



TECNOLOGIA DA INFORMAÇÃO

PROGRAMAÇÃO EM PYTHON

SENAI SANTOS DUMONT

Rua Pedro Rachid, 304 - Santana - São José dos Campos - SP

<https://saojosedoscamos.sp.senai.br> - Telefone: (12) 3519-4850

Conteúdo do Curso

- **Introdução à Computação**
- **Lógica de Programação**
- **Introdução ao Python**
- **Conceitos de Programação em Python**
- **Estruturas de Repetição**
- **Estrutura de Dados**
- **Módulos e Pacotes**



Introdução a Computação

Funcionamento de um Computador

Um computador é um dispositivo eletrônico controlado por um programa (chamado sistema operacional) , usado para processar dados. Ele é constituído por componentes eletrônicos, especialmente circuitos integrados, miniaturizados e encaixados em um pequeno pedaço de silício, mais conhecido como processador.



Introdução a Computação

Funcionamento de um Computador

O processador é a unidade central de processamento de um computador (CPU), que funciona como o cérebro do computador, pois interage e faz as conexões necessárias entre todos os programas instalados.



Introdução a Computação

Algoritimos e Programas de Computador

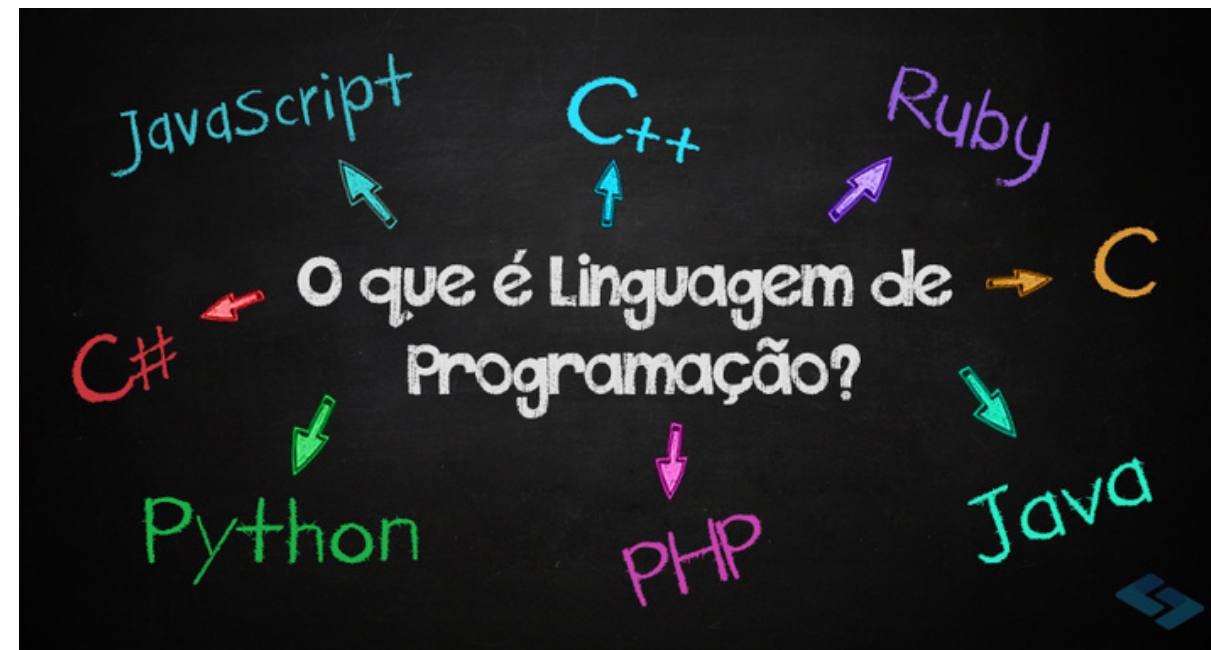
O algoritmo é o conjunto de instruções e regras que um programa de computador (mas não apenas ele) possui para executar suas funções. O conceito é bastante simples e mesmo tarefas simples podem ser descritas na forma de algoritmos, até mesmo as do dia a dia.



Introdução a Computação

Linguagens de Programação

O que é linguagem de programação? É por onde o hardware (máquina) e o programador se comunicam. É uma linguagem formal que funciona por meio de uma série de instruções, símbolos, palavras-chave, regras semânticas e sintáticas.



Introdução a Computação

Linguagens de Programação: Linguagens Mais Usadas Nos Últimos Anos

- Python 
- Java 
- JavaScript 
- C# 
- C++ 
- PHP 



Introdução a Computação

Linguagens de Programação: Curiosidade

O Python estreou no ranking PYPL como a linguagem de programação mais popular, com uma participação de mercado de 27,61%, tornando-se a linguagem de programação mais usada em 2022, embora exista há quase 30 anos



Introdução a Computação

Linguagem de Programação: Baixo nível

Linguagens de programação de baixo nível são aquelas cujos símbolos são uma representação direta do código de máquina.

```
felipe@debian: ~  
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda  
R12: 0x55555554580 (<_start>: xor    ebp,ebp)  
R13: 0x7fffffffef0 --> 0x1  
R14: 0x0  
R15: 0x0  
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)  
[-----code-----]  
0x555555546ab <frame_dummy+43>:  
  jmp     0x555555545f0 <register_tm_clones>  
0x555555546b0 <main>:      push    rbp  
0x555555546b1 <main+1>:    mov     rbp, rsp  
=> 0x555555546b4 <main+4>:    mov     edi, 0xf  
0x555555546b9 <main+9>:    call    0x555555546f5 <recursion_c>  
0x555555546be <main+14>:   mov     esi, eax  
0x555555546c0 <main+16>:   lea     rdi, [rip+0x10d]      # 0x555555547d4  
0x555555546c7 <main+23>:   mov     eax, 0x0  
[-----stack-----]  
0000| 0x7fffffffef10 --> 0x55555554750 (<_libc_csu_init>: push    r15)  
0008| 0x7fffffffef18 --> 0x7ffff7a5a2e1 (<_libc_start_main+241>: mov     edi, eax)  
0016| 0x7fffffffef20 --> 0x400000  
0024| 0x7fffffffef28 --> 0x7fffffffef18 --> 0x7fffffe4e4 ("/home/felipe/tst")  
0032| 0x7fffffffef30 --> 0x1f7b9b508  
0040| 0x7fffffffef38 --> 0x555555546b0 (<main>:      push    rbp)  
0048| 0x7fffffffef40 --> 0x0
```



Introdução a Computação

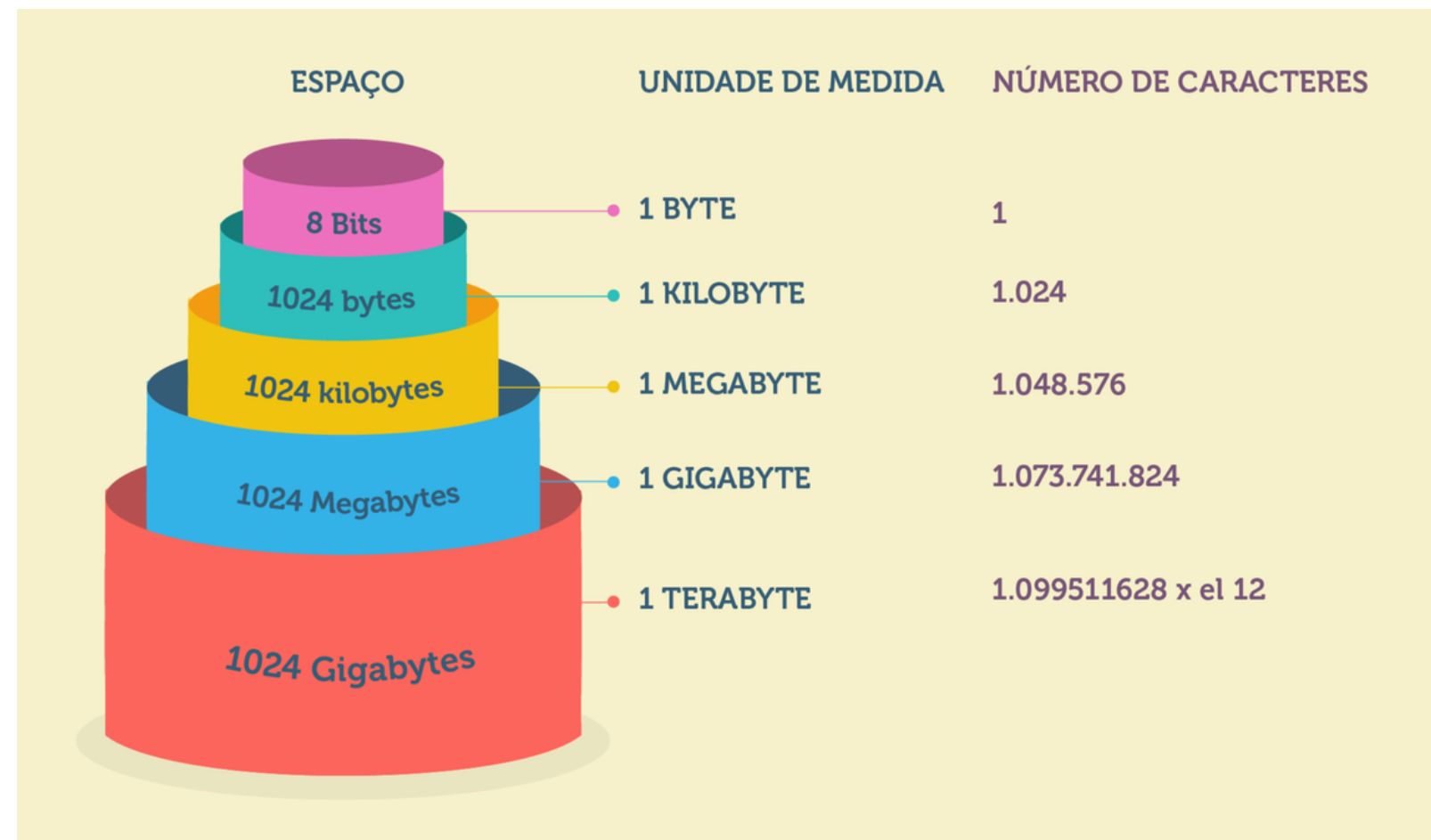
Linguagens de Programação: Linguagem de Máquina

Um programa em código de máquina consiste de uma sequência de bytes que correspondem a instruções a serem executadas pelo processador. Byte é uma unidade de informação digital equivalente a oito bits. Bit é a menor unidade de informação que pode ser armazenada ou transmitida e que pode assumir somente dois valores: 0 ou 1, verdadeiro ou falso e assim por diante.



Introdução a Computação

Linguagens de Programação: Unidade de Medida



O código de caracteres é formado pela união de 8 "zeros" e "uns". Cada 0 e 1 é chamado de BIT, e o conjunto de oito deles é chamado BYTE. Um BYTE consegue armazenar apenas um CARACTERE (letras, números, símbolos, pontuação, espaço em branco e outros caracteres especiais).



Introdução a Computação

Linguagem de Programação: Alto Nivel

Linguagem de programação de alto nível são aquelas composta de símbolos mais complexos, inteligível pelo ser humano e não-executável diretamente pela máquina.

- Exemplo de código PHP (alto nível)
- ```
print ("Bem vindos visitantes do InfoEscola!");
print (" Vamos contar até 50:");
for($x=1;$x<=50;$x++) {
 print $x;
 print " ";
}

if(4 == 2) {
 print ("Fim do mundo! 4 é igual a 2!");
} else {
 print ("Ufa! 4 é diferente de 2");
}
?>
```



# Introdução a Computação

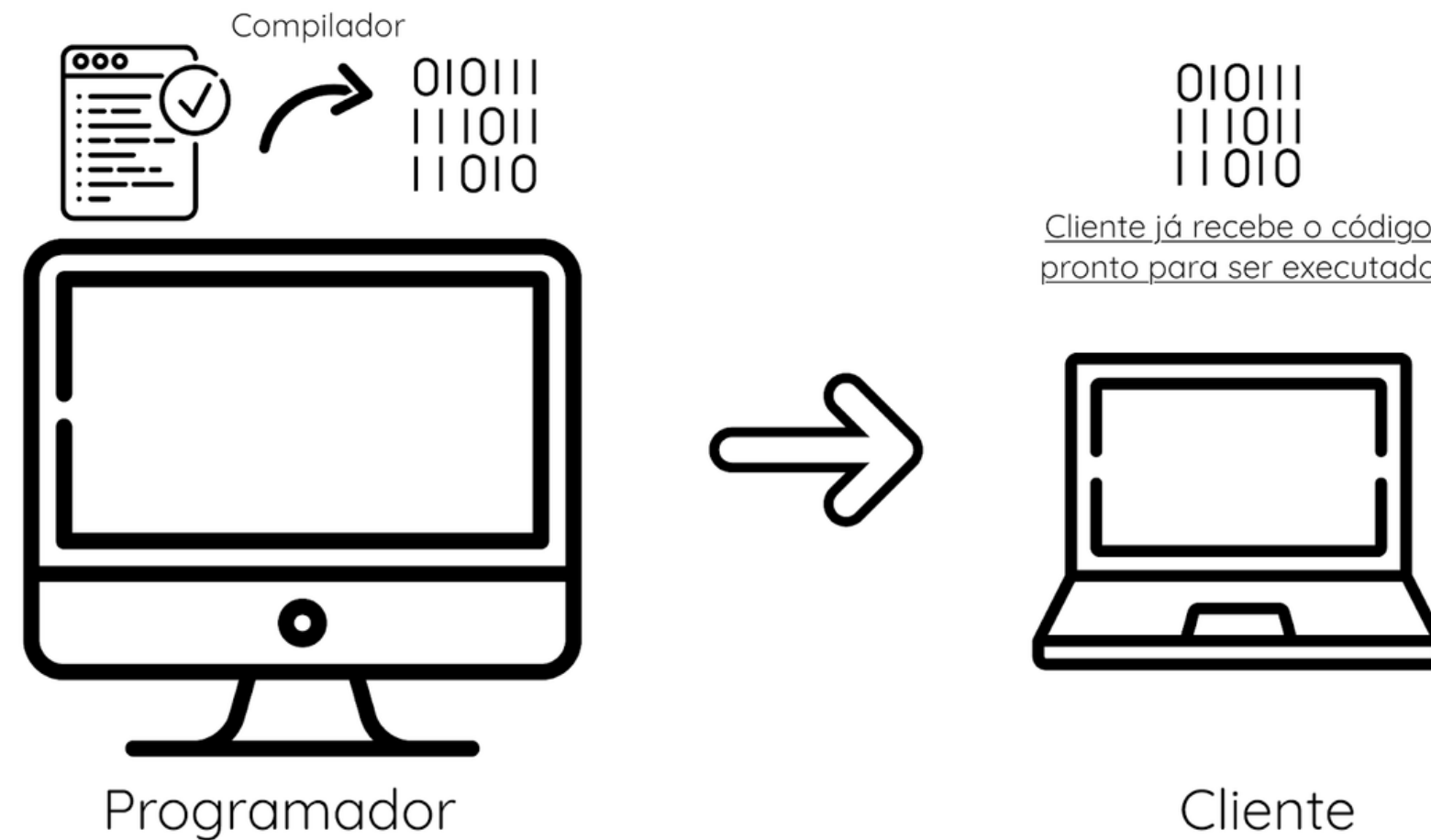
## Linguagem de Programação: Compilada

**O compilador pode ser definido como um programa que traduz todo o código escrito em uma linguagem de programação (código-fonte) em um código de máquina, gerando arquivos adicionais que consigam ser executados pelo computador.**



# Introdução a Computação

## Linguagem de Programação: Compilada

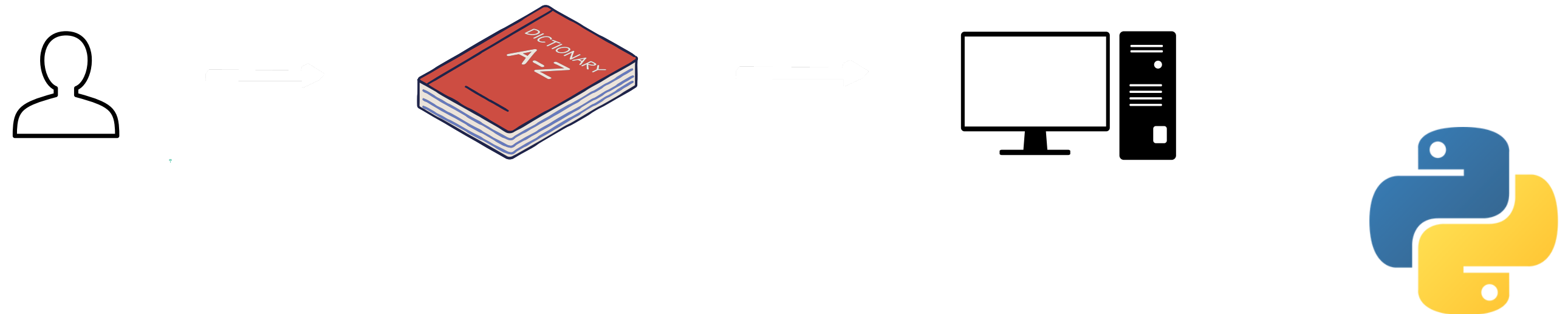




# Introdução a Computação

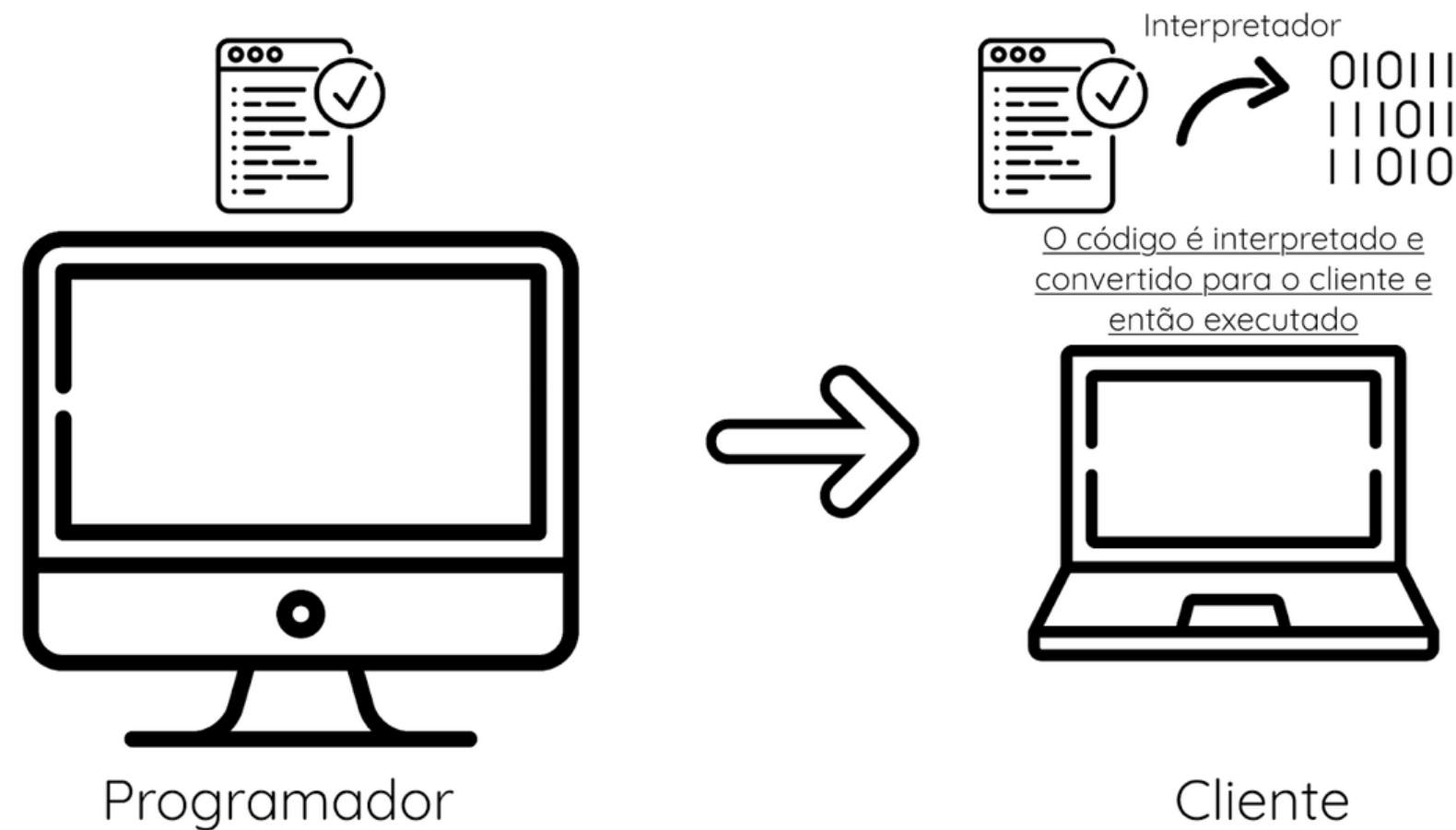
## Linguagem de Programação: Interpretada

**O interpretador, ao contrário do compilador, NÃO traduz o código-fonte inteiro para depois executá-lo, já que essa conversão ocorre simultaneamente à execução do código, deixando de lado a necessidade da criação de arquivos adicionais em código de máquina para serem executados posteriormente.**



# Introdução a Computação

## Linguagem de Programação: Interpretada



# Lógica de Programação

## Abstração Lógica

**Na programação, a abstração sugere a distinção que deve ser feita entre “o que” o programa faz e “como” ele é implementado. Por exemplo, quando um procedimento é chamado, pode-se concentrar somente no que ele faz; apenas quando se está escrevendo o procedimento é que deve-se concentrar em como implementá-lo.**



# Lógica de Programação

## Booleana

**A Lógica Booleana é assim denominada, de acordo com George Boole (1815-1864), matemático e lógico britânico que desenvolveu a teoria da lógica binária, na qual existem somente dois valores possíveis 0 ou 1/ verdadeiro ou falso.**



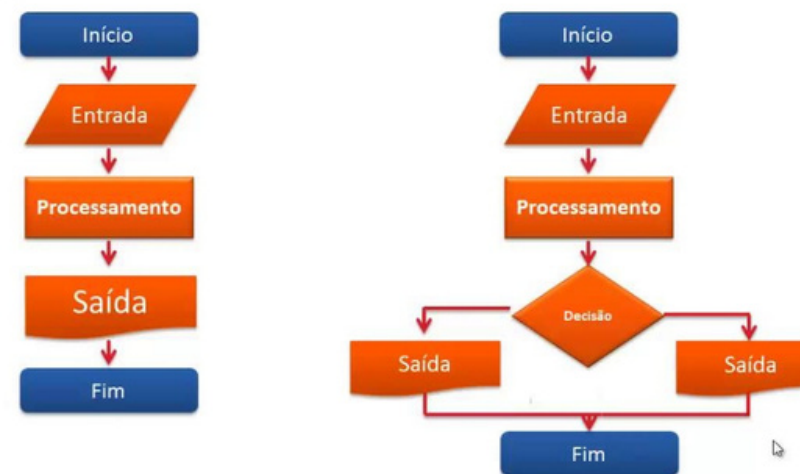
# Lógica de Programação

## Fluxograma

**O diagrama de blocos ou fluxograma é uma forma padronizada eficaz para representar os passos lógicos de um determinado processamento (algoritmos). Com o diagrama podemos definir uma sequência de símbolos, com significado bem definido.**

### ALGORITMO – FLUXOGRAMA / DIAGRAMA DE BLOCOS

Normalmente, os diagrama de blocos computacionais possuem as seguintes estruturas:



# Lógica de Programação

## Operadores Aritiméticos

**Os operadores aritméticos executam operações matemáticas, como adição e subtração com operandos.**



# Lógica de Programação

## Operadores Aritiméticos

| Operador                        | Conceito                                                                       | Exemplo          |
|---------------------------------|--------------------------------------------------------------------------------|------------------|
| + (Adição ou sinal positivo)    | - Realiza a soma entre operandos - Adiciona o sinal de positivo ao número      | - 10 + 7 -<br>+4 |
| - (Subtração ou sinal negativo) | - Realiza a subtração entre operandos - Adiciona o sinal de negativo ao número | - 10 - 7 - -4    |
| * (Multiplicação)               | Realiza a multiplicação entre operandos                                        | 3 * 4            |
| / (Divisão)                     | Realiza a divisão entre operandos                                              | 10 / 5           |
| // (Divisão inteira)            | Realiza a divisão entre operandos e a parte decimal do resultado               | 10 // 6          |
| % (Módulo)                      | Retorna o resto da divisão entre operandos                                     | 4 % 2            |
| ** (Exponenciação)              | Retorna um número elevado a potência de outro                                  | 4 ** 2           |



# Lógica de Programação

## Operadores Aritiméticos: Relacionais

**Operadores aritméticos relacionais são operadores que são usados para comparar ou relacionar valores ou expressões. Exemplos incluem:**

- **Igual (==):** Usado para verificar se dois valores são iguais.
- **Maior que (>):** Usado para verificar se um valor é maior do que outro.
- **Menor que (<):** Usado para verificar se um valor é menor do que outro.
- **Maior ou igual (>=):** Usado para verificar se um valor é maior ou igual a um outro.
- **Menor ou igual (<=):** Usado para verificar se um valor é menor ou igual a um outro.





# Lógica de Programação

## Operadores Aritiméticos: Lógicos

**Operadores aritméticos lógicos são usados para realizar operações matemáticas e lógicas em variáveis ou valores**

**Exemplos:**

- **Adição (+):  $5 + 2 = 7$**
- **Subtração (-):  $5 - 2 = 3$**
- **Multiplicação (\*):  $5 * 2 = 10$**
- **Divisão (/):  $5 / 2 = 2,5$**
- **Módulo (mod):  $5 \bmod 2 = 1$**
- **Potenciação (^):  $5^2 = 25$**
- **Maior que (>):  $5 > 2 = \text{Verdadeiro}$**
- **Menor que (<):  $5 < 2 = \text{Falso}$**



# Lógica de Programação

## Expressões Lógicas e Aritiméticas

**Expressões lógicas e aritméticas são formas de expressar comandos em uma linguagem de programação. Elas consistem em operadores lógicos e aritméticos, além de variáveis.**

**Exemplo de expressão lógica:**

**$(A > B) \text{ AND } (C < D)$**

**Exemplo de expressão aritmética:**

**$A + B * C - D$**



# Lógica de Programação

## Teste de Mesa

**Teste de mesa é uma técnica de teste usada para validar se um software está funcionando como esperado. É realizado por meio de uma tabela contendo todas as entradas possíveis para um sistema e as respectivas saídas esperadas para cada entrada.**

**Exemplo:**

**Entrada | Saída Esperada**

| ----- |  | -----       |
|-------|--|-------------|
| 1     |  | Resultado A |
| 2     |  | Resultado B |
| 3     |  | Resultado C |



# Lógica de Programação

## Refatoração

**Refatoração é o processo de melhoria de um código existente de maneira a torná-lo mais organizado e legível, sem alterar seu comportamento.**

**Exemplo:**

**Original:**

```
def calculate_total(x, y):
```

```
 total = x + y
 return total
```

**Refatorado:**

```
def calculate_total (num1, num2):
 return num1 + num2
```



# Introdução ao Python

## Histórico

**Python é uma linguagem de programação de alto nível criada por Guido van Rossum em 1991. Foi projetada para ser fácil de ler e escrever, e possui uma sintaxe clara que permite que os programadores escrevam programas com menos linhas de código do que outras linguagens.**

**Desde então, Python tornou-se uma das linguagens de programação mais populares e versáteis, sendo usada tanto para programação web quanto para desenvolvimento de aplicativos, análise de dados e muito mais.**

**Desde o lançamento, o Python sofreu várias atualizações, e a versão mais recente é a Python 3.9, lançada em outubro de 2020.**



# Introdução ao Python

## Contexto

**O objetivo do Python é a programação orientada a objetos. O Python é uma linguagem de programação imperativa que permite ao usuário criar scripts, classes, funções e objetos. Além disso, a linguagem também oferece suporte a bibliotecas, módulos e frameworks para facilitar a programação. O Python é considerado uma linguagem versátil, pois é usada para criar software de desktop, servidores web, aplicativos móveis, jogos, Inteligência Artificial, ciência de dados e muito mais.**



# Introdução ao Python

## Versões

**A linguagem de programação Python tem sido desenvolvida e mantida por uma grande comunidade de desenvolvedores em todo o mundo desde 1991. Existem atualmente quatro principais versões do Python:**

- 1. Python 2.7.x: Esta é a versão mais popular do Python, disponível desde 2000. Esta versão é totalmente compatível com todas as versões anteriores.**
- 2. Python 3.7.x: Esta versão foi lançada em 2015 e é considerada a versão mais moderna do Python. Esta versão não é totalmente compatível com versões anteriores.**
- 3. Python 3.8.x: Esta versão foi lançada em 2019 e fornece suporte a novos recursos e melhorias no desempenho.**
- 4. Python 3.9.x: Esta versão foi lançada em 2020 e fornece suporte a novos recursos e melhorias na segurança.**



# Introdução ao Python

## Linguagem compilada versus interpretada

**Python é uma linguagem de programação interpretada. Isto significa que, ao contrário de linguagens compiladas, como C e C++, o código Python não precisa ser compilado antes de ser executado. Em vez disso, o interpretador Python executa o código diretamente, traduzindo-o em ações que o computador pode executar.**





# Conceitos de Programação em Python

## Tipos de dados

**Python possui nove tipos de dados:**

- 1. Números Inteiros - int**
- 2. Números de Ponto Flutuante - float**
- 3. Strings - str**
- 4. Listas - list**
- 5. Tuplas - tuple**
- 6. Dicionários - dict**
- 7. Conjuntos - set**
- 8. Booleanos - bool**
- 9. Tipos de Dados Complexos - complex**



# Conceitos de Programação em Python

## Tipos de dados

### Variáveis:

**Variáveis são espaços reservados na memória do computador para armazenar dados. Elas podem ser usadas para armazenar diversos tipos de dados, como inteiros, strings, listas e dicionários.**

#### Exemplo:

**nome = "João"**

**idade = 20**

**altura = 1.75**

**lista\_de\_cursos = ["Matemática", "Português", "Física"]**

**dicionario\_de\_notas = {"Matemática": 10, "Português": 8, "Física": 9}**



# Conceitos de Programação em Python

## Tipos de dados: Constantes

A regra de nomeação das constantes no Python segue um padrão parecido com as de variáveis, com a diferença de que todas as letras são maiúsculas e separadas por underline "\_". Porém, por possuir tipagem dinâmica, os valores atribuídos à constantes podem ser alterados sem problemas:

```
MINHA_CONSTANTE = 10
```

```
print(MINHA_CONSTANTE) # 10
```

```
MINHA_CONSTANTE = 15
```

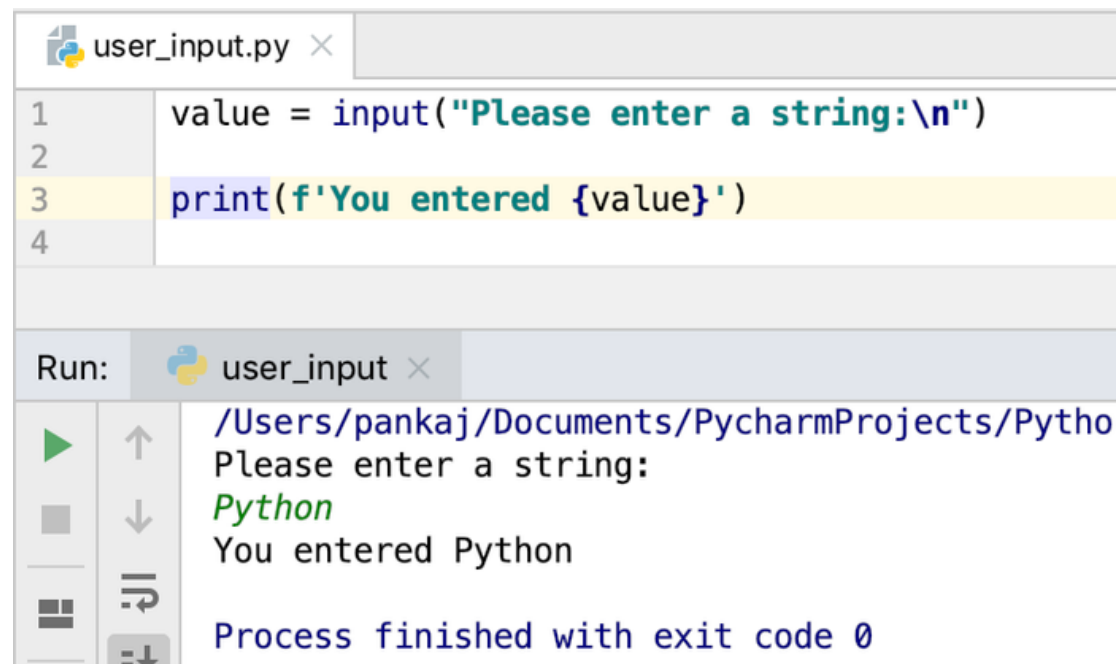
```
print(MINHA_CONSTANTE) # 10
```



# Conceitos de Programação em Python

## Operadores e Operação de E/S

A operação de entrada e saída é feita no Python usando os operadores de entrada e saída, como o operador de entrada (`input`) e o operador de saída (`print`). O operador de entrada solicita ao usuário que insira algum valor ou dado, enquanto o operador de saída imprime os valores ou dados na tela.



```
user_input.py x
1 value = input("Please enter a string:\n")
2
3 print(f'You entered {value}')
4

Run: user_input x
/Users/pankaj/Documents/PycharmProjects/Pytho
Please enter a string:
Python
You entered Python
Process finished with exit code 0
```



# Conceitos de Programação em Python

## **Operadores e Operação de E/S: Manipulação de Entrada e Saída de Dados**

**A manipulação de entrada e saída de dados é o processo de ler e gravar dados de e para arquivos externos como arquivos de texto, planilhas, banco de dados, etc. Em Python, a manipulação de entrada e saída de dados pode ser feita usando a biblioteca padrão "io" e várias outras bibliotecas de terceiros. A biblioteca padrão "io" é usada para manipular arquivos de texto. Ela fornece funções como open(), read(), write(), close() e outras que permitem ler, gravar e gerenciar arquivos.**



# Conceitos de Programação em Python

## Funções

**Funções em Python são peças fundamentais de código reutilizáveis que nos ajudam a evitar repetição de código. Elas nos permitem encapsular lógica de maneira simples, o que nos ajuda a manter o código limpo e organizado. Funções também nos permitem passar diferentes valores para sua lógica, por isso, elas são essenciais para a programação orientada a objetos.**



# Conceitos de Programação em Python

## **Funções:**

### **Variáveis locais e globais**

**Variáveis globais são aquelas que podem ser acessadas em qualquer local dentro do código. Elas são definidas fora de qualquer função ou classe. Variáveis locais são aquelas que só podem ser acessadas dentro da função ou classe onde foram definidas. Elas são definidas dentro de um escopo local e não são acessíveis de outros locais.**



# Conceitos de Programação em Python

## **Funções: Variáveis locais**

**Exemplos de variáveis locais:**

```
def exhibe_y():
```

```
y = 20 # Variável local
```

```
print(y) # Acessando a variável local
```

```
Tentando acessar a variável local y fora da função
```

```
print(y) # Erro, pois a variável y é local à função exhibe_y()
```





# Conceitos de Programação em Python

**Funções:  
Variáveis Globais**

**Exemplos de variáveis globais:**

**x = 10 # Variável global**

**def exibe\_x():  
print(x) # Acessando a variável global**



# Conceitos de Programação em Python

## Funções: Parâmetros

Os parâmetros são os argumentos que são passados para uma função ou método em Python. Os parâmetros podem ser passados de várias maneiras, incluindo por nome, por posição ou por valor padrão. Por exemplo, a função abaixo tem dois parâmetros, nome e idade:

```
def mostrar_info(nome, idade):
 print("Nome:", nome)
 print("Idade:", idade)
```

```
mostrar_info('John', 30)
```

Neste exemplo, os parâmetros nome e idade foram passados por posição.



# Conceitos de Programação em Python

## **Funções: Intrínsecas**

**Python tem um conjunto de funções intrínsecas que são nativamente incorporadas no interpretador de código. Essas funções fornecem acesso a recursos internos da linguagem, como conversão de tipos, operações matemáticas, manipulação de strings e muito mais. Algumas das principais funções Python intrínsecas são:**



# Conceitos de Programação em Python

## Funções: Intrínsecas

- **abs()** - Retorna o valor absoluto de um número
- **all()** - Retorna True se todos os elementos em um iterável são verdadeiros ou se o iterável estiver vazio
- **any()** - Retorna True se qualquer elemento em um iterável for verdadeiro ou se o iterável estiver vazio
- **bin()** - Retorna a representação binária de um inteiro
- **bool()** - Converte um valor em um booleano
- **callable()** - Retorna True se o argumento for chamável
- **chr()** - Retorna o caractere correspondente a um inteiro específico
- **complex()** - Retorna um número complexo
- **dict()** - Cria um dicionário
- **dir()** - Retorna uma lista de atributos e métodos de um objeto



# Conceitos de Programação em Python

## Funções: Intrínsecas

- **enumerate()** - Retorna um objeto enumerado
- **eval()** - Avalia uma string como expressão Python
- **filter()** - Retorna um iterador com elementos filtrados de um iterável
- **float()** - Converte um valor em um número de ponto flutuante
- **format()** - Formata uma string
- **frozenset()** - Retorna um conjunto congelado
- **getattr()** - Retorna o valor de um atributo de um objeto
- **help()** - Abre a documentação Python
- **hex()** - Retorna a representação hexadecimal de um inteiro
- **int()** - Converte um valor em um inteiro



# Conceitos de Programação em Python

## Funções: Intrínsecas

- **isinstance()** - Retorna True se um objeto for uma instância de uma classe
- **len()** - Retorna o comprimento (número de itens) de um objeto
- **list()** - Cria uma lista
- **map()** - Aplica uma função a todos os itens em um iterável e retorna um iterator
- **max()** - Retorna o maior elemento de um iterável
- **min()** - Retorna o menor elemento de um iterável
- **next()** - Retorna o próximo item em um iterável
- **oct()** - Retorna a representação octal de um inteiro
- **ord()** - Retorna o inteiro correspondente a um caractere
- **pow()** - Retorna o resultado da potência



# Conceitos de Programação em Python

## Funções: Intrínsecas

- **range()** - Retorna um objeto de intervalo de números
- **reversed()** - Retorna um iterator de elementos em ordem inversa
- **round()** - Arredonda um número para um determinado número de casas decimais
- **set()** - Cria um conjunto
- **sorted()** - Classifica os elementos de um iterável
- **str()** - Retorna uma representação de string de um objeto
- **sum()** - Retorna a soma dos elementos de um iterável
- **tuple()** - Cria uma tupla
- **type()** - Retorna o tipo de um objeto
- **vars()** - Retorna o `__dict__` de um objeto



# Conceitos de Programação em Python

## Funções: Utilização

**Funções no Python são pedaços de código úteis que podem ser reutilizados para realizar tarefas comuns. Elas são criadas usando a palavra-chave "def" e podem aceitar argumentos para ajudar a realizar suas tarefas. Além disso, elas podem retornar valores que são usados em outros lugares no código.**

**Uma das principais utilidades das funções no Python é a modularização do código. Isso significa que ao dividir o código em pequenas funções, você pode reutilizar o código, o que ajuda a reduzir a complexidade de um programa. Além disso, as funções também ajudam a tornar o código mais legível, pois elas podem ser usadas para abstrair seções complexas do código.**





# Conceitos de Programação em Python

## Exceções

Em Python, existem vários tipos de exceções que podem ocorrer. Alguns dos principais tipos incluem:

- **Erro de sintaxe:** Ocorre quando um programa Python contém código inválido.
- **Erro de nome:** Ocorre quando um programa Python tenta acessar um identificador que não existe.
- **Erro de tipo:** Ocorre quando um programa Python tenta executar operações com tipos inválidos.
- **Erro de valor:** Ocorre quando um programa Python tenta executar operações com valores inválidos.
- **Erro de índice:** Ocorre quando um programa Python tenta acessar uma posição inválida em uma lista ou outra estrutura de dados.
- **Erro de arquivo:** Ocorre quando um programa Python tenta acessar um arquivo que não existe.
- **Erro de chave:** Ocorre quando um programa Python tenta acessar uma chave inválida em um dicionário.
- **Exceção:** Ocorre quando um programa Python encontra uma condição anormal.



# Conceitos de Programação em Python

## Exceções:

### Fluxo de erro com try e except

Um fluxo de erros com try e except é usado para tratar erros específicos em um programa. O bloco try-except é usado para lidar com erros de forma segura e eficiente.

Exemplo:

**try:**

**# Código que pode gerar um erro**

**except:**

**# Código que é executado quando o erro é detectado**

**else:**

**# Código que é executado se nenhum erro ocorrer**

**finally:**

**# Código que é executado independentemente de ocorrer ou não um erro.**



# Conceitos de Programação em Python

## Manipulação de arquivos

**A manipulação de arquivos pode ser feita de diferentes maneiras utilizando o Python. Por exemplo, o Python pode ser usado para ler, escrever e manipular arquivos em formatos como TXT, CSV, HTML, XML, JSON, etc.**

**Para ler arquivos, o Python possui um conjunto de funções que podem ser usadas para abrir, ler o conteúdo do arquivo e fechá-lo. Por exemplo, a função `open()` pode ser usada para abrir um arquivo, enquanto a função `read()` pode ser usada para ler seu conteúdo.**

**Para escrever arquivos, o Python também possui uma série de funções úteis, incluindo a função `write()` para escrever dados em um arquivo existente. A função `create()` pode ser usada para criar novos arquivos**



# Conceitos de Programação em Python

## Manipulação de arquivos: Árvores de diretórios

**Python possui uma biblioteca que permite a manipulação de árvores de diretórios. Essa biblioteca, chamada os, fornece funções para criar, excluir, mover, caminhar e listar diretórios. Ela também fornece funções para manipular arquivos.**

**A função getcwd () fornece o diretório atual. As funções mkdir () e rmdir () podem ser usadas para criar e excluir diretórios, respectivamente. A função listdir () pode ser usada para listar os arquivos e diretórios em um diretório específico. A função chdir () pode ser usada para alterar o diretório corrente. A função isfile () pode ser usada para verificar se um determinado caminho é um arquivo. A função isdir () pode ser usada para verificar se um determinado caminho é um diretório.**

**Além desses, existem outras funções na biblioteca os que podem ser usadas para manipular árvores de diretórios. Portanto, a biblioteca os é uma ferramenta muito útil para manipular e explorar diretórios no Python.**



# Conceitos de Programação em Python

## Manipulação de arquivos: Sistemas de arquivos

**Manipular dados em sistemas de arquivos é uma tarefa comum e importante para desenvolvedores de Python. O Python possui várias bibliotecas que tornam a manipulação de arquivos muito mais fácil.**

**A biblioteca padrão do Python, o "os" e o "os.path", oferecem funções para manipular diretórios, manipular caminhos e listar arquivos em um diretório. O "shutil" também oferece funções para copiar, mover, renomear e excluir arquivos.**

**Também existem algumas bibliotecas de terceiros que oferecem recursos ainda mais avançados. A biblioteca "glob" pode ser usada para encontrar arquivos usando um padrão de caracteres. A biblioteca "pathlib" permite acessar informações sobre arquivos, como o caminho, o nome, o tamanho e a data de criação.**

**O Python também oferece funções para ler e escrever dados em arquivos. O "open" pode ser usado para abrir um arquivo para leitura ou gravação. O "read" e o "write" podem ser usados para ler e gravar dados do arquivo, respectivamente. O "close" deve ser usado para fechar o arquivo ao final do processo de leitura ou gravação.**



# Estruturas de repetição

**Python possui três estruturas de repetição: for, while e do-while. Elas são usadas para executar um bloco de código várias vezes, dependendo das condições especificadas. As estruturas de repetição podem ser usadas para executar um código específico até que uma determinada condição seja satisfeita. Isso permite que você execute seu código de forma mais eficiente e evite a necessidade de escrever o mesmo código várias vezes.**



# Estruturas de repetição

## Tomada de decisão

**Python Estruturas de repetição tomada de decisão é um conceito de programação que permite ao programador criar um código que possa tomar decisões com base em um conjunto de condições. Esta estrutura permite que a lógica de programação seja repetida até que uma condição seja satisfeita ou que um determinado resultado seja alcançado. Esta estrutura é comumente usada para programar loops de repetição, onde um conjunto de instruções deve ser executado até que certas condições sejam satisfeitas. Ela também é usada para criar estruturas de decisão, onde código diferente é executado dependendo do resultado de uma condição.**



# Estruturas de repetição

## Tomada de decisão

**# Exemplo de tomada de decisão usando uma estrutura de repetição**

```
valor = int(input("Digite um valor inteiro: "))
```

**# Se o valor for maior que 10, multiplicar por 3**

**if valor > 10:**

```
 resultado = valor * 3
```

```
 print("O resultado é:", resultado)
```

**# Se o valor for menor ou igual a 10, somar com 7**

**else:**

```
 resultado = valor + 7
```

```
 print("O resultado é:", resultado)
```





# Estruturas de repetição

## Tomada de decisão: if

**Uma estrutura de repetição if é uma estrutura de controle de programação que permite que um programa execute um conjunto de instruções de acordo com uma condição booleana. Se a condição for verdadeira, o programa executará as instruções dentro do bloco de código "if". Se a condição for falsa, o programa pulará essas instruções e continuará executando o restante do código. Por exemplo, você pode usar uma estrutura de repetição if para verificar se um determinado número é par ou ímpar. Se o número for par, o programa imprimirá "par". Se o número for ímpar, o programa imprimirá "ímpar".**



# Estruturas de repetição

## Tomada de decisão: if

**# Exemplo 1**

**number = 5**

**if number > 0:**

**print("O número é positivo")**

**# Exemplo 2**

**name = "John"**

**if name == "John":**

**print("O nome é John")**



# Estruturas de repetição

## Tomada de decisão: if-else

**As estruturas de repetição if são usadas para executar um bloco de código se uma determinada condição for atendida. Elas são usadas para controlar o fluxo do programa com base em algumas condições.**

**No Python, o if é usado para verificar se uma condição específica é verdadeira ou falsa. Se a condição for verdadeira, o bloco de código fornecido dentro do if será executado. Caso contrário, o código fornecido no else será executado.**



# Estruturas de repetição

## Tomada de decisão: if-else

**Exemplo:**

**# Programa para verificar se um número é par ou ímpar**

**num = int(input("Digite um número inteiro: "))**

**# Verifica se o número digitado é par**

**if num % 2 == 0:**

**print("O número digitado é par.")**

**else:**

**print("O número digitado é ímpar.")**



# Conceitos de Programação em Python

## Funções: Intrínsecas

**Estruturas de repetição if-elif-else são usadas para executar ações diferentes com base em diferentes condições. A estrutura if-elif-else começa com uma expressão if que é avaliada como verdadeira ou falsa. Se a expressão for avaliada como verdadeira, todas as declarações dentro do bloco if são executadas. Se a expressão for avaliada como falsa, a estrutura if-elif-else avalia a expressão elif. Se a expressão elif for avaliada como verdadeira, todas as declarações dentro do bloco elif são executadas. Se a expressão elif for avaliada como falsa, a estrutura if-elif-else avalia a expressão else. Se a expressão else for avaliada como verdadeira, todas as declarações dentro do bloco else são executadas. Se nenhuma das expressões for avaliada como verdadeira, nenhuma ação é executada.**



# Estruturas de repetição

**Tomada de decisão:  
if-elif-else**

```
numero = int(input("Digite um número: "))
```

```
if numero > 0:
 print("O número é positivo!")
elif numero < 0:
 print("O número é negativo!")else:
 print("O número é 0!")
```



# Estruturas de repetição

## Laços

**Um laço (loop) é uma estrutura de repetição na programação que permite que um bloco de código seja executado repetidamente enquanto uma condição for verdadeira. Existem dois tipos principais de laços em Python: for loops e while loops. O laço for é usado para iterar sobre sequências, como listas, tuplas, strings e dicionários. É usado para executar um bloco de código para cada elemento na sequência. O laço while é usado para iterar enquanto a condição for verdadeira. Ele é usado para executar um bloco de código repetidamente enquanto uma condição for verdadeira. Os laços são úteis para automatizar tarefas repetitivas e para processar dados. Eles também são usados para executar um bloco de código**



# Estruturas de repetição

## Laços

```
for numero in range(1, 11): # Loop de 1 a 10
print(numero) # Imprimir cada número
Saída:
1
2
3
4
5
6
7
8
9
10
```





# Estruturas de repetição

## Laços: for

**Um laço for é utilizado para iterar sobre uma sequência (como uma lista ou string). Isso permite que o usuário execute um bloco de código uma vez para cada elemento da sequência.**

**Exemplo:**

```
numbers = [1, 2, 3, 4, 5]
```

```
for num in numbers:
 print(num)
```



# Estruturas de repetição

## Laços: while

Um laço **while** executa um bloco de código até que uma condição especificada se torne falsa.

**Exemplo:**

```
n = 5
```

```
while n > 0:
 print(n)
 n = n - 1
```

```
print("Fim do loop")
```

**Saída:**

```
5
4
3
2
1
Fim do loop
```



# Estrutura de dados no Python

**Python oferece várias estruturas de dados para armazenar informações. Estas estruturas incluem listas, tuplas, dicionários e conjuntos.**

- **Listas:** Listas são estruturas de dados flexíveis que permitem armazenar e acessar facilmente os dados. Eles são mutáveis, o que significa que você pode adicionar, alterar ou remover elementos.
- **Tuplas:** Tuplas são similares às listas, mas são imutáveis. Isso significa que uma vez criadas, não podem ser alteradas.
- **Dicionários:** Dicionários permitem mapear chaves para valores. Eles são úteis para armazenar informações relacionadas a uma entidade, como um usuário ou produto.
- **Conjuntos:** Conjuntos são estruturas de dados não ordenadas que permitem armazenar elementos únicos. Eles são úteis para realizar operações de conjunto, como encontrar elementos comuns entre dois conjuntos.



# Estrutura de dados no Python

## Listas

**Uma lista é uma estrutura de dados no Python que consiste em uma sequência de itens. A lista é mutável, ou seja, os itens podem ser adicionados, removidos ou alterados. Elas são criadas usando colchetes [] e os itens são separados por vírgula. Por exemplo:**

```
lista = [1, 2, 3, 4, 5]
```

**Nesta lista, temos 5 itens: 1, 2, 3, 4 e 5. Podemos usar índices para acessar os itens, começando do 0. Então, lista [0] será 1, lista [1] será 2 e assim por diante. Podemos manipular os itens da lista fazendo operações como adicionar, remover, atualizar e pesquisar. Além disso, podemos usar métodos embutidos como sort (), append (), pop (), remove (), entre outros.**



# Estrutura de dados no Python

## Listas aninhadas

**Uma lista aninhada é uma lista de listas no Python. Por exemplo, a seguinte lista aninhada possui três listas internas, cada uma contendo três elementos:**

```
lista_aninhada = [['item1', 'item2', 'item3'], ['item4', 'item5', 'item6'], ['item7', 'item8', 'item9']]
```

**Você pode acessar elementos individuais na lista aninhada usando índices duplos. Por exemplo, para acessar o segundo item da segunda lista, você usaria o índice [1][1]:**

```
print(lista_aninhada[1][1])
```

**Saída:**

**item5**



# Estrutura de dados no Python

## Tuplas

**As tuplas são estruturas de dados imutáveis e sequenciais. Elas são geralmente usadas para armazenar conjuntos de dados relacionados que não devem ser alterados. Por exemplo, você pode usar tuplas para armazenar informações como nome, idade e endereço de um usuário:**

```
user_info = ("John", 23, "123 Main Street")
```



# Estrutura de dados no Python

## **Tuplas: Finalidade**

**Uma tupla é uma estrutura de dados imutável (não pode ser modificada) que armazena uma sequência ordenada de elementos. Ela é usada para armazenar conjuntos de dados que não podem ser alterados durante a execução do programa, como datas, horários, coordenadas geográficas, etc. Tuplas também são úteis para criar estruturas de dados que precisam ser acessadas por meio de índices, como listas. Além disso, elas são mais eficientes do que listas, pois usam menos memória.**



# Estrutura de dados no Python

## Tuplas: Construção

**Uma tupla é uma estrutura de dados imutável em Python que consiste em uma sequência de valores separados por vírgulas. Tuplas são usadas para armazenar dados que não devem ser alterados durante o programa. Elas também são úteis para passar dados entre funções.**

**Tuplas são criadas usando parênteses e cada elemento é separado por uma vírgula. Por exemplo:**

```
tupla = (1, 2, 3)
```

**Tuplas também podem ser criadas sem parênteses, desde que os elementos sejam separados por vírgulas. Por exemplo:**

```
tupla = 1, 2, 3
```

**Tuplas também podem conter outros objetos, como listas, strings, dicionários, etc. Por exemplo:**

```
tupla = (1, "dois", [3, 4])
```





# Estrutura de dados no Python

## Tuplas: Utilização

**Tuplas são estruturas de dados no Python usadas para armazenar um conjunto de dados imutável. Uma tupla é semelhante a uma lista, exceto que ela é escrita usando parênteses ().**

**Uma das principais vantagens da tupla é que ela é imutável, o que significa que não podemos adicionar, remover ou alterar os valores nela armazenados. Isso torna as tuplas ideais para armazenar dados que não precisam ser alterados, como informações de usuário ou dados de configuração.**

**Tuplas também são muito úteis para iterar através de dados. Por exemplo, podemos iterar através de uma tupla de nomes e imprimi-los na tela:**

```
names = ("John", "Paul", "George", "Ringo")
```

```
for name in names:
 print(name)
```

**O código acima imprimiria os nomes John, Paul, George e Ringo na tela.**



# Estrutura de dados no Python

## Dicionários

**Os dicionários são estruturas de dados no Python que permitem armazenar informações de forma organizada. Os dicionários são semelhantes a listas, mas ao invés de usar índices para acessar seus elementos, eles usam chaves. Esta característica permite que os dicionários armazenem informações de forma mais eficiente. Os dicionários são definidos usando chaves {}, e seus elementos são definidos como pares chave:valor.**



# Estrutura de dados no Python

## Dicionários: Finalidade

**Os dicionários no Python são usados para armazenar e acessar rapidamente dados não ordenados. Uma das principais finalidades de um dicionário é fornecer uma maneira eficiente de acessar dados usando chaves. As chaves fornecem um modo eficaz de localizar rapidamente um elemento específico no dicionário. Por exemplo, se você estiver procurando uma palavra específica em um dicionário, você pode usar a chave para localizá-la rapidamente. Além disso, os dicionários também podem ser usados para armazenar e acessar grandes quantidades de dados, como as informações de um banco de dados.**



# Estrutura de dados no Python

## Dicionários: Construção

**Um dicionário é uma estrutura de dados no Python que é uma coleção não ordenada de pares chave-valor. Uma chave é usada para identificar um item específico dentro do dicionário, enquanto o valor é o dado armazenado para a chave.**

**Para criar um dicionário, você usa chaves { } para envolver uma lista de itens. Cada item na lista é um par chave-valor, separado por vírgulas e consiste em uma chave seguida do seu valor. Por exemplo:**

```
dicionario = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

**Aqui, 'nome', 'idade' e 'cidade' são as chaves e 'João', 30 e 'São Paulo' são os valores.**



# Estrutura de dados no Python

## Dicionários: Utilização

**Dicionários são estruturas de dados mutáveis no Python que consistem em pares chave-valor. Eles são usados para armazenar informações que podem ser acessadas rapidamente usando sua chave. As chaves de um dicionário devem ser únicas e devem ser do tipo inteiro, string, float ou tupla. Os valores contidos nos dicionários também podem ser do tipo inteiro, string, float ou tupla.**

**Dicionários são muito úteis para armazenar informações que serão acessadas frequentemente. Por exemplo, um dicionário pode ser usado para armazenar os dados do usuário de um aplicativo, incluindo seu nome, endereço, data de nascimento e muito mais. Dessa forma, é possível acessar qualquer informação do usuário rapidamente, bastando para isso fornecer a chave desejada.**



# Módulos e pacotes

**Módulos são arquivos que contêm definições e declarações de código (funções, classes etc). Eles nos permitem reutilizar código em diferentes programas.**

**Pacotes são uma forma de organizar os módulos em Python. Um pacote é uma coleção de módulos que possuem funções relacionadas entre si. de texto**



# Módulos e pacotes

## Definição

**Módulos e pacotes são arquivos contendo códigos Python que podem ser reutilizados em aplicações e projetos. Um módulo é um arquivo contendo definições e declarações Python, enquanto um pacote é um diretório contendo múltiplos módulos. Os módulos e pacotes fornecem estruturas de dados e funções que podem ser usadas para simplificar a escrita de programas Python.**



# Módulos e pacotes

## Ferramenta PIP

**A ferramenta pip é um gerenciador de pacotes para o Python que permite instalar, atualizar e remover pacotes facilmente. Ele pode ser usado para instalar bibliotecas de terceiros, o que torna o Python particularmente poderoso e versátil. As bibliotecas permitem que você acesse funções e métodos específicos para realizar tarefas específicas.**

