

Python aplicado a Probabilidade e Estatística

# Aula 1

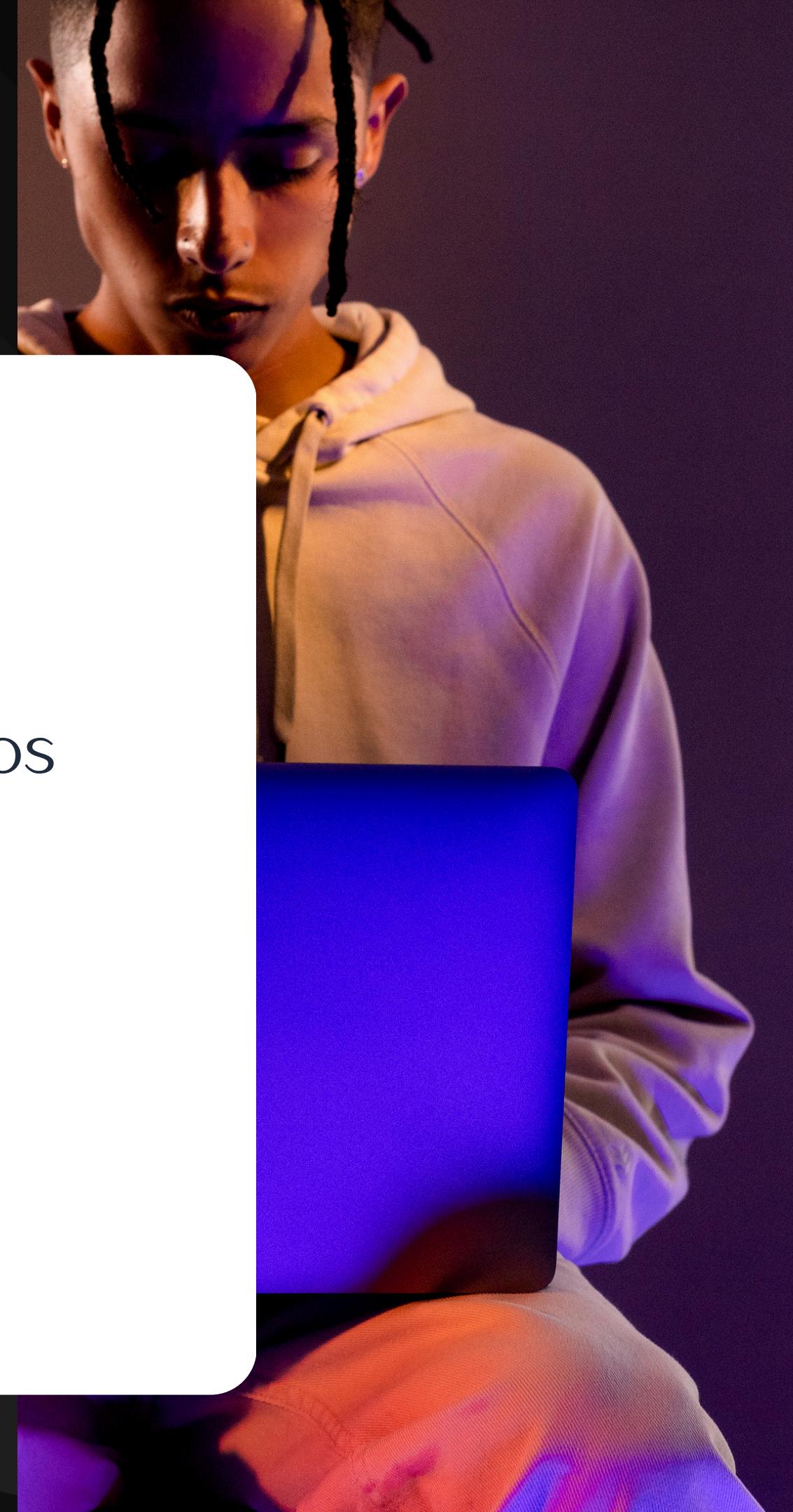
Diêgo Santos Queiroz



João Gabriel Oliveira Magalhães

# Tópicos

1. Introdução
2. Sintaxe e Estrutura básica
3. Operadores aritméticos e lógicos
4. Estruturas condicionais
5. Estruturas de repetição
6. Listas
7. Funções



# Introdução

O Python é uma linguagem de programação de alto nível, criada em 1991 por van Rossum.

- Sintaxe clara e limpa.
- 01** Fácil de aprender e ler.  
Grande aplicabilidade no mercado.
  
- Tipagem dinâmica.
- 02** Multiparadigma(orientada a objetos, funcional e imperativa).
  
- Roda em diferentes sistemas operacionais.
- 03** Possui uma grande comunidade e bibliotecas.  
Muito usada como introdução a programação.

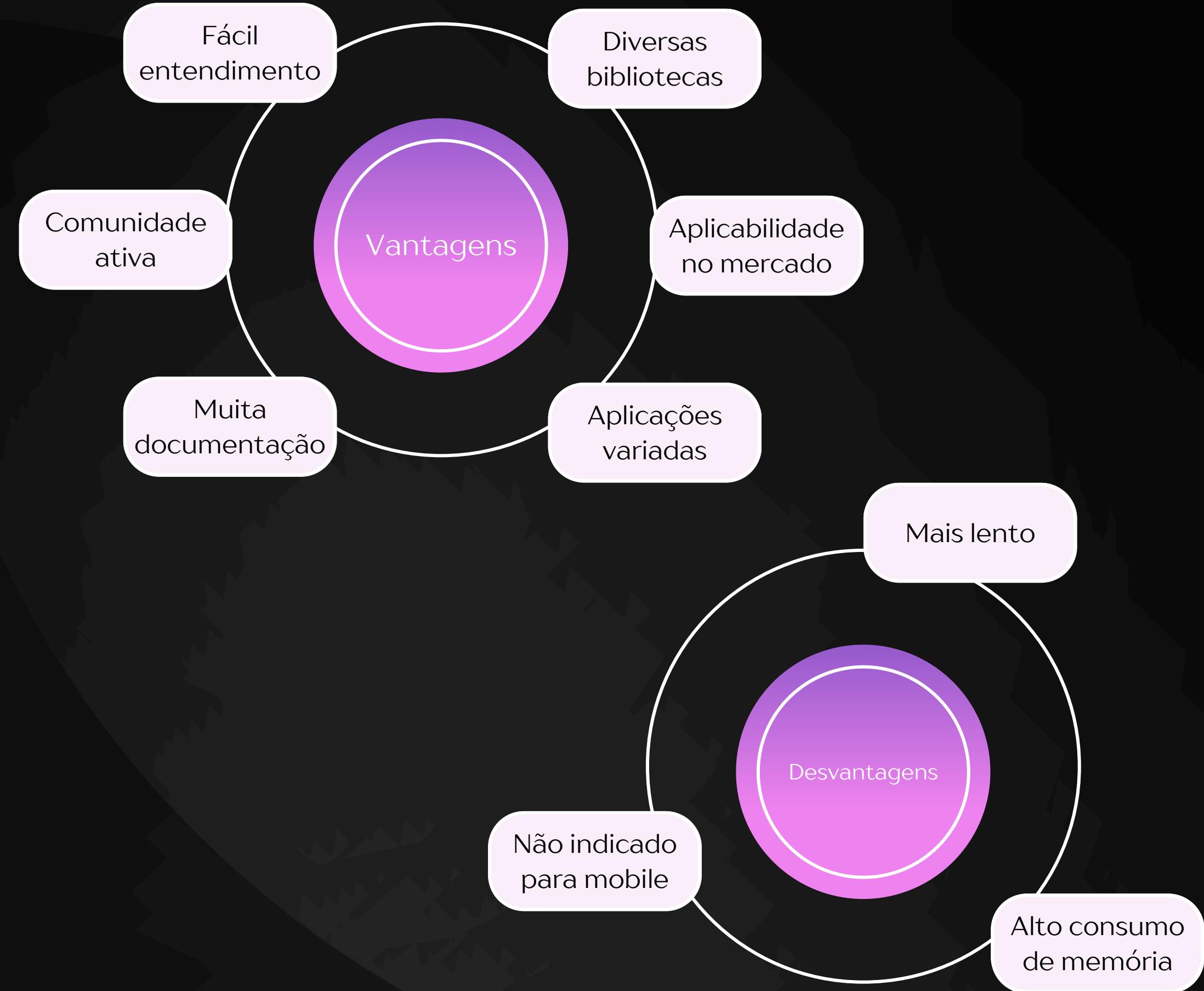
Ela se destaca bastante por sua simplicidade e legibilidade, sendo fácil para aspirantes à programação.



# Quem usa Python?

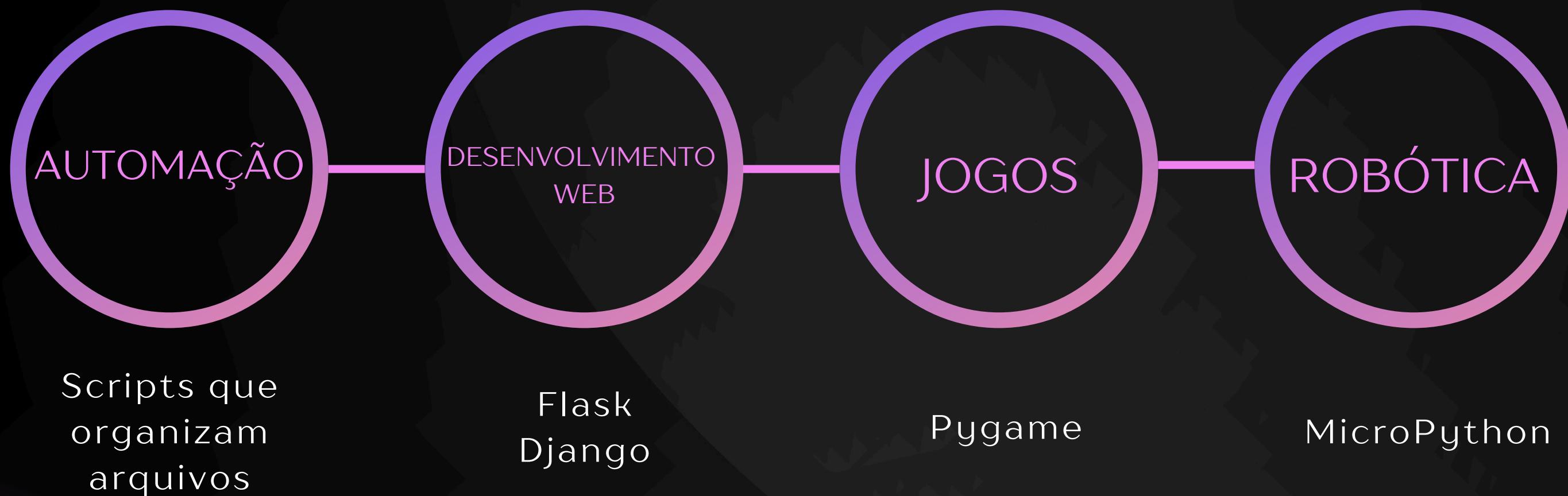
Grandes empresas como;  
Google, Netflix, Facebook,  
Nasa...

O Python é muito usado  
também em universidades,  
estando presente em  
diversas áreas.



# Usabilidade

O Python tem uma gama de aplicações práticas, em diversas áreas





# Comparação com outras línguagens

## Java

```
public class App {  
    Run | Debug | Run main | Debug main  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

Mais verbosa;  
Precisa de uma classe pública;  
Finaliza com ponto e vírgula;

## Python

```
1 print("Hello World")  
2
```

Menos verbos;  
Simples e direta;  
Sem necessidade de declarar classe,  
função principal ou tipo de dados;

# SINTAXE EM PYTHON

O Python possui uma sintaxe limpa e de fácil compressão.

Podemos notar que durante a escrita de códigos não existe a necessidade de usar {} ou ; ao encerrar uma linha de comandos.

Por outro lado possui uma indentação obrigatória ao definir blocos (if, for, funções). Que nada mais é do que recuo com espaço.

Ao contrário de outras linguagens não é necessário que se declare o tipo da variável, pois ele é interpretado automaticamente.

```
nome = "Maria"  
print("Olá,", nome)
```

```
if idade >= 18:  
    print("Maior de idade")  
else:  
    print("Menor de idade")
```

```
x = 10      # inteiro  
nome = "Ana" # string  
pi = 3.14   # float
```

# Estrutura Básica

A estrutura básica é simples e direta, o que facilita a escrita de código e aprendizado.

- Declaração de variáveis; atribui de forma direta, sem precisar declarar o tipo de variável.

```
nome = "Maria"  
print("Olá,", nome)
```

- Entrada de dados; utiliza de *input()* para capturar dados do usuário.

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

- Processamento; envolve qualquer cálculo ou lógica realizada pelo programa.

```
ano_nascimento = 2025 - idade
```

- Saída de dados; imprime uma informação usando *print()*.

```
print("Você nasceu em", ano_nascimento)
```

# Estrutura Básica

A estrutura básica é simples e direta, o que facilita a escrita de código e aprendizado.

- Condicionais; estabelecem uma condição.

```
if idade >= 18:  
    print("Maior de idade")  
else:  
    print("Menor de idade")
```

- Laços de repetição; repetem o código x vezes, ou até que uma condição seja satisfeita.

```
for i in range(5):  
    print("Repetição:", i)
```

- Funções; blocos que podem ser reutilizados.

```
def saudacao(nome):  
    print("Olá,", nome)  
  
saudacao("João")
```

- Importações(opcional); carrega módulos e bibliotecas.

```
import math
```

# Operadores Ariméticos

Lidam com números.

Operador	Descrição	Exemplo	Saída
+	Adição	$3 + 3$	6
-	Subtração	$10 - 8$	2
*	Multiplicação	$5 * 2$	10
/	Divisão	$15 / 3$	5
//	Divisão inteira	$8 // 3$	2
%	Divisão com resto	$14 \% 6$	2,333
**	Potência	$2 ** 2$	4

# Operadores Lógicos

Lidam com condições de verdadeiro ou falso (valores booleanos).

Operador	Significado	Exemplo	Saída
and	E	V and F	Falso
or	Ou	V ou F	Verdadeiro
not	Negação	não F	Verdadeiro

Quando vamos escrever valores de Verdadeiro ou Falso, denominamos esses valores em inglês, que fica True (Verdadeiro) e False (Falso).

```
idade = 20
tem_carteira = True

if idade >= 18 and tem_carteira:
    print("Pode dirigir")
```

# ESTRUTURAS CONDICIONAIS

# Definição

## O que são:

São estruturas que permitem ao programa tomar decisões, escolhendo qual caminho seguir com base em condições.

## Como funcionam:

O programa testa uma condição (como uma comparação) e executa um bloco de código apenas se ela for verdadeira. Caso contrário, pode testar outra ou seguir para uma alternativa.

## Comandos em python:

- if (se) - verifica a primeira condição.
- else (senão) - executa se nenhuma condição for satisfeita.
- elif (senão se) - verifica outra condição, se a anterior for falsa.

# If:

## O que é:

O if é a estrutura condicional principal do Python. Ele executa um bloco de código somente se uma condição for verdadeira.

## Pontos importantes:

- O bloco dentro do if deve estar indentado (geralmente com 4 espaços).
- A condição pode ser qualquer expressão que resulte em True ou False.

## Exemplo:

```
idade = 18  
if idade >= 18:  
    print("Você é maior de idade.")
```

# Else:

## O que é:

O else é a última opção em uma estrutura condicional. Ele executa um bloco de código se nenhuma das condições anteriores for verdadeira.

## Quando usar:

Quando você deseja que algo aconteça se a condição principal falhar, sem precisar verificar outras condições.

## Exemplo:

```
idade = 15  
if idade >= 18:  
    print("Você é maior de idade.")  
else:  
    print("Você é menor de idade.")
```

# Elif:

## O que é:

O elif (abreviação de "else if") permite testar outras condições quando o if inicial for falso.

## Para que serve:

Evita o uso de vários if separados, garantindo que apenas uma condição será executada.

## Exemplo:

```
nota = 7  
if nota >= 9:  
    print("Excelente!")  
elif nota >= 6:  
    print("Aprovado.")
```

```
nota = 4  
if nota >= 9:  
    print("Excelente!")  
elif nota >= 6:  
    print("Aprovado.")  
else:  
    print("Reprovado.")
```

# Problema com if/else:

## Exemplo:

### Excesso de verificações:

Quando precisamos tratar várias condições diferentes, usamos muitos if, elif e else, o que pode deixar o código longo, repetitivo e cansativo de escrever.

Também com múltiplas verificações, o código está mais sujeito ao erro

### Baixa legibilidade:

Quanto mais blocos condicionais houver, mais difícil se torna entender rapidamente o que o código está fazendo. Isso prejudica a manutenção e clareza do programa.

```
opcao = 3  
if opcao == 1:  
    print("Iniciar")  
elif opcao == 2:  
    print("Carregar")  
elif opcao == 3:  
    print("Sair")  
else:  
    print("Opção inválida")
```

# Solução match case (switch case)

## O que é:

O match case é uma estrutura condicional que funciona como um switch em outras linguagens, permitindo comparar um valor com várias possibilidades de forma simples e clara.

## Como funciona:

Ele compara um valor com vários casos (case) e executa o bloco correspondente ao primeiro que combinar. Se nenhum caso for atendido, pode haver um case \_ para o padrão (similar ao else).

## Exemplo:

```
opcao = 3

match opcao:
    case 1:
        print("Iniciar")
    case 2:
        print("Carregar")
    case 3:
        print("Sair")
    case _:
        print("Opção inválida")
```

# ESTRUTURAS DE REPETIÇÃO

# Definição

## O que são:

São estruturas que permitem executar um bloco de código várias vezes, repetindo uma ação enquanto uma condição for verdadeira ou por um número determinado de vezes.

## Para que serve:

Facilitam a automatização de tarefas repetitivas, evitando a necessidade de escrever o mesmo código diversas vezes, tornando o programa mais eficiente.

## Comandos em python:

- `for`: repete o código para cada item de uma sequência (como uma lista).
- `while`: repete o código enquanto uma condição for verdadeira.

# For:

## O que é:

O for é uma estrutura de repetição que executa um bloco de código para cada elemento de uma sequência (como listas, tuplas, strings ou intervalos).

## Pontos importantes:

- O loop termina automaticamente quando todos os itens da sequência forem processados.
- Muito usado para iterar listas, strings, ranges e outros iteráveis.

## Exemplo:

```
for i in range(1, 6):  
    print(i)
```

Contador de 1 a 5

# While:

## O que é:

O while repete um bloco de código enquanto uma condição for verdadeira. É usado quando não sabemos exatamente quantas vezes o código deve rodar.

## Pontos importantes:

- É essencial que a condição eventualmente se torne falsa, para evitar loops infinitos.
- Permite maior controle sobre a repetição, dependendo de condições dinâmicas.

## Exemplo:

```
contador = 1  
while contador <= 5:  
    print(contador)  
    contador += 1
```

Contador de 1 a 5

# LISTAS

# Definição

## O que são:

Listas são estruturas de dados que armazenam uma coleção ordenada e mutável de elementos, que podem ser de tipos diferentes (números, strings, etc.).

## Exemplo:

```
frutas = ["maçã", "banana", "laranja"]
print(frutas[0]) # imprime "maçã"
frutas.append("uva") # adiciona "uva" no final da lista
print(frutas)
```

## Características principais:

- São ordenadas, ou seja, cada elemento tem uma posição (índice).
- São mutáveis, você pode adicionar, remover ou alterar elementos.
- Permitem armazenar valores duplicados.
- Usam colchetes [ ] para criação e acesso aos elementos.

# Operações com listas

## Adicionar elementos:

`append(valor)`

→ adiciona um elemento no final.

`insert(posicao, valor)`

→ adiciona um elemento em uma posição específica.

## Exemplo:

```
lista = [1, 2, 3]
lista.append(4)          # [1, 2, 3, 4]
lista.insert(1, 10)       # [1, 10, 2, 3, 4]
```

# Operações com listas

## Remover elementos:

`remove(valor)`

→ remove a primeira ocorrência de um valor passado como parâmetro.

`pop()`

→ remove o último elemento.

`pop(indice)`

→ remove o elemento da posição passada como parâmetro.

## Exemplo:

```
lista.remove(10)      # [1, 2, 3, 4]  
lista.pop()          # [1, 2, 3]  
lista.pop(0)         # [2, 3]
```

# FUNÇÕES

# Definição

## O que são:

Funções são blocos de código reutilizáveis que executam uma tarefa específica. Elas ajudam a organizar, reutilizar e deixar o código mais limpo e modular.

## Exemplo:

```
def saudacao():
    print("Olá, bem-vindo!")
```

```
saudacao()
```

## Por que usar funções:

- Evitam repetição de código
- Facilitam a manutenção
- Tornam o programa mais organizado
- Permitem dividir tarefas complexas em partes menores

# Funções com parâmetros e retorno

## Parâmetros:

Permitem que a função receba valores quando for chamada. Assim, ela pode trabalhar com dados diferentes em cada execução.

## Retorno:

Usado para enviar um resultado de volta para quem chamou a função. Isso torna possível armazenar o resultado em variáveis ou usá-lo em outras operações.

## Exemplo:

```
def somar(a, b):  
    return a + b  
  
resultado = somar(3, 5)  
print(resultado) # Saída: 8
```

# Vantagens de usar funções

## 1. Reutilização de código:

**Evita escrever o mesmo código várias vezes. Você pode chamar a função quantas vezes quiser, com diferentes valores.**

## 2. Organização e legibilidade:

**Facilita a leitura e compreensão do código, separando as tarefas em blocos bem definidos.**

## 3. Facilidade na manutenção:

**Se precisar fazer ajustes, você modifica apenas dentro da função — isso economiza tempo e reduz erros.**

## 4. Modularidade:

**Permite dividir o programa em partes menores e independentes, tornando o desenvolvimento mais eficiente.**

## 5. Testes mais simples:

**É mais fácil testar e depurar funções isoladas do que o programa inteiro de uma vez.**