UFCD 10793_3/N – Fundamentos de Python

João Araújo

16-10-2024

PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU

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
- Bibiotecas
  - ❑ 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

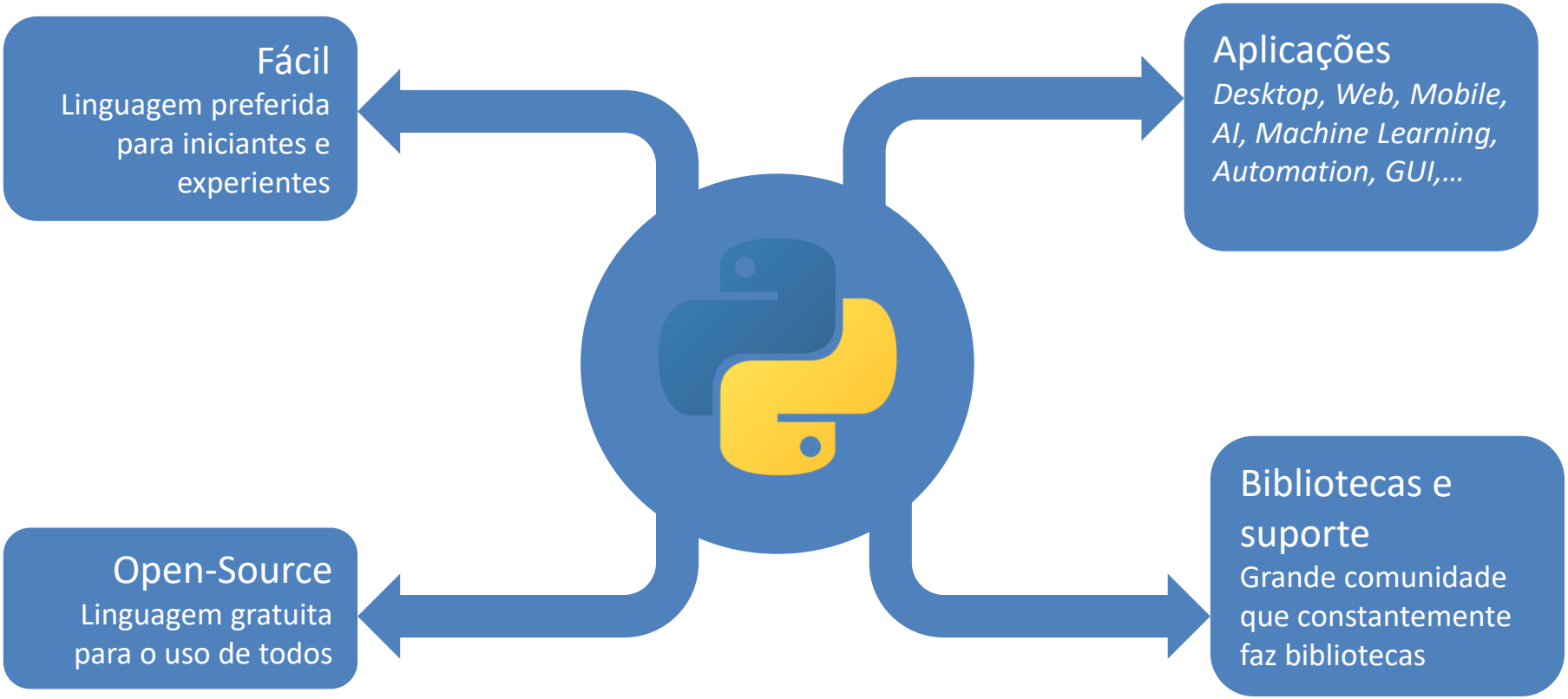
- O que é Python?
  - Python é uma linguagem de programação de uso **geral**, **alto nível**, **interpretada**, e sintaxe de uso fácil
  - Atualmente está na versão 3.12.5
  - Criada em 1989 por Guido Van Rossum, como sucessora da linguagem ABC
  - Primeira aparição em 1991
  - Leitura orientada de código
    - 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 regas
  - 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 obvia 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!

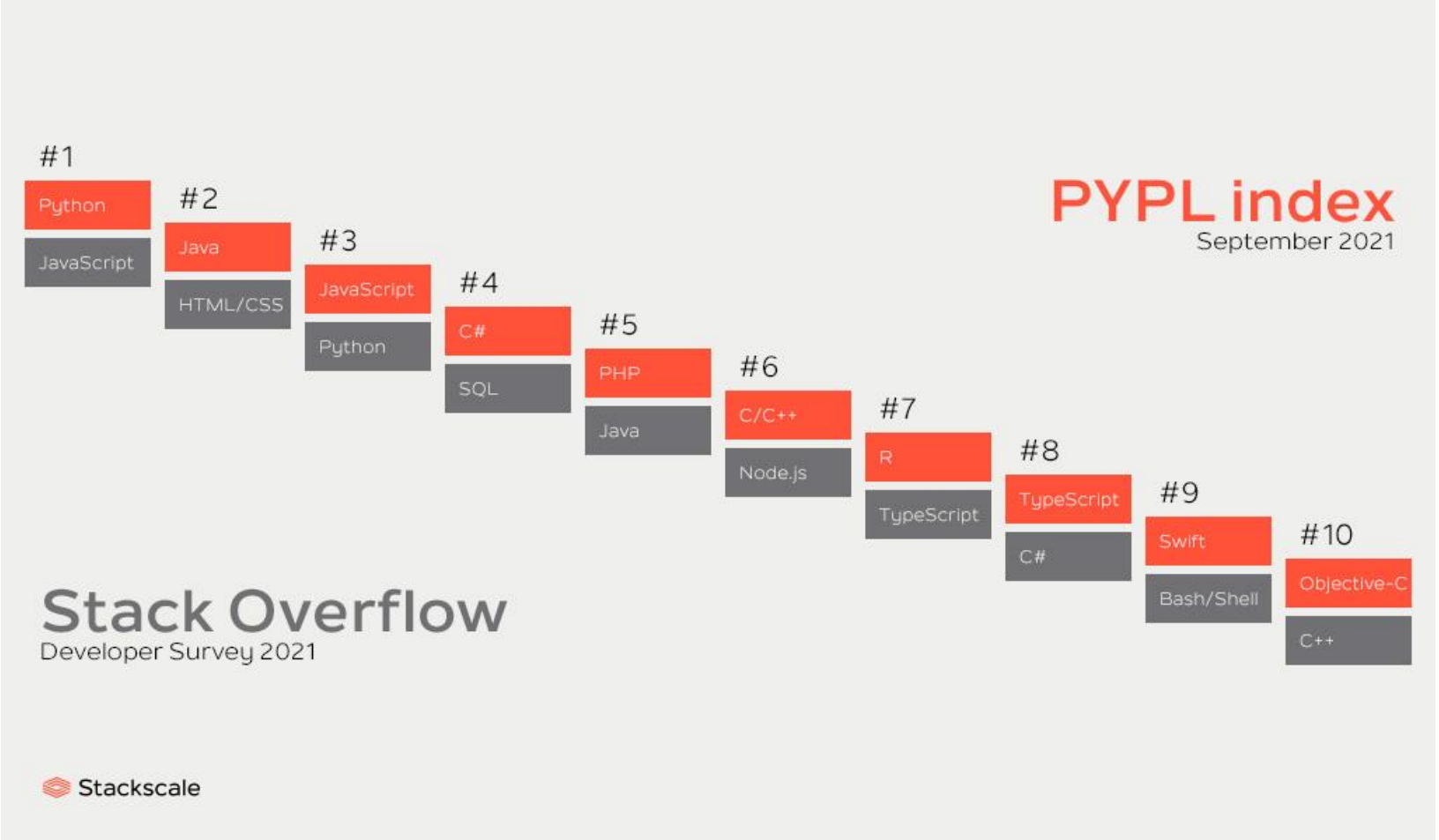
Introdução

- Porque o Python é popular?



Introdução

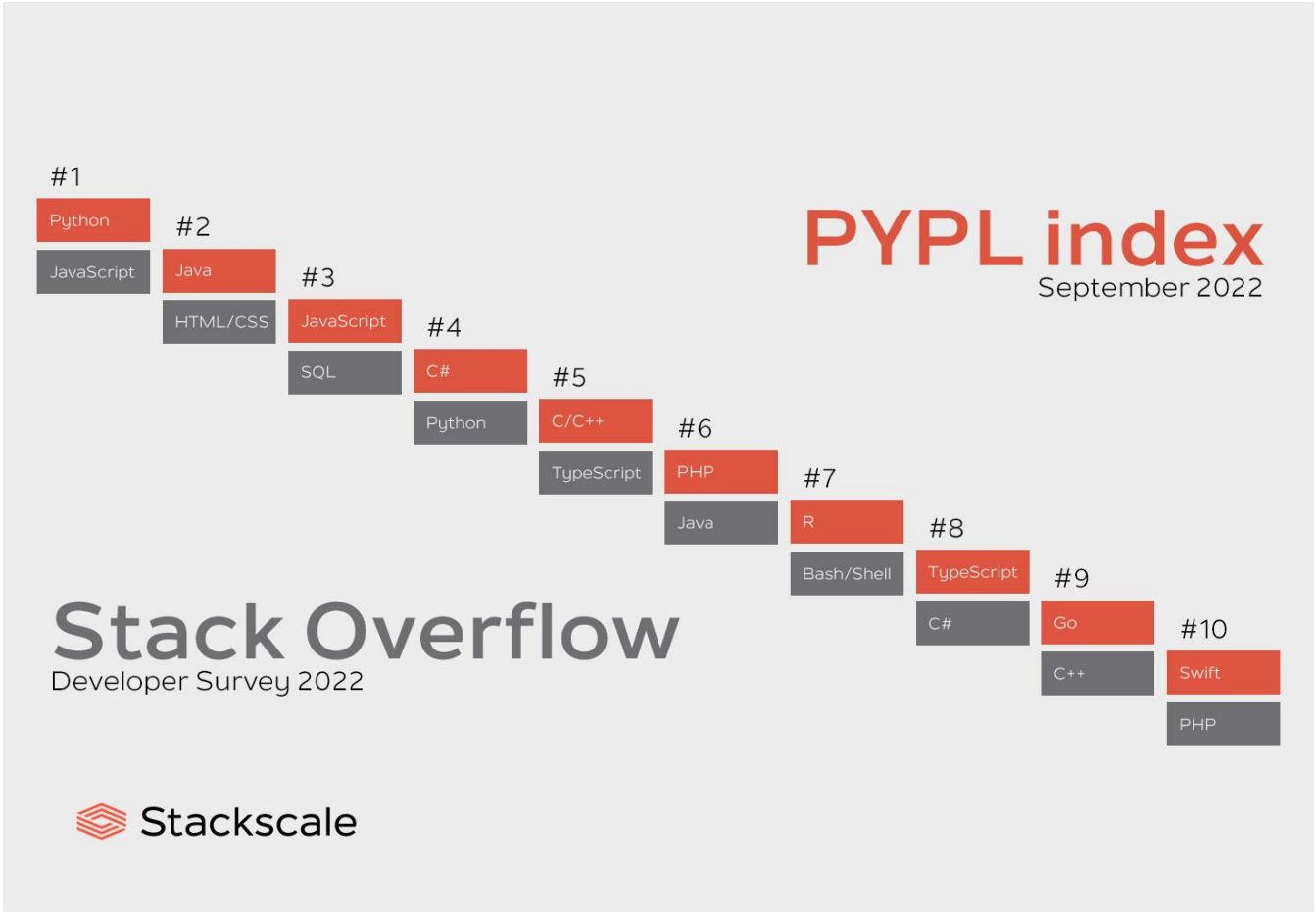
- Popularidade do Python



Fonte: PYPL e Stack Overflow

Introdução

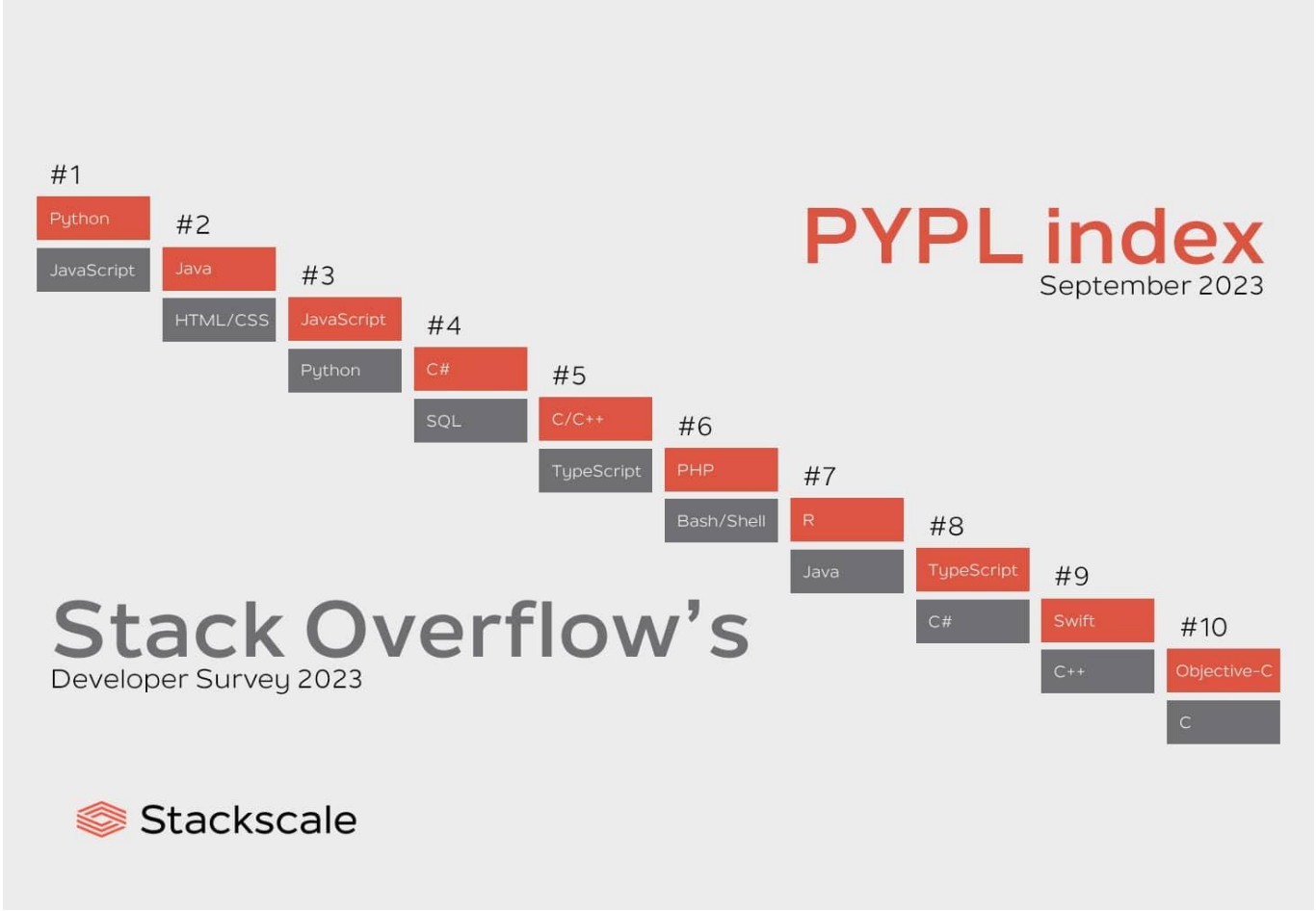
- Popularidade do Python



Fonte: PYPL e Stack Overflow

Introdução

- Popularidade do Python

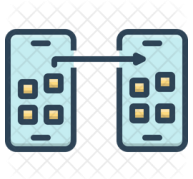


Fonte: PYPL e Stack Overflow

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 especificas, 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 *Strings*	*Strings* armazenadas como ASCII, por defeito	*Strings* armazenadas como UNICODE, por defeito
Divisão de inteiros	Nesta operação, obtem-se apenas a componente inteira da divisão	Nesta operação, obtem-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 compativeis com as versões seguintes	As bibliotecas apenas são compativeis 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 áeras como Software Engineering, Data Science, etc.

Versões de Python 3 e funções

Data de lançamento	Versão Python	Funcionalidades
13-03-2015	3.5.9	•Novas co-rotinas com *async* e *await* •Novo operador de matrizes a @ b •Acrescento de formatação % par *byte* e *bytearray* •Novos métodos *bytes.hex()*, *bytearray.hex()* e *memoryview.hex()* •*Memorybiew* suporta tuple •Atualização de geradores •Novos erros de recursão •Melhoramentos do Cpyton •Melhoramentos das bibliotecas standard •Melhoramentos de segurança
23-12-2016	3.6.15	•**Strings formatadas *fstrings*** •Underscore em numeros, para melhor legibilidade •Descrição de funções embutidas •Geradores assincronos •Criar co-rotinas assincronas •Melhoramentos das bibliotecas standard •Melhoramentos de segurança
27-06-2018	3.7.12	•Variáveis contextuais •Classes de dados •Importlib.resources •Função *breakpoint()* •Acesso personalizado a atributos de módulos •Suporte para escrita de módulos nucleares e tipos genéricos •**Manter a ordem de inserção no tipo de dado de dicionários** •Melhoramentos das bibliotecas standard •Melhoramentos de segurança

Versões de Python 3 e funções

Data de lançamento	Versão Python	Funcionalidades
14-10-2019	3.8.12	**•Nova expressão de :=, em que atribui um valor a uma variavel, fazendo parte de uma expressão maior** **•Argumentos de funções podem ser definidos como posicionais, posicionais ou por palavra chave, e palavra-chave obrigatoriamente**
02-01-2022	3.9.9	**•Operadores de união e atualização acrescentados em dicionários** **•Ajuda em variáveis do tipo de coleção de dados** **•Remoção de prefixos e sufixos em *strings*** •Melhoramentos das bibliotecas standard •Melhoramentos de segurança
02-01-2022	3.10.1	•Melhoramentos no interpetador •Documentação de sintaxe para normalização •Preparação da remoção de PyUnicodeObject •Especificação de variáveis de parâmetros
02-04-2023	3.11.0	•Melhoramento da localização de erros •Exception Groups e except* •Adicionar notas nas exeções •Melhoramento da velocidade de execução
06-08-2024	3.12.5	•PEP 695: Sintaxe de tipo de parametros em funções genéricas •PEP 701: Formalização sintaxe de fstring •PEP 684: Interpetador GIL – melhoramento da velocidade de execução •Mensagens de erro melhoradas

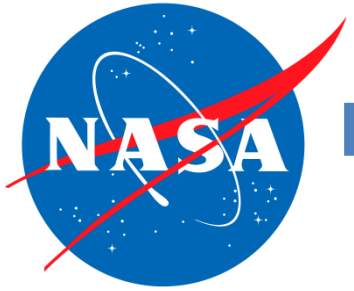
Python na indústria

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

Aplicações de cliente e servidor são feitas em Python

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

PHOENIX CONTACT

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

Oportunidades de carreira com Python

Web Development

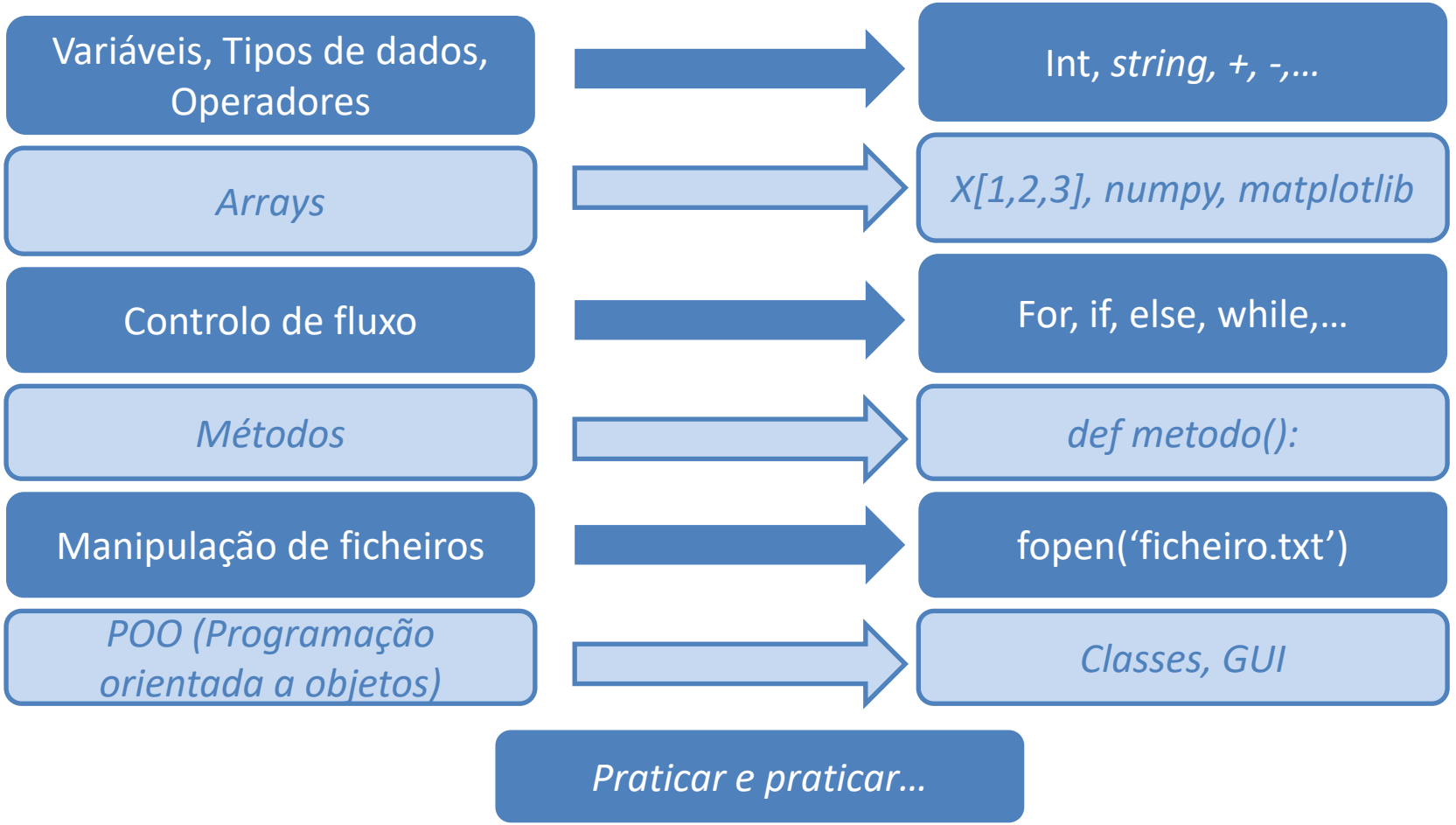
Web Testing

IOT

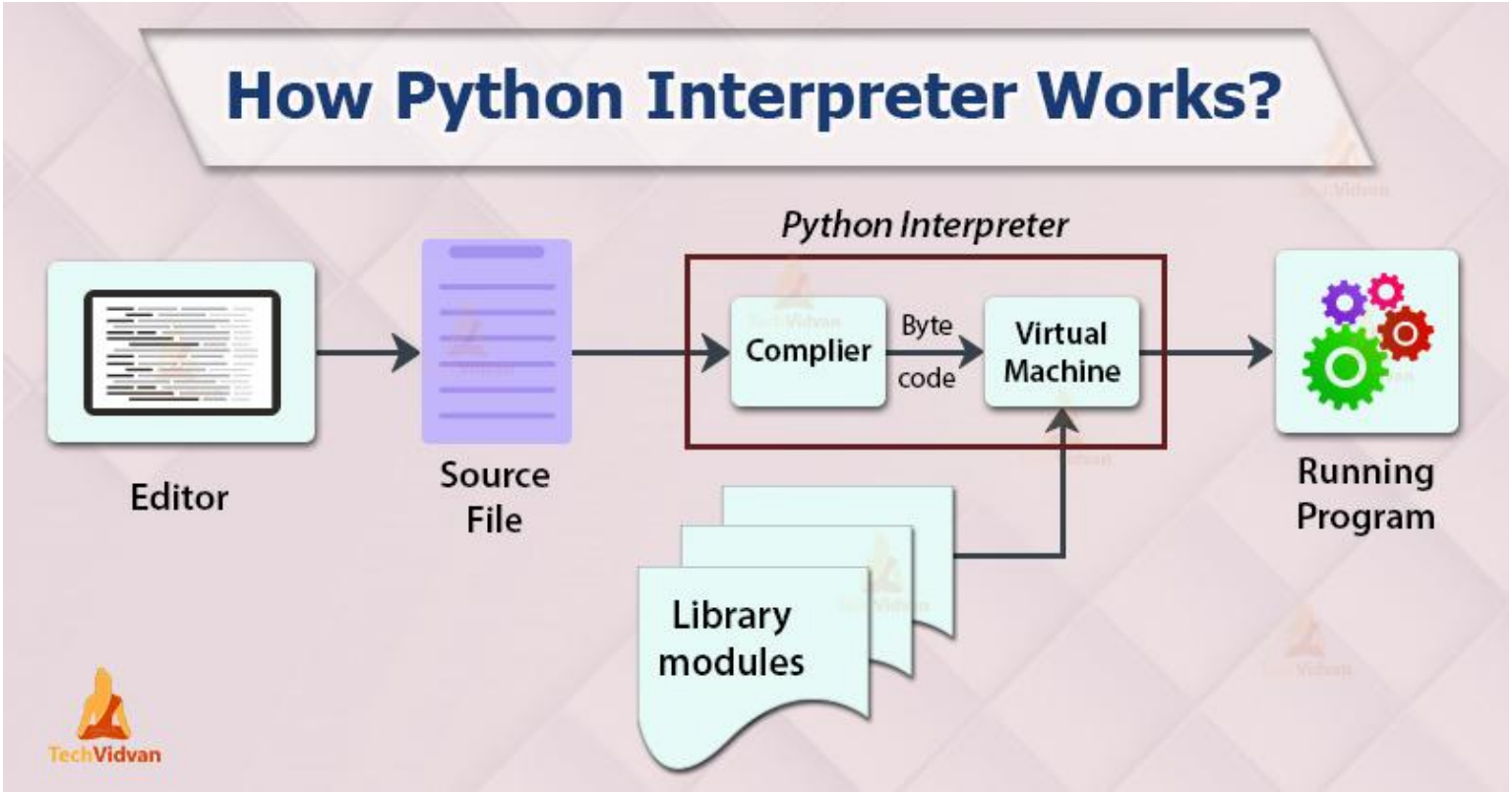
Game Development

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 virgula ;)

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

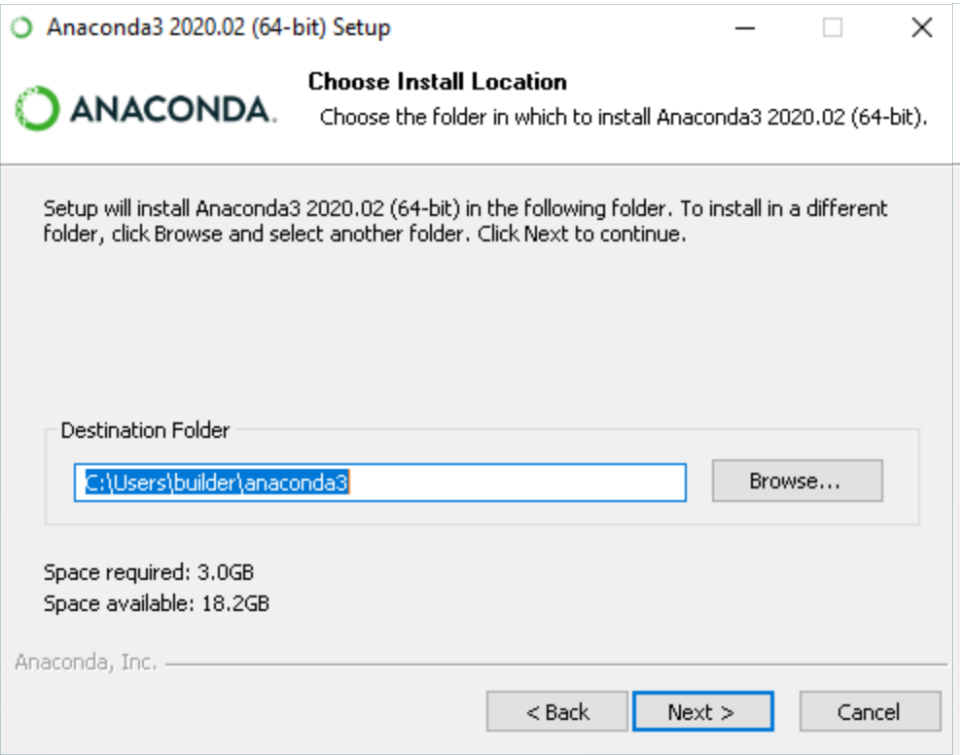
PEP8 – Style Guide for Python Code

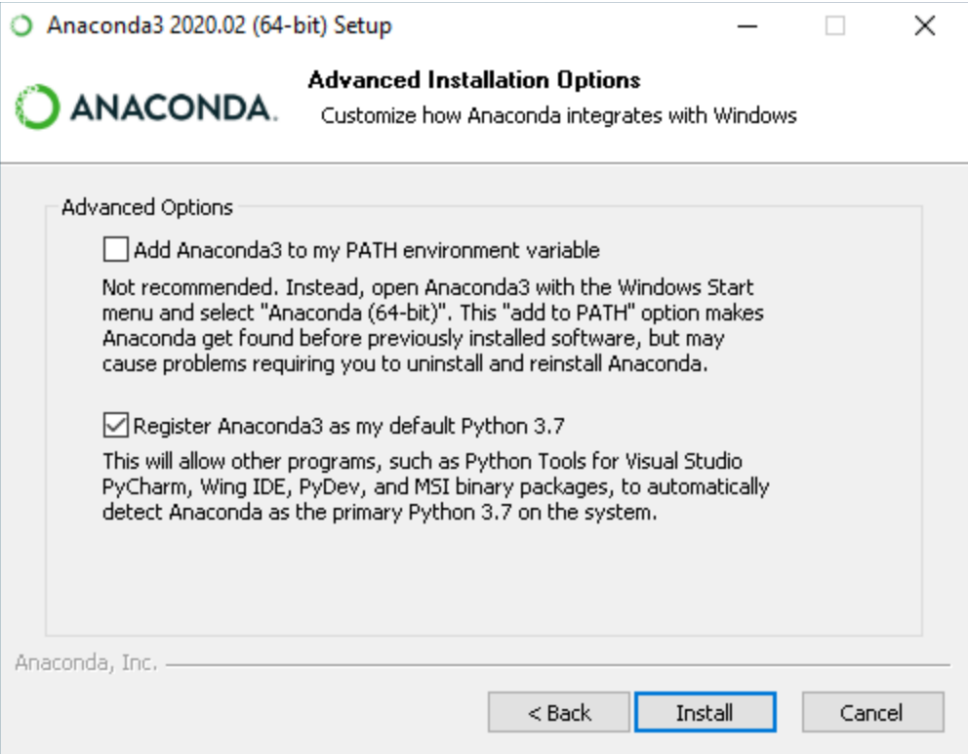
Software necessário

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 Sudio Code
  - Editor multi-linguagens de programação
  - Várias areas de aplicação (Web, UI, servidores,...)
  - Necessario instalar pacotes adicionais para utilizar as funcionalidades extra (Explorador de variáveis, correr areas 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

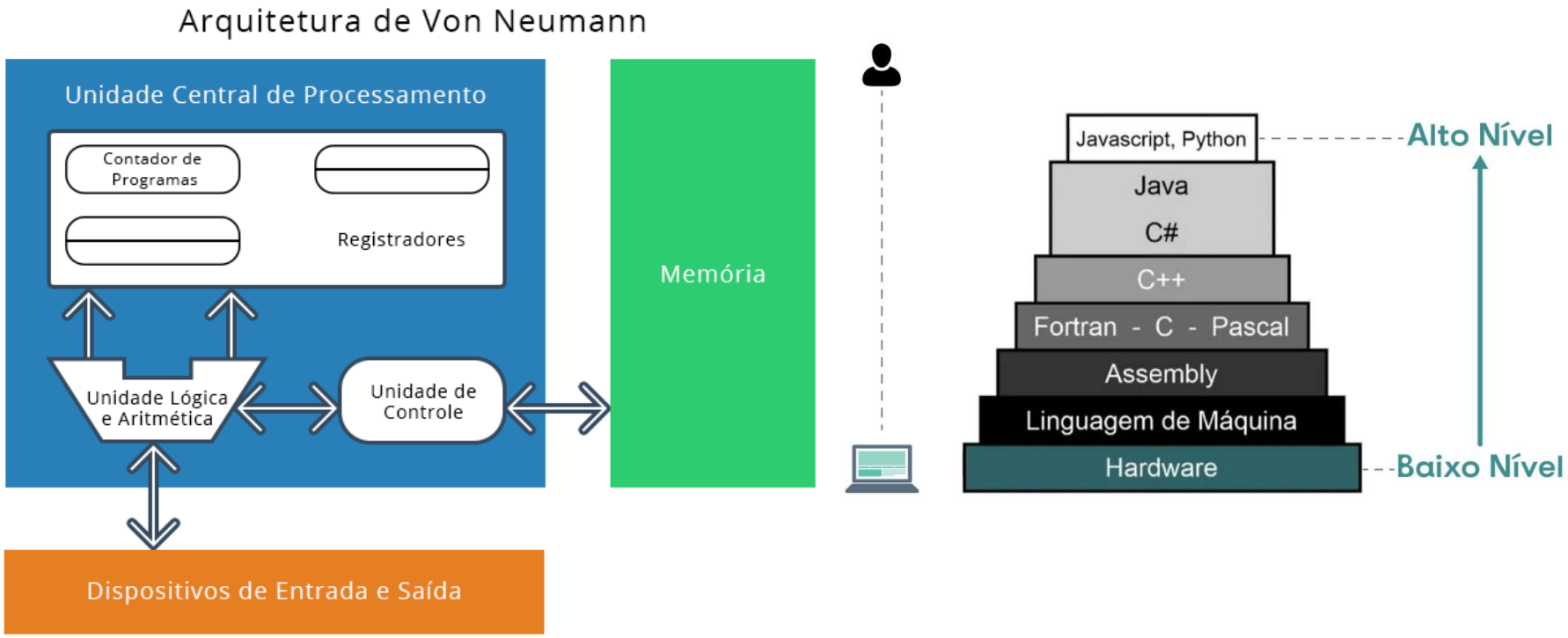
Jupyter notebook

- 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

Arquitectura de computadores

- Praticamente todos os sistemas compotacionais baseados na arquitectura von Neumann
- CPU – Unidade central de processamento que contém:
  - Unidade de controlo (UC): Responsavel por controlar o fluxo de dados e quando deve ocorrer transferências entre unidades
  - Unidade Aritmética e Lógica (ALU): responsável pelas operações lógicas e operações aritméticas
  - Registos: pequena memória de alta velocidade que armazena resultados temporários e realiza o controle de informações.
- Memória - responsável por armazenar dados e instruções;
- Dispositivos de entrada e saída - responsáveis por fornecer a entrada e saída de dados.

Arquitectura de computadores / linguagens de programação



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 cientifica ‘e’ ou ‘E’ (ex. 10x10^2 – 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 comlexos (ex. 2+1j)
- *String*
  - Conjunto de carateres, especificados por ' ' ou " "
- *Bool*
  - Especificado pelas duas condições de *True* e *False*
- *Bytes*
  - Conjunto de carateres, 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 indice
  - 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 sequencia de números, em que se pode definir o inicio, 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 = True)
  - **Comparação diferente (!=)** – Verifica se dois ou mais elementos são iguais (1 != 2 = True)
  - **Maior que (>)** - Verifica se dois ou mais elementos são maiores que o seguinte (2 > 1 = True)
  - **Menor que (<)** - Verifica se dois ou mais elementos são menores que o seguinte (1 < 2 = True)
  - **Maior ou igual que (>=)** - Verifica se dois ou mais elementos são maiores ou iguais que o seguinte (1 >= 1 = True)
  - **Menor ou igual que (<=)** - Verifica se dois ou mais elementos são menores ou iguais que o seguinte (1 <= 2 = True)
- Lógicos
  - **Operação AND (and)** – Comparação AND entre duas condições (2 > 1 **and** 1 <= 0 = True).
  - **Operação OR (or)** – Comparação OR entre duas condições (2 > 1 **or** 1 > 0 = True)
  - **Operação NOT (not)** - Inverte o resultado de uma comparação entre duas ou mais condições (**not(**2 > 1 **and** 1 <= 0 **)** = 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 multiplos elementos (lista, tuple, set,..)
  - **Operação NOT IN (not in)** – Verifica se um elemento está não contido numa variável de multiplos elementos (lista, tuple, set,..)
- *Bits* em inteiros
  - **Operação AND (&)** – Comparação AND entre duas variáveis (x & y)
  - **Operação OR (|)** – Comparação AND entre duas variáveis (x | y)
  - **Operação XOR (^)** – Comparação AND entre duas variáveis (x ^ y)
  - **Operação inversão (~)** – Inverte todos os bits (~x)
  - **Operação *shift-left* (<<)** – deslocação de *bits* à esquerda acrescentado zeros (x << 3)
  - **Operação *shift-right* (>>)** – deslocação de *bits* à direita acrescentado zeros (x >> 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 ^= 3)
  - **Shift-right e atribuir (>>=)** – Deslocação de bits à direita e atribuição. (x >>= 3)
  - **Shift-left e atribuir (<<=)** – Deslocação de bits à direita e atribuição. (x <<= 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/