Curso Preparatório

Semana 2:
Entrada e Saída / Variáveis

Admissão 2025

Instituto J&F
Germinare TECH

Linguagens de programação

Uma **linguagem de programação** pode ser definida como:

- **um conjunto** limitado **de símbolos e comandos**, usados para criar programas;
- **um método** padronizado **para dar instruções** para um computador;
- **o meio pelo qual** se estabelece uma comunicação com **o computador**, fazendo com que ele compreenda e **execute o que o programador determinar**.

Existem várias linguagens de programação, usadas no mundo todo.

Na tabela estão as mais populares segundo o índice **PYPL** (**P**opularit**Y** of **P**rogramming **L**anguages).

https://pypl.github.io/PYPL.html

Worldwide, Dec 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.34 %	-1.0 %
2		Java	16.93 %	-0.8 %
3		JavaScript	9.28 %	+0.3 %
4		C#	6.89 %	-0.3 %
5		C/C++	6.64 %	-0.3 %
6		PHP	5.19 %	-1.0 %
7		R	3.98 %	-0.1 %
8	↑↑	TypeScript	2.79 %	+1.1 %
9	↑↑	Swift	2.23 %	+0.6 %
10	↓↓	Objective-C	2.22 %	+0.1 %

Linguagens de programação

Qual melhor linguagem? Não existe.
Mas existem linguagens Dinâmicas e Estáticas

- Linguagem estática: JAVA
  - A verificação dos tipos de variáveis é feita no código (compilação).
  - Maior segurança e controle.
  - Mais rápidas.
  - Compila o código somente se estiver sem algum erro.

- Linguagem dinâmica: Python
  - As variáveis podem conter qualquer tipo de dado. A verificação é feita em tempo de execução.
  - Mais flexibilidade.
  - Códigos mais limpos.
  - Mais fáceis de usar.

Linguagens de Programação

- Compilador: JAVA
  - Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Ao compilar, gera outro código, que é, então, interpretado e executado.

  - **Vantagens:**
    - O código compilado é mais rápido de ser acessado;
    - Permite otimização do código por parte do compilador;
    - Compila o código somente se estiver sem algum erro.
  - **Desvantagens:**
    - Para ser utilizado o código precisa passar por muitos níveis de compilação;
    - Assim como vantagem a possibilidade de não poder visualizar o código-fonte, pode ser uma desvantagem;
    - Processo de correção ou alteração do código requer que ele seja novamente recompilado.

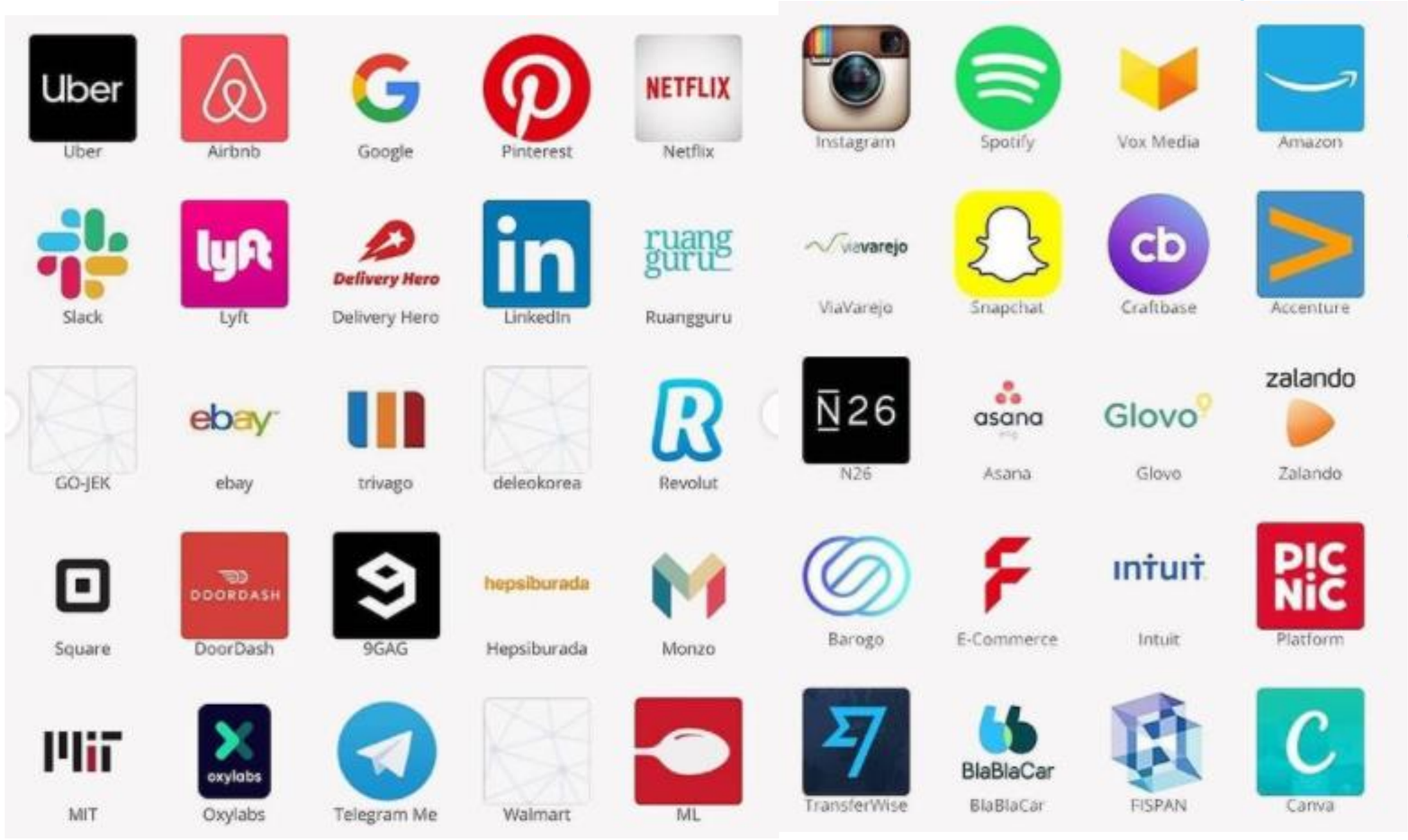
Linguagens de Programação

- Interpretador: Python
  - O interpretador ao contrário do compilador roda o código-fonte escrito como sendo o código objeto, ele traduz o programa linha a linha, o programa vai sendo utilizado na medida em que vai sendo traduzido. Cada execução do programa precisa ser novamente traduzido e interpretado.

  - **Vantagens:**
    - Correções e alterações são mais rápidas de serem realizadas;
    - Código não precisa ser compilado para ser executado;
    - Consomem menos memória.
  - **Desvantagens:**
    - Execução é mais lenta do programa;
    - Necessita sempre ser lido o código original para ser executado;

JAVA é RUIM?

Algumas empresas que usam:



Python é RUIM?

Python também é usado em muitas coisas, também é uma linguagem multiparadigma e de propósito geral, pode ser usado para diferentes fins. Mas não é portátil.

Python

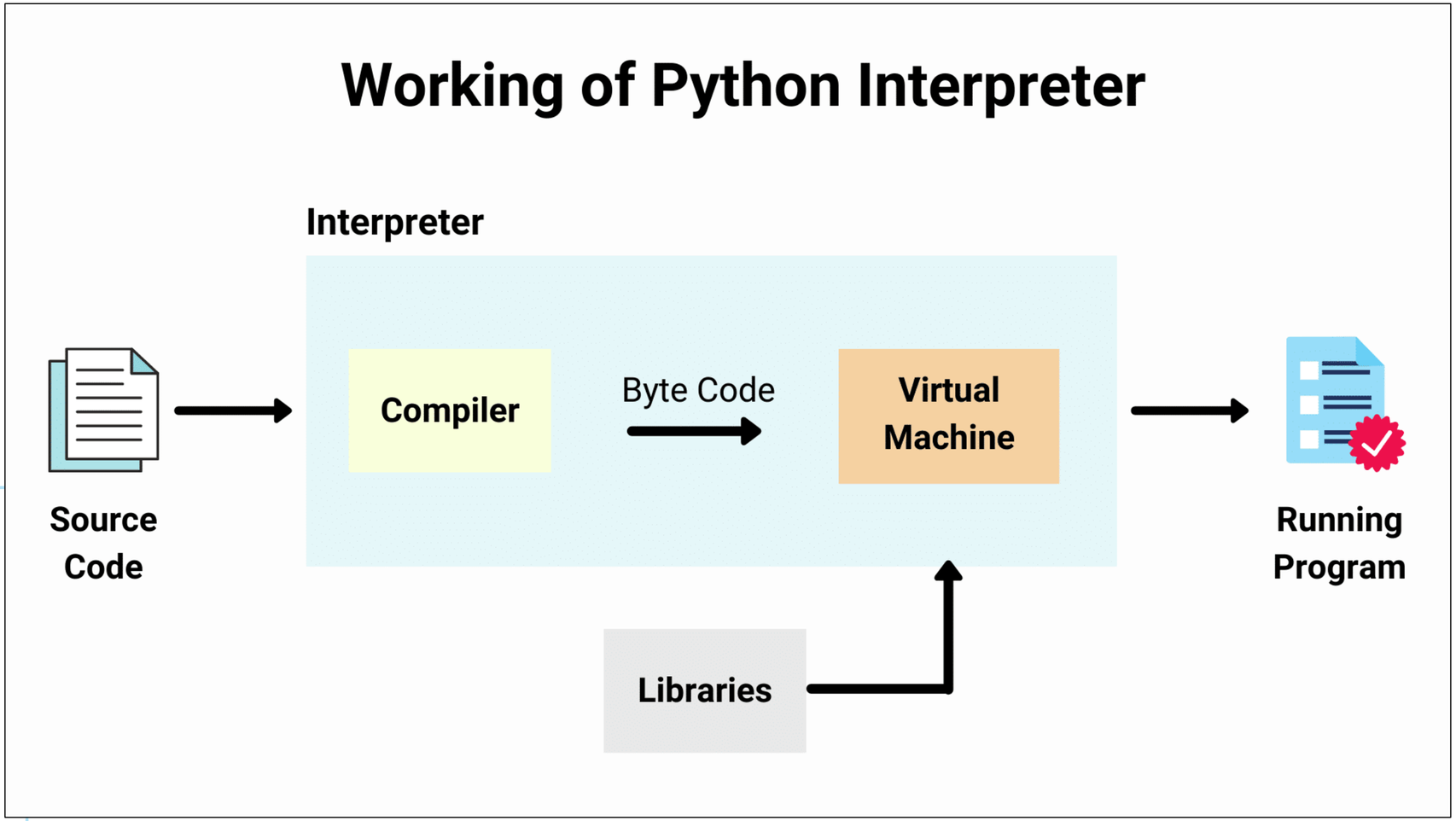
- Desenvolvida por **Guido van Rossum (1989)** no Centro de Matemática e Tecnológia da Informação (CWI, Centrum Wiskunde e Informatica), na Holanda.
- Atualmente, é gerenciada pela Python Software Foundation (https://www.python.org/).
- Portável a diferentes plataformas.
- Multiparadigma
- Dinâmica e Interpretada.
- Fácil aprendizado; Simples de programar; Sintaxe intuitiva; Open Source; Modularização; Multiplataforma; Grande quantidade de bibliotecas disponíveis; Grande comunidade de usuários;

Python

- O Python é uma linguagem interpretada, pois ela não gera arquivos executáveis como acontece na linguagem C e JAVA, por exemplo. Em vez disso, ela contém um interpretador, que é responsável por traduzir o código fonte em linguagem de máquina e, assim, executar o programa.

- Na prática, o interpretador transforma o código fonte para o formato *bytecode*, que corresponde à linguagem de máquina ou código binário, e envia para um ambiente chamado PVM — Python Virtual Machine —, que é uma máquina virtual Python que contém o ambiente capaz de executar o programa.

Python



Extensão

.py

Python

- **Sintaxe da Linguagem Python**

  - Declarações de variáveis
  - Atribuição
  - Saída de Dados
  - Entrada de Dados

Python – Tipos primitivos

Inteiro (int): números inteiros sem parte fracionária, podendo ser negativo, nulo ou positivo

Exemplos: 15, 0, 20, 4434

Real (float): números com parte fracionária, podendo ser negativo, nulo ou positivo

Exemplos: -234.4, 45, 98, 45

Caracter – cadeia (literais) (string): conjunto de caracteres alfanuméricos números (0.. .9), letras (A.. .Z, a.. .z) e símbolos (#, ?, !, @.....)

Para um caracter usar aspas simples - char

Para um conjunto de caracteres usar aspas dupla - String

Exemplos: “Não pise na grama”, “José da Silva”

Lógico (boolean): poderá assumir valores:

true – verdadeiro ou ***false*** – falso

Tipos de Dados

Tipos Primitivos

Descrição	Python	Em Java
Caractere (uma letra)	string	char
Literal	string	String (classe)
Inteiro	int	int
Inteiro longo	int	long
Real (ponto flutuante)	float	float
Real “longo”	float	double
Booleano	boolean	boolean

Por ser uma linguagem dinâmica, uma das grande vantagem de Python é que não precisa declarar as variáveis, mas precisa inicializar.

Variáveis

- Outra características, quer saber qual o tipo da variável você está lidando?
- Em python tem o comando **type**

```
1   nome = "Camila"
2   idade = 31
3   peso = 58.33
4   professor = False
5
6   print(type(nome))
7   print(type(idade))
8   print(type(peso))
9   print(type(professor))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Variáveis

Python

- p1 = 10
- media = 7.5
- nome = “Marcelo”

Python não precisa de ; no final

As variáveis precisam ser inicializadas
Não precisa o tipo da variável

Saída de Dados

Python

–print( )

Lembrar que texto vem entre “ “

- **print (“Digite o valor da compra “)**
- **print("%.2f" % 7.44444) -> saída = 7.44**
- **print("Media =", media)**
- **print("A saída vai ser", nome, "%.2f" % media)**

Concatenador -> Strings e variáveis

Python permite que você escolha entre usar o:

- **+ (mais)** <- sem espaço, mas apenas para concatenar strings
- **, (virgula)** <- com espaço

Saída de Dados

- **OBS Python**

Você pode usar o “+” para quando quer concatenar mais de 1 string no print()

Os print (linhas 4 e 5) do códigos abaixo tem a mesma saída: **“Marcelo Grilo”**

```
1  nome = "Marcelo"
2  sobrenome = "Grilo"
3
4  print(nome, sobrenome)
5  print(nome +" "+ sobrenome)
```

Saída de Dados

- **Saída formatada**

Python permite que ao invés de separar variáveis e textos com a virgula(,) ou o mais (+), você pode colocar uma MÁSCARA DE SUBSTITUIÇÃO que é dado por um **{ }** junto com o **.format(),** depois você precisa especificar o que vai ser inserido dentro dessa MÁSCARA, então usa-se a seguinte sintaxe:

```
x = 10
y = 5
print("A saída vai ser {} e {}".format(x,y))
```

As MÁSCARAS também permitem que você use variáveis dentro de um **input()**

A saída vai ser 10 e 5

Saída de Dados

- **Saída formatada 2**

Você também tem outra possibilidade de usar as variáveis dentro de uma MÁSCARA {}. Para isso você precisa adicionar antes do texto a letra **f**, observe o código abaixo:

```
x = 10
y = 5
print(f"A saída vai ser {x} e {y}")
```

A saída vai ser 10 e 5

Saída de Dados

- **Controlando casas decimais**

Observe no código abaixo as sintaxes que podem ser usadas para controlar as casas decimais de um numero real.

Ambos os prints abaixo mostrarão apenas 2 casas decimais (10.56 e 5.33).

```
x = 10.5554
y = 5.333333
print("A saída vai ser %.2f e %.2f" %(x,y))
print("A saída vai ser {:.2f} e {:.2f}".format(x,y))
print(f"A saída vai ser {x:.2f} e {y:.2f}")
```

OBS: Arredondamento de 2 casas,
Se < 4, arredonda para baixo.
Se >=5, arredonda para cima

A saída vai ser 10.56 e 5.33

Saída de Dados

- **Organizando os espaços**

Muitas vezes, temos que organizar os espaços em uma saída de dados, isso é dado pela quantidade de “espaços” reservados para uma variáveis.

```
x = 11
y = 54321
z = 3
print("QNT |  Código | Nome")
print(f"1   | {x} | Marcelo")
print(f"2   | {y} | Luis")
print(f"3   | {z} | Marcos")
```

```
QNT |  Código | Nome
1   | 11 | Marcelo
2   | 54321 | Luis
3   | 3 | Marcos
```

Saída de Dados

- **Organizando os espaços**

E se eu quiser que a saída fique assim:

```
QNT |  Código | Nome
1   |      11 | Marcelo
2   |   54321 | Luis
3   |       3 | Marcos
```

```
x = 11
y = 54321
z = 3
print("QNT |  Código | Nome")
print(f"1   | {x:7} | Marcelo")
print(f"2   | {y:7} | Luis")
print(f"3   | {z:7} | Marcos")
```

Adiciona o → **:7**
Para dizer quantas casas vai ocupar aquela informação, nesse caso 7.

Saída de Dados

- **Organizando os espaços**

E se eu quiser que a saída fique assim:

```
QNT | Código | Nome
1   |   11   | Marcelo
2   |  54321 | Luis
3   |    3   | Marcos
```

```
x = 11
y = 54321
z = 3
print("QNT |  Código | Nome")
print(f"1   | {x:^7} | Marcelo")
print(f"2   | {y:^7} | Luis")
print(f"3   | {z:^7} | Marcos")
```

Adiciona o → **:^7**
Para centralizar usa-se o circunflexo (^)

Saída de Dados

- **Símbolos com o print( )**

Limpar tela:

```
print("\x1b[2J")
```

Negrito

```
print("\033[;1m")
```

Emotion Sorrindo

```
print("\U0001f600")
```

Saída de Dados

- **Cores com Python (** Usar no **print( )** **)**

COR	Fonte	Fundo
Vermelho	\033[1;31m	\033[1;41m
Verde	\033[1;32m	\033[1;42m
Amarelo	\033[1;33m	\033[1;43m
Azul	\033[1;34m	\033[1;44m
Branco	\033[1;97m	\033[1;107m
Remover formatação	\033[0;0m	**PRETO: \033[1;30m**
https://raccoon.ninja/pt/dev-pt/tabela-de-cores-ansi-python/		

Saída de Dados

- **Cores com Python (** Usar no **print( )** **)**

Cuidado, ao mudar as cores, muda em todo o editor, então o que se recomenda é usar uma classe com as configurações de cores:

```
class color:
   AZUL = '\033[94m'
   VERDE = '\033[92m'
   AMARELO = '\033[93m'
   VERMELHO = '\033[91m'
   NEGRITO = '\033[1m'
   SUBLINHADO = '\033[4m'
   NORMAL = '\033[0;0m'

print(color.NEGRITO + 'testando configurações de cor' + color.NORMAL)
```

Entrada de Dados

Python

Input( )

```
# declaração das variáveis (só precisa se for inicializar)

salario = 0;

print(“ Digite a Idade : “)
idade = input()
print("Qual o salario?")
salario = input()
```

Lembra o **leia()**, né?

Python
#comentário
ou
"comentario"

Entrada de Dados

Python

```
# Mas Python tem outra vantagem:
# Você não precisa ter um print para cada pergunta, pode vir direto no input

# olhe o código abaixo:

idade = input("Digite sua idade ")
salario = input("Qual o salario? ")
```

Python
#comentário
ou
"comentario"

Entrada de Dados

- **Mas tem uma observação na entrada de dados:**
- Python reconhece o input como sendo sempre um string, se a entrada de dados (o que o usuário for digitar) for um numero (int ou float) você deve informar antes do input, assim:

```
numero = int( input("Digite um numero qualquer ") )
numero = float( input("Digite um numero qualquer ") )
```

Isso vai fazer com que o que o usuário digite, seja considerado um número. Se não colocar, o que o usuário digitar vai ser uma string.

Primeiro Código

- Vamos começar a praticar. Vamos ver a entrada e saída, testando as saídas formatas? Fazer um programa para saber o dobro de um número digitado pelo usuário

```
1  numero = int(input ("Digite um numero qualquer "))
2  dobro = numero*2
3  print("O dobro de", numero, "é", dobro)
```

- Mas cuidado! Veja o que acontece se você não coloca o int antes do input

```
Digite um numero qualquer 10
O dobro de 10 é 1010
```