior que y") ^ IndentationError: expected an indented block > </pre>

Acima, temos o mesmo código, mas sem a indentação para indicar o bloco de código da estrutura if. Nesse caso, o código não será executado e haverá um erro sobre a espera de uma indentação.

EXEMPLO

4:

main.py	Run	Shell
<pre>1 x = 15 2 y = 10 3 if x > y: 4 print("x é maior que y") 5 if x >= y: 6 print("x é maior que y")</pre>		<pre>x é maior que y x é maior que y > </pre>

No exemplo 4, temos uma situação interessante: o bloco da segunda estrutura if está mais recuado do que o bloco da primeira estrutura if. Porém, o que importa é que houve a indentação para delimitar o bloco de cada estrutura.

EXEMPLO

5:

main.py	Run	Shell
<pre>1 x = 15 2 y = 10 3 if x > y: 4 print("x é maior que y") 5 print("x é maior que y")</pre>		<pre>File "<string>", line 5 print("x é maior que y") ^ IndentationError: unexpected indent > </pre>

No quinto exemplo, o código contém um erro de indentação e não será executado, pois há duas linhas com indentações diferentes em uma mesma estrutura. Se as duas linhas participam do mesmo bloco de código, elas devem estar indentadas de forma igual, não importando a quantidade de recuo.

2.1. COMENTÁRIOS DE CÓDIGO

Os comentários na linguagem Python são indicados pelo caractere **cerquilha** (#). Podem ser usados para explicar um código, tornar o código mais legível ou evitar a execução de linhas específicas de código ao testá-lo.

EXEMPLO

6:

main.py	Run	Shell
<pre>1 # Variável x inicializada com valor 15. 2 x = 15 3 # Variável y inicializada com valor 10. 4 y = 10 5 # Se x for maior que y, então imprima uma mensagem. 6 if x > y: 7 print("x é maior que y")</pre>		<pre>x é maior que y > </pre>

3. VARIÁVEIS

3.1. INTRODUÇÃO

Meu(minha) consagrado(a), os **programas** são **compostos** por **dados** (variáveis) e **código** (instruções).

As **variáveis** e também as **constantes** são **recipientes** (endereços de memória) que **armazenam informações** de um **determinado tipo**, para que seja possível a manipulação delas pelos programas.

Destaco a diferença entre variáveis e constantes:

- As informações contidas nas variáveis podem ser modificadas no decorrer do programa;
- Enquanto as informações relacionadas a constantes não podem.

Faremos uma analogia entre variáveis e constantes com garagens de carros (figura 3.1):

- Os dados são os carros;
- Tanto as variáveis quanto as constantes são as garagens:
 - As variáveis são garagens com rotatividade de carros de mesma marca e de mesmo modelo;
 - Nem sempre teremos o mesmo carro estacionado;
 - As constantes, por sua vez, são garagens de carros de um colecionador, onde cada garagem recebe um carro de mesma marca e de mesmo modelo;
 - Uma vez um carro estacionado na garagem, ele não sai mais;

- O nome das variáveis e das constantes é a identificação das garagens;
- O tipo das variáveis e das constantes define qual a marca e qual o modelo de carros as garagens podem receber.

FIGURA 5 | **Analogia entre variáveis/constantes e garagens de carros.**




Dados	•Carros
Variáveis	•Garagens identificadas com rotatividade de carros de mesma marca e mesmo modelo
Constantes	•Garagens identificadas com carros de coleção de mesma marca e mesmo modelo
Nome das variáveis e constantes	•Identificação das garagens
Tipo das variáveis e constantes	•Marca e o modelo dos carros

3.2. VARIÁVEIS EM PYTHON

As **variáveis na linguagem Python** não possuem um comando para serem declaradas. Elas são criadas no momento em que se atribui um valor a elas.

EXEMPLO

1:

main.py	  	Shell
1 x = 15		15
2 print(x)		<class 'int'>
3 print(type(x))		Rogério Araújo
4		<class 'str'>
5 y = "Rogério Araújo"		>
6 print(y)		
7 print(type(y))		

As variáveis em Python também não precisam ser declaradas com nenhum tipo em particular. Podem até mudar de tipo depois de terem sido definidas:

EXEMPLO

2:

main.py	Run	Shell
<pre> 1 # Variável x do tipo int. 2 x = 15 3 print(x) 4 print(type(x)) 5 6 # Variável x do tipo str. 7 x = "Rogerão Araújo" 8 print(x) 9 print(type(x)) 10 11 # Variável x do tipo float. 12 x = 1.84 13 print(x) 14 print(type(x)) </pre>	Run	<pre> 15 <class 'int'> Rogerão Araújo <class 'str'> 1.84 <class 'float'> > </pre>

3.3. REGRAS DE NOMES DE VARIÁVEIS EM PYTHON

As **variáveis em Python** são **case-sensitive**, ou seja, uma variável com o nome xy é diferente de outra com o nome XY.

São sequências de **tamanho ilimitado**, contendo apenas **caracteres alfanuméricos** (A-z 0-9) e **sublinhados** (_).

Elas **podem iniciar** com **letra** (A-z) ou **sublinhado** (_), mas **não deve iniciar** com **números** (0-9).

As variáveis em Python não podem ter espaço em branco e não podem ser palavra-chave ou palavra reservada que faz parte da própria sintaxe da linguagem, tais como, print, for, if, etc.

Na tabela 1, temos a lista das palavras-chave da linguagem Python.

<i>and</i>	<i>as</i>	<i>assert</i>	<i>break</i>
<i>class</i>	<i>continue</i>	<i>def</i>	<i>del</i>
<i>elif</i>	<i>else</i>	<i>except</i>	<i>False</i>
<i>finally</i>	<i>for</i>	<i>from</i>	<i>global</i>
<i>if</i>	<i>import</i>	<i>in</i>	<i>is</i>
<i>lambda</i>	<i>None</i>	<i>nonlocal</i>	<i>not</i>
<i>or</i>	<i>pass</i>	<i>raise</i>	<i>return</i>
<i>True</i>	<i>try</i>	<i>while</i>	<i>with</i>
<i>yield</i>			

Tabela 1 | **Palavras-chave na linguagem Python.****EXEMPLO**

Exemplos de nomes de variáveis corretos:

main.py	Run	Shell
<pre>1 ab = "Python" 2 a_b = "Python" 3 _a_b = "Python" 4 aB = "Python" 5 AB = "Python" 6 ab2 = "Python"</pre>		<pre>></pre>

Exemplos de nomes de variáveis incorretos:

main.py	Run	Shell
<pre>1 1ab = "Python" 2 .ab = "Python" 3 a-b = "Python" 4 a b = "Python" 5 \$ab = "Python" 6 ab% = "Python" 7 while = "Python"</pre>		<pre>File "<string>", line 1 1ab = "Python" ^ SyntaxError: invalid syntax ></pre>

4. TIPOS DE DADOS

4.1. LINGUAGENS ESTÁTICA E DINAMICAMENTE TIPADAS

Linguagens estaticamente tipadas são linguagens em que o **tipo de uma variável** é conhecido em **tempo de compilação**:

// A variável x abaixo é do tipo int em Java:

```
int x = 15;
```

Linguagens dinamicamente tipadas são linguagens em que o **tipo de uma variável** pode ser alterado durante a execução do código, ou seja, é conhecido em **tempo de execução**:

A variável x abaixo é do tipo int em Python:

```
x = 15
```

Conclusão: Python é dinamicamente tipada!

4.2. LINGUAGENS FRACA E FORTEMENTE TIPADAS

Linguagens fracamente tipadas são linguagens em que se pode fazer **operações sem a necessidade** da realização de **cast**:

Código:

```
x = '11'
```

```
print(1 + x)
```

Resultado da execução:

```
111
```





No código acima, é um exemplo de linguagem que tem uma função print que faz um cast implícito para que o primeiro operando numérico 1 para ser concatenado com o segundo operando *string*.

DESTAQUE

Muitas vezes, há uma confusão entre os conceitos de uma linguagem ser fracamente tipada com uma linguagem ser estaticamente tipada. São conceitos diferentes.

Linguagens fortemente tipadas são linguagens em que se deve fazer **operações com a necessidade** da realização de **cast**.

EXEMPLO

main.py	  	Shell	
<pre>1 x = '11' 2 print(1 + x)</pre>		<pre>Traceback (most recent call last): File "<string>", line 2, in <module> TypeError: unsupported operand type(s) for +: 'int' and 'str' > </pre>	

No código acima, é um exemplo de linguagem que tem uma função print que não faz um cast implícito. Em Python, não há cast implícito, o que faz com que ela seja classificada como fortemente tipada.

Para corrigirmos o erro, temos que fazer um cast explícito:

main.py	Shell
1 <code>x = '11'</code>	12
2 <code>print(1 + int(x))</code>	111
3 <code>print(str(1) + x)</code>	>

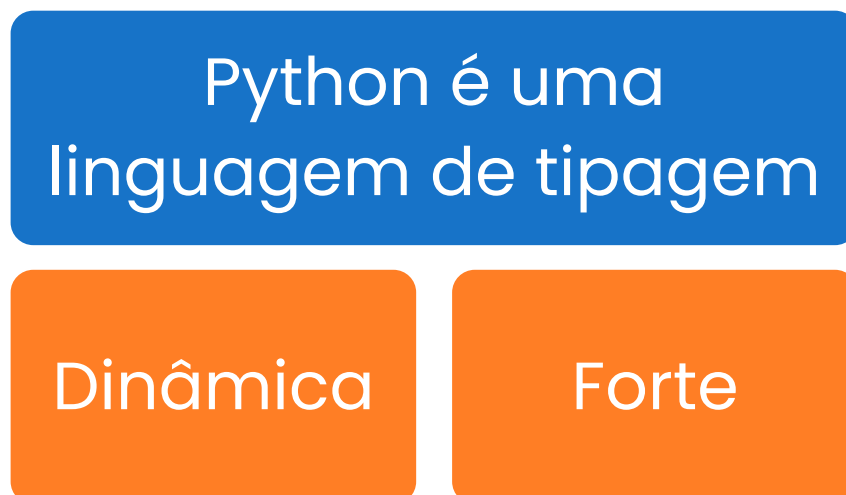
No primeiro print, transformamos o valor da variável `x` (*string*) em um número inteiro. Então houve a operação de adição ($1 + 11 = 12$).

No segundo print, transformamos o valor inteiro 1 em *string*. Então houve a concatenação de *string* ("`1`" + "`11`" = "`111`").

Conclusão: Python é fortemente tipada!

4.3. CATEGORIAS DA LINGUAGEM PYTHON

FIGURA 6 | **Categorias da linguagem Python.**



Python é:

- **Dinamicamente tipada;**
 - É uma linguagem em que o tipo de uma variável pode ser alterado durante a execução do código; e
- **Fortemente tipada;**
 - É uma linguagem em que se deve fazer operações com a necessidade da realização de cast.

4.4. CATEGORIAS DE TIPOS DE DADOS DA LINGUAGEM PYTHON

Irei mostrar apenas quais são os principais tipos de dados da linguagem Python, com alguns exemplos de cada tipo. Na segunda parte do nosso curso, iremos estudá-los com mais detalhes.

FIGURA 7 | **Categorias de tipos da linguagem Python.**



5. EXPRESSÕES, INSTRUÇÕES E BLOCOS

As **expressões** são construções feitas de variáveis, operadores e invocações de métodos. Elas retornam um valor único. O tipo do valor retornado por elas depende dos elementos utilizados.

Exemplos de expressões:

main.py	Shell
<pre>1 x = 15 # Retorna um valor inteiro. 2 print("O valor de x é " + str(x)) # Retorna um valor string. 3 y = (x + 10) / 100; # Retorna um valor de ponto fluante. 4 if x > y: # Retorna um valor boolean. 5 print("x (" + str(x) + ") > y (" + str(y) + ")") # Retorna um valor string.</pre>	<pre>O valor de x é 15 x (15) > y (0.25) ></pre>

As **instruções**, por sua vez, também chamadas de **comandos** ou **declarações**, são unidades de código que o Python pode executar e tem um efeito, tais como:

- Criar uma variável; e
- Exibir um valor.

As instruções são unidades completas de execução e são formadas por expressões.

Podemos ter instruções de declaração de variáveis ou instruções de controle de fluxo.

Os **blocos** são grupos de zero ou mais instruções. O início e o fim de um **bloco em Python** são delimitados por **indentação**.

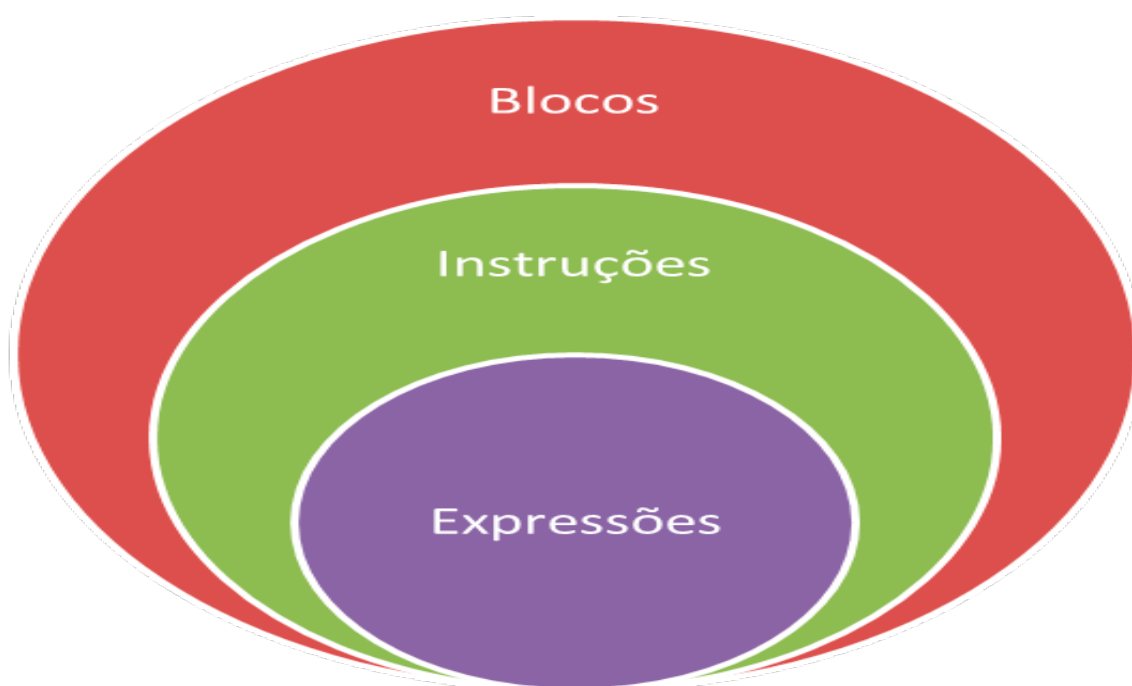
EXEMPLO

```
1 passei = True
2 if passei: # Início do bloco 1.
3     print("Churrasco garantido!!!")
4     # Fim do bloco 1.
5 else: # Início do bloco 2.
6     print("Não foi dessa vez.")
7     print("Vou continuar a estudar e consegui o
8         objetivo!")
9     # Fim do bloco 2.
```

Churrasco garantido!!!

> |

FIGURA 8 | **Relação entre expressões, instruções e blocos.**



RESUMO

Meu(minha) prezado(a) aluno(a), chegamos ao final da nossa aula. Vou destacar os principais pontos da aula para sua revisão.

Python é uma **linguagem de programação**:

- De **alto nível**;
 - É uma linguagem possui a sintaxe que se aproxima da linguagem humana;
- **Interpretada**;
 - Cada linha do código em Python é lida e executada por um interpretador;
- De **script**;
 - Em um script, por exemplo, código em Python, descreve-se uma sequência de comandos e tarefas que um interpretador deve executar;
- **Imperativa**;
 - É uma linguagem orientada a ações, onde a computação é vista como uma sequência de instruções que manipulam valores de variáveis;
- **Orientada a objetos**;
 - Suporta os conceitos da orientação a objetos;
- **Funcional**;
 - É um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e evita estados ou dados mutáveis;
- De **tipagem**:
 - **Dinâmica**; e
 - **Forte**.

Python é **software livre** e é **distribuído** através da licença **Python Software Foundation License** (compatível com a **GNU GPL**). Isso torna a linguagem **gratuita, reutilizável e distribuível** até mesmo para **software comercial**.

Para muitas linguagens de programação, a indentação é usada apenas para uma boa legibilidade, mas, para o Python, é de muita importância: Python usa a indentação para indicar

(delimitar) um bloco de código. Se a indentação não for feita de forma correta em um código nem Python, ele não será executado.

As **variáveis na linguagem Python** não possuem um comando para serem declaradas. Elas são criadas no momento em que se atribui um valor a elas.

As **variáveis em Python** são **case-sensitive**, ou seja, uma variável com o nome xy é diferente de outra com o nome XY.

São sequências de **tamanho ilimitado**, contendo apenas **caracteres alfanuméricos** (A-z 0-9) e **sublinhados** (_).

Elas **podem iniciar** com **letra** (A-z) ou **sublinhado** (_), mas **não deve iniciar** com **números** (0-9).

As variáveis em Python não podem ter espaço em branco e não podem ser palavra-chave ou palavra reservada que faz parte da própria sintaxe da linguagem, tais como, print, for, if, etc.

Python é:

- **Dinamicamente tipada;**
 - É uma linguagem em que o tipo de uma variável pode ser alterado durante a execução do código; e
- **Fortemente tipada;**
 - É uma linguagem em que se deve fazer operações com a necessidade da realização de cast.

As **expressões** são construções feitas de variáveis, operadores e invocações de métodos. Elas retornam um valor único. O tipo do valor retornado por elas depende dos elementos utilizados.

As **instruções**, por sua vez, também chamadas de **comandos** ou **declarações**, são unidades de código que o Python pode executar e tem um efeito, tais como:

- Criar uma variável; e
- Exibir um valor.

Os **blocos** são grupos de zero ou mais instruções. O início e o fim de um **bloco em Python** são delimitados por **indentação**.

MAPA MENTAL

FIGURA 1 | **características da linguagem Python.**

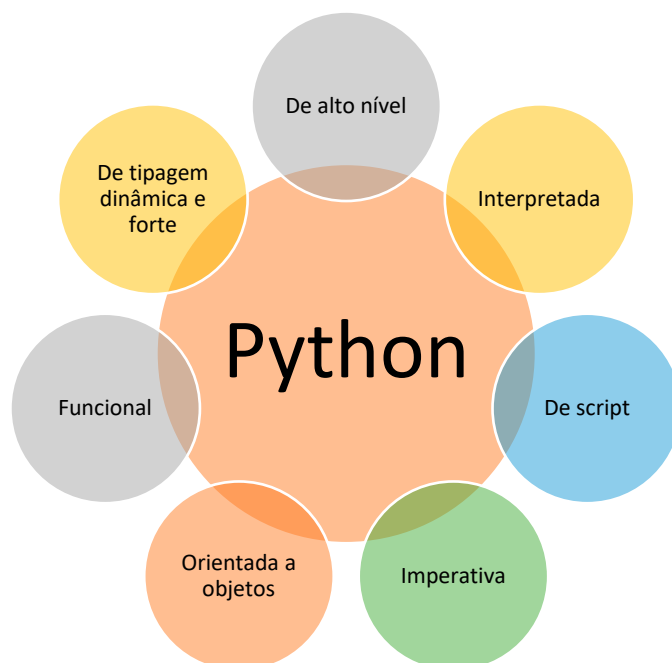


FIGURA 2 | **Classificações de linguagens.**

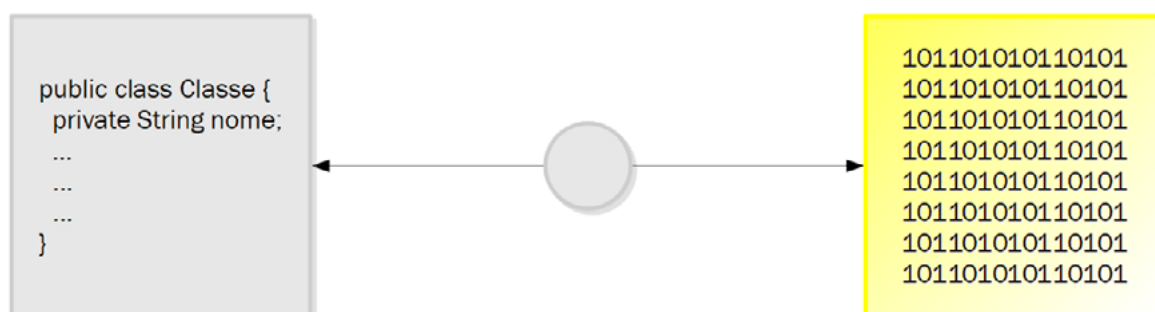


FIGURA 3 | processo de compilação de um programa.

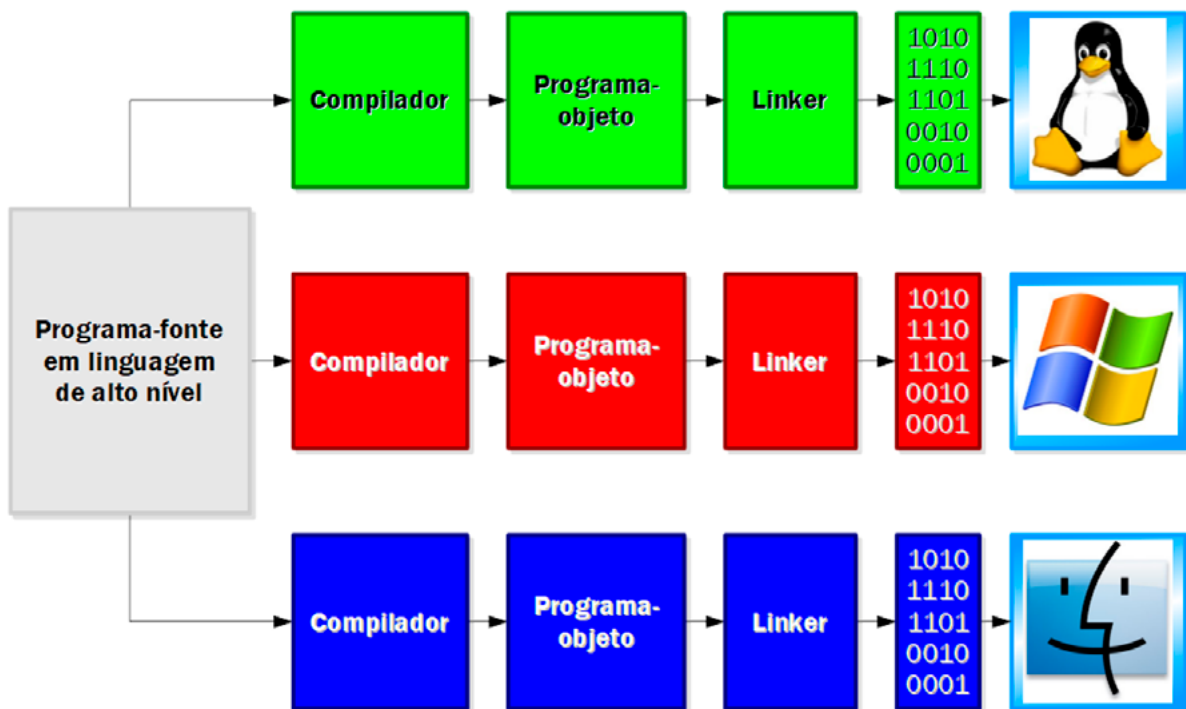


FIGURA 4 | Resumo da conceituação da linguagem Python.

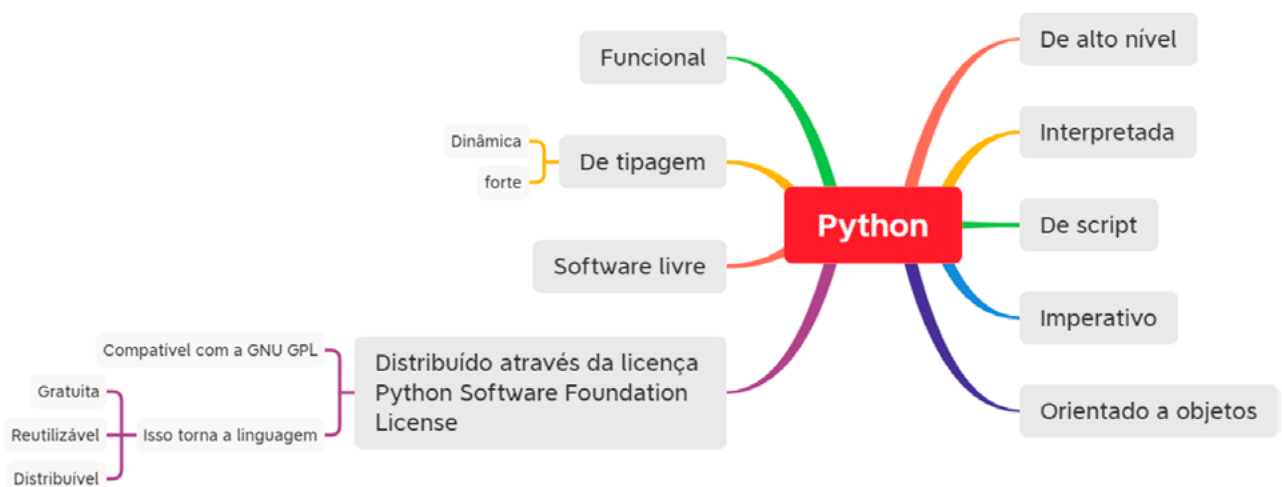


FIGURA 5 | **Analogia entre variáveis/constantes e garagens de carros.**

Dados	•Carros
Variáveis	•Garagens identificadas com rotatividade de carros de mesma marca e mesmo modelo
Constantes	•Garagens identificadas com carros de coleção de mesma marca e mesmo modelo
Nome das variáveis e constantes	•Identificação das garagens
Tipo das variáveis e constantes	•Marca e o modelo dos carros

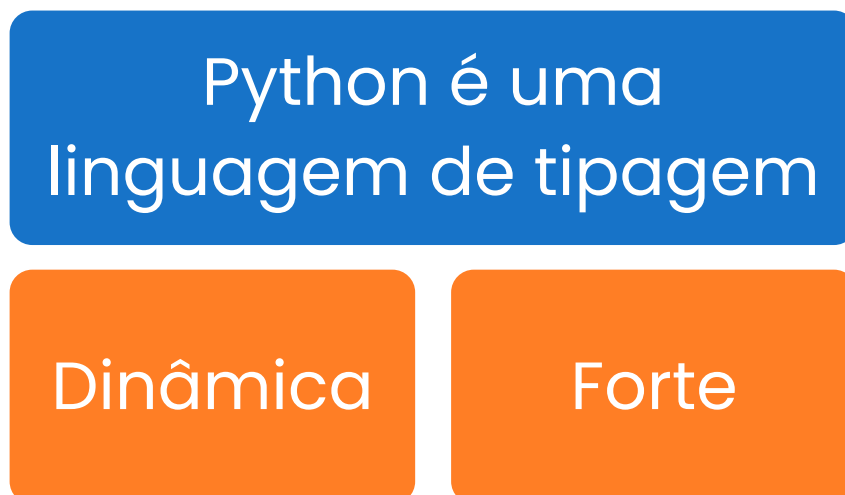
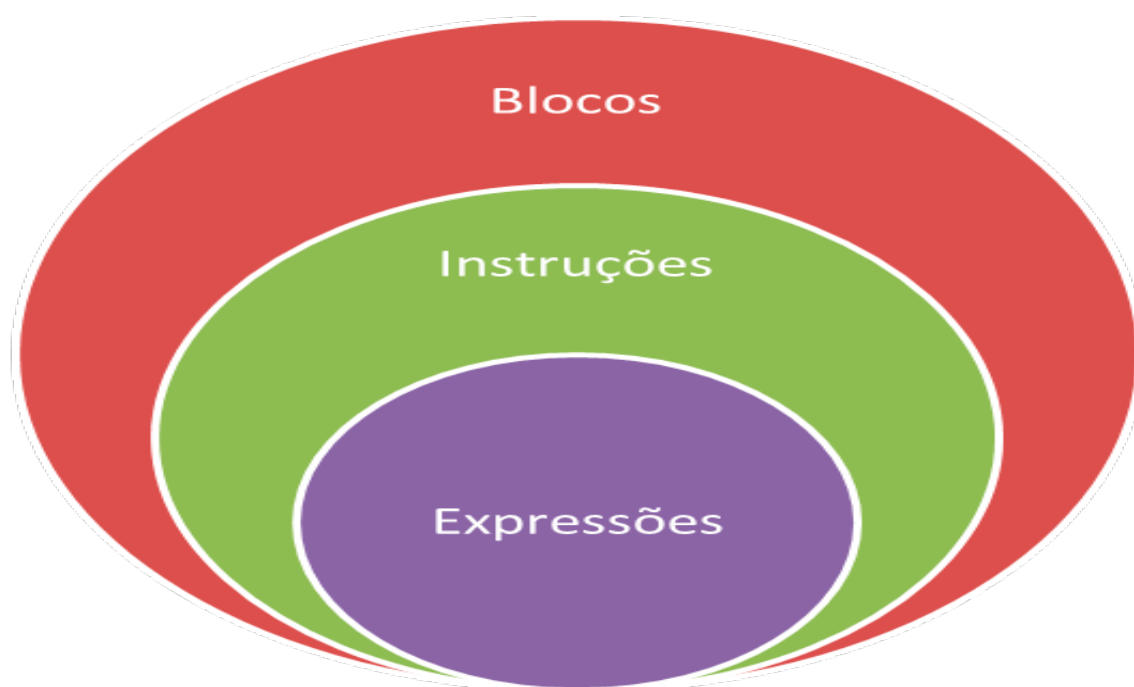
FIGURA 6 | **Categorias da linguagem Python.**FIGURA 7 | **Categorias de tipos da linguagem Python.**

FIGURA 8 | **Relação entre expressões, instruções e blocos.**

REFERÊNCIAS

1. Apostila de programação fácil em Python em pdf. Disponível em: <https://www.passeidireto.com/arquivo/30903709/apostila-de-programacao-facil-em-python-em-pdf>
2. Para quê é utilizado a programação em Python?. Disponível em: <https://br.bitdegree.org/tutoriais/programacao-em-python/>
3. PensePython2e - Tradução do livro Pense em Python (2ª ed.), de Allen B. Downey. Disponível em: <https://penseallen.github.io/PensePython2e/>
4. Por que Cientistas de Dados escolhem Python?. Disponível em: <http://www.cienciaedados.com/por-que-cientistas-de-dados-escolhem-python/>
5. Python. Disponível em: <https://www.python.org/>
6. Python e Orientação a Objetos. Disponível em: <https://www.caelum.com.br/apostila-python-orientacao-a-objetos>
7. Python Para Programadores. Disponível em: <https://python-para-programadores.readthedocs.io/pt/latest/index.html>
8. Python Tutorial. Disponível em: <https://www.w3schools.com/python/>