

## UFCD 10793\_04\_AG – Fundamentos de Python

João Araújo

22-02-2024



# Estrutura da formação

- Apresentação do formador / formandos
- Python
  - ☐ Introdução
  - ☐ Funcionalidades
  - ☐ Uso de Python na indústria
  - ☐ Oportunidades de carreira
  - ☐ O interpretador Python
  - ☐ Básicos de Python
- Ferramentas de programação Python
  - ☐ Ambientes de desenvolvimento
  - ☐ Variáveis
  - ☐ *Input e output de dados*
  - ☐ *Tipos de dados*
  - ☐ Primeiros programas em Python
- Utilizações de Python
  - ☐ Tratamentos de dados
  - ☐ Controlo de fluxo
  - ☐ Ficheiros e iteradores
  - ☐ Benchmark
  - ☐ Profilers de memória e CPU
  - ☐ Widgets
  - ☐ Geradores



# Estrutura da formação

- Conceitos genéricos de programação em Python
  - ☐ Tipos de dados
  - ☐ Programação condicional
  - ☐ Funções
  - ☐ Iterações
- Classes
  - ☐ Construtores
  - ☐ Métodos e atributos
  - ☐ Herança
  - ☐ Decoradores
- Bibliotecas
  - ☐ Pandas e Numpy
  - ☐ Matplotlib
  - ☐ Importação SQL e CSV
  - ☐ Testes unitários



# Paradigmas de programação

- Funcional
  - Conjunto de funções relacionadas entre si
  - Não existe distinção entre dados e funções
  - Programa = Dados + Funções
    - Lisp
    - Prolog
    - Scheme
    - ...
- Procedimentais ou imperativas
  - Sequência de instruções que atuam sobre dados
  - Altera de forma destrutiva o valor dos dados
  - Programa = Estrutura de dados + Algoritmo
    - C
    - Pascal
    - PHP(inferior à versão 4.0)
    - ...
- Orientado a objetos
  - Conjunto de objetos que são manipulados pelas operações que estão ligados
  - Objetos encapsulam não apenas dados mas também operações sobre os mesmos
  - Programa = Objetos + Operações
  - Orientação a objetos = Objetos + Classes + Herança
  - POO = Orientação de objetos + Mensagens
    - Java
    - C#
    - Javascript
    - ..

# Introdução

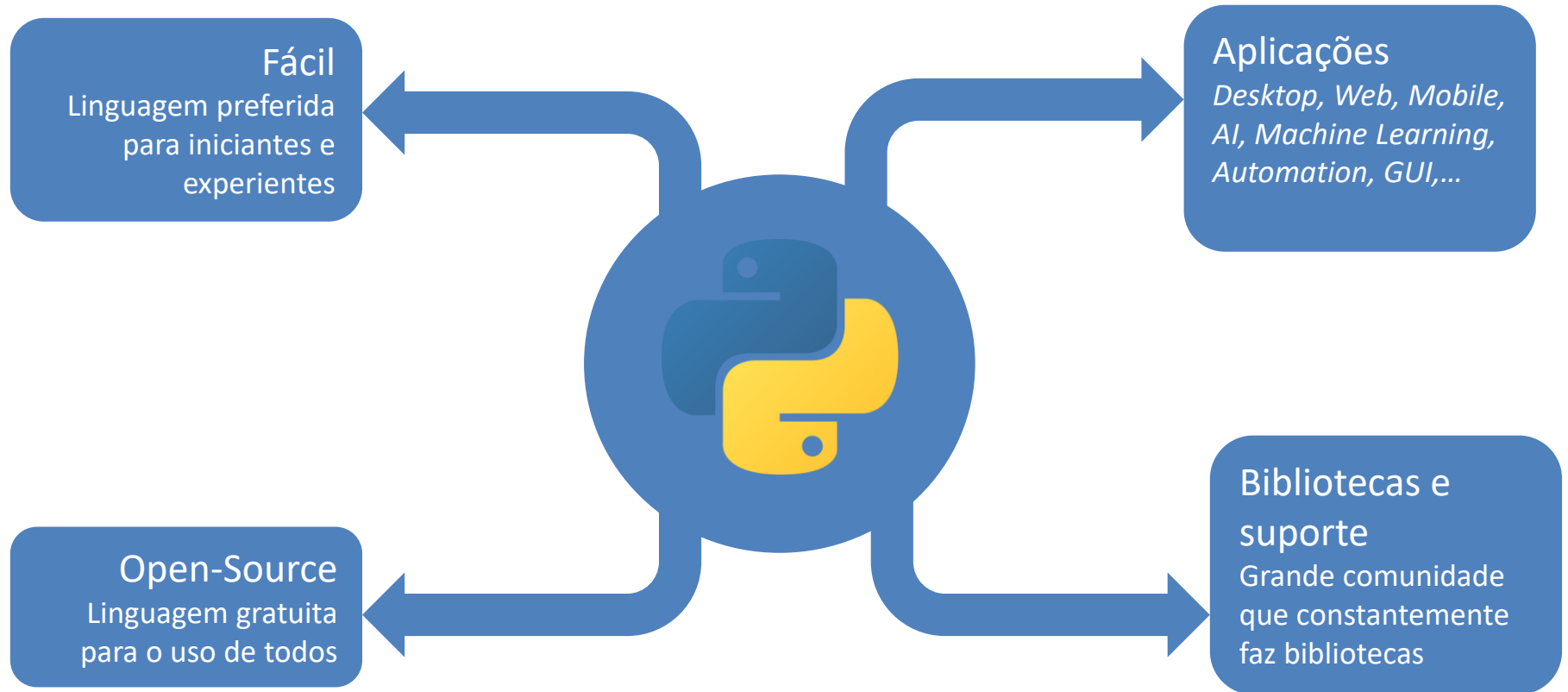
- O que é Python?
  - Python é uma linguagem de programação de uso **geral**, **alto nível**, **interpretada**, e sintaxe de uso fácil
  - Atualmente está na versão 3.11.5
  - Criada em 1989 por Guido Van Rossum, como sucessora da linguagem ABC
  - Primeira aparição em 1991
  - Leitura orientada de código
    - Uso de indentação
  - Programação estruturada e orientada a objetos
  - Guia de boas práticas de programação no *Python Enhancement Proposals* (PEP)
  - Suporta vários tipos de paradigmas de programação (procedimental, funcional e orientado a objetos)

# Introdução

- Princípios de programação em Python (*Zen of Python* - PEP20)
  - Bonito é melhor que feio
  - **Explícito é melhor que implícito**
  - **Simples é melhor que complexo**
  - Complexo é melhor que complicado
  - Linear é melhor que encaixado
  - Disperso é melhor que compacto
  - **Legibilidade interessa**
  - Os casos especiais não são o suficiente para quebrar as regras
  - Apesar que a praticabilidade vence a pureza
  - Erros não devem ser silenciados
  - Exceto quando o sejam, explicitamente
  - No caso de ambiguidade, recusar a cair na tentação de adivinhar
  - Deve haver uma, e preferencialmente apenas uma, maneira de fazer
  - A maneira de fazer pode não ser óbvia ao início, a não ser que seja holandês
  - Agora é melhor que nunca
  - Apesar que nunca, muitas vezes é melhor que imediatamente agora
  - Se a implementação é difícil de explicar, é uma má ideia
  - Se a implementação é fácil de explicar, pode ser uma boa ideia
  - Os **namespaces** são uma grande ideia – vamos fazer mais disso!

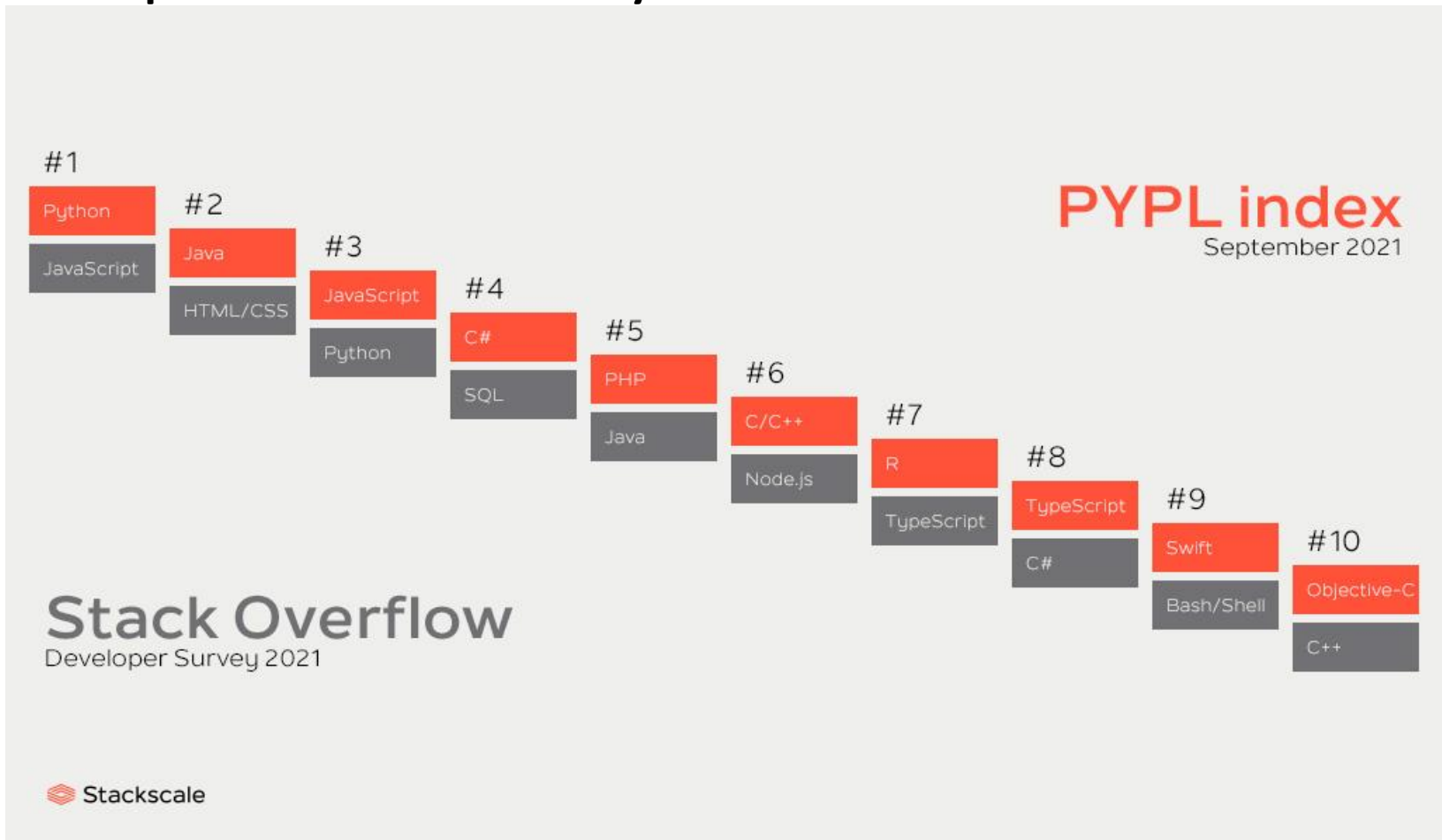
# Introdução

- Porque o Python é popular?



# Introdução

- Popularidade do Python

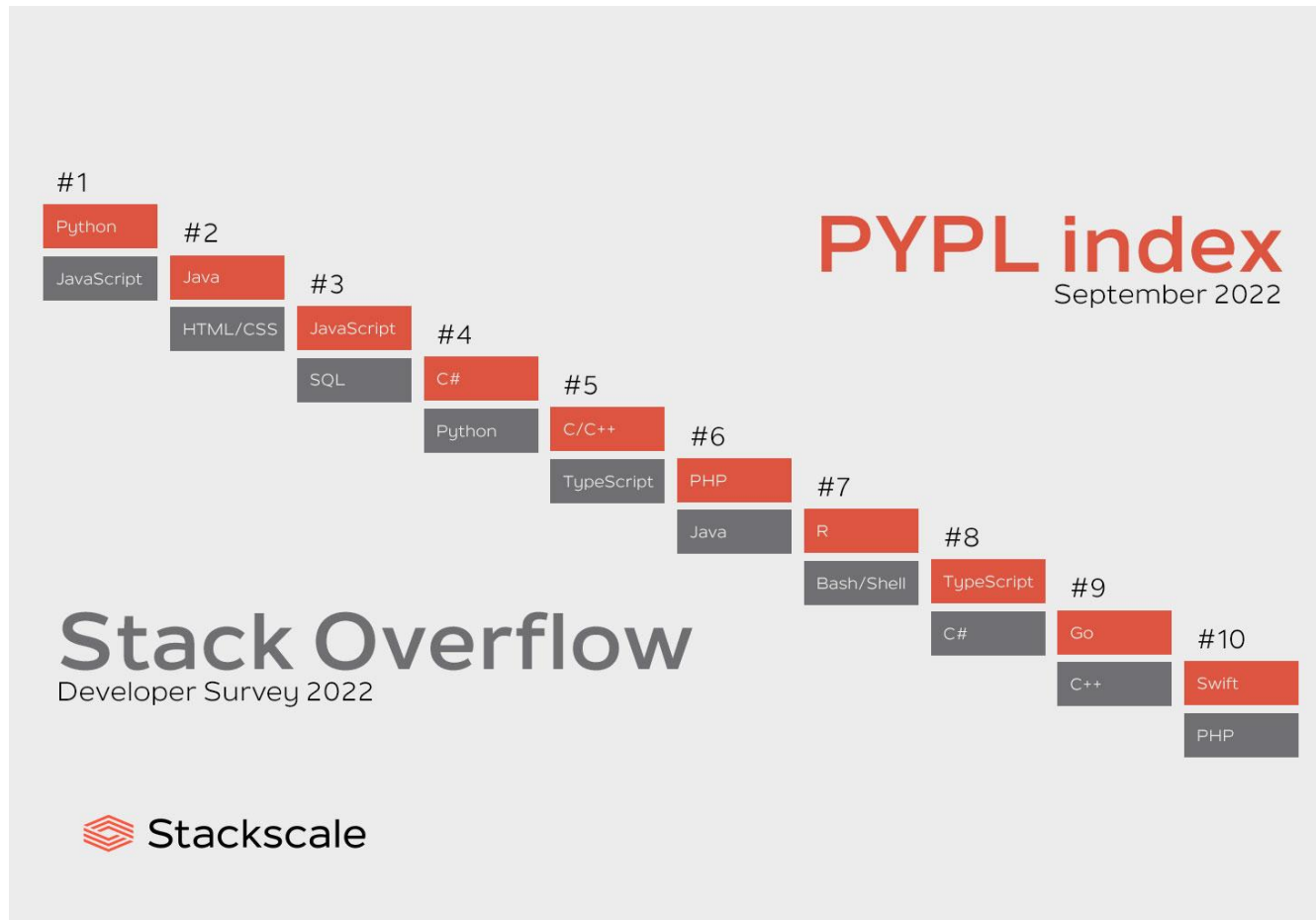


Fonte: PYPL e Stack Overflow



# Introdução

- Popularidade do Python



Fonte: PYPL e Stack Overflow

# Características



**Simplicidade** – Foco na solução, invés da sintaxe



**Open source** – Gratuito para todos os utilizadores



**Portabilidade**– Aplicações desenvolvidas executam em qualquer plataforma



**Embutido e extensível**– Capacidade de embutir o código Python em outras línguas como C, C++, Java,...



**Interpretada**– Não requer compilador para correr os programas



**Bibliotecas** – Suporte enorme de bibliotecas que se aplicam a quase todos os domínios



**Programação orientada a objetos** – Aplica-se os conceitos de OOP em Python, assim como a segurança

# Python vs linguagens de programação



- Mais tempo para desenvolvimento
- Código equivalente 3 a 5 vezes maior
- Necessário declarar tipo de variáveis
- Obriga a seguir um paradigma de programação

- Não suporta programas de grande dimensão
- Não permite reutilização eficiente do código (herança em classes)

- Utilizado em aplicações específicas, num universo mais restrito

- Mais tempo para desenvolvimento
- Código equivalente 5 a 10 vezes maior
- Necessário declarar tipo de variáveis
- Obriga a seguir um paradigma de programação

# Diferenças entre versões Python

Elemento de comparação	Python 2	Python 3
Ano de lançamento	Lançado no ano 2000	Lançado no ano 2008
Palavra chave “print”	É considerado uma declaração, não uma função	É considerado uma função, não uma declaração
Armazenamento de <i>Strings</i>	<i>Strings</i> armazenadas como ASCII, por defeito	<i>Strings</i> armazenadas como UNICODE, por defeito
Divisão de inteiros	Nesta operação, obtém-se apenas a componente inteira da divisão	Nesta operação, obtém-se o resultado com componentes inteira e decimal
Exceções	São colocadas em notações	São colocadas em parêntises
Alteração do valor de variáveis	As variáveis globais usadas para iteração num ciclo for, alteram o seu valor	As variáveis globais usadas para iteração num ciclo for, não alteram o seu valor
Iterações	Foi definida a função xrange() para para iterações	Foi introduzida a função range() para realizar iterações
Sintaxe	Tem uma sintaxe mais complicada que o Python 3	Tem uma sintaxe mais simples que o Python 2
Bibliotecas	As bibliotecas usadas não são compatíveis com as versões seguintes	As bibliotecas apenas são compatíveis em Python 3
Utilização atual	Python 2 não possui suporte desde 2020	É mais popular que o Python 2 e é a versão atual
Aplicabilidade	Mais usado em DevOps Engineering.	Usado em muitas áreas como Software Engineering, Data Science, etc.

# Versões de Python 3 e funções

Data de lançamento	Versão Python	Funcionalidades
13-03-2015	3.5.9	<ul style="list-style-type: none"><li>• Novas co-rotinas com <i>async</i> e <i>await</i></li><li>• Novo operador de matrizes a <i>@</i> b</li><li>• Acrescento de formatação % par <i>byte</i> e <i>bytearray</i></li><li>• Novos métodos <i>bytes.hex()</i>, <i>bytearray.hex()</i> e <i>memoryview.hex()</i></li><li>• <i>Memorybiew</i> suporta tuple</li><li>• Atualização de geradores</li><li>• Novos erros de recursão</li><li>• Melhoramentos do Cpython</li><li>• Melhoramentos das bibliotecas standard</li><li>• Melhoramentos de segurança</li></ul>
23-12-2016	3.6.15	<ul style="list-style-type: none"><li>• <b>Strings formatadas <i>fstrings</i></b></li><li>• Underscore em numeros, para melhor legibilidade</li><li>• Descrição de funções embutidas</li><li>• Geradores assincronos</li><li>• Criar co-rotinas assincronas</li><li>• Melhoramentos das bibliotecas standard</li><li>• Melhoramentos de segurança</li></ul>
27-06-2018	3.7.12	<ul style="list-style-type: none"><li>• Variáveis contextuais</li><li>• Classes de dados</li><li>• <i>Importlib.resources</i></li><li>• Função <i>breakpoint()</i></li><li>• Acesso personalizado a atributos de módulos</li><li>• Suporte para escrita de módulos nucleares e tipos genéricos</li><li>• <b>Manter a ordem de inserção no tipo de dado de dicionários</b></li><li>• Melhoramentos das bibliotecas standard</li><li>• Melhoramentos de segurança</li></ul>

# Versões de Python 3 e funções

Data de lançamento	Versão Python	Funcionalidades
14-10-2019	3.8.12	<ul style="list-style-type: none"><li>• Nova expressão de <code>:=</code>, em que atribui um valor a uma variável, fazendo parte de uma expressão maior</li><li>• Argumentos de funções podem ser definidos como posicionais, posicionais ou por palavra chave, e palavra-chave obrigatoriamente</li></ul>
02-01-2022	3.9.9	<ul style="list-style-type: none"><li>• Operadores de união e atualização acrescentados em dicionários</li><li>• Ajuda em variáveis do tipo de coleção de dados</li><li>• Remoção de prefixos e sufixos em <i>strings</i></li><li>• Melhoramentos das bibliotecas standard</li><li>• Melhoramentos de segurança</li></ul>
02-01-2022	3.10.1	<ul style="list-style-type: none"><li>• Melhoramentos no interpretador</li><li>• Documentação de sintaxe para normalização</li><li>• Preparação da remoção de PyUnicodeObject</li><li>• Especificação de variáveis de parâmetros</li></ul>
02-04-2023	3.11.0	<ul style="list-style-type: none"><li>• Melhoramento da localização de erros</li><li>• Exception Groups e <code>except*</code></li><li>• Adicionar notas nas exeções</li><li>• Melhoramento da velocidade de execução</li></ul>

# Python na indústria

Google



Melhores resultados de pesquisa dos utilizadores, baseado no *ranking* dos *websites* e muito mais



Dropbox



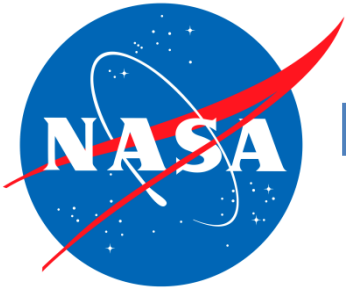
Aplicações de cliente e servidor são feitas em Python

NETFLIX



Algoritmos de *Machine Learning* para identificar os interesses dos utilizadores e classifica-los

# Python na indústria



Python usado para computação de cálculos científicos

**facebook**



Responsável por múltiplos serviços da sua infraestrutura, como efetuar manutenções agendadas



Automação industrial, Robótica, Visão computadorizada, *Machine Learning*



# Oportunidades de carreira com Python



*Web Development*



*Web Testing*



*IOT*

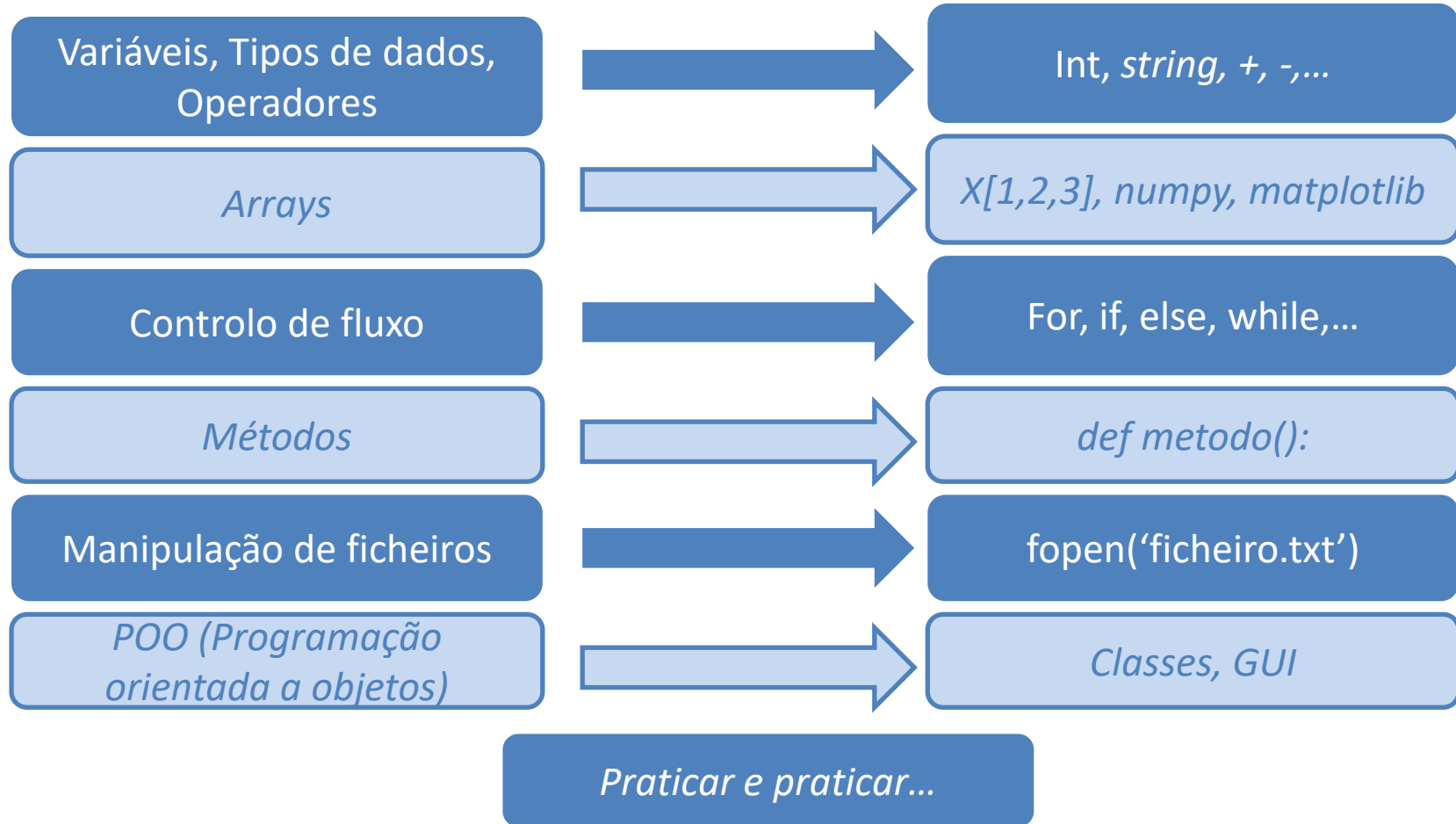


*Game Development*

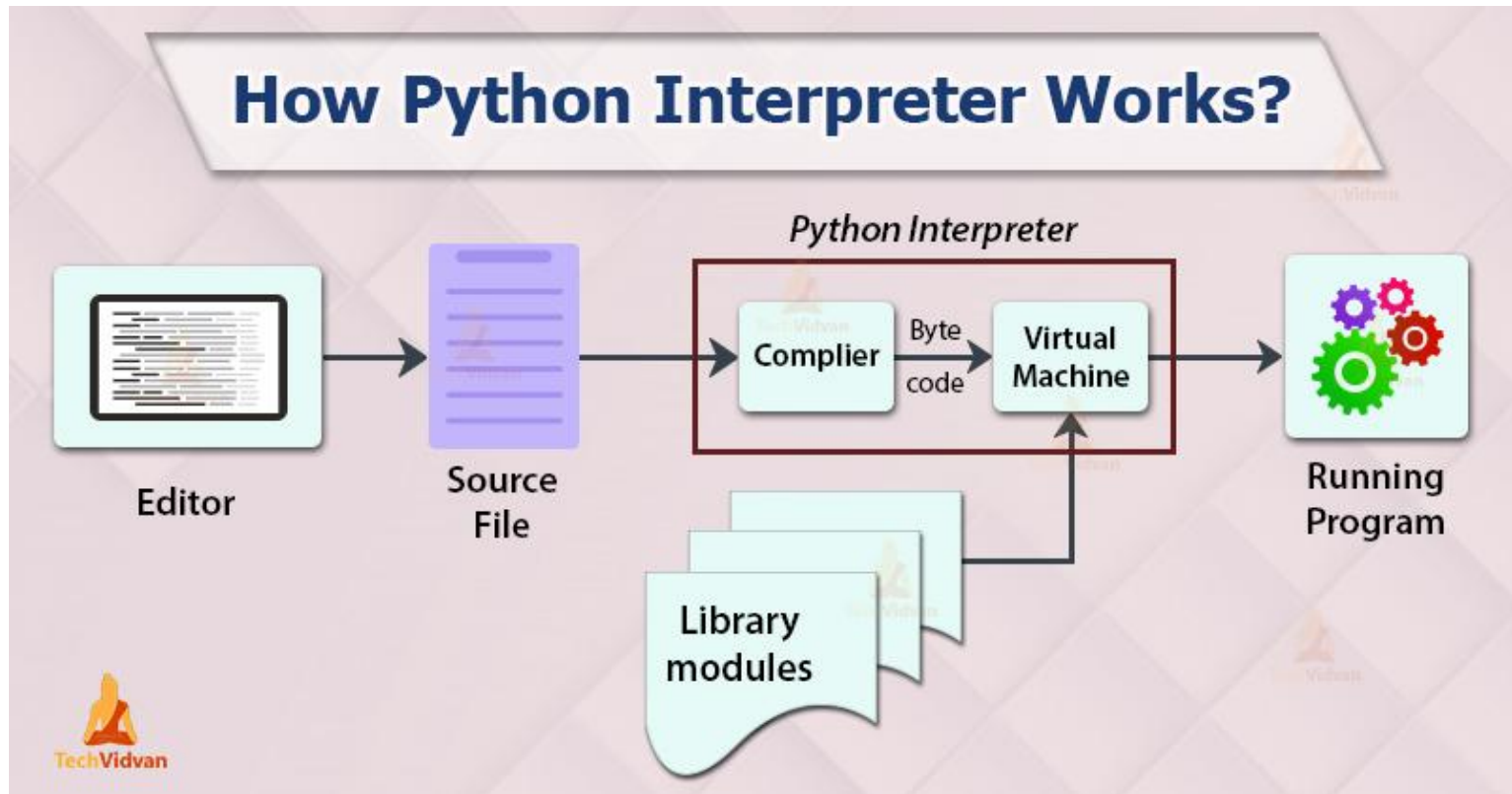


*AI, Data Science,  
Machine Learning,  
Big Data...*

# Conceitos para aprender Python



# O interpretador Python



# Conceitos básicos de Python

1. A linguagem Python é *case sensitive*

*Python* ≠ *python* ≠ *pYtHoN*

2. Endereços de ficheiros diferentes entre Windows e Linux

*r'C:\pasta\ficheiro.txt'* ≠ *'pasta/ficheiro.txt'*

3. Não existe comando para mudança de linha

# Conceitos básicos de Python

4. Deve ser escrito apenas um comando por linha  
(caso seja necessário, pode ser utilizado ponto e vírgula ;)
5. Para escrita de *strings* pode ser usadas aspas simples ou duplas : '*string*' ou "*string*"
6. Podem ser escritas linhas de comentários (#) ou várias linhas comentadas (""")

# Conceitos básicos de Python

7. Para continuar o comentário na linha seguinte (\)
8. Quando a linha de comando está entre parênteses () [] ou {} não necessita de \
9. As linhas em branco são sempre ignoradas
10. Indentação do código: **Regra mais importante do Python**. Muitas linhas de comando necessitam de *tabs* para o seu fluxo de código

# Conceitos básicos de Python

1. **Sugestão 1:** Para facilitar a legibilidade do programa, usar espaços entre operadores e linhas em branco
2. **Sugestão 2:** Comentar todos os passos do programa
3. **Sugestão 3:** Parametrizar sempre explicitamente
4. **Sugestão 4:** Limitar o comprimento das linhas a 72 char

PEP8 – Style Guide for Python Code

# Software necessário



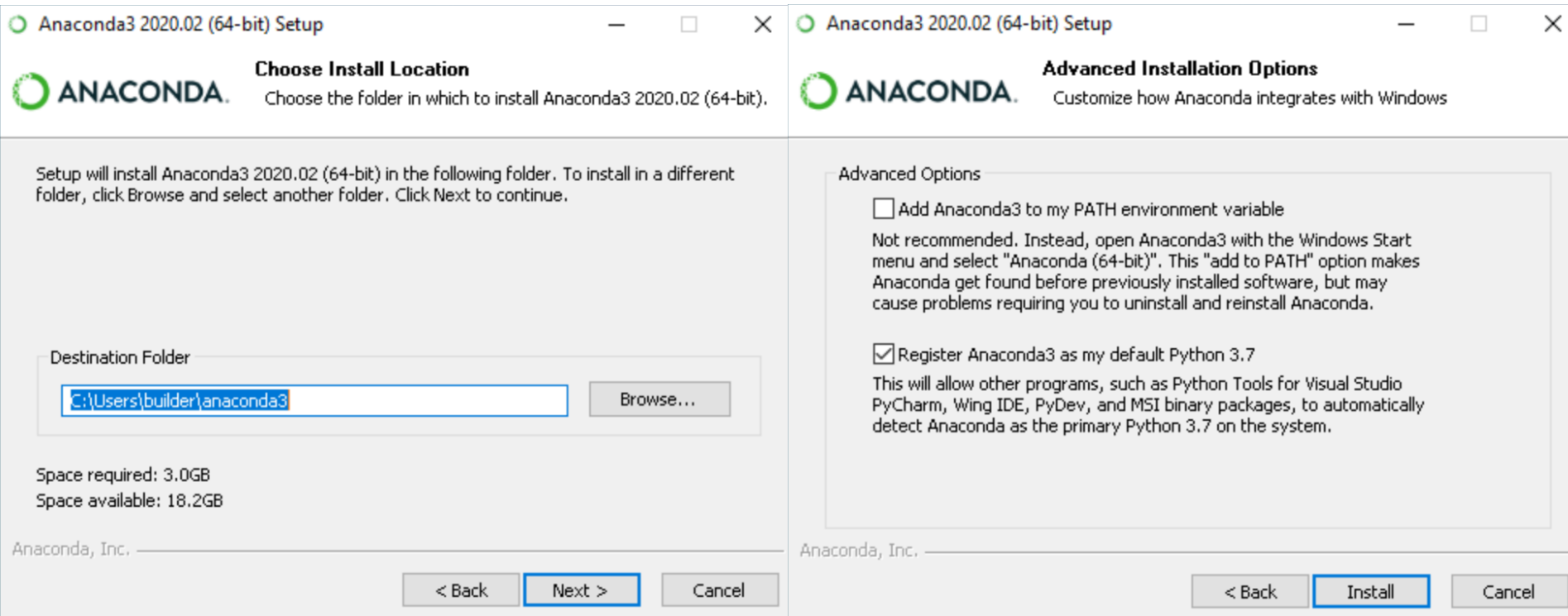
<https://www.anaconda.com/products/individual>





# Instalação do Anaconda

- Selecionar a opção “Just Me”
- Escolher caminho de instalação que não contenha espaços nem caracteres especiais
- Não instalar como administrador
- Não adicionar Anaconda ao caminho de ambientes de variáveis



# Ambientes de desenvolvimento

- Visual Studio Code



Visual Studio Code

- Editor multi-linguagens de programação
- Várias áreas de aplicação (Web, UI, servidores,...)
- Necessário instalar pacotes adicionais para utilizar as funcionalidades extra (Explorador de variáveis, correr áreas específicas de código, etc...)
- Instalar os pacotes ipykernel, e pandas no Anaconda
- Instalar a extensão Python e Jupyter da Microsoft no VS Code

- Spyder

- Editor dedicado ao Python
- Mais dedicado ao Data Science
- Não é necessário instalar extensões



SPYDER

# Jupyter notebook



- Programação interativa no Jupyter notebook
- Programação dividida por células
- Permite visualizar o resultado da execução do código da respetiva célula
- Permite visualizações de todas as células por ordem de execução

# Criação de variáveis

- O tipo de variável depende do valor que é associado
- É possível atribuir um tipo de variável usando funções Python que explicitamente atribuem a um elemento, desde que este seja válido – ***Casting*** ou **Construtores**
- Output do tipo de variável
- *Case-sensitive*

# Nome das variáveis

- Deve começar com uma letra ou com o caracter *underscore* (`_`)
- Não pode começar com numero
- O nome de variável só pode conter caracteres alfanuméricos, e *underscore* (`A a z, 0 a 9, _`)
- Variáveis são *case-sensitive*

# Múltiplas variáveis

- Atribuir diferentes valores a variáveis na mesma linha de comando
- Podem ser tipos de variáveis diferentes
- Atribuir o mesmo valor a múltiplas variáveis
- Atribuição direta de uma variável do tipo lista para variáveis individuais

# Comandos de *Input* e de *Output*

- Usa-se o comando *input()* para o utilizador introduzir dados
- Usa-se o comando *print()* para colocar um *output* para o utilizador
- Usa-se o sinal + para combinar texto no comando print() e uma variável do tipo *string*
- Usa-se o sinal + para unir duas ou mais variáveis do tipo *string*
- Em números, o sinal + funciona como operador matemático
- No comando print(), usa-se \n para mudar de linha
- Para inserir elementos específicos, numa ordem à escolha, deve-se usar as referências %, em que
  - print (“numero int %d”, 1) – elemento do tipo inteiro inserido no output
  - print (“numero float %f”, 1.5) – elemento do tipo float inserido no output
  - print (“string %s”, ‘palavra’) – elemento do tipo string inserido no output

# Tipos de dados - Singulares

- **Integer**
  - Numero sem casas decimais, na base de 10 (decimal), limitado pela memória do sistema que se utiliza
- **Float**
  - Numero com casas decimais, sendo representado por ‘,’ ou notação científica ‘e’ ou ‘E’ (ex.  $10 \times 10^2$  – 10e2)
  - Tem uma precisão de 64 bits, de dupla precisão de acordo com o IEEE 754, de  $5e-324$  a  $1.79e308$ .
- **Complex**
  - Especificado pela parte real e imaginária dos numeros complexos (ex.  $2+1j$ )
- **String**
  - Conjunto de caracteres, especificados por ' ' ou " "
- **Bool**
  - Especificado pelas duas condições de *True* e *False*
- **Bytes**
  - Conjunto de caracteres, especificados por b' ' ou b" ", em que o interpretador compila em bytes



# Tipos de dados – Coleção

- *List* – Especificado em [ ... , ... , ... ]
  - Ordenados – Têm sempre a mesma ordem
  - Indexados – cada item tem o seu índice, começando em lista[0], lista[1]...
  - Alteráveis – Permite alterar, remover ou adicionar o seu conteúdo, depois de ser criado
  - Permitem duplicados
- *Tuple* – Especificado em ( ... , ... , ... )
  - Ordenados – Têm sempre a mesma ordem
  - Indexados – cada item tem o seu índice, começando em tuple[0], tuple[1]...
  - Não alteráveis – O seu conteúdo não pode ser alterado, depois de este ser criado
  - Permitem duplicados
- *Set* – Especificado em { ... , ... , ... }
  - Não ordenados – Não possui sempre a mesma ordem
  - Não indexados – Não se consegue aceder a um elemento através do índice
  - Alteráveis – O seu conteúdo pode ser alterado depois de este ser criado, mas não permite adicionar elementos
  - Não permitem duplicados
- *Dict* – Especificado em { ... : ... , ... : ... , ... : ... }
  - Ordenados – Têm sempre a mesma ordem, nas versões de Python 3.7 e acima
  - Indexados – cada item tem o seu índice, começando em dict[0], dict[1]...
  - Alteráveis – Permite alterar, remover ou adicionar o seu conteúdo, depois de ser criado
  - Não permitem duplicados

# Tipos de dados – Funções implícitas

- *Range*
  - Cria uma sequência de números, em que se pode definir o início, fim e o passo
  - ( `x = range(start, stop, step)` )
- *Frozenset*
  - Transforma uma coleção do tipo *list* em não alterável
  - ( `x = frozenset(list)` )
- *Bytearray*
  - Cria uma coleção de bytes definida
  - ( `x = bytearray(n)` )
- *Memoryview*
  - Especifica a vista de memória de um objeto do tipo *byte*
  - ( `x = memoryview(byte)` )

# Operadores

- Aritméticos
  - **Soma (+)** – Soma de dois ou mais elementos. No caso de *strings*, junta todos os elementos na soma, formando uma. ( $2 + 2 = 4$ ; `'hello' + 'world' = 'helloworld'`)
  - **Subtração (-)** – Subtração de dois ou mais elementos ( $2 - 2 = 0$ )
  - **Multiplicação (\*)** – Multiplicação de dois ou mais elementos ( $2 * 2 = 4$ )
  - **Divisão (/)** – Divisão de dois ou mais elementos ( $2 / 2 = 1$ )
  - **Resto divisão (%)** – Resultado do resto da divisão de dois elementos ( $2 \% 2 = 0$ )
  - **Expoente (\*\*)** – Potência de um elemento ( $2 ** 4 = 16$ )
  - **Quociente divisão (//)** – Resultado do quociente de uma divisão ( $2 // 2 = 1$ )
- Comparação
  - **Comparação igual (==)** – Verifica se dois ou mais elementos são iguais ( $1 == 1 = \text{True}$ )
  - **Comparação diferente (!=)** – Verifica se dois ou mais elementos são iguais ( $1 != 2 = \text{True}$ )
  - **Maior que (>)** - Verifica se dois ou mais elementos são maiores que o seguinte ( $2 > 1 = \text{True}$ )
  - **Menor que (<)** - Verifica se dois ou mais elementos são menores que o seguinte ( $1 < 2 = \text{True}$ )
  - **Maior ou igual que (>=)** - Verifica se dois ou mais elementos são maiores ou iguais que o seguinte ( $1 >= 1 = \text{True}$ )
  - **Menor ou igual que (<=)** - Verifica se dois ou mais elementos são menores ou iguais que o seguinte ( $1 <= 2 = \text{True}$ )
- Lógicos
  - **Operação AND (and)** – Comparação AND entre duas condições ( $2 > 1$  **and**  $1 <= 0 = \text{True}$ ).
  - **Operação OR (or)** – Comparação OR entre duas condições ( $2 > 1$  **or**  $1 > 0 = \text{True}$ )
  - **Operação NOT (not)** - Inverte o resultado de uma comparação entre duas ou mais condições (**not**( $2 > 1$  **and**  $1 <= 0$ ) =  $\text{False}$ )
- Identidade
  - **Operação IS (is)** – Verifica se duas variáveis são o mesmo objeto ( $x$  **is**  $y$ )
  - **Operação IS NOT (is not)** – Verifica se duas variáveis não são o mesmo objeto ( $x$  **is not**  $y$ )

# Operadores

- Membro
  - **Operação IN (in)** – Verifica se um elemento está contido numa variável de múltiplos elementos (lista, tuple, set,..)
  - **Operação NOT IN (not in)** – Verifica se um elemento está não contido numa variável de múltiplos elementos (lista, tuple, set,..)
- *Bits* em inteiros
  - **Operação AND (&)** – Comparação AND entre duas variáveis (  $x \& y$  )
  - **Operação OR (|)** – Comparação OR entre duas variáveis (  $x | y$  )
  - **Operação XOR (^)** – Comparação XOR entre duas variáveis (  $x \wedge y$  )
  - **Operação inversão (~)** – Inverte todos os bits ( $\sim x$ )
  - **Operação *shift-left* (<<)** – deslocação de *bits* à esquerda acrescentado zeros ( $x \ll 3$ )
  - **Operação *shift-right* (>>)** – deslocação de *bits* à direita acrescentado zeros ( $x \gg 3$ )
- Atribuição
  - **Igualar (=)** – Atribuição de elemento a uma variável ( $x = 5$ )
  - **Incrementar (+=)** – incrementar o valor de uma variável. ( $x += 1$ )
  - **Decrementar (-=)** – decrementar o valor de uma variável. ( $x -= 1$ )
  - **Multiplicar e atribuir (x\*=)** – multiplicação de uma variável e atribuição. ( $x *= 3$ )
  - **Dividir e atribuir (/=)** – divisão de uma variável e atribuição. ( $x /= 3$ )
  - **Resto de divisão e atribuir (%=)** – resto de divisão de uma variável e atribuição. ( $x \% = 3$ )
  - **Quociente divisão e atribuir (//=)** – quociente de divisão de uma variável e atribuição. ( $x //= 3$ )
  - **Expoente e atribuir (\*\*=)** – expoente de uma variável e atribuição. ( $x ** = 3$ )
  - **AND e atribuir (&=)** – Comparação AND e atribuição. ( $x \& = 1$ )
  - **OR e atribuir (|=)** – Comparação OR e atribuição. ( $x | = 3$ )
  - **XOR e atribuir (^=)** – Comparação XOR e atribuição. ( $x \wedge = 3$ )
  - **Shift-right e atribuir (>>=)** – Deslocação de bits à direita e atribuição. ( $x \gg = 3$ )
  - **Shift-left e atribuir (<<=)** – Deslocação de bits à esquerda e atribuição. ( $x \ll = 3$ )

# Referências

- <https://www.python.org/>
- <https://www.python.org/dev/peps>
- <https://github.com/>
- <https://en.wikipedia.org/>
- <https://www.plcnext-community.net/en/hn-business-lounge/559-python-in-industrial-automation.html>
- <https://techvidvan.com/tutorials/python-interpreter/>