APOSTILA DE LÓGICA DE PROGRAMAÇÃO EM PYTHON

Brasília DF

2025

APOSTILA DE LÓGICA DE PROGRAMAÇÃO EM PYTHON

Autor: Prof. Karython Gomes

Ano: 2025

Ilustrações

Tabelas

Sumário

GLOSSÁRIO

Algoritmo: Sequência de passos para realizar uma tarefa ou resolver um problema.

Booleano (bool): Tipo de dado que representa verdadeiro (True) ou falso (False).

Código-fonte: Conjunto de instruções escritas em linguagem de programação.

Compilar: Processo de transformar código-fonte em linguagem de máquina.

Função: Bloco de código reutilizável que realiza uma tarefa específica.

IDE: Ambiente de desenvolvimento.

Laço (Loop): Estrutura que permite repetir um bloco de código.

Lista: Estrutura de dados que armazena múltiplos valores e pode ser alterada.

Parâmetro: Valor passado para uma função.

Python: Linguagem de programação de alto nível, simples e poderosa.

Script: Arquivo contendo instruções em linguagem de programação.

String: Sequência de caracteres (texto).

Terminal: Interface onde comandos são digitados diretamente.

Tupla: Estrutura de dados imutável que armazena múltiplos valores.

Variável: Espaço de armazenamento para dados usados em um programa.

Mensagem do Professor

Caro(a) estudante,

Seja bem-vindo(a) à sua jornada com a **Apostila de Lógica de Programação em Python**. Aqui, você terá a oportunidade de aprender uma linguagem moderna, versátil e amplamente utilizada em áreas como desenvolvimento web, automação, ciência de dados e inteligência artificial. Python é conhecida por sua sintaxe clara e pela facilidade de aprendizado, o que a torna ideal para quem está começando na programação.

Para obter o melhor aproveitamento deste material, é essencial que você **pratique constantemente**, compreenda os **fundamentos da linguagem**, explore as **estruturas de dados** e use o **vocabulário técnico** com familiaridade. As atividades práticas, as tabelas comparativas e os diagramas visuais são ferramentas pensadas para reforçar seu aprendizado de forma clara e progressiva.

Ao final da apostila, você encontrará o **Projeto Integrador**, um desafio que simula um cenário real de desenvolvimento. Essa é a sua chance de aplicar todos os conceitos aprendidos: listas, dicionários, funções, controle de fluxo e manipulação de arquivos. Encare os erros como parte natural do processo e lembre-se: programar é construir soluções — passo a passo, linha por linha. Estou à disposição para apoiá-lo(a) nesse caminho.

Prof. Karython Gomes
Instrutor SENAI DF

1. INTRODUÇÃO À PROGRAMAÇÃO

1.1 O que é Programação?

Programação é o processo de criar instruções para que um computador execute tarefas específicas. Essas instruções são escritas em linguagens de programação, como o Python.

1.2 Por que Python?

Python é uma linguagem moderna, versátil e fácil de aprender. É amplamente usada em áreas como ciência de dados, desenvolvimento web, automação e inteligência artificial.

1.3 Primeira Aplicação: Olá, Mundo!

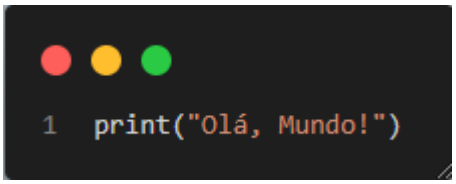
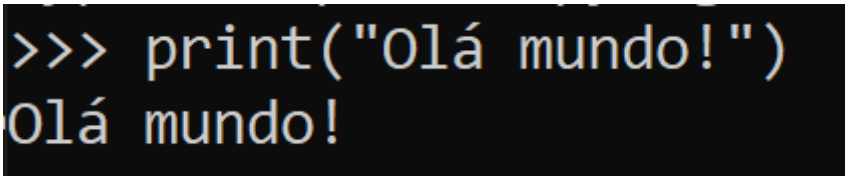


Imagem ilustrativa:



1.4 Vocabulário Técnico

- **Código-fonte:** conjunto de instruções escritas em linguagem de programação.
- **Terminal:** interface onde comandos podem ser executados diretamente no computador.
- **Compilar:** processo de transformar o código-fonte em um formato que o computador consiga executar.

Atividades Práticas

1. Instale o Python em seu computador e execute o exemplo "Olá, Mundo!".
2. Pesquise e escreva uma breve definição sobre o que é uma linguagem de programação.
3. Explique com suas palavras o motivo de escolher Python como linguagem inicial.
4. Crie um programa que exiba seu nome, idade e cidade.
5. Escreva em seu caderno o significado de “código-fonte” e “terminal” com exemplos.

2. INSTALAÇÃO E AMBIENTE DE DESENVOLVIMENTO

2.1 Instalação do Python

Para iniciar a programação com Python, é necessário instalar a linguagem em seu computador. O processo consiste em:

1. Acessar o site oficial do Python: https://www.python.org
2. Clicar em “Downloads” e selecionar a versão recomendada para seu sistema operacional.
3. Executar o instalador e marcar a opção “Add Python to PATH” antes de clicar em “Install Now”.

Após a instalação, é possível verificar se está funcionando corretamente abrindo o terminal e digitando:

Se o retorno for algo semelhante a Python 3.12.0, a instalação foi realizada com sucesso.

Imagem ilustrativa:

2.2 Escolhendo um Ambiente de Desenvolvimento

Um ambiente de desenvolvimento é o conjunto de ferramentas que permite escrever, testar e executar códigos. Para alunos iniciantes, recomenda-se o uso do **IDE** (ambiente de desenvolvimento Python) ou do **VS Code (Visual Studio Code)**.

IDE

- IDE de desenvolvimento Python.
- Simples e direto para pequenos testes.
- Desenvolvida pela JetBrains, oferece recursos como autocompletar inteligente, depuração, testes integrados e controle de versão.
- Disponível nas versões Community (gratuita) e Professional (com recursos adicionais).

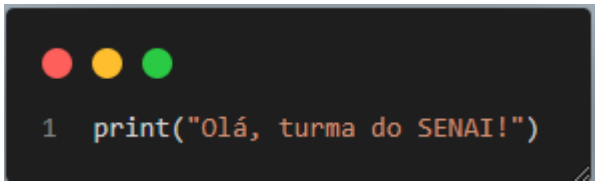
Visual Studio Code

- Editor moderno com suporte a extensões.
- Requer instalação à parte no site: https://code.visualstudio.com
- Para ativar o suporte ao Python, instale a extensão “Python” pela galeria de extensões.

Imagem ilustrativa:

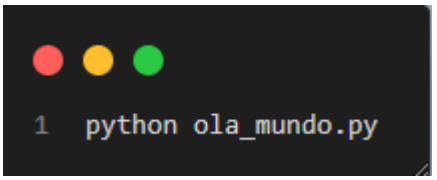
2.3 Criando e Executando seu Primeiro Script

1. Abra o editor de sua preferência.
2. Crie um novo arquivo com extensão .py, por exemplo: ola_mundo.py
3. Digite o seguinte código:



4. Salve o arquivo e execute:

    - No IDE: pressione F5.
    - No VS Code: abra o terminal integrado e digite:



Vocabulário Técnico

- **PATH:** variável do sistema que permite o reconhecimento de comandos no terminal.
- **Script:** arquivo de texto que contém código-fonte.
- **Editor de código:** programa utilizado para escrever e modificar scripts.

Atividades Práticas

1. Instale o Python em seu computador seguindo as instruções apresentadas.
2. Crie um arquivo ola_mundo.py e execute-o no IDLE.
3. Instale o Visual Studio Code e configure a extensão Python.
4. Comente, em um documento, as diferenças entre o IDLE e o VS Code.
5. Pesquise o significado da variável PATH e explique com suas palavras seu papel na execução de comandos.

---

3. TIPOS DE DADOS E OPERAÇÕES BÁSICAS

3.1 Tipos de Dados em Python

Os tipos de dados representam diferentes categorias de valores que podem ser manipulados em um programa. Abaixo estão os principais tipos básicos em Python:

- **Inteiros (int)**: números inteiros, positivos ou negativos.
  Exemplo: 10, -5

- **Ponto flutuante (float)**: números decimais.
  Exemplo: 3.14, -0.5
- **Texto (str)**: sequência de caracteres entre aspas.
  Exemplo: "Olá", 'Python'
- **Booleanos (bool)**: representam valores lógicos, verdadeiro ou falso.
  Exemplo: True, False

3.2 Operações Aritméticas

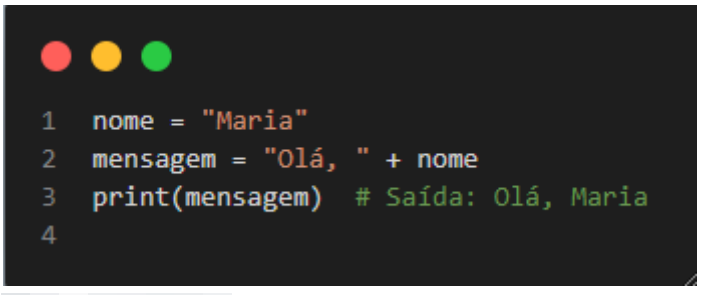
Python realiza operações matemáticas com facilidade. Veja abaixo os operadores mais comuns:

Tabela 1 - operadores aritméticos

Operador	Função	Exemplo	Resultado
+	Adição	5 + 2	7
-	Subtração	7 - 3	4
*	Multiplicação	4 * 3	12
/	Divisão	10 / 2	5.0
**	Potenciação	2 ** 3	8
%	Módulo (resto)	10 % 3	1 //

3.3 Concatenando Textos

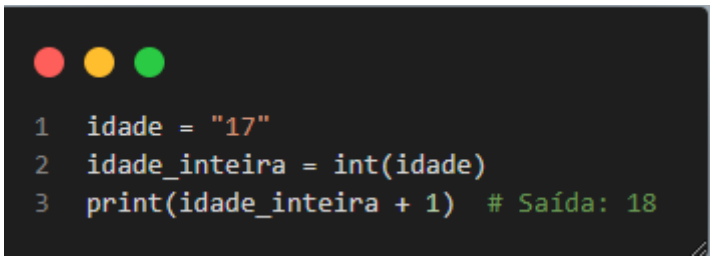
Com o operador +, é possível unir textos (strings):



```
1  nome = "Maria"
2  mensagem = "Olá, " + nome
3  print(mensagem)  # Saída: Olá, Maria
4
```

3.4 Conversão de Tipos

Às vezes é necessário converter um valor de um tipo para outro. Exemplo:



```
idade = "17"
idade_inteira = int(idade)
print(idade_inteira + 1)  # Saída: 18
```

Vocabulário Técnico

- **Tipo de dado:** categoria que define o formato e comportamento de um valor.
- **String:** texto entre aspas simples ou duplas.
- **Booleano:** tipo lógico que representa verdadeiro ou falso.
- **Concatenar:** unir duas ou mais strings.
- **Conversão de tipo:** transformação de um valor de um tipo para outro.

Atividades Práticas

1. Crie variáveis com os tipos **int, float, str** e **bool** e imprima seus valores.
2. Realize as operações `12 + 7`, `15 % 4`, `3 ** 2` e verifique os resultados.
3. Crie um programa que peça o nome do usuário e exiba uma mensagem de boas-vindas.
4. Converta uma string `"20"` para inteiro e somar com um número.
5. Identifique e escreva no caderno os operadores aritméticos estudados e seus significados.

---

4. CONTROLE DE FLUXO (CONDICIONAIS E LAÇOS)

4.1 Estruturas Condicionais (Decisão)

As estruturas condicionais permitem que o programa tome decisões com base em uma condição.

Exemplo:

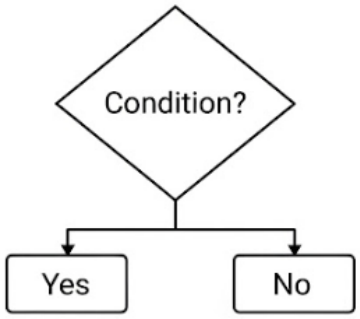
```python
idade = 17

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

```

Esse código verifica se a idade é maior ou igual a 18. Se verdadeiro, executa o primeiro bloco; caso contrário, o segundo.

Imagem ilustrativa



4.2 Operadores de Comparação

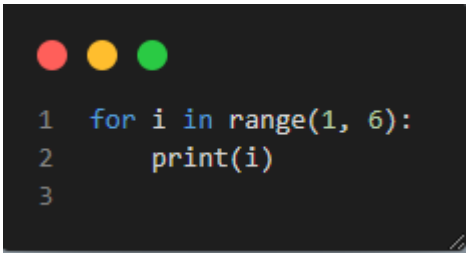
Tabela 2 - operadores de comparação

Operador	Significado	Exemplo	Resultado
==	Igualdade	5 == 5	True
!=	Diferença	3 != 4	True
>	Maior	10 > 8	True
<	Menor	7 < 9	True
>=	Maior ou igual	4 >= 4	True
<=	Menor ou igual	6 <= 5	False

4.3 Estruturas de Repetição (Laços)

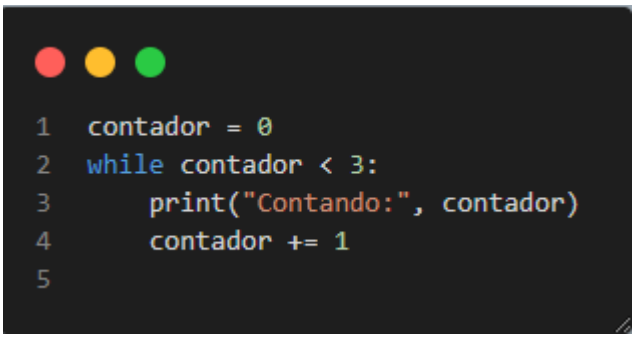
Permitem repetir ações diversas vezes, de forma automática.

Laço for – usado para repetir algo um número conhecido de vezes.



Este laço imprime os números de 1 a 5.

Laço while – executa enquanto a condição for verdadeira.

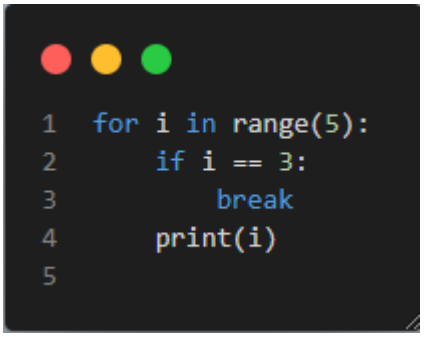


```
contador = 0
while contador < 3:
    print("Contando:", contador)
    contador += 1

```

4.4 Controle de Laço com break e continue

- break: interrompe o laço imediatamente.
- continue: pula a iteração atual e vai para a próxima.



```
for i in range(5):
    if i == 3:
        break
    print(i)

```

Vocabulário Técnico

- **Condicional:** estrutura que permite tomar decisões.
- **Laço de repetição:** mecanismo para executar blocos de código várias vezes.
- **for:** laço com número definido de repetições.
- **while:** laço baseado em condição.
- **break e continue:** comandos de controle de repetição.

Atividades Práticas

1. Crie um programa que peça a idade do usuário e diga se ele é maior ou menor de idade.
2. Use um laço for para imprimir os números pares de 0 a 10.
3. Escreva um programa que solicite números até que o usuário digite 0, usando while.
4. Faça um programa que, ao encontrar o número 7 em uma sequência, interrompa a execução usando break.
5. Reescreva uma das atividades anteriores utilizando continue.

---

5. FUNÇÕES

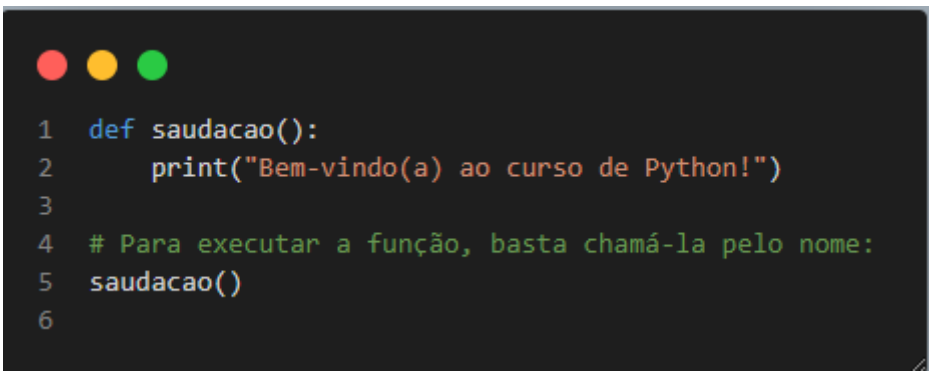
5.1 O que são Funções?

Funções são blocos de código que executam tarefas específicas. Elas permitem reutilizar instruções sem repetir o mesmo código várias vezes. Podem receber informações, executar ações e retornar resultados.

5.2 Criando uma Função

A estrutura básica para definir uma função utiliza a palavra-chave def, seguida pelo nome da função, parênteses e dois-pontos.

Exemplo:

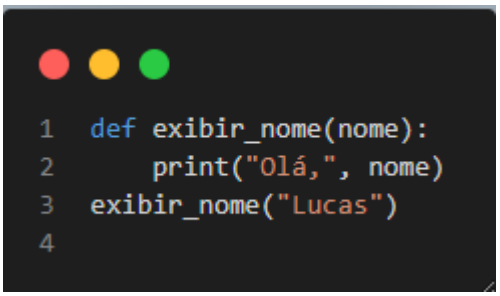


```
1  def saudacao():
2      print("Bem-vindo(a) ao curso de Python!")
3
4  # Para executar a função, basta chamá-la pelo nome:
5  saudacao()
6
```

5.3 Funções com Parâmetros

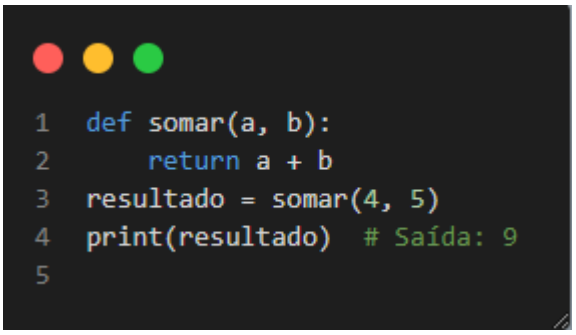
Podemos tornar a função mais flexível ao receber valores como entrada.

Exemplo:



5.4 Funções com Retorno

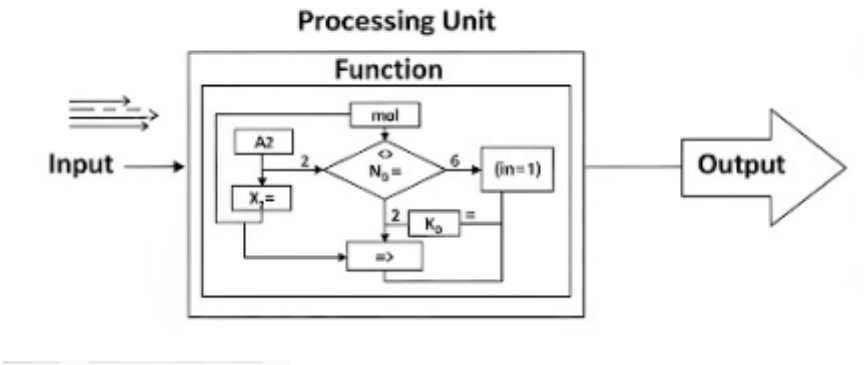
Algumas funções devolvem um valor após a execução. Para isso, usamos a palavra return.



5.5 Vantagens do Uso de Funções

- Organização do código.
- Reutilização de lógica.
- Facilidade na manutenção.
- Clareza na estrutura dos programas.

Imagem ilustrativa:



Vocabulário Técnico

- **Função:** bloco de código reutilizável que realiza uma tarefa.
- **Parâmetro:** valor passado à função para processamento.
- **Retorno (return):** valor que uma função entrega após execução.
- **Chamada de função:** momento em que se executa uma função.
- **Modularização:** prática de dividir o programa em partes menores e independentes.

Atividades Práticas

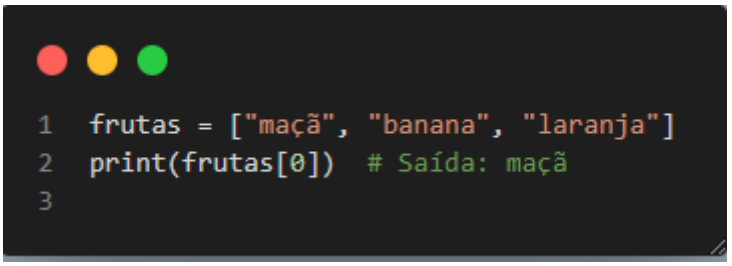
1. Crie uma função chamada boas_vindas que exiba uma mensagem de recepção.
2. Escreva uma função dobro que receba um número e retorne o dobro.
3. Implemente uma função que receba dois números e retorne a média.
4. Crie uma função que receba um nome e uma idade, e exiba uma frase personalizada.
5. Pesquise e explique com suas palavras o que significa “modularização” no contexto da programação.

6. LISTAS E TUPLAS

6.1 O que são Listas?

Listas são coleções de valores que podem ser modificadas. Os elementos são organizados em uma sequência e acessados por índices numéricos, iniciando em zero.

Exemplo de Lista:



```
1  frutas = ["maçã", "banana", "laranja"]
2  print(frutas[0])  # Saída: maçã
3
```

Nesse exemplo, o primeiro elemento da lista é acessado com o índice 0.

6.2 Operações com Listas

Tabela 3 - operações com listas

Operação	Exemplo	Resultado
Adicionar elemento	frutas.append("uva")	["maçã", "banana", "laranja", "uva"]
Remover elemento	frutas.remove("banana")	["maçã", "laranja", "uva"]
Acessar por índice	frutas[1]	laranja
Ver tamanho da lista	len(frutas)	3

6.3 O que são Tuplas?

Tuplas são semelhantes às listas, mas **não podem ser modificadas** após a sua criação (são imutáveis). Usam parênteses em vez de colchetes.

Exemplo de Tupla:

```
cores = ("azul", "verde", "vermelho")
print(cores[2])  # Saída: vermelho
```

6.4 Diferenças entre Lista e Tupla

Tabela 4 - diferença lista e tupla

Característica	Lista	Tupla
Símbolo	Colchetes [ ]	Parênteses ( )
Mutabilidade	Pode ser alterada	Não pode ser alterada
Aplicação comum	Dados variáveis	Dados fixos ou constantes

Vocabulário Técnico

- **Lista:** estrutura que armazena múltiplos valores e permite alterações.
- **Tupla:** sequência de dados imutável.
- **Índice:** posição de um elemento dentro da lista ou tupla.
- **Método:** função que opera diretamente sobre um objeto (exemplo: .append()).

Atividades Práticas

1. Crie uma lista chamada nomes com pelo menos três nomes e imprima o segundo.
2. Adicione um novo elemento à lista e remova um dos anteriores.
3. Crie uma tupla com três cores e imprima todas utilizando um laço for.
4. Faça um programa que receba uma lista de notas e calcule a média.
5. Explique em um parágrafo a diferença entre listas e tuplas e quando usar cada uma.

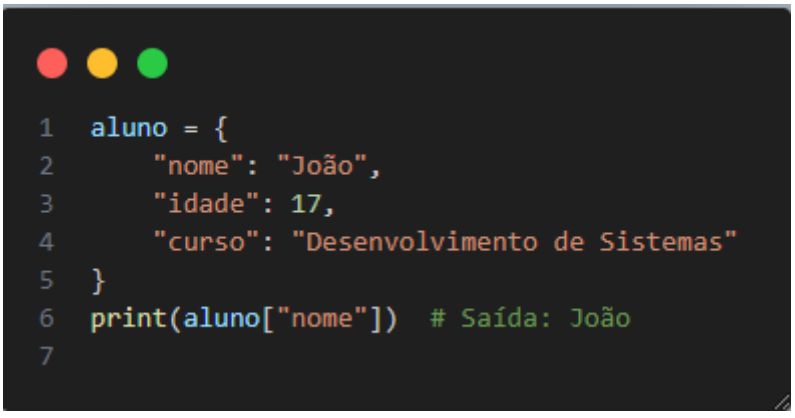
---

7. DICIONÁRIOS

7.1 O que são Dicionários em Python?

Dicionários são estruturas que armazenam **pares de chave e valor**, permitindo acesso direto aos dados por meio de identificadores, em vez de posições numéricas como nas listas.

Exemplo de Dicionário:



```
1  aluno = {
2      "nome": "João",
3      "idade": 17,
4      "curso": "Desenvolvimento de Sistemas"
5  }
6  print(aluno["nome"])  # Saída: João
7
```

Nesse exemplo, acessamos o valor relacionado à chave "nome".

7.2 Operações com Dicionários

Tabela 5 - operações com dicionarios

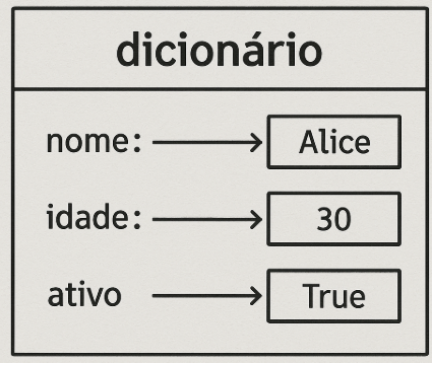
Operação	Exemplo	Resultado
Acessar valor	aluno["idade"]	17
Adicionar item	aluno["cidade"] = "Brasília"	Adiciona nova chave cidade
Alterar valor	aluno["idade"] = 18	Atualiza a idade para 18
Remover item	del aluno["curso"]	Remove a chave curso
Ver todas as chaves	aluno.keys()	dict_keys(['nome', 'idade', 'cidade'])
Ver todos os valores	aluno.values()	dict_values(['João', 18, 'Brasília'])

7.3 Dicionário com Laço for

Dicionários podem ser percorridos com um laço para exibir todos os pares:

```
for chave, valor in aluno.items():
    print(chave, ":", valor)
```

Imagem ilustrativa:



Vocabulário Técnico

- **Chave (key):** identificador exclusivo de um elemento.
- **Valor (value):** informação associada a uma chave.
- **Par chave-valor:** estrutura que compõe um item do dicionário.
- **Método .items():** permite acessar chave e valor simultaneamente.

Atividades Práticas

1. Crie um dicionário com informações de um aluno: nome, idade e curso.
2. Adicione a cidade e o ano de ingresso ao dicionário.
3. Altere o valor da idade para um número diferente.
4. Imprima todas as chaves e valores usando o laço for.
5. Remova uma das chaves do dicionário e explique o que ocorreu.

---

8. MANIPULAÇÃO DE ARQUIVOS

8.1 Introdução a Manipulação de Arquivos

Em muitas aplicações, é necessário ler dados salvos em arquivos ou gravar informações para consultas futuras. Python oferece recursos simples para realizar essas tarefas com arquivos de texto.

8.2 Abrindo Arquivos com a Função open()

A função open() permite acessar arquivos e definir o modo de operação:

- "r" – leitura
- "w" – escrita (substitui o conteúdo)
- "a" – acrescentar conteúdo
- "x" – cria um novo arquivo

Exemplo de leitura:

```python
arquivo = open("dados.txt", "r")
conteudo = arquivo.read()
print(conteudo)
arquivo.close()
```

8.3 Escrevendo em Arquivos

Para criar ou modificar um arquivo de texto:

```python
arquivo = open("saida.txt", "w")
arquivo.write("Curso de Python no SENAI")
arquivo.close()

```

8.4 Método with para Abertura Segura

O método with garante que o arquivo será fechado automaticamente após a operação, mesmo em caso de erros.

```python
with open("dados.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)

```

8.5 Lendo Linha por Linha

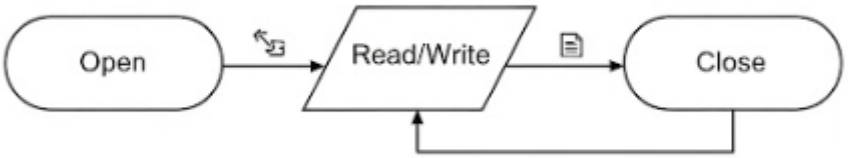
Podemos percorrer um arquivo linha por linha com um laço:

```python
with open("notas.txt", "r") as arquivo:
    for linha in arquivo:
        print(linha.strip())

```

O método .strip() remove espaços em branco e quebras de linha.

Imagem ilustrativa:



Vocabulário Técnico

- **Arquivo:** conjunto de dados armazenado em mídia digital.
- **Modo de abertura:** especifica se o arquivo será lido, escrito ou alterado.
- **.read() / .write():** métodos para leitura e escrita em arquivos.
- **.strip():** método que remove caracteres invisíveis das extremidades da linha.
- **Bloco with:** estrutura de segurança para manipulação de arquivos.

Atividades Práticas

1. Crie um arquivo chamado aluno.txt e escreva seu nome, idade e curso.
2. Leia o conteúdo do arquivo e imprima no terminal.
3. Adicione uma nova linha ao arquivo com a cidade onde mora, sem apagar o restante.
4. Crie um programa que leia cada linha de um arquivo de notas e exiba as médias dos alunos.
5. Escreva com suas palavras a importância do uso do bloco with na leitura e escrita de arquivos.

---

9. PROJETO INTEGRADOR

9.1 Objetivo

O Projeto Integrador é uma atividade prática que visa aplicar os conteúdos estudados de forma estruturada e funcional, simulando um cenário real do desenvolvimento de sistemas backend com Python.

9.2 Proposta do Projeto

Desenvolver uma aplicação simples em linha de comando para o gerenciamento de alunos de um curso técnico. A aplicação deve permitir:

- Cadastro de alunos com nome, idade e curso.
- Exibição da lista de alunos cadastrados.
- Pesquisa de aluno por nome.
- Cálculo da média de notas de um aluno.
- Armazenamento e leitura dos dados em arquivos .txt.

9.3 Requisitos Técnicos

- Uso de listas e dicionários para estruturação dos dados.
- Utilização de funções para organizar as ações.
- Implementação de controle de fluxo (condicionais e laços).
- Manipulação de arquivos para salvar e recuperar informações.

- Interface baseada em menu textual para interação com o usuário.

9.4 Estrutura Sugerida

```python
def menu():
    print("1 - Cadastrar aluno")
    print("2 - Listar alunos")
    print("3 - Buscar aluno")
    print("4 - Calcular média")
    print("5 - Sair")

def cadastrar_aluno():
    # implementação...

def listar_alunos():
    # implementação...

# Demais funções...

while True:
    menu()
    opcao = input("Escolha uma opção: ")

    if opcao == "1":
        cadastrar_aluno()
    elif opcao == "2":
        listar_alunos()
    # Demais condições...
    elif opcao == "5":
        break
```

9.5 Avaliação

Tabela - exemplo de estrutura

A aplicação será avaliada conforme os critérios abaixo:

Tabela 7 - critério avaliativo

Critério	Pontuação Máxima
Funcionalidade do programa	30 pontos
Organização do código (funções)	20 pontos
Apresentação e domínio do conteúdo	35 pontos
Manipulação de arquivos	5 pontos
Documentação e comentários	10 pontos

Vocabulário Técnico

- **Aplicação em linha de comando:** programa executado via terminal com interação textual.
- **Interface textual:** apresentação de opções para o usuário em formato de texto.
- **Persistência de dados:** prática de armazenar informações para acesso posterior.
- **Documentação:** explicações incluídas no código para facilitar a compreensão.

Orientações para o projeto

1. Planeje o funcionamento da sua aplicação em papel antes de começar a programar.
2. Implemente os módulos de cadastro e listagem de alunos.
3. Salve os dados em um arquivo e teste sua leitura.
4. Adicione funcionalidades de busca e cálculo de médias.
5. Apresente o projeto com explicações sobre sua estrutura e o funcionamento do programa.

10. REFERÊNCIAS

Abaixo estão listadas as obras, sites e materiais utilizados ou recomendados para consulta complementar. A formatação segue os padrões da ABNT para referências bibliográficas.

SILVA, José Carlos. *Introdução à Programação com Python*. 2. ed. São Paulo: Novatec, 2021.

LOPES, Mário. *Lógica de Programação: A construção de algoritmos e estruturas de dados*. 6. ed. Rio de Janeiro: Elsevier, 2020.

BEAZLEY, David. *Python: Guia Prático*. Porto Alegre: Bookman, 2022.

PYTHON SOFTWARE FOUNDATION. *Python Documentation*. Disponível em: https://docs.python.org/. Acesso em: 20 jul. 2025.

SENAI. *Currículo Técnico em Desenvolvimento de Sistemas*. Departamento Nacional, 2024.