Introdução ao Python

Introdução e Conceitos Básicos

Joyce Teixeira - jvt@cesar.school

C.E.S.A.R school

POR QUE PROGRAMAR?

Steve Jobs

"Todos neste país deveriam aprender a programar um computador, pois isto ensina a pensar."

Bill Gates

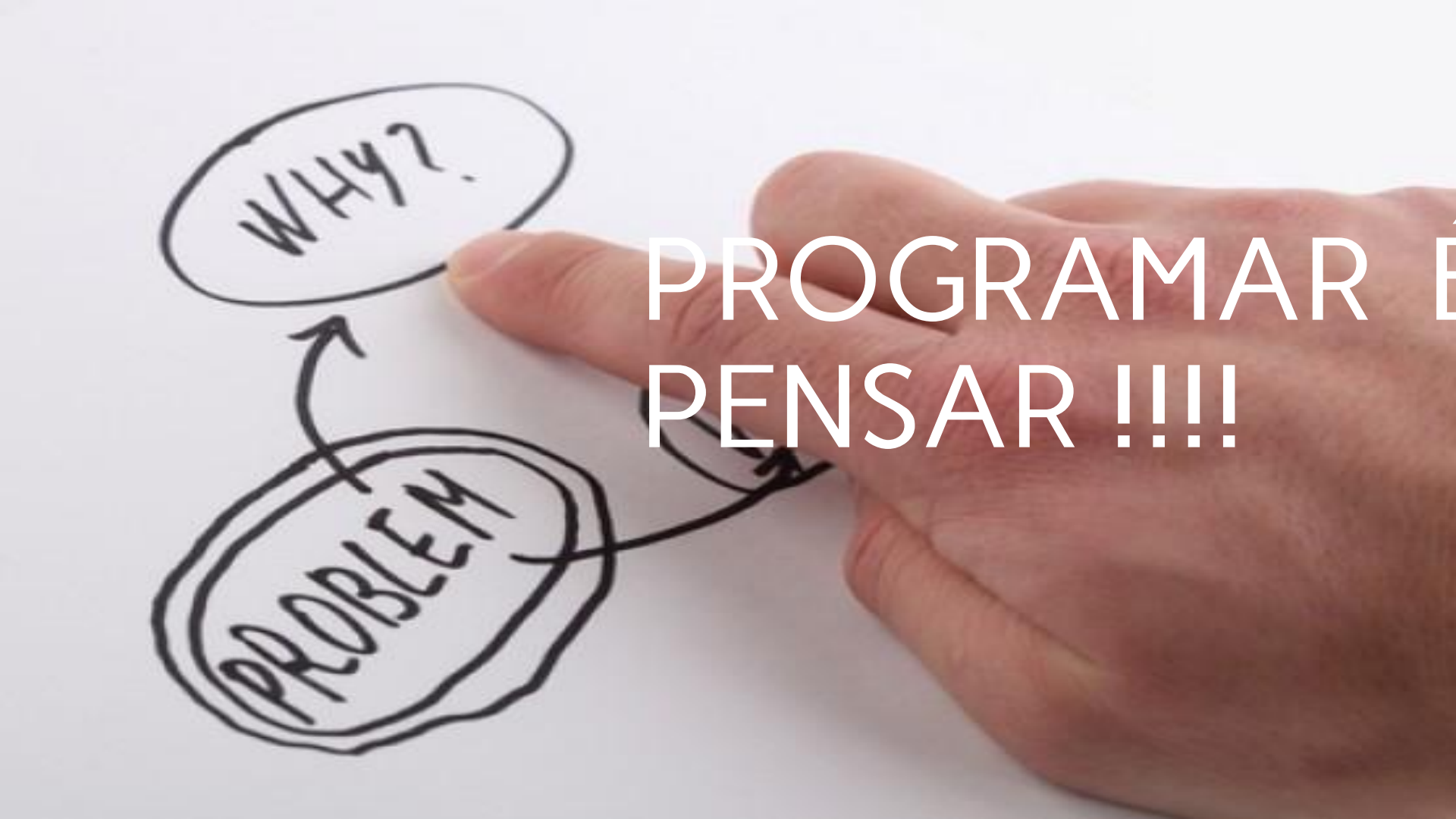
"Aprender a escrever programas expande sua mente e o ajuda a pensar melhor, cria uma maneira de pensar sobre as coisas que são úteis em todos os domínios."

Silvio Meira

"Se você quer uma profissão do futuro, entenda de software, de programação. Isso vale para todas as áreas. A mola motriz da sociedade da informação e do conhecimento é software."


PROGRAMADORES CRIAM SOLUÇÕES PARA PROBLEMAS !


PROGRAMAR NÃO É
DIGITAR !!!



PROGRAMAR É PENSAR !!!!

Programar é a "menor" das partes do processamento de criar um software.

Trata-se da consolidação de todo Pensamento Computacional.

NÃO SEJA FANÁTICO POR UMA LINGUAGEM DE PROGRAMAÇÃO ESPECÍFICA!

Há uma infinidade de linguagens!

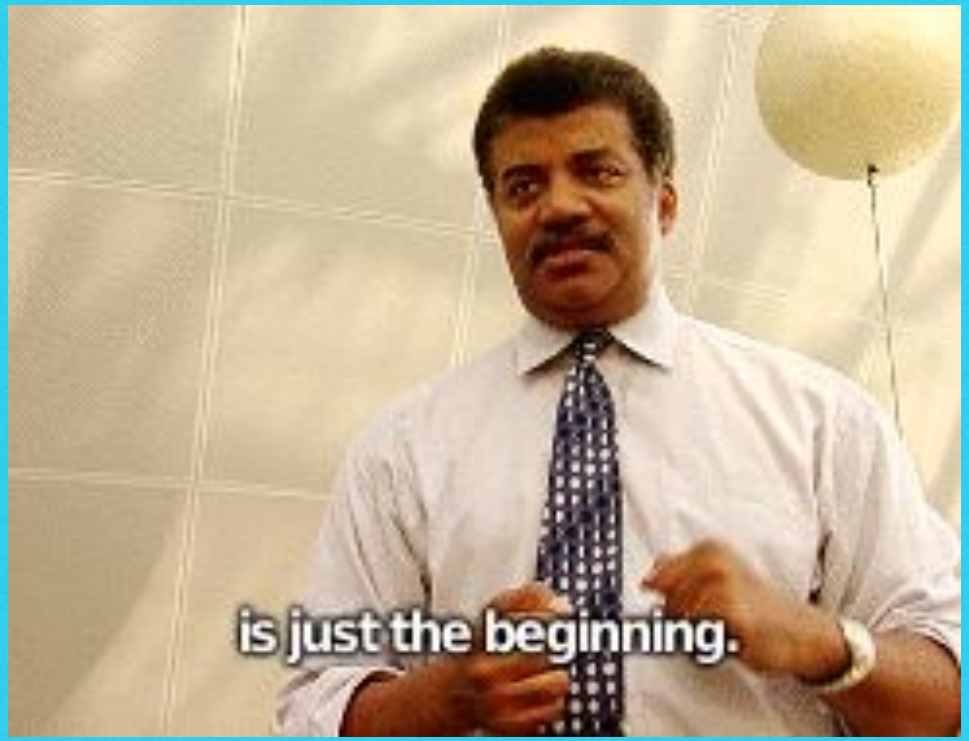

A LINGUAGEM "DA MODA"
HOJE, PODE NÃO ESTAR TÃO
EM ALTA AMANHÃ

APRENDA LÓGICA DE PROGRAMAÇÃO

Aprenda a pensar antes de aprender a programar.

E FINALMENTE...

ENJOY!!!!


is just the beginning.

Conceitos Básicos de Programação

Joyce Teixeira

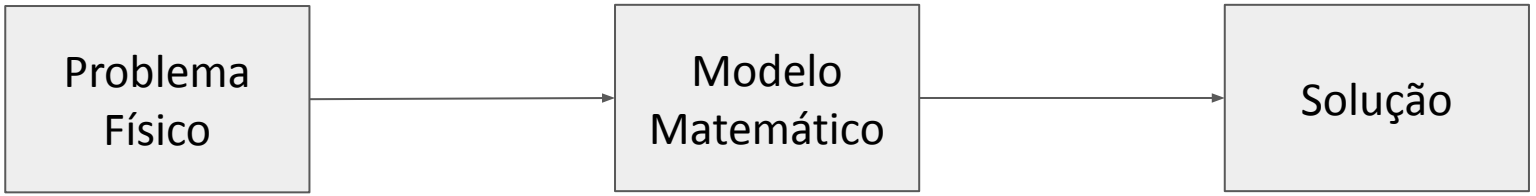
Roteiro

- *Problemas e Soluções*
- *Algoritmo*
- *Programa*
- *Linguagens de Programação*
- *Construção de Algoritmos*
- *Representação de Algoritmos*

Problemas e Soluções

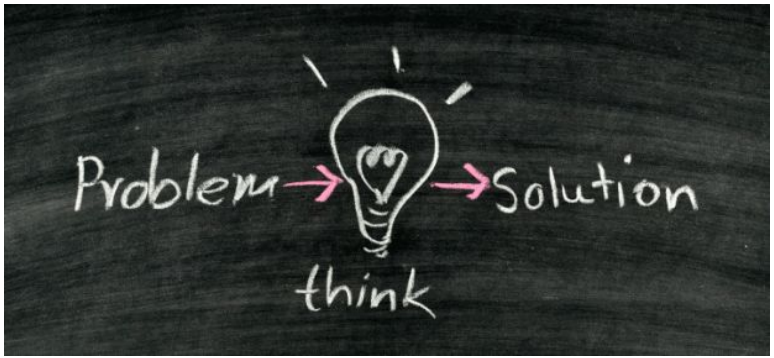
No mundo real temos:

- **Problemas**: representam situações do mundo físico que podem ser transformadas em modelos que os representem
- **Soluções**: são as respostas possíveis para problemas do mundo físico; utilizam métodos e cálculos para resolução



Problemas e Soluções

- *Pode existir **mais de uma solução** para os problemas*
- *Algumas soluções são melhores que outras sob algum aspecto*
- *Alguns problemas são casos particulares de outros similares*
- *Sempre que possível, é melhor resolver o problema mais genérico*



Problemas e Soluções

- Exemplo
  - Contar o número de pessoas em um evento

Contando pessoas em um evento

Solução 1: contar pessoa por pessoa, um por vez

- Não contar a mesma pessoa mais de uma vez
- Não se esquecer de contar alguém

Vantagens: simples, fácil de executar, não exige conhecimento prévio, sem recursos extras

Desvantagens: tempo de contagem (se sala for grande e estiver cheia), altas chances de errar

Contando pessoas em um evento

Solução 2: contar cadeiras vazias

Vantagens: simples, fácil de executar, sem recursos extras

Desvantagens: requer saber quantas cadeiras há, tempo de contagem (se sala for grande e estiver quase vazia)

E se fosse um comício?

Contando pessoas em um evento

Solução 3: contagem por estatística

Vantagens: mais rápido que outras soluções para públicos maiores

Desvantagens: deve-se saber a metragem da praça de antemão e estimar pessoas por metro quadrado, número impreciso

Contando pessoas em um evento

Solução 4: roletas/catracas

Vantagens: numero exato

Desvantagens: instalação dos equipamentos, ambiente controlado (evitar de burlar catracas)

Contando pessoas em um evento

Outras soluções

Solução 5: filas organizadas e de mesmo tamanho

Solução 6: trabalho distribuído em paralelo (cada primeiro conta sua fila e ao final soma-se os valores de cada fileira)

Algoritmo

- Algoritmo é uma sequência **finita** de instruções bem definidas e não ambíguas, cada uma das quais devendo ser executadas mecânica ou eletronicamente em um intervalo de tempo finito e com uma quantidade de esforço finita.
  - Quando executado, produz uma solução
  - Registra-se uma solução para um problema
  - Outra pessoa pode executar
- Resumindo: É uma sequência finita de passos visando resolver um determinado problema.

Algoritmo

- Então tudo pode ser representado por um algoritmo?
- Podemos construir um algoritmo para qualquer coisa?

Algoritmo

- Para um problema ser representado por um algoritmo, ele precisa ter lógica;
- Como encontrar a lógica?

Algoritmo

- Mas como dizer ao computador para executar esses passos?
  - Ele entende português?
  - Inglês?
  - Espanhol?
  - ...

Programa

Um **programa** é:

- Um algoritmo de forma detalhada tal qual o necessário
- Programa de computador, linguagem de computador
  - Quem faz vai entender?
  - Computador vai entender?

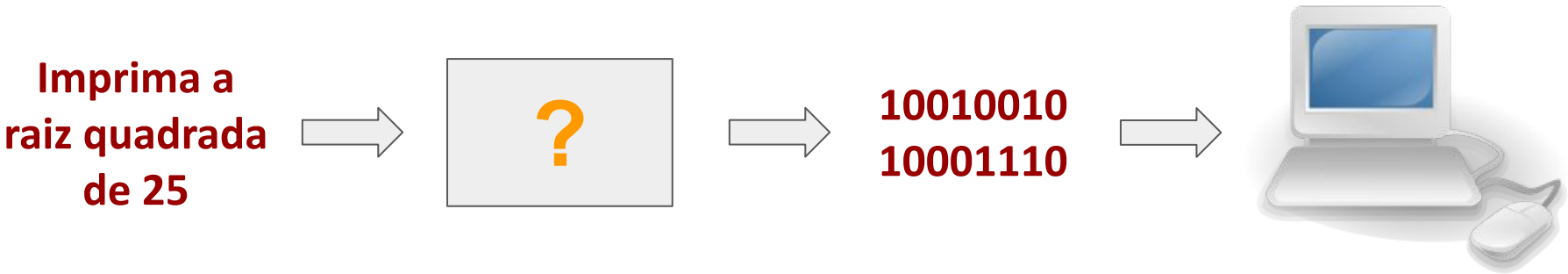
Linguagens de programação

- Os programas têm que ser escritos em uma linguagem de programação:
  - uma linguagem que pode ser **entendida** pelo computador

10010010
10001110

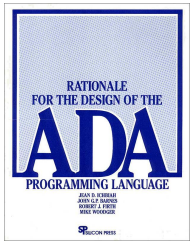
Linguagens de programação

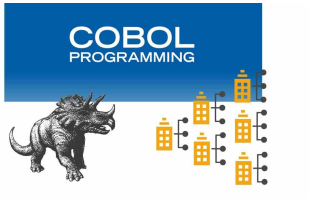
- Uma linguagem que **entendemos** e que possa ser **traduzida** para a linguagem entendida pelo computador

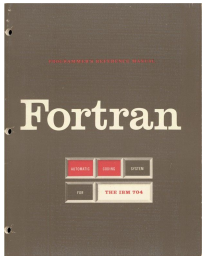


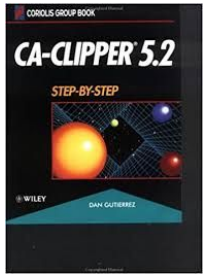
Linguagens de programação

- Existem muitas linguagens de programação









Linguagens de programação

- Usamos a programação para criar um programa através de algoritmos
- Um algoritmo deve passar por algumas fases antes de se tornar programa
- Escrevemos o algoritmo em uma linguagem
  - *O algoritmo deve ser processado, adaptado para o computador*
  - *Em linguagem do computador, o programa deve ser carregado em memória*

Linguagens de programação

- Linguagens existentes:
  - *Linguagem natural*
  - *Linguagem de alto nível*
  - *Linguagem de baixo nível*
  - *Linguagem de máquina*

Linguagens de programação

- Linguagens existentes:
  - **Linguagem natural**: São as línguas faladas e escritas pelos seres humanos para se comunicarem entre si. Elas são complexas e cheias de nuances, ambiguidades e contextos. Exemplo: português, inglês, alemão, etc.

Linguagens de programação

- Linguagens existentes:
  - **Linguagem de alto nível**: São linguagens de programação mais próximas da linguagem humana e mais distantes da linguagem de máquina. São mais fáceis de ler e escrever, e geralmente são independentes de plataforma. Exemplo: C, C++, Java, Pascal, Fortran, Cobol, Python

```
print("Olá, Mundo!")
```

Linguagens de programação

- Linguagens existentes:
  - **Linguagem de baixo nível**: São linguagens de programação mais próximas da linguagem de máquina e mais distantes da linguagem humana. Elas são mais difíceis de ler e escrever, mas podem ser mais eficientes em termos de desempenho. Exemplo: Assembly.

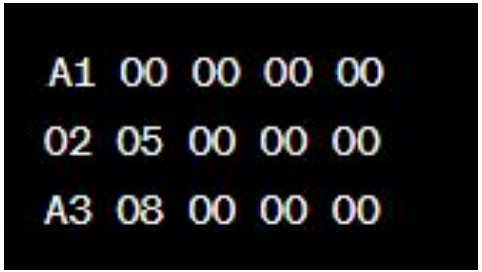
Linguagens de programação

- Linguagens existentes:
  - **Linguagem de baixo nível**:

```
section .data
hello db 'Olá, Mundo!',0

section .text
global _start

_start:
    ; escrever 'Olá, Mundo!' para a tela
    mov eax, 4
    mov ebx, 1
    mov ecx, hello
    mov edx, 11
    int 0x80

    ; sair do programa
    mov eax, 1
    int 0x80
```

Linguagens de programação

- Linguagens existentes:
  - **Linguagem de máquina**: É a linguagem de mais baixo nível, composta por códigos binários (frequentemente representados em formas como hexadecimal) que podem ser diretamente executados por um processador. É específica para cada arquitetura de processador.



Linguagens de programação

- Passos “grosseiros” para escrever um programa:
  - 1 – Entender o problema
  - 2 – Planejar a lógica
  - 3 – Escrever o programa (codificar)
  - 4 – Testar o programa
  - 5 – Escrever documentação
  - 6 – Otimizar o programa
  - 7 – Implantar o programa

Construção de Algoritmos

- Como um algoritmo é construído?

Representação de Algoritmos

- As formas de representação de algoritmos mais conhecidas são:
  - Descrição narrativa;
  - Fluxograma convencional;
  - Pseudo-código:
    - Linguagem estruturada.

Descrição Narrativa

- Os algoritmos são expressos em linguagem natural.

Trocar um pneu furado

1. Afrouxar ligeiramente as porcas
2. Suspender o carro
3. Retirar as porcas e o pneu
4. Colocar o pneu reserva
5. Apertar as porcas
6. Abaixar o carro
7. Dar o aperto final nas porcas

Calcular média de um aluno

1. Obter as suas 2 notas de provas
2. Calcular a média aritmética
3. Se a média for maior que 7, o aluno foi aprovado, senão ele foi reprovado

Fluxograma Convencional

- Representação do algoritmo em formato gráfico;
- Formas geométricas implicam ações distintas;
- Facilita o entendimento do algoritmo.

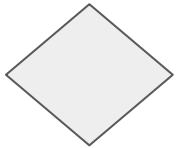
Fluxograma Convencional

Início ou final do fluxograma

Entrada de dados

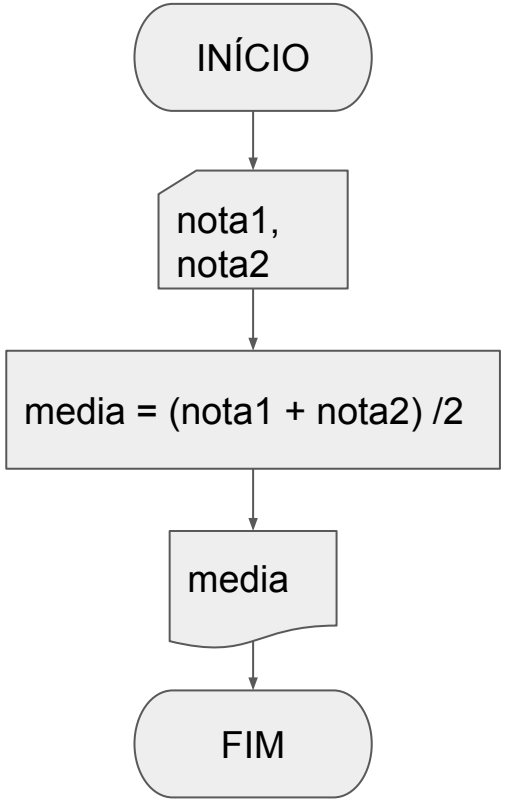
Saída de dados

Operação de atribuição



Decisão

Fluxograma Convencional



Como ficaria esse fluxograma se adicionássemos uma “decisão”?

Fluxograma Convencional

- Figuras conhecidas mundialmente;
- As figuras podem facilitar o entendimento do que deve ser feito no algoritmo;
- Não oferece recursos para descrever dados;
- A medida que o fluxograma cresce, o algoritmo fica complicado de entender.

Pseudo-código

- Informações ricas em detalhes;
- Assemelha-se com a forma em que os programas serão escritos.

```
programa
{

    funcao inicio()
    {
        inteiro nota_1 = 10
        inteiro nota_2 = 5
        inteiro media

        media = (nota_1 + nota_2) / 2

        escreva("A média é: ", media)
    }
}
```

Introdução ao Python

Joyce Teixeira

Roteiro

- Introdução
- Instalação e Preparação do Ambiente
- Primeiro Projeto
- Constantes e Variáveis
- Comandos de Entrada e Saída
- Operadores:
  - Atribuição
  - Aritméticos
  - Relacionais
  - Lógicos

Por que Python?

- Simples
- Acessível
- Fácil de usar

Um pouco de história...

- **Linguagem de programação:**
  - Desenvolvida por ***Guido van Rossum*** em 1989
  - Lançada em 1991 - versão 0.9.0
  - Linguagem de alto nível
  - Linguagem orientada a objetos
  - Linguagem interpretada
  - Duas grandes vertentes: **Python 2 (descontinuada em 04/2020)** e **Python 3**

Por que tão especial?

- Interoperabilidade (comunica-se de forma transparente com outros sistemas ou linguagens)
- Multiplataforma (pode ser utilizada em sistemas Linux, Mac OS ou Windows)
- Robustez
- Velocidade de aprendizado
- Simplicidade

Foi pensada para ser simples

- Trata-se de uma linguagem de alto nível
- Planejada para ser produtiva e de fácil entendimento
- Reduz a utilização de caracteres especiais (vírgula, dois pontos, ponto e vírgula, sinal de igual, chaves, etc)
- Possibilita que você determine uma sequência e monte uma lista com apenas uma linha de código
- Sintaxe clara e direta

Não se engane, é simples! Porém, robusta!

- Back-end de sistemas web
- Pesadas simulações de engenharia
- Processamento pesado de efeitos especiais de filmes
- Soluções de análise de dados (data analytics)
- Aprendizado de máquina (machine learning – ML)

Quem usa Python?

Projetada com foco no programador

- Ampla variedade de tipos básicos:
  - Números - ponto flutuante, complexos e inteiros longos de comprimento ilimitado
  - Strings (ASCII e Unicode)
  - Listas
  - Tuplas
  - Sequências
  - Conjuntos
  - Dicionários

Código aberto e comunidade Python

- Linguagem livre, de código aberto
- Programadores podem compartilhar problemas, soluções e binários da linguagem, sem ter que anexar as fontes
- Documentação extremamente rica e completa. Acesse o site oficial da linguagem
- Comunidade Python Brasil

Então, como começar?

- Instalação do interpretador Python
- Neste primeiro momento, instalação de uma IDE de desenvolvimento: PyCharm.

Verificando Versões Instaladas

- Verifique se há alguma versão do Python instalada em sua máquina:
- No terminal/console digite alguns dos comandos:
  - `py -V`
  - `python -V`
  - `python3 -V`

```
C:\Users\Matheus>py -V
'py' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.
```

```
C:\Users\Matheus>py -V
Python 3.7.7
```

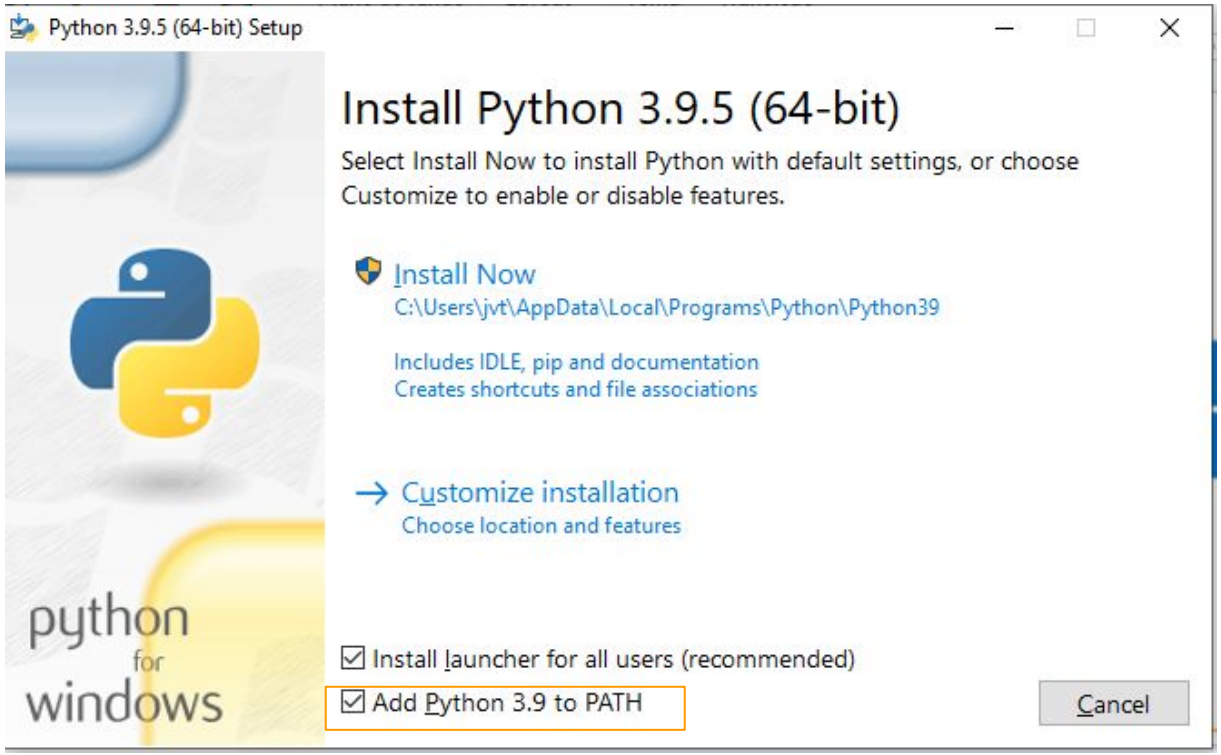
Preparando o ambiente

- Instalação do interpretador da linguagem Python
    - Ele permitirá que os códigos escritos na linguagem Python sejam executados pelo computador.
- Usaremos a versão 3.x do Python (Ex: Python 3.9.0)
    - Se você já possui uma versão mais antiga instalada, não tem problema. Várias versões podem coexistir no seu computador.
- Instalação do PyCharm

Instalando Python 3 - Windows

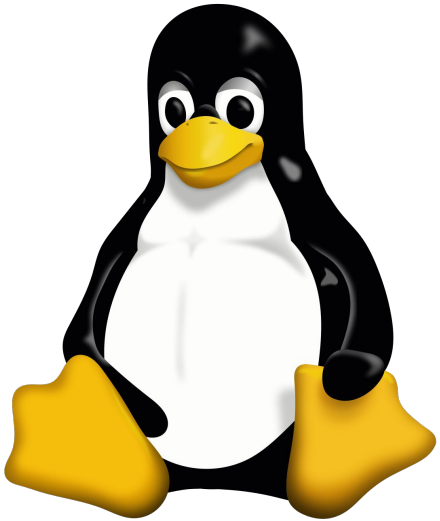
1. Visite https://www.python.org/downloads/windows/
2. Procure pela versão estável
3. Faça o download da versão estável tipo **Windows installer**
4. Instale o arquivo obtido no passo 3
5. É importante marcar a opção ***Add Python 3.x to PATH***

Instalando Python 3 - Windows



Instalando Python 3 - Linux

- O Python 3 já está presente nas últimas versões das distribuições mais famosas de GNU/Linux, como Ubuntu, Debian openSUSE e Fedora.
- Ubuntu/Debian/Mint:
  - **sudo apt-get update**
  - **sudo apt-get install python3.x**

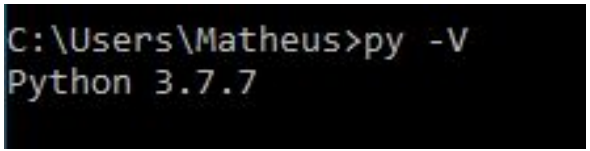


Instalando Python 3 - Mac OS

1. Através do Homebrew
    a. brew update
    b. brew install python3
2. Através do site
    a. Visite https://www.python.org/downloads/windows/
    b. Procure pela versão recomendada
    c. Faça o download de macOS 64-bit installer
    d. Instale o arquivo obtido no passo c
    e. É importante marcar a opção **Add Python 3.x to PATH**

Verificando a Instalação

- Verifique se a instalação do Python foi concluída com sucesso
- No terminal/console digite alguns dos comandos:
  - `py -V`
  - `python -V`
  - `python3 -V`



Interpretador Python

- Verificando o interpretador Python:
  - Modo ***shell***:

```
$ python3
Python 3.2 (r32:88445, Mar 25 2011, 19:28:28)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3
5
>>>
```

Instalando o PyCharm - Windows

1. Visite https://www.jetbrains.com/pycharm/download/#section=windows
2. Selecionar a versão ***Community***
3. Com o download finalizado, executa o arquivo .exe e next...next...next

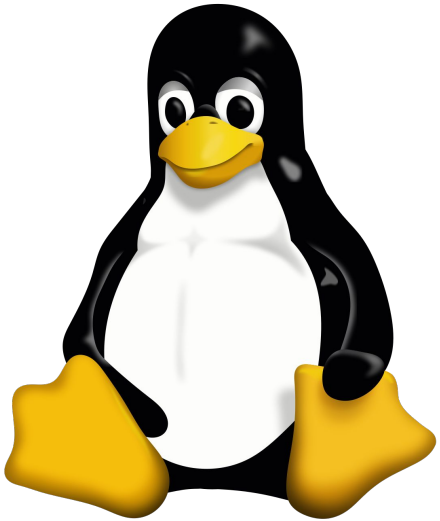
Instalando o PyCharm - Linux Ubuntu

- Há algumas maneiras de instalar o PyCharm no Linux - Ubuntu
- A mais fácil é via Ubuntu Software Center. Procure por **Pycharm CE**
- Caso não seja possível, há alternativas:
  - Usando **snap**

```
sudo snap install pycharm-community --classic
```

  - Usando **umake**

```
sudo add-apt-repository
ppa:ubuntu-desktop/ubuntu-make
sudo apt-get update
sudo apt-get install ubuntu-make
sudo snap install pycharm-community --classic

umake ide pycharm
```



C.E.S.A.R school

Instalando o PyCharm - Mac OS

1. Visite https://www.jetbrains.com/pycharm/download/#section=mac
2. Selecionar a versão ***Community***
3. Com o download finalizado, monta o arquivo:

```
$ hdiutil mount Downloads/pycharm-community-2019.2.2.dmg
```

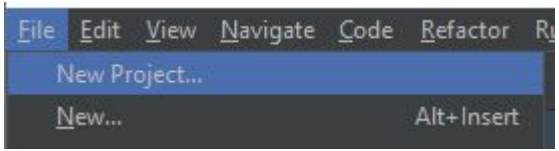
4. Para iniciar a instalação

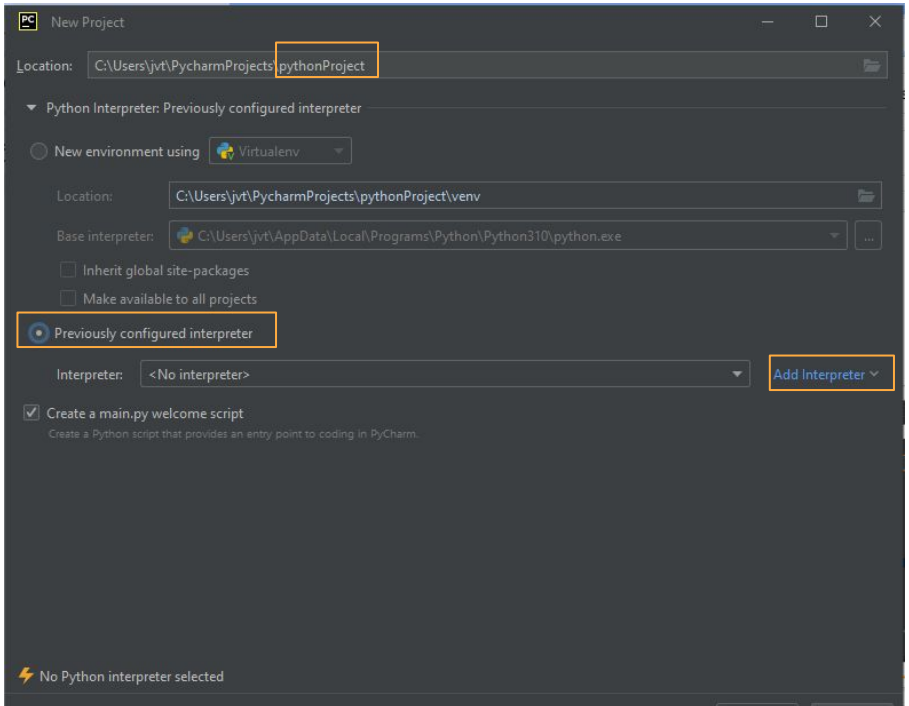
```
$sudo /Volumes/PyCharm\ CE/PyCharm\ CE.app/Contents/MacOS/pycharm
```

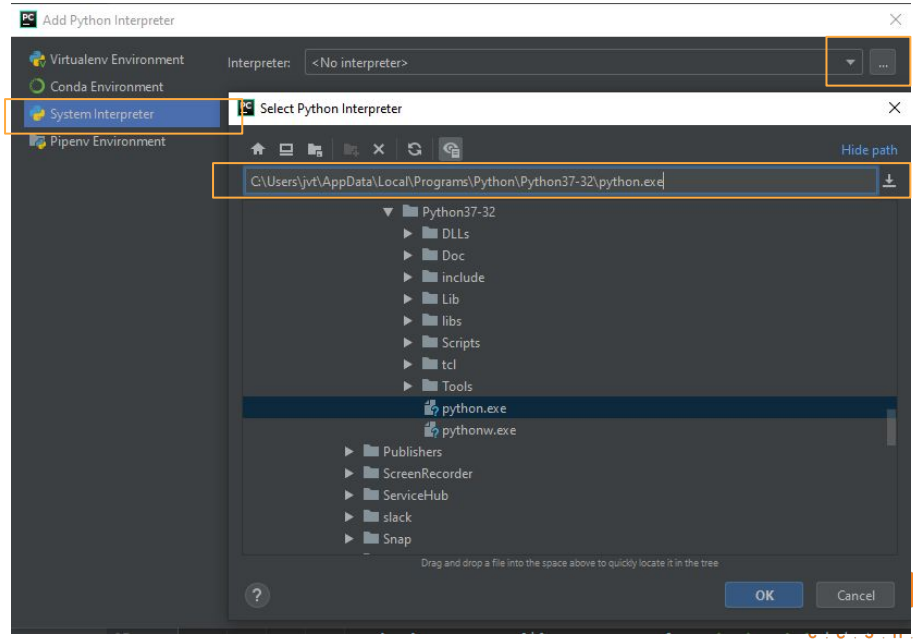
Criando um Projeto no Pycharm

- Para criar um projeto é necessário selecionar o interpretador Python.

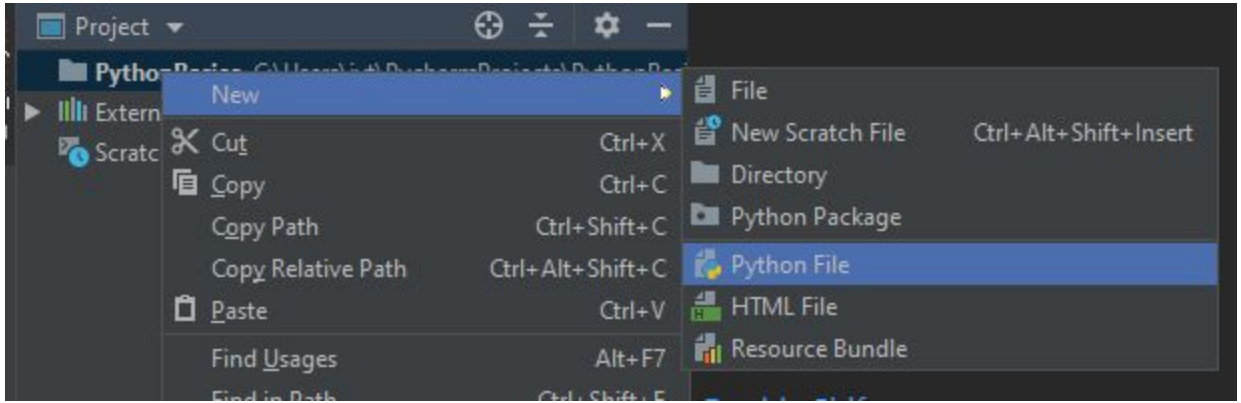
Criando um Projeto no Pycharm







Um primeiro programa típico...



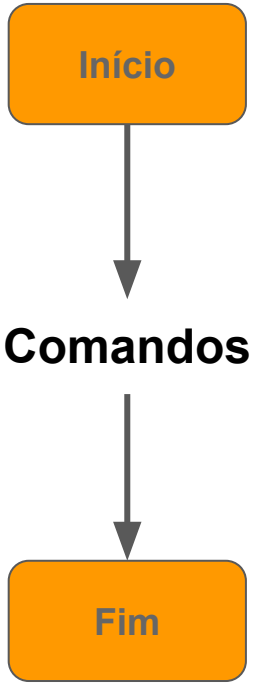
Um primeiro programa típico...

- Monte o ambiente para programação em Python no seu computador
- Verifique a instalação
- Imprima a mensagem "Olá Mundo" no PyCharm

```
print("Olá, mundo!")
```

Execução de Programas

- *Um programa executa instrução após instrução*
- *A execução é linear*
- *O interpretador ignora comentários*
  - *Úteis apenas aos programadores*
- *Extensão **.py***
  - *Ou .pyc, .pyo (quando compilado)*



Variáveis e Constantes

Joyce Teixeira

Variáveis

- Uma variável é um espaço na memória do computador usado para armazenar um valor ou uma referência a um valor. Esse valor pode ser alterado durante a execução do programa.
- Exemplo em Python:

```
numero = 10
numero = numero + 5  # Agora, 'numero' é 15
```

Variáveis

- Em muitas linguagens de programação, incluindo Python, o valor de uma variável pode ser alterado a qualquer momento. Além disso, em linguagens tipadas dinamicamente como Python, o tipo de dado que uma variável armazena também pode mudar:

```
dado = 10        # 'dado' é um inteiro
dado = "texto"   # Agora, 'dado' é uma string
```

Constantes

- Uma constante é semelhante a uma variável no sentido de que também é um espaço na memória usado para armazenar um valor. No entanto, uma vez que um valor é atribuído a uma constante, ele não pode ser alterado.
- Python, por si só, não possui um mecanismo nativo para definir constantes. No entanto, por convenção, os desenvolvedores usam nomes em letras maiúsculas para indicar que uma variável não deve ser modificada e é tratada como uma constante:

```
PI = 3.14159
TAXA_MAXIMA = 0.25
```

Variáveis e Constantes

- Diferença entre Constantes e Variáveis:
  - Mutabilidade: A principal diferença entre constantes e variáveis é que o valor de uma variável pode ser alterado após sua inicialização, enquanto o valor de uma constante não deve ser alterado.
  - Convenção vs. Regra: Em algumas linguagens, a distinção entre constantes e variáveis é estritamente aplicada pelo compilador ou pelo ambiente de execução. Em Python, a distinção é mais uma questão de convenção e boas práticas.
  - Uso: Variáveis são usadas para valores que podem precisar ser alterados, como contadores em loops, entradas do usuário ou resultados de cálculos. Constantes são usadas para valores que são conhecidos para não mudar, como valores matemáticos (por exemplo, PI) ou configurações fixas em um programa.

Variáveis e Constantes

- Tipos de Variáveis
  - Em Python, as variáveis não têm um tipo intrínseco definido na declaração, como em algumas outras linguagens.
  - O tipo é determinado dinamicamente com base no valor que você atribui à variável.

Variáveis e Constantes

- Tipos de Variáveis
  - Númericas:
    - Inteiro (int): Representa números inteiros, positivos ou negativos.
      - Exemplo: idade = 25; quant_filhos = 2.
    - Ponto Flutuante (float): Representa números reais, ou seja, números que têm uma parte decimal.
      - Exemplo: altura = 1.75; peso = 70.0.
  - Textual:
    - String (str): Representa sequências de caracteres.
      - Exemplo: nome = "Maria"; endereco = “Rua da Hora, 45”.

Variáveis e Constantes

- Tipos de Variáveis
    - Coleções:
        - Lista (list): Coleção ordenada de valores.
            - Ex: frutas = ["maçã", "banana", "cereja"]
        - Tupla (tuple): Coleção ordenada e imutável.
            - Ex: coordenadas = (4, 5)
        - Dicionário (dict): Pares chave-valor.
            - Ex: pessoa = {"nome": "João", "idade": 30}
        - Conjunto (set): Coleção de valores únicos.
            - Ex: cores = {"vermelho", "azul", "verde"}

Variáveis e Constantes

- Tipos de Variáveis
  - Bolleana:
    - Bolleano (bool): Representa uma sentença verdadeira ou falsa.
    - Exemplo: esta_chovendo = True
  - Especial:
    - NoneType (None): Representa a ausência de valor.
    - Exemplo: resultado = None

Variáveis e Constantes

- Lembrando que o uso de variáveis no Python é bem simples;
- Possui tipagem dinâmica, ou seja, o tipo da variável será definido de acordo com o valor atribuído;

```
nome = 'Maria'
idade = 25
tamanho = 1.66
sexo = 'F'
matriculada = True
```

str
int
float
str
bool

Variáveis e Constantes

```
nome = "Maria"
idade = 25
tamanho = 1.66
sexo = "F"
matriculada = True

print("A variável nome tem o tipo", type(nome), "e o valor ", nome)
print("A variável idade tem o tipo", type(idade), "e o valor ", idade)
print("A variável tamanho tem o tipo", type(tamanho), "e o valor ", tamanho)
print("A variável sexo tem o tipo", type(sexo), "e o valor ", sexo)
print("A variável matriculada tem o tipo", type(matriculada), "e o valor ", matriculada)
```

Variáveis e Constantes

```
A variável nome tem o tipo <class 'str'> e o valor  Maria
A variável idade tem o tipo <class 'int'> e o valor  25
A variável tamanho tem o tipo <class 'float'> e o valor  1.66
A variável sexo tem o tipo <class 'str'> e o valor  F
A variável matriculada tem o tipo <class 'bool'> e o valor  True
```

Nomeando Variáveis

- Regras:
  - Caracteres Permitidos:
    - Os nomes das variáveis devem começar com uma letra (a-z, A-Z) ou um sublinhado (_).
    - O restante do nome da variável pode conter letras, números (0-9) e sublinhados.
  - Sensibilidade a Maiúsculas e Minúsculas:
    - Python é sensível a maiúsculas e minúsculas. Assim, variavel, Variavel e VARIaVEL seriam considerados identificadores distintos.
  - Palavras Reservadas:
    - Nomes de variáveis não podem ser palavras reservadas em Python. Por exemplo, você não pode nomear uma variável como if, else, while, etc.

Nomeando Variáveis

- Convenções:
  - Nomes Descritivos:
    - Use nomes descritivos para variáveis para tornar o código mais autoexplicativo. Por exemplo, nome_do_usuario em vez de n.
  - Tudo Minúsculo:
    - Para nomes de variáveis e funções, use letras minúsculas com sublinhados para separar palavras. Por exemplo, contador_de_itens ou carregar_dados.
  - Constantes:
    - Nomeie constantes usando todas as letras maiúsculas com sublinhados. Por exemplo, PI, TAXA_MAXIMA.
  - Evite Abreviações:
    - A menos que a abreviação seja amplamente aceita, prefira nomes completos. Por exemplo, use computador em vez de comp.

Comandos de Entrada e Saída

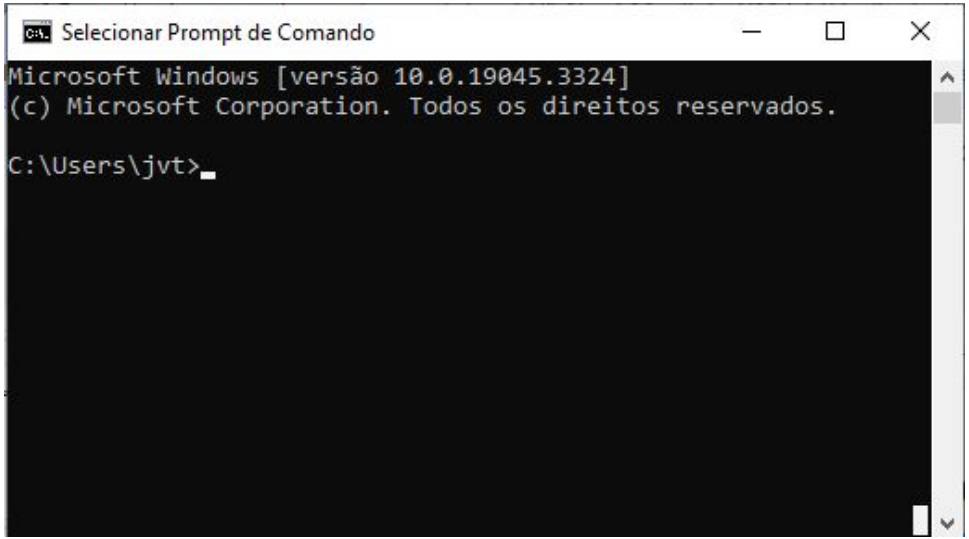
Joyce Teixeira

Comandos de Entrada e Saída

- Como interagir com o usuário e exibir informações na tela?

Comando de Entrada

- **input()** permite que você colete informações do usuário.




```
nome = input("Digite seu nome: ")
```

Comando de Entrada

- Capturando Entradas Diferentes com **input()**
  - Por padrão, **input()** sempre retorna uma string. Para trabalhar com outros tipos de dados, como números, é necessário converter essa entrada.

```
texto = input("Digite algo: ")
print(type(texto))
```

```
Digite algo: Olá
<class 'str'>
```

Comando de Entrada

- Capturando Entradas Diferentes com **input()**
    - A conversão para um tipo de dado é chamada de 'casting'. Em Python, podemos usar funções como int(), float(), e str() para converter entre tipos de dados.

```
idade = int(input("Digite sua idade: "))
print(type(idade))
```

```
Digite sua idade: 34
<class 'int'>
```

Comando de Saída

- **print()** exibe informações para o usuário.

```
print("Olá, Mundo!")
```

Comando de Saída

- Personalizando Mensagens com *f-strings*
  - As *f-strings*, introduzidas no Python 3.6, permitem a incorporação de expressões dentro de strings, usando **{}**.

```
idade = int(input("Digite sua idade: "))
nome = input("Digite seu nome: ")
print(f'{nome}, você tem {idade} anos.')
```

```
Digite sua idade: 34
Digite seu nome: Joyce
Joyce, você tem 34 anos.
```

Comando de Saída

- Personalizando Mensagens com *f-strings*
  - As *f-strings* não só tornam o código mais legível, mas também são úteis para formatar saídas de maneira dinâmica e eficiente.
  - Para criar uma *f-string*, comece a string com a letra 'f' ou 'F' antes das aspas.

Comando de Saída

- Personalizando Mensagens com *f-strings*

```
nome = input("Digite seu nome: ")
altura = float(input("Digite sua altura: "))
peso = float(input("Digite seu peso: "))

imc = peso / (altura * altura)

print(f'{nome}, você tem um imc de {imc}.')
```

```
Digite sua altura: 1.66
Digite seu peso: 62.0
Maria, você tem um imc de 22.49963710262738.
```

Vamos Praticar!

- Faça um algoritmo que receba o ano de nascimento do usuário, o ano atual, e imprima a idade.

Vamos Praticar!

- Faça um algoritmo que receba o ano de nascimento do usuário, o ano atual, e imprima a idade.

```
ano_nasc = int(input("Digite o ano de nascimento: "))
ano_atual = int(input("Digite o ano atual: "))
idade = ano_atual - ano_nasc
print(f'A idade é {idade}.')
```

Vamos Praticar!

- Altere o algoritmo anterior para receber o ano atual como constante.

Vamos Praticar!

- Altere o algoritmo anterior para receber o ano atual como constante.

```
ANO_ATUAL = 2023
ano_nasc = int(input("Digite o ano de nascimento: "))
idade = ANO_ATUAL - ano_nasc
print(f'A idade é {idade}.')
```

Vamos Praticar!

- Faça um programa para ler os três lados de um triângulo e exiba o perímetro do mesmo. Lembrar que o perímetro se calcula a partir da soma de todos os lados do triângulo.
- Ler um número em metros e transformar em centímetros.

Vamos Praticar!

- Lista 01 disponível.

Operadores

Joyce Teixeira

Operadores

É possível realizar operações sobre as variáveis e sobre algumas constantes:

- Atribuição: guardar valor em variáveis
- Aritmética: soma, subtração, multiplicação, divisão
- Relacional: maior, menor, igual, diferente, maior ou igual, menor ou igual
- Lógica: e, ou, não

Operadores - Atribuição

A atribuição guarda conteúdos na memória:

- Símbolo: **=**
- Sintaxe: **VARIAVEL = VALOR**;

  VARIAVEL = EXPRESSAO;
- Primeiro avalia o lado direito do operador
- O valor de **VALOR** ou **EXPRESSAO** é guardado na posição de memória endereçada por **VARIAVEL**

Operadores - Atribuição

```
a = 10
b = 4
a = b
b = 7

print(f'a: {a}')
print(f'b: {b}')
```

Operadores – Aritmética

- Para construir algoritmos que realizam cálculos matemáticos precisamos utilizar os operadores aritméticos;
- As expressões aritméticas devem ser linearizadas;
- Deve ser feito o mapeamento dos operadores utilizados tradicionalmente para a linguagem Python.

Operadores – Aritmética

- Como realizar a linearização de expressões?

Tradicional

$$\left\{\left[\frac{7}{3}-(7+2)\right]+5\right\}.2$$

Computacional

$$((7/3-(7+2)+5))*2$$

Operadores – Aritmética

Operadores Aritméticos	Python	Descrição
Adição	+	Operador tradicional de adição
Subtração	-	Operador tradicional de subtração
Multiplicação	*	Operador tradicional de multiplicação
Divisão	/	Operador tradicional de divisão
Módulo (Resto da Divisão)	%	Resto da divisão inteira (9%2=1)

Operadores – Aritmética

Precedência dos Operadores

Operadores Aritméticos	Python	Prioridade
Módulo (resto da divisão)	%	1º
Divisão	/	1º
Multiplicação	*	2º
Subtração	-	3º
Adição	+	3º

Operadores – Aritmética

- Os operadores “fracos” podem ser executados primeiro com a utilização de parênteses:
  - 1+5*8/4*2 = ?
  - (1+5)*(8/4)*2 = ?
  - ((1+5)*(8/4))*2 = ?

Exercícios

Criar algoritmos para:

1. Mostrar a soma, produto e média aritmética de três números informados pelo usuário
2. Ler o tempo de duração de um exame expresso em horas e mostrar em segundos

Operadores – Relacionais

- Realizam comparações e retornam valores lógicos: TRUE ou FALSE;
- Geralmente envolvem os operadores de comparação

Operadores Relacionais	Python
Maior	>
Menor	<
Maior ou igual	>=
Menor ou igual	<=
Igual	==
Diferente	!=

Operadores – Relacionais

- Vamos praticar?
  - a = 10.
  - b = 8.

Operadores – Lógicos

- Servem para **combinar** resultados de expressões retornando se o resultado final é **TRUE (True)** ou **FALSE (False)**.

Operadores Lógicos	Python	Significado
Multiplicação Lógica ou Conjunção	and	O resultado será TRUE se uma parte **E** a outra parte forem verdadeiras.
Adição Lógica ou Disjunção	or	O resultado será TRUE se uma parte **OU** a outra parte forem verdadeiras.
Não Lógico	not	O resultado será a inversão do valor lógico. Se for TRUE torna-se FALSE.

Operadores – Lógicos

- Como chego a este valores?
- Há um método chamado tabela-verdade, que mostra os resultados das aplicações dos operadores lógicos.

Operadores – Lógicos

A	B	A and B	A or B	not A	not B
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

Operadores – Lógicos

- Vamos praticar?
  - A = 10; B = 15; C = 2.

Expressões			Resultado
A == B	E	B > C	
A != B	OU	B < C	
A > B	NAO		
A >= B	E	B == C	
A <= B	OU	B != C	

Operadores – Lógicos

- Precedência de Operadores Lógicos

Operadores Lógicos	Python	Significado
Multiplicação Lógica ou Conjunção	e	1º
Adição Lógica ou Disjunção	ou	2º
Não Lógico	nao	3º

Operadores – Lógicos

- Quais as saídas?

```
x = 5
print(x > 3 and x < 10)
```

True

```
x = 5
print(x > 3 or x > 10)
```

True

```
x = 5
print(not(x > 3 and x < 10))
```

False

Operadores

- Tantos os operadores relacionais como os lógicos são fundamentais para a construção de expressões condicionais em Python e são amplamente utilizados em estruturas de controle, como if, elif e while.

Operadores

- Lista de Exercício 02 disponível.