





**UNIP**  
**UNIVERSIDADE PAULISTA**

# Roteiros

**Introdução à Programação Estruturada**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada <b>Título da Aula:</b> Ambiente de programação e saídas	<b>ROTEIRO 1</b>
---	--	------------------

## Roteiro da Aula Prática – Introdução à Programação Estruturada: O Comando `print()` em Python

Este roteiro é um material de apoio para a aula prática focada no comando essencial `print()` em Python. Exploraremos suas diversas funcionalidades para exibir informações na tela, formatar saídas e comunicar-se efetivamente com o usuário.

Público Alvo: Alunos com conhecimento básico de variáveis e tipos de dados em Python.

Duração: 1h40min (ajustável conforme o ritmo da turma)

### 1. Objetivos da Aula

- Compreender a função do comando `print()` para exibição de dados.
- Aprender a exibir cadeias de caracteres (strings) na tela.
- Entender como exibir o conteúdo de variáveis.
- Explorar o uso de múltiplos argumentos no `print()` e o parâmetro `sep`.
- Dominar a formatação de saída utilizando f-strings e o método `.format()`.
- Conhecer o parâmetro `end` para controlar o final da linha de saída.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão com a internet.
- Material de apoio (este roteiro e o notebook Google Colab correspondente, se disponível).
- Projetor para o professor.

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e objetivos: Apresentar os objetivos da aula, destacando a importância do comando `print()` como a principal ferramenta de comunicação do programa com o usuário.
- Discussão inicial: Perguntar aos alunos: "Como um programa nos informa o que está acontecendo ou o resultado de uma operação? Qual foi a primeira coisa que vocês aprenderam a fazer em programação para ver algo na tela?"
- Contextualização: Explicar que `print()` é fundamental para depuração, feedback ao usuário e construção de interfaces textuais simples.

#### 3.2. Revisão conceitual (20 minutos)

- O comando `print()`:
  - Exibição na tela de qualquer informação relevante ao usuário do programa.
  - É a maneira mais fácil para que o programa "se comunique" com quem o está utilizando.
- Exibindo cadeias de caracteres (Strings):
  - Strings podem ser delimitadas por aspas duplas ("" ) ou aspas simples (").
  - Exemplo:

```
print("Estamos estudando Introdução à Programação Estruturada")  
print('Estamos estudando Introdução à Programação Estruturada')
```

- Strings com aspas internas:
  - Para incluir aspas dentro de uma string, use o tipo oposto de aspas para delimitá-la.
  - Exemplo:

```
print("Estamos estudando 'Introdução à Programação Estruturada'")  
print('Estamos estudando "Introdução à Programação Estruturada"')
```

Exibindo o conteúdo de variáveis:

O `print()` pode exibir o valor de variáveis de diferentes tipos.

Exemplo:

```
a = 1
```

```
print(a)
```

```
b = 2.357
```

```
print(b)
```

- Múltiplos argumentos e o parâmetro `sep`:
- O `print()` pode receber múltiplos argumentos, que são separados por um espaço por padrão.
- O parâmetro `sep` permite definir um separador diferente.
- Exemplo:

```
a = 1
```

```
b = 2.357
```

```
print(a, b)
```

```
print(a, b, sep=" e ")
```

```
print(a, b, sep=",")
```

```
print(a, b, sep="-")
```

```
print(a, b, 11, 13, 17, sep=" e ")
```

- Combinando texto e variáveis:
- É possível mostrar mensagens de texto, conteúdos de objetos ou uma combinação das duas coisas.
- Exemplo:

```
a = 1
print("Valor de A=", a)
vcomida = "hamburger"
vOpinioao = "nota dez"
print("Este {prato} é {qualidade}".format(prato=vcomida, qualidade=vOpinioao))
```

- Formatação de saída com `.format()` e f-strings:
  - O método `.format()` permite inserir valores em placeholders (`{}`) e formatá-los.
  - f-strings (formatted string literals) são uma forma mais concisa e legível de formatar strings (disponível a partir do Python 3.6).
  - Exemplo:

```
a = 1
b = 2.357
# Usando .format() com índices numéricos
print("Valor de A={0} e B={1} certo?".format(a, b))
print("Valor de B={1} e A={0} certo?".format(a, b))
print("Valor de A={1} e B={0} certo?".format(b, a))
```

```
# Usando .format() com palavras-chave
print("Este {prato} é {qualidade}".format(prato="peixe", qualidade="ruim demais"))
```

```
# Formatação de números inteiros (d: decimal)
# :7d - reserva 7 dígitos para o número inteiro
print("Valor de A={0} e B={1} certo?".format(a, b))
print("Valor de A={0:7d} e B={1} certo?".format(a, b))
```

```
# Alinhamento de números inteiros
# :<7d - alinha à esquerda em 7 dígitos
# :^7d - centraliza em 7 dígitos
print("Valor de A={0:<7d} e B={1} certo?".format(a, b)) # esquerda
print("Valor de A={0:^7d} e B={1} certo?".format(a, b)) # centralizado
```

```
# Formatação de números de ponto flutuante (f: float)
# :.xf - x casas decimais
# :y.xf - y largura total, x casas decimais
print("Valor de A={0:7d} e B={1:f} certo?".format(a, b)) # Padrão: 6 casas decimais
print("Valor de A={0:7d} e B={1:.3f} certo?".format(a, b)) # 3 casas decimais
print("Valor de A={0:7d} e B={1:6.3f} certo?".format(a, b)) # 6 de largura total, 3 casas
decimais
```

```
# Demonstração de diferentes larguras e casas decimais para floats
print("Valor de A={0:7d} e B={1:1.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:2.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:3.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:4.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:5.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:6.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:7.3f} certo?".format(a, b))
print("Valor de A={0:7d} e B={1:8.3f} certo?".format(a, b))
```

```
# Compatibilidade com printf da linguagem C (usando %)  
print("Valor de A=%7d e B=%10.4f certo?" % (a, b))
```

```
# Usando f-strings (forma moderna e recomendada)  
print(f"Valor de A={a} e B={b} certo?")  
print(f"Valor de A={a:7d} e B={b:6.3f} certo?")
```

- O Parâmetro end:
  - Por padrão, o print() adiciona uma quebra de linha (\n) ao final de cada saída.
  - O parâmetro end permite especificar um caractere ou string diferente para o final da linha.
  - Exemplo:

```
print("Linha 1")  
print("Linha 2")
```

```
print("Linha 1 ", end="")  
print("Linha 2")
```

```
print("Primeira parte", end="...")  
print("Continuação")
```

### 3.3. Demonstração prática (30 minutos)

- Apresentação (utilizando o Google Colab):
  - O professor deve executar todas as células de código das seções 3.2 e 3.3, explicando a saída e os conceitos por trás de cada exemplo.
  - Incentivar os alunos a modificar os exemplos e observar as mudanças na saída.

## 4. Atividade prática (40 minutos)

### 4.1. Desafio inicial (a ser desenvolvido pelo aluno, guiado pelo professor)

Objetivo: Criar um programa simples que utilize o comando `print()` para exibir informações de um perfil de usuário.

Instruções:

1. Crie variáveis para armazenar seu nome, idade, cidade e profissão.
2. Utilize o `print()` para exibir cada uma dessas informações em linhas separadas.
3. Combine as informações em uma única frase, utilizando f-strings, por exemplo: "Olá, meu nome é [nome], tenho [idade] anos, moro em [cidade] e sou [profissão]."
4. Exiba a idade com 3 dígitos, preenchendo com zeros à esquerda se necessário (ex.: 025 para 25 anos).
5. Exiba um número de ponto flutuante (ex.: sua altura 1.75) formatado para ter apenas 2 casas decimais.
6. Tente exibir seu nome e sua cidade na mesma linha, separados por um traço (-).

### 4.2. Ampliação do desafio

Objetivo: Aprofundar o uso do `print()` e explorar a interação com o usuário.

Instruções:

1. Crie um pequeno "questionário" onde você pergunta ao usuário seu nome, idade e cor favorita (utilize a função `input()` para ler as respostas).
2. Após coletar as informações, utilize o `print()` para resumir as respostas do usuário em uma frase amigável, como: "Que legal, [Nome do Usuário]! Então você tem [Idade do Usuário] anos e adora a cor [Cor Favorita do Usuário]."
3. Adicione um toque de criatividade: use diferentes separadores (`sep`) e finais de linha (`end`) para que a saída seja mais interessante.
4. (Desafio extra) Pesquise sobre caracteres de escape (como `\n` para nova linha e `\t` para tabulação) e incorpore-os em suas mensagens `print()`.



## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para perguntas e discutir as principais dificuldades enfrentadas pelos alunos durante a prática.
  - "Qual formatação foi mais útil para vocês?"
  - "Vocês conseguiram usar os caracteres de escape? Qual o efeito deles?"
  - "Como o comando `print()` pode ajudar a encontrar erros em programas?"
- Orientação sobre o relatório final: Reforçar a importância de documentar o código e explicar os conceitos aprendidos.
  - Sugestão de próximos passos: Incentivar a exploração de outros tipos de entrada e saída em Python, ou aprofundar a formatação de strings para relatórios mais complexos.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Explicação sobre o comando `print()` em Python, suas funcionalidades básicas (exibição de strings e variáveis), e as diferentes formas de formatação de saída (parâmetros `sep`, `end`, método `.format()` e f-strings).
- Código-fonte comentado: Apresentação do código-fonte completo desenvolvido nas atividades práticas, com comentários claros explicando o funcionamento de cada trecho de código implementado no projeto.
- Resultados e discussão: Capturas de tela do programa em funcionamento e uma breve discussão sobre o que foi aprendido e as dificuldades superadas ao utilizar o comando `print()`.

## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre o comando <code>print()</code> e suas funcionalidades.
Estrutura e Organização do Código	3,0	Uso adequado de variáveis, formatação e comentários. Código limpo, legível e bem estruturado.
Funcionamento da Solução	3,0	O programa deve executar corretamente todas as funcionalidades propostas (exibição de dados, formatação, interação com o usuário).
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo (ex.: uso de caracteres de escape, mensagens mais elaboradas).

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá compreendido a importância do comando `print()` como a principal ferramenta de saída em Python, praticando a exibição de diferentes tipos de dados, a combinação de texto e variáveis, e a formatação de saída para tornar a comunicação com o usuário mais clara e eficaz. O comando `print()` é fundamental em todas as etapas do desenvolvimento de software, sendo usado para exibir resultados de cálculos, mensagens de erro, instruções para o usuário e para depurar o código, servindo como a base para qualquer interação textual com o programa. O domínio do `print()` é um pré-requisito essencial para todos os tópicos subsequentes da disciplina, pois será utilizado constantemente para verificar o funcionamento de algoritmos, exibir o estado de variáveis em estruturas de controle e funções, e para construir a interface de programas mais complexos, permitindo que o aluno progrida com confiança em seus estudos de programação.

**Bom estudo e boa prática!**



Instituto de Ciências  
Exatas e Tecnologia

**Disciplina:** Introdução à Programação Estruturada

**Título da Aula:** Variáveis tipo e operações

**ROTEIRO 2**

## Roteiro da Aula Prática – Introdução à Programação Estruturada

### 1. Objetivos da aula

- Compreender os conceitos de variáveis, tipos de dados e atribuição em Python.
- Aprender a interagir com o usuário utilizando a função `input()` para entrada de dados.
- Dominar a conversão de tipos de dados (type casting) para realizar operações adequadas.
- Praticar a exibição de informações na tela utilizando a função `print()` e diferentes métodos de formatação.
- Desenvolver a capacidade de traduzir algoritmos simples para código Python.
- Familiarizar-se com o ambiente de desenvolvimento integrado (IDE) do Google Colab.
- Compreender o funcionamento e a aplicação dos operadores de divisão inteira (`//`) e módulo (`%`) em Python.
- Aprender a extrair dígitos de um número inteiro (unidade, dezena, centena etc.) utilizando esses operadores.
- Desenvolver a lógica para converter números entre bases numéricas simples (ex.: decimal para binário, binário para decimal).

## 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) instalado e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Este notebook do Google Colab como material de apoio.

## 3. Estrutura da aula

### 3.1. Abertura (10 minutos)

- **Boas-vindas e explicação dos objetivos:** Iniciaremos a aula entendendo e aplicando os operadores de divisão inteira e módulo em Python para manipulação de números e conversão de bases. Também abordaremos os fundamentos da programação estruturada em Python, focando em variáveis, tipos de dados, entrada e saída, e como usar o Google Colab para praticar.
- **Discussão inicial:** Abriremos uma breve discussão sobre o que vocês já entendem por "programação" e qual a importância de conceitos como variáveis e tipos de dados em um programa de computador. Além disso, discutiremos como os números são representados e a importância de conseguir "desmontar" e "montar" números em diferentes contextos (ex.: sistemas de numeração, validação de dados).

### 3.2. Revisão conceitual (20 minutos)

- **Variáveis e atribuição:** Conceito de dado, variáveis (nome e tipo), e o sinal de atribuição (=).
- **Tipos de dados:** Revisão de int (inteiro), float (ponto flutuante), bool (booleano), complex (complexo) e str (string).

- **Entrada de dados (input()):** Como usar a função input() para receber dados do usuário, e a observação importante de que ela retorna uma string por padrão.
- **Conversão de tipos (Type Casting):** Necessidade e sintaxe para converter para int() e float().
- **Saída de dados (print()):** Diferentes formas de formatar a saída utilizando print(), incluindo f-strings, .format() e formatação antiga (%).
- **Operadores aritméticos básicos:** Soma, subtração, multiplicação.
- **Operadores de divisão em Python:** Divisão (/), divisão inteira (//) e operador módulo (%).
- **Aplicação: extraindo dígitos de um número:** Como usar % 10 e // 10 para decompor um número.

### 3.3. Demonstração prática (30 minutos)

- Apresentação (utilizando o Colab) da tradução de algoritmos simples para Python:
  - Cálculo de  $Y=3X+2$ .
  - Extração da dezena e unidade de um número de dois dígitos.

## 4. Atividade prática (40 minutos)

### 4.1. Desafio inicial

- **Tradução do "Algoritmo Consumo Médio":** Traduzir o pseudocódigo para um programa Python que calcule o consumo médio de um veículo.
- **Decompondo um Número de Três Dígitos:** Traduzir o pseudocódigo "ncdu" para um programa Python que leia um número de três dígitos e exiba separadamente sua unidade, dezena e centena.

## 4.2. Ampliação do desafio

- **Cálculo de IMC (Índice de Massa Corporal):** Desenvolver um programa em Python que calcule o IMC de uma pessoa, solicitando peso e altura, e exibindo o resultado formatado.
- **Conversão de decimal para binário (0 a 15):** Desenvolver um programa em Python que leia um número inteiro entre 0 e 15 e o converta para sua representação binária de 4 dígitos.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, abordando erros comuns e como superá-los.
- Revisão das soluções dos desafios, destacando diferentes abordagens e boas práticas de código (se o tempo permitir).
- Orientação sobre a entrega do relatório final, explicando detalhadamente a estrutura e os requisitos para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- **Resumo teórico:** Uma explicação clara e concisa sobre os conceitos de variáveis, tipos de dados (int, float, str, bool), entrada de dados (input()), saída de dados (print()) e conversão de tipos (type casting) em Python. Demonstre seu entendimento da teoria abordada na aula.
- **Código-fonte comentado:** O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Consumo Médio e Decomposição de Número) e a "Ampliação do desafio" (Cálculo de IMC e Conversão Decimal para Binário). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento e propósito.

## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (variáveis, tipos, input/output, type casting).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada, mensagens de erro personalizadas, ou formatação avançada da saída.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, os alunos terão adquirido os fundamentos essenciais da programação estruturada em Python, capacitando-os a manipular dados com variáveis e diversos tipos, interagir com o usuário para entrada e saída de informações, e aplicar esses conhecimentos na resolução de problemas computacionais básicos, além de se familiarizarem com o ambiente do Google Colab. Esses conceitos formam a base para futuras aplicações em automação de tarefas, análise de dados e desenvolvimento web simples, e são cruciais para a compreensão de tópicos mais avançados como estruturas de controle, funções e manipulação de arquivos, garantindo uma base sólida para a construção de programas robustos e interativos.

**Bom estudo e boa prática!**



**Instituto de Ciências  
Exatas e Tecnologia**

**Disciplina:** Introdução à Programação Estruturada

**Título da Aula:** Estruturas Condicionais

**ROTEIRO 3**

## **Roteiro da Aula Prática – Estruturas Condicionais em Python**

Este roteiro detalha uma aula prática focada nos fundamentos das estruturas condicionais (if, else, elif) em Python, utilizando o ambiente interativo do Google Colab. A aula abordará como tomar decisões em programas e resolver problemas que exigem diferentes caminhos de execução.

### **1. Objetivos da aula**

- Compreender o conceito e a importância das estruturas condicionais na programação.
- Aprender a utilizar os comandos if, if-else e if-elif-else em Python.
- Desenvolver a lógica para criar programas que tomam decisões com base em condições.
- Praticar a resolução de problemas que exigem diferentes fluxos de execução.

### **2. Recursos necessários**

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).



### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando claramente os objetivos: entender e aplicar as estruturas condicionais em Python para criar programas que reagem a diferentes situações.
- Discussão inicial: Abrir uma breve discussão sobre como tomamos decisões no dia a dia e como essa lógica pode ser traduzida para um programa de computador. Perguntar sobre exemplos de situações em que um programa precisaria "decidir" algo.
- Relembrando o Google Colab: Breve recordatório de como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais para a aula.

Estrutura condicional:

- Trata-se de uma estrutura responsável pelo desvio na sequência de execução das instruções em um programa.
- Executam o desvio da execução do programa em conformidade com o resultado lógico de uma expressão relacional ou lógica (CONDIÇÃO).
- Mecanismos de decisão em Python:
- if ...: Executa um bloco de código somente quando uma condição é verdadeira.
  - Exemplo: Programa para informar quando um número inteiro é par.

```
numero = int(input("Entre com um número: "))
```

```
resto = numero % 2
```

```
if resto == 0:
```

```
    print("O número é par.")
```

- if... else: Bifurca a execução do código em função de uma condição (verdadeira ou falsa).

- Exemplo: Programa para informar se um número é par ou ímpar.

```
numero = int(input("Entre com um número: "))
```

```
if numero % 2 == 0:
```

```
    print("O número é par.")
```

```
else:
```

```
    print("O número é ímpar.")
```

- if... elif... else: Permite verificar múltiplas condições em sequência. O bloco de código da primeira condição verdadeira será executado, e as demais serão ignoradas. Se nenhuma condição for verdadeira, o bloco else (se existir) será executado.

- Exemplo: Classificar um número como positivo, negativo ou zero.

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

```
if valor > 0:
```

```
    print("O valor é positivo.")
```

```
elif valor < 0:
```

```
    print("O valor é negativo.")
```

```
else:
```

```
    print("O valor é zero.")
```

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação das estruturas condicionais utilizando o Google Colab. Será feita a tradução de um algoritmo para ordenar três números em ordem decrescente.

Algoritmo "descresc" (Pseudocódigo):

algoritmo "descresc"

var

    n1,n2,n3:inteiro

inicio

    escreva("Entre com n1")

    leia(n1)

    escreva("Entre com n2")

    leia(n2)

    escreva("Entre com n3")

    leia(n3)

    se n1>n2 entao

        se n2>n3 entao

            escreva(n1,n2,n3)

        senao

            se n1>n3 entao

                escreva(n1,n3,n2)

            senao

                escreva(n3,n1,n2)

        fimse

    fimse

senao

    se n1>n3 entao

        escreva(n2,n1,n3)

    senao

        se n2>n3 entao

```
        escreva(n2,n3,n1)
    senao
        escreva(n3,n2,n1)
    fimse
fimse
fimse
finalgoritmo
```

Tradução para Python (utilizando elif):

```
n1 = int(input("Entre com n1: "))
n2 = int(input("Entre com n2: "))
n3 = int(input("Entre com n3: "))
```

```
if n1 >= n2 and n1 >= n3:
    if n2 >= n3:
        print(n1, n2, n3)
    else:
        print(n1, n3, n2)
elif n2 >= n1 and n2 >= n3:
    if n1 >= n3:
        print(n2, n1, n3)
    else:
        print(n2, n3, n1)
else:
    if n1 >= n2:
        print(n3, n1, n2)
    else:
        print(n3, n2, n1)
```

## 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos.

### 4.1 Desafio inicial

Verificação de Idade para CNH:

Faça um programa em Python que leia a data de hoje (dia, mês, ano) e a data de nascimento (dia, mês, ano) de uma pessoa. O programa deve verificar e informar se a pessoa já tem idade suficiente para tirar a Carteira Nacional de Habilitação (CNH), ou seja, 18 anos ou mais.

- Dica: Considere os anos, meses e dias para uma verificação precisa.

### 4.2 Ampliação do desafio

Classificação de Triângulos:

Escreva um programa que receba as três medidas dos lados de um triângulo. Primeiro, verifique se os valores realmente podem formar um triângulo (para ser um triângulo, cada lado tem que ser menor que a soma dos outros dois). Caso seja um triângulo, o programa deve classificar e escrever se ele é Equilátero, Isósceles ou Escaleno:

- Triângulo Equilátero: possui os 3 lados iguais.
- Triângulo Isósceles: possui 2 lados iguais.
- Triângulo Escaleno: possui 3 lados diferentes.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios. O professor deve abordar os conceitos de estruturas condicionais, aninhamento e como eles foram aplicados nos exercícios.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- Orientação sobre o relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- **Resumo teórico:** Uma explicação clara e concisa sobre as estruturas condicionais (if, else, elif) em Python, seu propósito e como elas controlam o fluxo de execução de um programa.
- **Código-fonte comentado:** O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Verificação de Idade para CNH) e a "Ampliação do desafio" (Classificação de Triângulos). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento e propósito.

## 7. Critérios de avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (estruturas condicionais).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, os alunos terão um entendimento sólido sobre as estruturas condicionais em Python e como aplicá-las para criar programas que tomam decisões, sendo capazes de implementar lógicas de ramificação para resolver problemas que exigem diferentes caminhos de execução, uma habilidade fundamental para a construção de programas mais complexos e interativos. As estruturas condicionais são a base para a criação de qualquer programa que precise de lógica de decisão, sendo usadas em sistemas de validação de dados, jogos (para determinar ações do jogador), interfaces de usuário (para responder a interações), e em algoritmos que precisam de diferentes comportamentos baseados em condições específicas. O domínio das estruturas condicionais é crucial para os próximos tópicos da disciplina, sendo um pré-requisito para a compreensão de laços de repetição (como `for` e `while`), funções mais avançadas, e para o desenvolvimento de algoritmos mais sofisticados que exigem controle de fluxo complexo.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada <b>Título da Aula:</b> Laços não contados	<b>ROTEIRO 4</b>
---	--	------------------

### Roteiro da Aula Prática – Laço while e Estruturas do-while

Este roteiro detalha uma aula prática focada no laço de repetição while e em como simular a estrutura do-while em Python, utilizando o ambiente interativo do Google Colab. A aula abordará como automatizar tarefas repetitivas baseadas em condições e resolver problemas que exigem que um bloco de código seja executado pelo menos uma vez.

#### 1. Objetivos da aula

- Compreender o conceito e a importância do laço de repetição while na programação.
- Aprender a utilizar o comando while para repetições baseadas em condições.
- Entender a diferença entre while e a estrutura do-while presente em outras linguagens.
- Aprender a simular a funcionalidade do do-while em Python.
- Desenvolver a lógica para criar programas que automatizam tarefas repetitivas e que precisam ser executadas pelo menos uma vez.
- Praticar a resolução de problemas que exigem iteração e controle de fluxo.

#### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).



### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando claramente os objetivos: aprender a usar o laço `while` e a simular a estrutura `do-while` em Python para automatizar tarefas.
- Discussão Inicial: Abrir uma breve discussão sobre situações do dia a dia ou programas em que uma ação precisa ser repetida enquanto uma condição for verdadeira (ex.: um jogo que continua enquanto o jogador tiver vidas, um sistema que pede uma senha até que seja correta).
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais para a aula.

Laços de Repetição (Loops):

- Permitem que um bloco de código seja executado repetidamente.
- Essenciais para automatizar tarefas e processar coleções de dados.

Comando `while`:

- Executa um bloco de código enquanto uma condição for verdadeira.
- A condição é verificada antes de cada iteração do laço. Se a condição for falsa desde o início, o bloco de código dentro do `while` nunca será executado.
- Sintaxe:

`while condicao_verdadeira:`

```
# Bloco de código a ser repetido
# É crucial que algo dentro do loop mude a condição
# para que ela eventualmente se torne falsa e o loop termine.
```

- Exemplo: Contar de 0 a 5.

```
contador = 0
```

```
while contador <= 5:
```

```
    print(contador)
```

```
    contador += 1 # Incrementa o contador para evitar loop infinito
```

Estrutura do-while (e sua simulação em Python):

- Em algumas linguagens de programação (como C++, Java), existe uma estrutura do-while que garante que o bloco de código seja executado pelo menos uma vez, antes que a condição seja verificada.
- Python não possui um do-while nativo. No entanto, podemos simular esse comportamento de duas maneiras principais:
  - Utilizando um while True com break:
  - O laço começa com uma condição sempre verdadeira (while True).
  - O bloco de código é executado.
  - Dentro do bloco, a condição real é verificada. Se for falsa, o comando break é usado para sair do laço.

```
while True:
```

```
    # Bloco de código que será executado pelo menos uma vez
```

```
    # ...
```

```
    if not condicao_para_continuar: # Se a condição for falsa, sai do loop
```

```
        break
```

- Inicializando a variável de controle antes do while:
- Damos um valor inicial à variável de controle que garante que a condição do while seja verdadeira na primeira vez.
- A condição é verificada no início do while como de costume.
- Essa abordagem é mais próxima do while tradicional, mas com uma inicialização estratégica.

```
variavel_controle = valor_inicial_que_garante_primeira_execucao
```

```
while condicao_verdadeira_baseada_em_variavel_controle:
```

```
    # Bloco de código
```

```
    # ...
```

```
    variavel_controle = novo_valor # Atualiza a variável para a próxima verificação
```

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação do laço while e a simulação do do-while utilizando o Google Colab.

- Apresentação (utilizando o Colab): O professor apresentará os seguintes exemplos:
  - Contagem Regressiva (Exemplo while simples):

```
# Contagem regressiva simples com while
```

```
segundos = 5
```

```
print("Iniciando contagem regressiva...")
```

```
while segundos > 0:
```

```
    print(segundos)
```

```
    segundos -= 1
```

```
print("Lançar!")
```

- Validação de Senha (Simulando do-while com while True e break):
  - # Simulação de do-while: Pedir senha até que seja correta
  - senha\_correta = "python123"
  - senha\_digitada = "" # Inicializa para garantir que o loop execute pelo menos uma vez

while True:

```
senha_digitada = input("Digite a senha: ")
if senha_digitada == senha_correta:
    print("Senha correta! Acesso concedido.")
    break # Sai do loop quando a senha está correta
else:
    print("Senha incorreta. Tente novamente.")
```

#### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

##### 4.1 Desafio inicial

Imprimindo os Cinco Primeiros Números Pares:

Faça um programa em Python que leia números do teclado e imprima apenas os cinco primeiros números pares digitados pelo usuário. O programa deve parar de solicitar números assim que cinco pares forem encontrados.

- Dica: Você precisará de um laço while e uma forma de contar quantos números pares já foram encontrados. Use o operador módulo (%) para verificar se um número é par.

## 4.2 Ampliação do desafio

Login com Tentativas Limitadas:

Modifique o exercício de validação de senha demonstrado pelo professor para que haja um limite de 3 tentativas antes de o programa se encerrar. Se o usuário exceder as tentativas, o programa deve exibir uma mensagem informando que o acesso foi bloqueado.

- Dica: Utilize um contador de tentativas dentro do laço `while` e verifique se o limite foi atingido.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente no que diz respeito ao controle de laço, condições de parada e a simulação do `do-while`. O professor deve abordar a importância do controle de laço para evitar loops infinitos.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- Orientação sobre o relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o laço de repetição `while` em Python, seu propósito, sintaxe e como ele controla o fluxo de execução de um programa. Inclua também a explicação sobre a estrutura `do-while` (presente em outras linguagens) e as formas de simulá-la em Python.
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Imprimindo os Cinco Primeiros Números Pares) e a "Ampliação do Desafio" (Login com Tentativas Limitadas). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento e propósito.

## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (laço while e simulação de do-while).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, os alunos terão um entendimento sobre o laço while em Python e como aplicá-lo para automatizar tarefas baseadas em condições. Eles também aprenderão a simular a estrutura do-while, garantindo que um bloco de código seja executado pelo menos uma vez. O laço while é amplamente utilizado em cenários em que o número de repetições é desconhecido e depende de uma condição, com exemplos que incluem validação de entrada de usuário (pedir até que seja válido), loops principais de jogos, leitura de dados de arquivos até o final e simulações. O domínio do laço while é importante para os próximos tópicos da disciplina, sendo um pré-requisito para a compreensão de algoritmos que exigem repetição condicional, como busca em estruturas de dados dinâmicas, e para o desenvolvimento de programas que interagem de forma contínua com o usuário ou com sistemas externos.

**Bom estudo e boa prática!**



Instituto de Ciências  
Exatas e Tecnologia

**Disciplina:** Introdução à Programação Estruturada

**Título da Aula:** Laços contados

**ROTEIRO 5**

### **Roteiro da Aula Prática – Laço for e Função range() em Python**

Este roteiro detalha uma aula prática focada no laço de repetição for e na função range() em Python, utilizando o ambiente interativo do Google Colab. A aula abordará como automatizar tarefas repetitivas e resolver problemas que exigem iteração sobre sequências.

#### **1. Objetivos da aula**

- Compreender o conceito e a importância do laço de repetição for na programação.
- Aprender a utilizar o comando for para iterar sobre sequências e ranges.
- Dominar o uso da função range() para gerar sequências numéricas.
- Desenvolver a lógica para criar programas que automatizam tarefas repetitivas.
- Praticar a resolução de problemas que exigem iteração e controle de fluxo.
- Reforçar o uso do Google Colab como ferramenta de desenvolvimento.

#### **2. Recursos necessários**

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando claramente os objetivos: aprender a usar o laço for e a função range() em Python para automatizar tarefas e como usar o Google Colab para praticar.
- Discussão inicial: Abrir uma breve discussão sobre situações do dia a dia ou em programas em que a mesma ação precisa ser repetida várias vezes (ex.: processar todos os itens de uma lista, executar algo um número fixo de vezes). Perguntar sobre a importância de não ter que escrever o mesmo código repetidamente.
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais para a aula.

Laços de Repetição (Loops):

- Permitem que um bloco de código seja executado repetidamente.
- Essenciais para automatizar tarefas e processar coleções de dados.
- Comando for:
- Utilizado para iterar sobre uma sequência (como uma lista, string ou um range).
- A cada iteração, uma variável recebe o próximo item da sequência.
- Sintaxe:

for item in sequencia:

    # Bloco de código a ser executado para cada item da sequência



- Exemplo: Iterar sobre uma lista de frutas.

```
frutas = ["maçã", "banana", "laranja"]
```

```
for fruta in frutas:
```

```
    print(fruta)
```

Função range():

- É uma função built-in do Python que gera uma sequência de números. É muito comum usá-la com o laço for para iterar um número específico de vezes.

- Sintaxe e usos:

- range(fim): Gera números inteiros de 0 até fim-1.

- Exemplo:

range(5) gera 0, 1, 2, 3, 4.

- range(inicio, fim): Gera números inteiros de inicio até fim-1.

- Exemplo:

range(2, 7) gera 2, 3, 4, 5, 6.

- range(inicio, fim, passo): Gera números inteiros de inicio até fim-1, pulando de passo em passo.

- Exemplo:

range(0, 10, 2) gera 0, 2, 4, 6, 8.

- Exemplo: Contar de 0 a 5 usando for e range().

```
for i in range(6): # Gera números de 0 a 5
```

```
    print(i)
```

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação do laço for e da função range() utilizando o Google Colab. Será feita a tradução de um algoritmo simples de contagem para Python.

Algoritmo "ZeroAcem" (Pseudocódigo adaptado para for):

algoritmo "ZeroAcem"

var

n:inteiro

inicio

para n de 0 ate 100 faca

escreva(n)

fimpara

fimalgoritmo

Tradução para Python (utilizando for):

```
for n in range(101): # range(101) gera números de 0 a 100
```

```
    print(n)
```

### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

#### 4.1 Desafio inicial

Soma dos N Primeiros Números Inteiros:

Faça um programa em Python que solicite ao usuário um número inteiro positivo N. Em seguida, o programa deve calcular e exibir a soma de todos os números inteiros de 1 até N (inclusive).

- Dica: Utilize o laço for e a função range() para iterar pelos números.

## 4.2 Ampliação do desafio

Verificação de Número Primo:

Faça um programa que receba um número natural maior que zero e verifique se ele é um número primo. Um número primo é divisível apenas por 1 e por ele mesmo.

- Por definição, o número 1 não é primo.
- Os primeiros números primos são: 2, 3, 5, 7, 11, 13, 17, 19, 23...
- Dica: Você pode usar um laço for para testar divisões do número por todos os inteiros de 2 até o número - 1. Se encontrar algum divisor, o número não é primo.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios. O professor deve abordar os conceitos do laço for e range(), e como eles foram aplicados nos exercícios, incluindo a importância de definir corretamente os limites do range().
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- Orientação sobre o relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o laço for e a função range() em Python, seu propósito, sintaxe e como eles controlam o fluxo de execução de um programa.
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Soma dos N Primeiros Números Inteiros) e a "Ampliação do desafio" (Verificação de Número Primo). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento e propósito.

## 7. Critérios de avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (laço for e função range()).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, os alunos terão um entendimento sólido sobre o laço for e a função range() em Python e como aplicá-los para automatizar tarefas. Eles serão capazes de implementar lógicas de iteração sobre sequências para resolver problemas que exigem repetição de código. O laço for é onipresente na programação, sendo usado em processamento de listas e outras coleções de dados, leitura de arquivos, jogos (para a lógica de iteração sobre elementos), simulações, e qualquer situação em que uma ação precisa ser repetida um número conhecido de vezes ou para cada item em uma sequência. O domínio do laço for e da função range() é importante para os próximos tópicos da disciplina, sendo um pré-requisito para a compreensão de estruturas de dados mais complexas (como listas e dicionários), algoritmos de busca e ordenação, e para o desenvolvimento de programas que interagem com grandes volumes de dados.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada  <b>Título da Aula:</b> Funções	<b>ROTEIRO 6</b>
---	---	------------------

### Roteiro da Aula Prática – Funções

Este roteiro detalha uma aula prática focada na introdução ao conceito de funções em Python, utilizando o ambiente interativo do Google Colab. A aula abordará a definição, uso e importância das funções para organizar e reutilizar código, com foco na aplicação para o cálculo de fatorial.

#### 1. Objetivos da aula

- Compreender o conceito de funções e sua importância na programação (modularidade e reusabilidade).
- Aprender a definir funções em Python, utilizando a palavra-chave `def`, parâmetros e o comando `return`.
- Desenvolver a lógica para implementar funções que resolvem problemas específicos, como o cálculo de fatorial.
- Praticar a chamada de funções e a passagem de argumentos.

#### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando que o foco será em “funções”, um conceito fundamental para escrever programas mais organizados e eficientes. Apresentar os objetivos da aula, destacando a importância de dividir problemas complexos em partes menores e reutilizáveis.
- Discussão inicial: Abrir uma breve discussão sobre a necessidade de evitar a repetição de código. Perguntar aos alunos se eles conseguem pensar em situações em que a mesma sequência de instruções é necessária em diferentes partes de um programa. Introduzir a ideia de que funções são a solução para isso.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais que serão a base para as atividades práticas.

O que são funções?

- Um bloco de código organizado e reutilizável que executa uma única ação relacionada.
- Permitem dividir um programa grande em partes menores e gerenciáveis.
- Aumentam a legibilidade do código e facilitam a manutenção.

Definição de funções em Python:

- Funções são definidas usando a palavra-chave `def`.
- Sintaxe básica:

```
def nome_da_funcao(parametro1, parametro2):  
    # Bloco de código da função  
    # Pode incluir operações, estruturas de controle, etc.  
    return resultado # Opcional: retorna um valor
```

- Parâmetros: Variáveis listadas dentro dos parênteses na definição da função. Eles atuam como entradas para a função.
- Comando return: Usado para enviar um valor de volta para o local onde a função foi chamada. Se return não for usado, a função retorna None por padrão.

Exemplo: Cálculo de Fatorial

- Definição de fatorial: Dado um número natural N maior ou igual a zero, o fatorial de N (escreve-se N!) é definido como:
  - Se  $N=0$ ,  $0!=1$  (por definição);
  - Se  $N>0$ ,  $N!=1 \times 2 \times 3 \times \dots \times (N-1) \times N$ .
  - Exemplo:

Para  $N=5$ , o fatorial é  $5!=1 \times 2 \times 3 \times 4 \times 5=120$ .

- A validação para garantir que o número digitado pelo usuário é positivo é importante.

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação de funções utilizando o Google Colab, implementando o cálculo de fatorial.

- Apresentação (utilizando o Colab): O professor apresentará a implementação de uma função para calcular o fatorial de um número.
  - Serão demonstradas duas versões da função  $\text{fat}(n)$ : uma utilizando o laço while e outra utilizando o laço for, conforme o notebook.
  - O professor executará os códigos no Colab, explicando cada parte da função: a definição, os parâmetros, o laço de repetição e o comando return.
  - Será enfatizada a validação para números positivos.

# Versão com while

```
def fat_while(n):  
    if n < 0:  
        return "Número deve ser positivo"  
    elif n == 0:  
        return 1  
    else:  
        f = 1  
        i = 1  
        while i <= n:  
            f *= i  
            i += 1  
        return f
```

# Versão com for

```
def fat_for(n):  
    if n < 0:  
        return "Número deve ser positivo"  
    elif n == 0:  
        return 1  
    else:  
        f = 1  
        for i in range(1, n + 1):  
            f *= i  
        return f
```



# Exemplos de uso

```
print(f"Fatorial de 5 (while): {fat_while(5)}")
print(f"Fatorial de 0 (while): {fat_while(0)}")
print(f"Fatorial de -3 (while): {fat_while(-3)}")
print(f"Fatorial de 8 (for): {fat_for(8)}")
```

#### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

##### 4.1 Desafio inicial

Cálculo de Fatorial com Validação:

- Descrição: Crie uma função em Python que calcule o fatorial de um número inteiro e positivo. A função deve receber um número como parâmetro e retornar o seu fatorial.
- Validação: Inclua uma validação dentro da função para garantir que o número digitado pelo usuário é positivo (maior ou igual a zero). Se o número for negativo, a função deve retornar uma mensagem de erro ou um valor indicativo de erro.
- Orientação: Os alunos devem adaptar a lógica demonstrada pelo professor, focando na estrutura da função e na validação da entrada.

##### 4.2 Ampliação do desafio

Cálculo de Combinação:

- Descrição: Faça um programa para calcular a combinação de dois números inteiros,  $n$  e  $k$ , utilizando a fórmula da combinação:  $C(n,k) = \frac{n!}{k!(n-k)!}$ .
- Reaproveitamento: Utilize a função de cálculo de fatorial desenvolvida no "Desafio inicial".
- Validação: Garanta que  $n \geq k$  e que  $n$  e  $k$  são números não negativos.

- Orientação: Os alunos deverão pensar em como chamar a função de fatorial múltiplas vezes e combinar os resultados para obter a combinação.

# Escreva seu código para a Ampliação do Desafio aqui:

## 5. Encerramento e orientações finais (20 minutos)

- Discussão sobre as principais dificuldades: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente no que diz respeito à definição de funções, passagem de parâmetros e o uso do return. O professor abordará a importância da validação de entrada e do reaproveitamento de código.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código, como a clareza dos nomes das funções e variáveis.
- Orientação sobre a entrega do relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o que são funções em Python, por que são importantes (modularidade, reusabilidade, abstração), como são definidas (sintaxe com def, parâmetros) e como o comando return funciona. Inclua a definição de fatorial e como funções podem ser usadas para calculá-lo.
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Cálculo de Fatorial) e a "Ampliação do desafio" (Cálculo de Combinação). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento, o propósito de cada linha ou bloco, e como as funções foram utilizadas e reutilizadas.

## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (funções, parâmetros, retorno, fatorial).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários explicativos, nomes de funções e variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados para diferentes entradas.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá um entendimento fundamental sobre o uso de funções em Python, sendo capaz de criar suas próprias funções para encapsular lógicas específicas, passar informações para elas através de parâmetros e obter resultados através do return. Funções são a espinha dorsal de quase todo programa complexo, sendo usadas em desenvolvimento web (para processar requisições), análise de dados (para aplicar transformações repetitivas), jogos (para a lógica de personagens ou interações), e qualquer aplicação que se beneficie da reutilização de código e da divisão de tarefas. O domínio de funções é um pré-requisito essencial para os próximos tópicos da disciplina, sendo fundamental para a compreensão de módulos e pacotes, programação orientada a objetos, e para a construção de algoritmos mais complexos que exigem a decomposição de problemas em subproblemas.

**Bom estudo e boa prática!**



Instituto de Ciências  
Exatas e Tecnologia

**Disciplina:** Introdução à Programação Estruturada

**Título da Aula:** Manipulação de Strings

**ROTEIRO 7**

## Roteiro da Aula Prática – Manipulação de Strings em Python

Este roteiro detalha uma aula prática focada na introdução e manipulação de strings em Python, utilizando o ambiente interativo do Google Colab. A aula abordará as características das strings, operações básicas e métodos úteis para trabalhar com texto.

### 1. Objetivos da aula

- Compreender o conceito de strings como sequências de caracteres e sua indexação.
- Aprender as operações básicas com strings: concatenação (+), multiplicação (\*) e obtenção do tamanho (len()).
- Familiarizar-se com métodos comuns de manipulação de strings, como replace(), count(), find(), split(), upper() e lower().
- Entender e aplicar as funções ord() e chr() para manipulação de caracteres ASCII.
- Desenvolver a lógica para resolver problemas que envolvem a manipulação de strings, como verificar palíndromos e realizar criptografia simples.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando que o foco será em "strings", um tipo de dado essencial para lidar com texto em programação. Apresentar os objetivos da aula, destacando a importância de manipular texto em diversas aplicações.
- Discussão inicial: Abrir uma breve discussão sobre onde os alunos veem a manipulação de texto no dia a dia (ex.: mensagens, nomes de usuário, senhas, busca em sites). Introduzir a ideia de que strings são a representação fundamental desses textos em um programa.
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais que serão a base para as atividades práticas.

O que são strings?

- Uma string é uma sequência imutável de caracteres.
- Pode ser considerada como uma "tabela" ou "vetor" de caracteres, onde cada caractere ocupa uma posição (índice).
- A indexação sempre se inicia em zero. A última posição terá como índice o tamanho da string menos 1.

Operações básicas com strings:

- Concatenação (+): Une duas ou mais strings.
  - Exemplo: "Olá" + " " + "Mundo" resulta em "Olá Mundo".
- Multiplicação (\*): Repete uma string um número N de vezes.
  - Exemplo: "Python" \* 3 resulta em "PythonPythonPython".
- Tamanho (len()): Retorna o número de caracteres em uma string.
  - Exemplo: len("Programação") retorna 11.

Métodos comuns de strings:

- `string.replace('antigo', 'novo')`: Substitui todas as ocorrências de uma substring por outra.
- `string.count('x')`: Indica quantas vezes um determinado caractere ou substring aparece.
- `string.find('x')`: Retorna o índice da primeira ocorrência de um caractere ou substring.
- `string.split(separador)`: Divide a string em uma lista de substrings, usando um separador (padrão é espaço).
- `string.upper()`: Converte todas as letras para maiúsculas.
- `string.lower()`: Converte todas as letras para minúsculas.

Métodos aplicáveis a caracteres (ASCII):

- `ord(caractere)`: Devolve o código numérico (ASCII) de um caractere.
  - Exemplo: `ord('A')` retorna 65.
- `chr(valor_ascii)`: Devolve o caractere correspondente a um valor inteiro ASCII.
  - Exemplo: `chr(97)` retorna 'a'.

Imutabilidade de strings:

- Um ponto importante: strings são imutáveis em Python. Isso significa que você não pode alterar um caractere específico em uma string existente. Para "modificar" uma string, você precisa criar uma nova string com as alterações desejadas, geralmente concatenando caracteres ou usando métodos que retornam uma nova string.

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a manipulação de strings utilizando o Google Colab, com exemplos práticos.

- Apresentação (utilizando o Colab): O professor apresentará os seguintes exemplos, explicando cada um:
  - Exibição caractere a caractere: Mostrar como iterar sobre uma string usando `for` e `while` para acessar cada caractere e sua posição.
  - Concatenação e multiplicação: Exemplos práticos das operações `+` e `*`.
  - Uso de métodos de string: Demonstração de `replace()`, `count()`, `split()`, `upper()` e `lower()` com diferentes strings.
  - Funções `ord()` e `chr()`: Explicar o conceito de tabela ASCII e como essas funções são usadas para converter entre caracteres e seus valores numéricos.
  - Criptografia de César (exemplo do material): Implementar e explicar o exemplo de criptografia de César, em que cada caractere avança um certo número de posições no alfabeto. Será enfatizado como lidar com a imutabilidade das strings, criando uma nova string resultado.

### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

## 4.1 Desafio inicial

Verificador de Palíndromos:

- Descrição: Crie um programa que receba uma palavra ou frase do usuário e verifique se ela é um palíndromo. Um palíndromo é uma palavra ou frase que pode ser lida da mesma forma de trás para frente, ignorando espaços e diferenças entre maiúsculas e minúsculas.
- Exemplos: "arara", "ovo", "Apos a sopa", "Roma me tem amor".
- Dicas:
  1. Converta a string para minúsculas.
  2. Remova todos os espaços.
  3. Compare a string original (tratada) com sua versão invertida.

## 4.2 Ampliação do desafio

Contador de Ocorrências de Palavras (Sem `count()`):

- Descrição: Dadas duas strings (uma contendo uma frase e outra contendo uma palavra), determine o número de vezes que a palavra ocorre na frase. NÃO USE O MÉTODO `.count()`
- Exemplo:
  - Frase: "ANA E MARIANA GOSTAM DE BANANA"
  - Palavra: "ANA"
  - Resultado esperado: 4
- Dicas:
  1. Converta ambos os strings para minúsculas para evitar problemas com maiúsculas/minúsculas.
  2. Use um laço de repetição para percorrer a frase.



3. Utilize fatiamento de strings ou métodos como `find()` para verificar a ocorrência da palavra.

4. # Escreva seu código para a Ampliação do Desafio aqui:

### **5. Encerramento e orientações finais (20 minutos)**

- Discussão sobre as principais dificuldades: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente no que diz respeito à manipulação de índices, uso dos métodos de string e a lógica para os desafios.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código, como a clareza dos nomes das variáveis e a eficiência da lógica.
- Orientação sobre a entrega do relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

### **6. Orientações para o relatório final**

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o que são strings em Python, suas características (imutabilidade, indexação), as operações básicas (`+`, `*`, `len()`) e os principais métodos de manipulação (`replace()`, `count()`, `find()`, `split()`, `upper()`, `lower()`). Inclua a explicação de `ord()` e `chr()` e a importância da tabela ASCII.
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Verificador de Palíndromos) e a "Ampliação do desafio" (Contador de Ocorrências). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento, o propósito de cada linha ou bloco, e como os conceitos e métodos de string foram aplicados.

## 7. Critérios de avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (strings, operações, métodos, ASCII).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários explicativos, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados para diferentes entradas.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, você terá um conhecimento sobre a manipulação de strings em Python. Será capaz de realizar operações básicas, utilizar métodos para transformar e analisar texto, e aplicar funções de conversão ASCII. Essa habilidade é importante para processar e interagir com dados textuais em diversas aplicações. A manipulação de strings é onipresente na programação, sendo utilizada em processamento de texto (análise de sentimentos, extração de informações, formatação de documentos), validação de entrada (verificação de senhas, e-mails, formatos de dados), desenvolvimento web (criação de URLs amigáveis, processamento de formulários), criptografia (implementação de algoritmos de segurança simples) e jogos (criação de diálogos, nomes de personagens). O domínio de strings é um pilar para tópicos mais avançados, como expressões regulares, manipulação de arquivos de texto, e desenvolvimento de aplicações que interagem com usuários através de interfaces textuais.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada  <b>Título da Aula:</b> Coleções	<b>ROTEIRO 8</b>
---	--	------------------

## Roteiro da Aula Prática – Coleções em Python

Este roteiro detalha uma aula prática focada na introdução e manipulação de coleções em Python, com ênfase em listas e tuplas.

### 1. Objetivos da aula

- Compreender o conceito de sequências e coleções em Python.
- Distinguir entre sequências mutáveis (listas) e imutáveis (tuplas, strings, ranges).
- Dominar a criação e manipulação de listas em Python.
- Aprender a acessar elementos de listas e tuplas por índice.
- Realizar operações básicas com listas: adicionar, remover, modificar e buscar elementos.
- Entender o conceito de fatiamento (slicing) para extrair subconjuntos de listas e tuplas.
- Desenvolver a lógica para resolver problemas que envolvem a manipulação de coleções.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando que o foco será em "coleções" ou "sequências" em Python, com destaque para as "listas" e a introdução às "tuplas". Apresentar os objetivos da aula, enfatizando a importância de organizar e manipular múltiplos dados em um único lugar.
- Discussão inicial: Abrir uma breve discussão sobre onde os alunos veem a necessidade de agrupar informações no dia a dia (ex.: lista de compras, contatos telefônicos, notas de alunos, inventário de um jogo). Introduzir a ideia de que listas e tuplas são ferramentas fundamentais para isso em Python.
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais que serão a base para as atividades práticas.

O que são sequências?

- Sequências são tipos de dados que contêm outros dados, organizados em uma ordem específica.
- Em Python, os principais tipos de sequência são:
  - list (lista): Coleção ordenada e mutável de itens.
  - tuple (tupla): Coleção ordenada e imutável de itens.
  - range (objeto de intervalo): Sequência imutável de números.
  - string (cadeia de caracteres): Sequência imutável de caracteres (já vimos na aula anterior).

## Listas em Python:

- São a estrutura de dados mais versátil para armazenar coleções de itens.
- Podem conter elementos de qualquer tipo de dado, inclusive misturados (listas heterogêneas).
- Sintaxe: Listas são delimitadas por colchetes `[]` e seus elementos são separados por vírgulas.
  - Exemplo: `lista1 = [1, 2, 3, 4]`
  - Exemplo heterogêneo: `lista_mista = [1, 'python', 3.5, True]`

## Tuplas em Python:

- São coleções ordenadas e imutáveis de itens.
- Uma vez criada, uma tupla não pode ser modificada (não é possível adicionar, remover ou alterar elementos).
- Sintaxe: Tuplas são delimitadas por parênteses `()` e seus elementos são separados por vírgulas.
  - Exemplo: `tupla1 = (1, 2, 3)`
  - Exemplo heterogêneo: `tupla_mista = ('nome', 25, 1.75)`
- Quando usar tuplas? Para coleções de dados que não devem mudar, como coordenadas geográficas, dados de registro imutáveis, ou como chaves em dicionários (já que listas não podem ser chaves por serem mutáveis).

Acesso a elementos (indexação):

- Tanto listas quanto tuplas têm seus elementos identificados por um índice, que começa em 0.
- Para acessar um elemento, usamos o operador de colchetes [] com o índice.
- Exemplo:

```
minha_lista = ['a', 'b', 'c'], minha_tupla = (10, 20, 30)
```

```
minha_lista[0] retorna 'a'
```

```
minha_tupla[1] retorna 20
```

- Índices negativos: Permitem acessar elementos a partir do final da sequência.

```
minha_lista[-1] retorna o último elemento.
```

```
minha_tupla[-2] retorna o penúltimo elemento.
```

Fatiamento (Slicing):

- Permite extrair uma parte (sub-lista ou sub-tupla) de uma sequência.
- Sintaxe: `sequencia[inicio:fim:passo]`
  - `inicio`: Índice de onde o fatiamento começa (inclusive). Padrão é 0.
  - `fim`: Índice de onde o fatiamento termina (exclusive). Padrão é o final da sequência.
  - `passo`: Intervalo entre os elementos selecionados. Padrão é 1.
- Exemplo:

```
numeros = [0, 1, 2, 3, 4, 5]
```

```
numeros[1:4] retorna [1, 2, 3]
```

```
numeros[::-1] retorna [5, 4, 3, 2, 1, 0] (lista invertida)
```

- O fatiamento em tuplas funciona da mesma forma, mas o resultado será uma nova tupla.

## Operações comuns com listas:

- Adicionar elementos:
  - `lista.append(item)`: Adiciona um item ao final da lista.
  - `lista.insert(indice, item)`: Insere um item em uma posição específica.
- Remover elementos:
  - `lista.remove(item)`: Remove a primeira ocorrência de um item pelo seu valor.
  - `lista.pop(indice)`: Remove e retorna o item de uma posição específica (se o índice não for fornecido, remove o último).
  - `del lista[indice]`: Remove um item pelo seu índice.
- Modificar elementos:
  - `lista[indice] = novo_valor`: Altera o valor de um elemento em uma posição específica.
- Buscar elementos:
  - `item in lista`: Retorna True se o item estiver na lista, False caso contrário.
  - `lista.index(item)`: Retorna o índice da primeira ocorrência de um item. Gera erro se o item não for encontrado.
- Outros métodos úteis:
  - `lista.sort()`: Ordena a lista (modifica a lista original).
  - `sorted(lista)`: Retorna uma nova lista ordenada (não modifica a original).
  - `lista.reverse()`: Inverte a ordem dos elementos na lista (modifica a lista original).
  - `lista.count(item)`: Retorna o número de vezes que um item aparece na lista.
  - `lista.clear()`: Remove todos os elementos da lista.

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a manipulação de listas e tuplas utilizando o Google Colab, com exemplos práticos.

- Apresentação (utilizando o Colab): O professor apresentará os seguintes exemplos, explicando cada um:
  - Criação e acesso de listas e tuplas: Demonstrar a criação de ambas as coleções, com diferentes tipos de dados, e o acesso a elementos usando índices positivos e negativos.
  - Fatiamento: Exemplos detalhados de fatiamento para listas e tuplas.
  - Operações de listas: Mostrar o uso de `append()`, `insert()`, `remove()`, `pop()`, `del`, modificação por índice, `in`, `index()`, `sort()`, `sorted()` e `reverse()`.
  - Imutabilidade da tupla: Tentar modificar um elemento de uma tupla para demonstrar o erro e explicar o conceito de imutabilidade.
  - Iteração: Como percorrer listas e tuplas usando laços `for`.

# Demonstração Prática de Coleções (Listas e Tuplas) em Python

```
print("--- Criação e Acesso de Listas ---")
# Lista de números inteiros
numeros_lista = [10, 20, 30, 40, 50]
print(f"Lista de números: {numeros_lista}")
print(f"Primeiro elemento da lista: {numeros_lista[0]}")
print(f"Último elemento da lista (índice negativo): {numeros_lista[-1]}")
```



```
# Lista heterogênea
```

```
dados_lista_variados = ["Python", 3.14, True, 100, "Colab"]
```

```
print(f"\nLista heterogênea: {dados_lista_variados}")
```

```
print(f"Elemento na posição 1 da lista: {dados_lista_variados[1]}")
```

```
print("\n--- Criação e Acesso de Tuplas ---")
```

```
# Tupla de números inteiros
```

```
numeros_tupla = (100, 200, 300, 400, 500)
```

```
print(f"Tupla de números: {numeros_tupla}")
```

```
print(f"Primeiro elemento da tupla: {numeros_tupla[0]}")
```

```
print(f"Último elemento da tupla (índice negativo): {numeros_tupla[-1]}")
```

```
# Tupla heterogênea
```

```
dados_tupla_variados = ("Java", 2.71, False, 50, "IDE")
```

```
print(f"\nTupla heterogênea: {dados_tupla_variados}")
```

```
print(f"Elemento na posição 1 da tupla: {dados_tupla_variados[1]}")
```

```
print("\n--- Demonstração de Imutabilidade da Tupla ---")
```

```
# Tentando modificar um elemento de uma tupla (isso causará um erro)
```

```
try:
```

```
    # numeros_tupla[0] = 999 # Descomente para ver o erro
```

```
    print("Tentativa de modificar tupla (comentado para evitar erro).")
```

```
except TypeError as e:
```

```
    print(f"Erro ao tentar modificar tupla: {e}")
```

```
print("Tuplas são imutáveis: seus elementos não podem ser alterados após a criação.")
```

```
print("\n--- Fatiamento (Slicing) em Listas e Tuplas ---")
lista_original = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
tupla_original = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
print(f"Lista original para fatiamento: {lista_original}")
print(f"Tupla original para fatiamento: {tupla_original}")
```

```
print(f"Fatiamento da lista (índice 2 ao 6): {lista_original[2:7]}")
print(f"Fatiamento da tupla (índice 2 ao 6): {tupla_original[2:7]}")
print(f"Lista invertida: {lista_original[::-1]}")
print(f"Tupla invertida: {tupla_original[::-1]}")
```

```
print("\n--- Operações de Adicionar/Remover/Modificar em Listas ---")
frutas = ["maçã", "banana"]
print(f"Lista inicial de frutas: {frutas}")
frutas.append("laranja") # Adiciona ao final
print(f"Após append('laranja'): {frutas}")
frutas.insert(1, "uva") # Insere na posição 1
print(f"Após insert(1, 'uva'): {frutas}")
```

```
cores = ["vermelho", "azul", "verde", "amarelo", "azul"]
print(f"\nLista inicial de cores: {cores}")
cores.remove("azul") # Remove a primeira ocorrência
print(f"Após remove('azul'): {cores}")
elemento_removido_pop = cores.pop(2) # Remove e retorna o elemento na posição 2
print(f"Após pop(2): {cores}, Elemento removido: {elemento_removido_pop}")
del cores[0] # Remove o elemento na posição 0
print(f"Após del cores[0]: {cores}")
```

```
precos = [15.50, 20.00, 12.75]
print(f"\nLista inicial de preços: {precos}")
precos[1] = 22.50 # Modifica o elemento na posição 1
print(f"Após modificação: {precos}")
```

```
print("\n--- Buscar Elementos ---")
nomes = ["Alice", "Bob", "Charlie", "David"]
print(f"Lista de nomes: {nomes}")
print(f"'Bob' está na lista? {'Bob' in nomes}")
print(f"'Eve' está na lista? {'Eve' in nomes}")
```

```
try:
```

```
    print(f"Índice de 'Charlie': {nomes.index('Charlie')}")
```

```
    # print(f"Índice de 'Eve': {nomes.index('Eve')}") # Isso causará um erro, descomente
```

para testar

```
except ValueError as e:
```

```
    print(f"Erro ao buscar 'Eve': {e}")
```

```
print("\n--- Ordenação e Inversão em Listas ---")
numeros_desordenados = [5, 2, 8, 1, 9, 3]
print(f"Lista desordenada: {numeros_desordenados}")
numeros_desordenados.sort() # Ordena a lista in-place
print(f"Após sort(): {numeros_desordenados}")
```

```
letras = ['c', 'a', 'b']
print(f"Lista de letras: {letras}")
letras_ordenadas = sorted(letras) # Retorna uma nova lista ordenada
print(f"Lista original de letras (não modificada por sorted()): {letras}")
print(f"Nova lista ordenada (sorted()): {letras_ordenadas}")
```

```
minha_lista_inverter = [1, 2, 3, 4, 5]
print(f"Lista para inverter: {minha_lista_inverter}")
minha_lista_inverter.reverse() # Inverte a lista in-place
print(f"Após reverse(): {minha_lista_inverter}")
```

```
print("\n--- Iteração com Listas e Tuplas ---")
itens_lista = ["item A", "item B", "item C"]
itens_tupla = ("dado X", "dado Y", "dado Z")
```

```
print("Percorrendo a lista com 'for':")
for item in itens_lista:
    print(item)
```

```
print("\nPercorrendo a tupla com 'for':")
for dado in itens_tupla:
    print(dado)
```

```
print("\nPercorrendo a lista com 'for' e índice:")
for i in range(len(itens_lista)):
    print(f"Lista - Posição {i}: {itens_lista[i]}")
```

```
print("\nPercorrendo a tupla com 'for' e índice:")
for i in range(len(itens_tupla)):
    print(f"Tupla - Posição {i}: {itens_tupla[i]}")
```

## 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

### 4.1 Desafio inicial

Gerenciador de tarefas simples:

- Descrição: Crie um programa que permita ao usuário gerenciar uma lista de tarefas.

O programa deve ser capaz de:

1. Adicionar uma nova tarefa.
2. Visualizar todas as tarefas.
3. Marcar uma tarefa como concluída (removê-la da lista).
4. Sair do programa.

- Requisitos:

- Use uma lista para armazenar as tarefas.
- Implemente um menu interativo para o usuário escolher as opções.
- Considere a entrada do usuário para o número da tarefa a ser removida.

### 4.2 Ampliação do desafio

Análise de dados de vendas:

- Descrição: Você tem uma lista de vendas diárias. Crie um programa que:

1. Calcule a venda total do período.
2. Encontre a venda máxima e a venda mínima.
3. Calcule a média das vendas.
4. Permita adicionar novas vendas à lista.

- Requisitos:

- Use uma lista de números (inteiros ou floats) para representar as vendas.
- Utilize funções built-in do Python como `sum()`, `max()`, `min()` para facilitar os cálculos.
- Inclua um menu interativo similar ao desafio anterior.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão sobre as principais dificuldades: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente no que diz respeito à escolha da estrutura de dados, uso dos métodos de lista e a lógica para os desafios.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código, como a clareza dos nomes das variáveis e a eficiência da lógica.
- Orientação sobre a entrega do relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o que são sequências em Python, com foco em listas e tuplas. Inclua suas características (mutabilidade/imutabilidade, indexação, fatiamento) e os principais métodos de manipulação de listas (adicionar, remover, modificar, buscar, ordenar).
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Gerenciador de Tarefas) e a "Ampliação do desafio" (Análise de Dados de Vendas). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento, o propósito de cada linha ou bloco, e como os conceitos e métodos de listas foram aplicados.

## 7. Critérios de avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (sequências, listas, tuplas, operações, métodos).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários explicativos, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados para diferentes entradas.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá um conhecimento sobre o uso de listas e tuplas em Python, sendo capaz de criar, manipular (listas) e iterar sobre essas coleções, aplicando os métodos mais comuns para resolver problemas práticos, uma habilidade fundamental para organizar e processar conjuntos de dados em diversas aplicações. O uso de listas e tuplas é essencial em quase todas as áreas da programação, incluindo armazenamento de dados (listas de usuários, produtos, registros de banco de dados, e tuplas para registros imutáveis como coordenadas), jogos (inventários de itens, posições de inimigos, placares), análise de dados (coleta e processamento de séries de números como vendas e temperaturas), desenvolvimento web (listas de itens em carrinhos de compra, menus de navegação) e algoritmos (implementação de pilhas, filas, grafos e outras estruturas de dados). O domínio das listas e tuplas é um passo crucial para a compreensão de estruturas de dados mais complexas e algoritmos, permitindo que os alunos comecem a pensar em como organizar e processar grandes volumes de informações, abrindo caminho para tópicos como manipulação de arquivos, bancos de dados e desenvolvimento de aplicações mais eficientes.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada  <b>Título da Aula:</b> Dicionário	<b>ROTEIRO 9</b>
---	--	------------------

## Roteiro da Aula Prática – Introdução à Programação Estruturada com Dicionários em Python

### 1. Objetivos da aula

- Compreender o conceito de dicionários em Python como uma estrutura de dados chave-valor.
- Aprender a criar, acessar e manipular elementos em dicionários.
- Explorar os principais métodos de dicionários (`keys()`, `values()`, `items()`, `update()`, `pop()`, `clear()`, `setdefault()`, etc.).
- Desenvolver um sistema de agenda simples utilizando dicionários para armazenar e gerenciar dados.
- Praticar a escrita de funções em Python para organizar o código.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Projetor para o professor.



### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e objetivos: Apresentar os objetivos da aula, destacando a importância dos dicionários na programação para organizar dados de forma eficiente, similar a registros em bancos de dados.
- Discussão inicial: Perguntar aos alunos: "Onde vocês acham que dados como endereços e telefones são armazenados em sistemas reais? Como podemos organizar informações que não são apenas uma sequência, mas que precisam de um 'nome' para serem encontradas?"

#### 3.2. Revisão conceitual (20 minutos)

- O que são dicionários?
  - Revisar o conceito de dicionário em Python: uma coleção de elementos ordenada (a partir do Python 3.7) e mutável, que armazena pares de "chave" e "valor".
  - Diferença de listas e tuplas: Dicionários usam chaves (que podem ser de qualquer tipo imutável, como strings ou números) em vez de índices numéricos.
  - Explicar que chaves são únicas e valores podem ser duplicados.
  - Exemplo: Apresentar a estrutura básica de um dicionário: `carro = {"marca": "Ford", "modelo": "Ka", "ano": 2005, "cor": "preto"}`.
- Acessando elementos:
  - Demonstrar como acessar valores usando suas chaves: `print(carro["modelo"])`.
  - Mostrar `len()` para obter o tamanho do dicionário e `type()` para verificar o tipo.
- Iterando em dicionários:
  - Explicar como percorrer um dicionário usando `for` para chaves, `carro.keys()` para chaves, `carro.values()` para valores e `carro.items()` para pares chave-valor.

### 3.3. Demonstração prática (30 minutos)

- Apresentação (utilizando o Google Colab):
  - Abrir o notebook IPELab0Dicionarios\_24final.ipynb no Google Colab.
  - Executar os exemplos de código do notebook, explicando cada um:
    - Criação e acesso a dicionários (carro).
    - Uso de `keys()`, `values()`, `items()`.
    - Iteração com `for` loops.
  - Principais Métodos de Manipulação:
    - Demonstrar `clear()`, `copy()`, `fromkeys()`, `get()`, `pop()`, `popitem()`, `setdefault()`, `update()`.
    - Explicar a utilidade de cada método com exemplos práticos.
    - Enfatizar a diferença entre `copy()` e atribuição direta para evitar referências indesejadas.

## 4. Atividade prática (40 minutos)

### 4.1. Desafio inicial (A ser desenvolvido pelo aluno, guiado pelo professor)

- Sistema de Agenda Simples:
  - Passo 1: Criar um registro de dados.
    - Crie um dicionário vazio chamado `dados`.
    - Insira os registros "Endereço" (ex: "Rua Aqui, 0") e "Telefone" (ex: 11996660000) neste dicionário.
    - Mostre o registro `dados`.
  - Passo 2: Ler endereço e telefone do usuário.
    - Faça um programa que solicite ao usuário o endereço e o telefone.
    - Armazene essas informações no dicionário `dados`.

- Passo 3: Encapsular a leitura em uma função.
  - Transforme o código de leitura de endereço e telefone em uma função chamada `ledados()` que retorne o dicionário dados lido.
- Passo 4: Criar a agenda principal.
  - Crie um dicionário chamado agenda e insira um registro com a chave "Paulo" e o valor sendo o dicionário dados lido pela função `ledados()`.
  - Mostre o dicionário agenda.
- Passo 5: Acessar informações específicas.
  - Mostre o número de telefone do "Paulo" a partir do dicionário agenda.
- Passo 6: Função para mostrar item da agenda.
  - Crie uma função `mostrarItem(chave)` que receba uma chave (nome) e mostre a chave e seus dados (endereço e telefone) se a chave existir na agenda. Se não existir, a função deve mostrar "Não Localizado".
  - Teste a função com "Paulo" e com um nome que não existe (ex.: "Maria").

## 4.2. Ampliação do desafio

- Adicionar Novo Contato: Crie uma função `criarItem()` que solicite ao usuário uma nova chave (nome) e os dados (endereço e telefone) para adicionar um novo contato à agenda.
- Remover Contato: Crie uma função `removerItem(chave)` que remova um contato da agenda a partir de uma chave fornecida.
- Menu de Interação: Implemente um menu simples (usando while loop) que permita ao usuário escolher entre as opções: Criar agenda, Mostrar item, Criar item, Remover item, Mostrar todos os itens, Sair.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para perguntas e discutir as principais dificuldades enfrentadas pelos alunos durante a prática.
  - "Quais foram os maiores desafios ao trabalhar com chaves e valores?"
  - "Como a organização em funções ajudou a manter o código limpo?"
  - "Quais outras aplicações vocês imaginam para dicionários?"
- Orientação sobre o relatório final: Reforçar a importância de documentar o código e explicar os conceitos aprendidos.
- Sugestão de próximos passos: Incentivar a exploração de outros tipos de dados estruturados em Python (conjuntos) ou aprofundar o uso de dicionários em cenários mais complexos (ex.: dicionários aninhados).

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Explicação sobre o conceito de dicionários em Python, suas características (mutáveis, ordenados, chave-valor) e sua importância na programação estruturada para organização de dados.
- Código-fonte comentado: Apresentação do código-fonte completo do sistema de agenda desenvolvido, com comentários claros explicando o funcionamento de cada função e trecho de código implementado no projeto.
- Resultados e discussão: Capturas de tela do programa em funcionamento (pelo menos uma para cada funcionalidade principal: adicionar, mostrar, remover), e uma breve discussão sobre o que foi aprendido e as dificuldades superadas.

## 7. Critérios de Avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre o conceito de dicionários e sua aplicação.
Estrutura e Organização do Código	3,0	Uso adequado de funções, variáveis e comentários. Código limpo, legível e bem estruturado.
Funcionamento da Solução	3,0	O sistema de agenda deve executar corretamente todas as funcionalidades propostas (adicionar, mostrar, remover).
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo (ex.: validação de entrada, persistência de dados simples, interface mais amigável).

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá compreendido a importância e a versatilidade dos dicionários em Python como uma ferramenta poderosa para organizar e manipular dados de forma eficiente, tendo praticado a criação de estruturas de dados complexas e o desenvolvimento de funções para gerenciar essas estruturas, consolidando conceitos de programação estruturada. Dicionários são a espinha dorsal de muitas aplicações modernas, sendo usados para representar registros de usuários, configurações de sistemas, dados de APIs (como JSON), informações de produtos em e-commerce, e muito mais; a habilidade de trabalhar com dicionários é crucial para interagir com bancos de dados e serviços web. O domínio dos dicionários é um pré-requisito fundamental para tópicos mais avançados da disciplina, como manipulação de arquivos (CSV, JSON), integração com bancos de dados, desenvolvimento web (onde dados são frequentemente transmitidos em formato de dicionário/JSON) e até mesmo conceitos de programação orientada a objetos, onde dicionários podem ser usados para representar atributos de objetos ou configurações. Uma base sólida neste conceito permitirá que o aluno construa programas cada vez mais robustos e eficientes.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada <b>Título da Aula:</b> Arquivo sequencial	<b>ROTEIRO 10</b>
---	--	-------------------

## Roteiro da Aula Prática – Manipulação de Arquivos .txt em Python

Este roteiro detalha uma aula prática focada na leitura e escrita de arquivos de texto (.txt) em Python, utilizando o ambiente interativo do Google Colab. A aula abordará os conceitos fundamentais para interagir com o sistema de arquivos e persistir dados.

### 1. Objetivos da aula

- Compreender o conceito de manipulação de arquivos em Python.
- Aprender a abrir e fechar arquivos utilizando a função `open()` e o método `close()`.
- Dominar os diferentes modos de abertura de arquivos: leitura ("r"), escrita ("w") e adição ("a").
- Utilizar o método `write()` para gravar conteúdo em arquivos.
- Utilizar o método `readlines()` para ler o conteúdo de arquivos e o método `rstrip()` para formatar a saída.
- Desenvolver a lógica para criar programas que interagem com arquivos de texto para armazenar e recuperar informações.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e/ou acesso a plataformas interativas para o desenvolvimento de aplicativos Python.
- Conexão à internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando que o foco será em "manipulação de arquivos", um conceito crucial para que os programas possam armazenar e recuperar dados de forma persistente. Apresentar os objetivos da aula, destacando a importância de lidar com informações que não se perdem ao final da execução do programa.
- Discussão inicial: Abrir uma breve discussão sobre onde os alunos veem a necessidade de armazenar dados em arquivos (ex.: configurações de programas, listas de contatos, logs de atividades, dados de jogos).
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente, incluindo como verificar arquivos no sistema de pastas (`ls -l`).

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais que serão a base para as atividades práticas.

Arquivos .txt com Python:

- Python permite criar, ler e escrever em arquivos. Para simplificar, focaremos em arquivos de texto (.txt).
- Para acessar um arquivo, utilizamos o comando `open()`.

Sintaxe Básica de Abertura:

```
variável_arquivo = open("nome_do_arquivo", "modo")
```

- `nome_do_arquivo`: O nome do arquivo (com extensão) que você deseja manipular. Pode incluir o caminho completo se o arquivo não estiver no mesmo diretório do script.
- `modo`: Uma string que especifica a finalidade da abertura do arquivo.

Modos de abrir um arquivo:

- `"w"` (write):
  - Cria um novo arquivo para escrita.
  - Se o arquivo já existir, seu conteúdo será apagado e o arquivo será reescrito do zero.
- `"r"` (read):
  - Abre um arquivo para leitura.
  - É o modo padrão se nenhum for especificado.
  - Não permite gravação no arquivo. Se o arquivo não existir, um erro será gerado.
- `"a"` (append):
  - Abre um arquivo para adicionar conteúdo ao final.
  - Se o arquivo não existir, ele será criado.
  - Os dados são inseridos no final do conteúdo existente, sem apagar o que já está lá.

Fechando um arquivo:

- Após encerrar o acesso a um arquivo (leitura ou escrita), é crucial fechá-lo usando o método `arquivo.close()`. Isso libera os recursos do sistema e garante que todas as alterações sejam salvas.
- `arquivo.close()`



Codificação de caracteres (encoding="utf-8"):

- Para garantir a correta conversão dos caracteres do arquivo para o Python e vice-versa (especialmente caracteres acentuados ou especiais), é recomendado utilizar o parâmetro encoding="utf-8" no método open().
- `variável_arquivo = open("nome_do_arquivo", "modo", encoding="utf-8")`

Gravando dados em um arquivo (.write()):

- Para gravar conteúdo em um arquivo, utilizamos o método .write(), passando a string a ser escrita como argumento.
- Se você quiser que cada dado seja gravado em uma nova linha, deve adicionar explicitamente o caractere de quebra de linha \n ao final da string.
- `arquivo.write("Minha primeira linha\n")`
- `arquivo.write("Esta é a segunda linha.\n")`

Lendo dados de um arquivo (.readlines()):

- Para ler todas as linhas de um arquivo e transformá-las em uma lista de strings dentro do programa, utilizamos o método .readlines(). Cada elemento da lista será uma linha do arquivo, incluindo o caractere de quebra de linha \n ao final.
- `lista_de_linhas = arquivo.readlines()`

Removendo caracteres de quebra de linha (.rstrip()):

- Ao ler linhas de um arquivo, o caractere \n (quebra de linha) geralmente é incluído. Para removê-lo e obter apenas o conteúdo da linha, podemos usar o método .rstrip(). Ele remove caracteres de espaço em branco (incluindo \n) do final da string.
- `linha_limpa = linha_com_quebra_de_linha.rstrip()`

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação da manipulação de arquivos .txt utilizando o Google Colab, com exemplos práticos.

- Apresentação (utilizando o Colab): O professor apresentará os seguintes exemplos, explicando cada um:
  - Criando um arquivo e escrevendo nele (usando %%writefile para conveniência do Colab):

```
# Criando um arquivo de exemplo para demonstração
# Este comando é específico do Colab para criar um arquivo rapidamente
# Em um ambiente Python normal, você usaria apenas o open() e write()
%%writefile exemplo.txt
Linha 1 do arquivo.
Linha 2 do arquivo.
Última linha do arquivo.
```

- Lendo o conteúdo de um arquivo:

```
# Lendo o conteúdo de um arquivo
nome_arquivo_leitura = input("Digite o nome do arquivo para leitura: ")
arq_leitura = open(nome_arquivo_leitura, "r", encoding="utf-8")
conteudo_lista = arq_leitura.readlines()
print(f"\nConteúdo lido (como lista de linhas): {conteudo_lista}")

print("\nConteúdo formatado (linha por linha):")
for linha in conteudo_lista:
    print(linha.rstrip()) # Remove o \n do final da linha
arq_leitura.close()
```

- Criando um novo arquivo e escrevendo nele (modo "w"):

```
# Criando um novo arquivo (modo "w" - sobrescreve se existir)
nome_novo_arquivo = input("Digite o nome do novo arquivo a ser criado: ")
arq_escrita = open(nome_novo_arquivo, "w", encoding="utf-8")
arq_escrita.write("Este é o conteúdo da primeira linha.\n")
arq_escrita.write("Esta é a segunda linha do novo arquivo.\n")
print(f"Arquivo '{nome_novo_arquivo}' criado e conteúdo escrito.")
arq_escrita.close()
```

- Adicionando conteúdo a um arquivo existente (modo "a"):

```
# Adicionando conteúdo a um arquivo (modo "a" - anexa ao final)
nome_arquivo_anexar = input("Digite o nome do arquivo para adicionar conteúdo: ")
arq_anexar = open(nome_arquivo_anexar, "a", encoding="utf-8")
texto_adicionar = input("Digite o texto a ser adicionado: ")
arq_anexar.write(texto_adicionar + "\n")
print(f"Texto adicionado ao arquivo '{nome_arquivo_anexar}'.")
arq_anexar.close()
```

- Exemplo prático de leitura de arquivo texto e criação de arquivo com texto criptografado usando Cifra de César.
- Exemplo prático de leitura de arquivo texto criptografado e criação de arquivo com texto descriptografado usando Cifra de César.

# Esta função criptografa apenas letras (maiúsculas e minúsculas),  
# mantendo outros caracteres inalterados.

```
def criptografar_cesar(texto, chave):
```

```
    texto_criptografado = ""
```

```
    for char in texto:
```

```
        if 'a' <= char <= 'z':
```

```
            start = ord('a')
```

```
        elif 'A' <= char <= 'Z':
```

```
            start = ord('A')
```

```
        else:
```

```
            texto_criptografado += char # Não criptografa caracteres que não são letras
```

```
            continue
```

```
        # Calcula a nova posição do caractere no alfabeto.
```

```
        # O operador '%' (módulo) garante que, se a chave mover o caractere
```

```
        # além do 'Z' ou 'z', ele "volte" para o início do alfabeto ('A' ou 'a').
```

```
        nova_posicao = (ord(char) - start + chave) % 26 + start
```

```
        texto_criptografado += chr(nova_posicao)
```

```
    return texto_criptografado
```

```
# Função para descriptografar usando Cifra de César (voltando 'chave' caracteres)
def descriptografar_cesar(texto, chave):
    texto_descriptografado = ""
    for char in texto:
        if 'a' <= char <= 'z':
            start = ord('a')
        elif 'A' <= char <= 'Z':
            start = ord('A')
        else:
            texto_descriptografado += char # Caracteres não-alfabéticos permanecem
inalterados
            continue

        # Calcula a posição original do caractere no alfabeto.
        # Adicionamos 26 antes de aplicar o módulo para garantir que o resultado seja
positivo,
        # mesmo se (ord(char) - start - chave) for negativo.
        nova_posicao = (ord(char) - start - chave + 26) % 26 + start
        texto_descriptografado += chr(nova_posicao)
    return texto_descriptografado

# --- Bloco Principal do Programa ---
print("\n--- Criptografia de Arquivo ---")
nome_arquivo_entrada = input("Digite o nome do arquivo de texto a ser lido (ex.: entrada.
txt): ")
nome_arquivo_saida = input("Digite o nome do arquivo para gravar o texto criptografado
(ex.: saida_criptografada.txt): ")
```

```
# Abrir o arquivo de entrada para leitura
# Se o arquivo não existir, este comando causará um FileNotFoundError
arquivo_entrada = open(nome_arquivo_entrada, 'r', encoding='utf-8')
conteudo = arquivo_entrada.read()
arquivo_entrada.close() # Fechar o arquivo após a leitura
print(f"\nConteúdo lido do arquivo '{nome_arquivo_entrada}':\n{conteudo}")

# Definir a chave para a criptografia (ex.: 5 para César +5)
chave_criptografia = 5

# Criptografar o conteúdo
conteudo_criptografado = criptografar_cesar(conteudo, chave_criptografia)

# Abrir o arquivo de saída para escrita
# 'w' abre o arquivo para escrita, sobrescrevendo se já existir.
arquivo_saida = open(nome_arquivo_saida, 'w', encoding='utf-8')
arquivo_saida.write(conteudo_criptografado)
arquivo_saida.close() # Fechar o arquivo após a escrita
print(f"\nConteúdo criptografado com chave {chave_criptografia} e salvo em '{nome_arquivo_saida}':")
print(conteudo_criptografado)

print("\n--- Descriptografia de Arquivo ---")
nome_arquivo_criptografado = input("Digite o nome do arquivo criptografado a ser lido (ex.: saida_criptografada.txt): ")
nome_arquivo_descriptografado = input("Digite o nome do arquivo para gravar o texto descriptografado (ex.: texto_original.txt): ")
```

```
# Abrir o arquivo criptografado para leitura
arquivo_criptografado = open(nome_arquivo_criptografado, 'r', encoding='utf-8')
conteudo_criptografado = arquivo_criptografado.read()
arquivo_criptografado.close() # Fechar o arquivo após a leitura
print(f"\nConteúdo lido do arquivo '{nome_arquivo_criptografado}':\n{conteudo_criptografado}")
```

```
# A chave de descriptografia deve ser a mesma chave usada na criptografia
chave_descriptografia = 5
```

```
# Descriptografar o conteúdo
conteudo_descriptografado = descriptografar_cesar(conteudo_criptografado, chave_descriptografia)
```

```
# Abrir o arquivo de saída para escrita do texto descriptografado
arquivo_descriptografado = open(nome_arquivo_descriptografado, 'w', encoding='utf-8')
arquivo_descriptografado.write(conteudo_descriptografado)
arquivo_descriptografado.close() # Fechar o arquivo após a escrita
print(f"\nConteúdo descriptografado com chave {chave_descriptografia} e salvo em '{nome_arquivo_descriptografado}':")
print(conteudo_descriptografado)
```

#### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

## 4.1 Desafio inicial

Sistema Simples de Gerenciamento de Arquivos:

- Descrição: Crie um programa que ofereça um menu interativo ao usuário para realizar operações básicas com um arquivo de texto. O programa deve continuar executando até que o usuário escolha sair.

- Opções do Menu:

1. Criar um arquivo .txt: Solicita o nome do arquivo e o cria (modo "w").
2. Acessar (ler) um arquivo .txt: Solicita o nome do arquivo e exibe todo o seu conteúdo, linha por linha, removendo as quebras de linha.
3. Inserir conteúdo em um arquivo .txt: Solicita o nome do arquivo e um texto para ser inserido. O texto deve ser adicionado ao final do arquivo, em uma nova linha.
4. Sair: Encerra o programa.

- Requisitos:

- Utilize funções para cada operação (criar, acessar, inserir, menu).
- O programa deve repetir até que a opção 4 seja selecionada.
- Observação: Como não estamos utilizando try-except, o programa pode gerar um erro se tentar abrir um arquivo que não existe no modo de leitura ("r") ou adição ("a") se ele não tiver sido criado previamente. Certifique-se de criar o arquivo antes de tentar lê-lo ou adicionar conteúdo.



## 4.2 Ampliação do desafio

### Sistema para gerenciamento de criptografia:

- Criar um programa para gerenciar um sistema de criptografia utilizando a Cifra de César.
- Funcionalidades:
  - Criar um arquivo texto.
  - Inserir um texto.
  - Ler um arquivo texto.
  - Criptografar um arquivo texto: deve solicitar a chave de acesso.
  - Descriptografar um arquivo texto: deve solicitar a chave de acesso.

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente no que diz respeito ao controle de fluxo em programas com arquivos e a lógica do jogo da forca.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- Orientação sobre o Relatório Final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- **Resumo teórico:** Uma explicação clara e concisa sobre a manipulação de arquivos .txt em Python. Inclua os conceitos de abertura (`open()`), modos ("`r`", "`w`", "`a`"), fechamento (`close()`), leitura (`readlines()`) e escrita (`write()`), e a importância da codificação (`encoding="utf-8"`).
- **Código-fonte comentado:** O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Sistema Simples de Gerenciamento de Arquivos) e a "Ampliação do desafio" (Jogo da Forca). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento, o propósito de cada linha ou bloco, e como os conceitos de manipulação de arquivos foram aplicados.

## 7. Critérios de avaliação



Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (manipulação de arquivos, modos, métodos).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários explicativos, nomes de funções e variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados para diferentes entradas.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá um entendimento sólido sobre como manipular arquivos de texto em Python, sendo capaz de criar, ler e escrever dados em arquivos, o que é fundamental para persistir informações e construir aplicações que interagem com o sistema de arquivos. A manipulação de arquivos é essencial em diversas áreas da programação, como armazenamento de dados (salvar configurações de usuário, logs de sistema, dados de jogos), processamento de texto (análise de grandes volumes de texto, geração de relatórios), automação (leitura de arquivos de entrada para scripts, escrita de resultados) e desenvolvimento web (servir arquivos estáticos, armazenar dados simples). O domínio da manipulação de arquivos é um pré-requisito crucial para tópicos mais avançados, como o trabalho com formatos de dados estruturados (JSON, CSV), interação com bancos de dados, e o desenvolvimento de aplicações que exigem persistência de dados complexos. Uma base sólida neste conceito permitirá que os alunos construam programas cada vez mais completos e funcionais.

**Bom estudo e boa prática!**

  <b>Instituto de Ciências Exatas e Tecnologia</b>	<b>Disciplina:</b> Introdução à Programação Estruturada <b>Título da Aula:</b> Módulos, sys e from...import	<b>ROTEIRO 11</b>
---	--	-------------------

## Roteiro da Aula Prática – Módulos, sys e from...import em Python

Este roteiro detalha uma aula prática focada no conceito de módulos em Python, na instalação de pacotes com pip, na interação com o ambiente Python através do módulo sys, e nas diferentes formas de importação (import e from...import). Utilizaremos o ambiente interativo do Google Colab para as demonstrações e atividades.

### 1. Objetivos da aula

- Compreender o conceito de módulos em Python e sua importância para a organização e reusabilidade do código.
- Aprender a instalar módulos de terceiros utilizando o gerenciador de pacotes pip.
- Explorar o módulo sys para interagir com o interpretador Python e obter informações sobre o ambiente de execução.
- Dominar as declarações import e from...import para importar elementos de módulos de forma eficiente.
- Entender o que são arquivos .pyc e sua função na otimização de desempenho.

### 2. Recursos necessários

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e conexão com a internet.
- Material de apoio (o próprio notebook do Colab com o roteiro e exemplos).

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- Boas-vindas e explicação dos objetivos: Iniciar a aula explicando que o foco será em "módulos", que são como "caixas de ferramentas" que nos ajudam a organizar e reutilizar código. Apresentar os objetivos da aula, destacando como os módulos tornam a programação mais eficiente e colaborativa.
- Discussão inicial: Abrir uma breve discussão sobre a importância de não "reinventar a roda" na programação. Perguntar aos alunos se eles já pensaram em como os grandes programas são construídos sem que tudo seja escrito do zero. Introduzir a ideia de que módulos são a resposta.
- Relembrando o Google Colab: Breve recordatório sobre como usar o Google Colab para executar código Python, criar novas células e interagir com o ambiente.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais para a aula.

Introdução aos Módulos em Python:

- O que são módulos? São arquivos Python (.py) que contêm definições de funções, classes e variáveis. Eles permitem organizar o código em unidades lógicas e reutilizáveis.
- Benefícios: Organização do código, reusabilidade (evita repetição), facilita a colaboração em projetos grandes.
- Módulos Built-in vs. Módulos de Terceiros:
  - Built-in: Já vêm com a instalação padrão do Python (ex.: math, random, os, sys).
  - Terceiros: Desenvolvidos pela comunidade e precisam ser instalados (ex.: numpy, pandas, requests).

Instalando Módulos com pip:

- O que é pip? É o gerenciador de pacotes padrão para Python, usado para instalar, atualizar e remover pacotes (módulos) de terceiros do Python Package Index (PyPI).
- Utilizando pip no Google Colab: No Colab, os comandos de terminal são prefixados com `!`.
  - Exemplo: `!pip install requests`

Explorando o Módulo `sys`:

- O que é o módulo `sys`? Fornece acesso a variáveis e funções que interagem diretamente com o interpretador Python e o ambiente de execução.
- Principais atributos e funções:
  - `sys.version`: Retorna a versão do Python em uso.
  - `sys.platform`: Retorna a plataforma do sistema operacional (ex.: `'linux'`, `'win32'`).
  - `sys.path`: Uma lista de strings que especifica os diretórios onde o interpretador Python procura por módulos.
  - `sys.argv`: Uma lista de argumentos de linha de comando passados para um script Python. (Mais comum em scripts executados via terminal).
  - `sys.exit()`: Usado para terminar a execução do programa.

A Declaração `from...import`:

- Problema: Usar `import nome_do_modulo` exige que você sempre prefixe os elementos com `nome_do_modulo`. (ex.: `math.pi`). Para módulos grandes ou quando se usa muitos elementos, isso pode ser repetitivo.
- Sintaxe: `from nome_do_modulo import elemento1, elemento2, ...`
- Benefícios: Melhora a legibilidade do código, permite importar apenas o que é necessário.
- Renomeando elementos: `from nome_do_modulo import elemento as novo_nome`

Arquivos Byte-Compiled .pyc:

- São arquivos "byte-compiled" (compilados para bytecode).
- Criados automaticamente pelo interpretador Python quando um módulo é importado pela primeira vez.
- Função: Otimizam o desempenho, pois o interpretador não precisa recompilar o código-fonte (.py) toda vez que o módulo é importado, resultando em carregamento mais rápido.
- No Google Colab, não vemos esses arquivos diretamente, mas o mecanismo de otimização ainda ocorre.

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar a aplicação dos conceitos de módulos e importação utilizando o Google Colab, com exemplos práticos.

- Apresentação (utilizando o Colab): O professor apresentará os seguintes exemplos, explicando cada um:

- Importação básica de math:

```
import math
print(f"Valor de pi: {math.pi}")
print(f"Raiz quadrada de 16: {math.sqrt(16)}")
```

- Instalação e uso de um módulo de terceiros (requests):

```
!pip install requests
import requests
response = requests.get('https://www.google.com')
print(f"Status da requisição para Google: {response.status_code}")
```

- Explorando `sys.version` e `sys.platform`:

```
import sys
print(f"Versão do Python: {sys.version}")
print(f"Plataforma do sistema operacional: {sys.platform}")
```

- Demonstrando `from...import`:

```
from math import pi, sqrt
print(f"Valor de pi (importação direta): {pi}")
print(f"Raiz quadrada de 25 (importação direta): {sqrt(25)}")
```

```
from math import sqrt as raiz_quadrada
print(f"Raiz quadrada de 49 (com renomeação): {raiz_quadrada(49)}")
```

#### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos.

##### 4.1 Desafio inicial

Este exercício tem como objetivo familiarizar o aluno com a instalação e o uso da biblioteca Faker em Python para a geração de dados fictícios, bem como a utilização do módulo random para a criação de números aleatórios.

- **Instruções**

##### 1. Instalação do Módulo Faker:

- No ambiente Google Colab, execute o comando necessário para instalar a biblioteca Faker.



## 2. Importação e Configuração do Faker:

- Importe a classe Faker do módulo faker.
- Crie uma instância do gerador de dados Faker, configurando-o para gerar dados em português do Brasil ('pt\_BR').

## 3. Geração e Impressão de Dados Pessoais Fictícios:

- Utilize o objeto Faker criado para gerar e imprimir na tela:
  - Um nome completo aleatório.
  - Um endereço aleatório.
  - Um endereço de e-mail aleatório.

## 4. Geração de Notas Aleatórias:

- Importe o módulo random do Python.
- Gere duas notas aleatórias (números de ponto flutuante) entre 0 (inclusive) e 10 (inclusive) usando a função apropriada do módulo random.
- Imprima as duas notas, formatando-as para exibir com duas casas decimais.

Seu código aqui:

## 4.2 Ampliação do desafio

Crie um programa Python que siga os passos abaixo:

### 1. Configuração inicial:

- Instale a biblioteca Faker utilizando o comando apropriado para ambientes como o Google Colab.
- Importe os módulos `sys`, `random`, `math` e a classe `Faker`.

### 2. Informações do sistema (opcional, mas recomendado para exploração):

- Imprima o caminho do executável Python.
- Liste os diretórios onde o Python procura por módulos (`sys.path`).

### 3. Gerador de dados fictícios:

- Crie uma instância do gerador de dados Faker configurada para gerar dados em português do Brasil ('pt\_BR').

### 4. Criação da agenda fictícia:

- Crie uma lista vazia chamada `agenda_ficticia` que será utilizada para armazenar os registros.
- Implemente um laço de repetição que gere **10 (dez) registros** completos. Para cada registro:
  - Gere um nome completo, um endereço e um endereço de e-mail utilizando os métodos apropriados da biblioteca Faker.
  - Gere duas notas aleatórias (números de ponto flutuante) entre 0 (inclusive) e 10 (inclusive) usando o módulo `random`.

- Calcule a média geométrica dessas duas notas. Lembre-se de que a média geométrica de dois números  $x$  e  $y$  é  $\sqrt{x \cdot y}$ . Garanta que seu cálculo lide corretamente com o caso em que uma ou ambas as notas sejam zero (a média geométrica deve ser zero nesse caso para evitar erros de raiz quadrada de números negativos ou para seguir a lógica de que um fator zero anula o produto).
- Crie um dicionário para representar o registro da pessoa, contendo as seguintes chaves e seus respectivos valores:
  - "Nome"
  - "Endereço"
  - "Email"
  - "Notas" (armazene as duas notas como uma tupla)
  - "Média Geométrica"
- Adicione este dicionário à lista `agenda_ficticia`.
- Imprima uma mensagem indicando que o registro foi gerado, incluindo o nome da pessoa.

## 5. Exibição dos registros:

- Após a geração de todos os 10 registros, percorra a lista `agenda_ficticia`.
- Para cada registro, imprima de forma organizada todas as informações: nome, endereço, e-mail, as duas notas (formatadas com duas casas decimais) e a média geométrica (formatada com duas casas decimais). Utilize separadores para facilitar a leitura de cada registro.

Seu código aqui:

## 5. Encerramento e orientações finais (20 minutos)

- Discussão: Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente com a instalação de módulos e a compreensão dos atributos do sys.
- Revisão das soluções: Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- Orientação sobre a entrega do relatório final: Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- Resumo teórico: Uma explicação clara e concisa sobre o conceito de módulos em Python, a importância do pip para a instalação de pacotes, a função do módulo sys para interagir com o ambiente, e as diferenças e casos de uso das declarações import e from...import. Inclua uma breve menção aos arquivos .pyc e sua finalidade.
- Código-fonte comentado: O código-fonte completo das soluções desenvolvidas para o "Desafio Inicial" (Faker) e a "Ampliação do Desafio" (sys.executable e sys.path). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento e propósito.

## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (módulos, pip, sys, importações, pyc).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do "Desafio inicial" e da "Ampliação do desafio" devem funcionar corretamente, produzindo os resultados esperados.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, os alunos terão um entendimento sólido sobre a modularização de código em Python, a gestão de pacotes com pip, e a interação com o ambiente de execução através do módulo sys. Eles serão capazes de importar e utilizar bibliotecas externas, além de compreender como o Python organiza e carrega seus módulos. O uso de módulos e pacotes é fundamental em qualquer projeto Python, com aplicações que incluem desenvolvimento de software (organização de grandes bases de código), análise de dados e Machine Learning (utilização de bibliotecas como pandas, numpy, scikit-learn), desenvolvimento web (frameworks como Django e Flask são compostos por módulos), e automação de tarefas (scripts que interagem com o sistema operacional via os module ou com a web via requests module). O domínio de módulos e pacotes é crucial para os próximos tópicos da disciplina, pois permite que os alunos utilizem ferramentas e bibliotecas avançadas, sendo um pré-requisito para a programação orientada a objetos, o desenvolvimento de aplicações mais complexas e a participação em projetos de software maiores. Uma base sólida neste conceito permitirá que os alunos construam programas cada vez mais robustos, eficientes e escaláveis.

**Bom estudo e boa prática!**



**Instituto de Ciências  
Exatas e Tecnologia**

**Disciplina:** Introdução à Programação Estruturada

**Título da Aula:** Conexão com Banco de Dados

**ROTEIRO 12**

### **Roteiro da Aula Prática – Conexão com Banco de Dados**

Este roteiro detalha uma aula prática focada na conexão e manipulação de dados em um banco de dados SQLite utilizando Python, diretamente no ambiente interativo do Google Colab. A aula abordará as operações básicas de leitura, gravação e alteração de dados.

#### **1. Objetivos da aula**

- Compreender o conceito de banco de dados SQLite como um SGBD embarcado e sem servidor.
- Aprender a conectar-se a um banco de dados SQLite a partir do Google Colab.
- Executar um script SQL para criar e popular a tabela Aluno.
- Realizar operações de leitura (SELECT), gravação (INSERT) e alteração (UPDATE) na tabela Aluno usando Python e SQLite.
- Utilizar o módulo sqlite3 do Python para interagir com o banco de dados.

#### **2. Recursos necessários**

- Computadores com acesso ao site [colab.research.google.com](https://colab.research.google.com) e conexão com a internet.
- Este notebook do Google Colab como material de apoio.

- **Script SQL para configuração do banco de dados:**

-- A conexão a um arquivo .db cria o arquivo se ele não existir.

-- Para reiniciar o banco de dados, basta excluir o arquivo .db ou recriar a tabela.

```
DROP TABLE IF EXISTS Aluno;
```

```
CREATE TABLE Aluno
```

```
(
```

```
    Codigo  INTEGER PRIMARY KEY, -- SQLite usa INTEGER PRIMARY KEY para auto-  
incremento
```

```
    Nome    CHAR(15),
```

```
    Semestre INTEGER,
```

```
    Curso   CHAR(10)
```

```
);
```

```
INSERT INTO Aluno VALUES (1, 'AMANDO DIAS', 3, 'CC');
```

```
INSERT INTO Aluno VALUES (2, 'DANILA CARMO', 4, 'SI');
```

```
INSERT INTO Aluno VALUES (3, 'ISABELO PAIVA', 2, 'CC');
```

```
INSERT INTO Aluno VALUES (4, 'GUSTAVA MARTINS', 1, 'SI');
```

```
INSERT INTO Aluno VALUES (5, 'ORLANDA MOURA', 2, 'SI');
```

```
INSERT INTO Aluno VALUES (6, 'REGINO SILVA', 2, 'CC');
```

```
INSERT INTO Aluno VALUES (7, 'GILFREDO SANTOS', 8, 'CC');
```

```
INSERT INTO Aluno VALUES (8, 'ABDUL SATO', 4, 'CC');
```

-- COMMIT é automático por padrão em SQLite após cada instrução DML,

-- mas é boa prática usar connection.commit() no Python.

**Importante:** Este script será executado diretamente no Python dentro do Google Colab para configurar o banco de dados SQLite.

### 3. Estrutura da aula

#### 3.1. Abertura (10 minutos)

- **Boas-vindas e explicação dos objetivos:** Iniciar a aula explicando que o foco será na integração de Python com bancos de dados, especificamente SQLite, que é um banco de dados leve e ideal para aplicações embarcadas e testes. Apresentar os objetivos da aula, destacando a importância de persistir dados em um banco de dados para aplicações reais.
- **Discussão inicial:** Abrir uma breve discussão sobre a necessidade de bancos de dados para armazenar informações de forma organizada. Introduzir o SQLite como uma alternativa prática ao MySQL para cenários onde um servidor completo não é necessário.
- **Relembrando o Google Colab e preparação:** Breve recordatório sobre como usar o Google Colab. Mencionar que o SQLite já vem embutido no Python, não sendo necessária instalação adicional de bibliotecas.

#### 3.2. Revisão conceitual (20 minutos)

Nesta seção, revisaremos os conceitos fundamentais para a aula.

##### **Introdução ao SQLite e Python:**

- **SQLite:** É um sistema de gerenciamento de banco de dados relacional (SGBDR) leve, sem servidor e de código aberto. Ao contrário de outros SGBDs como MySQL ou PostgreSQL, o SQLite não requer um processo de servidor separado. Ele armazena o banco de dados em um único arquivo no disco.
- **Interação Python-SQLite:** Python possui um módulo embutido chamado `sqlite3` que permite interagir facilmente com bancos de dados SQLite. Isso significa que você não precisa instalar nada extra para começar a usar SQLite com Python.
- **Vantagens do SQLite para treinamento:**



- **Configuração Zero:** Não há servidor para instalar ou configurar.
- **Portabilidade:** Um banco de dados SQLite é um arquivo único que pode ser facilmente copiado e transportado.
- **Embutido:** Já faz parte da biblioteca padrão do Python.

### Conectando ao Banco de Dados SQLite:

- Para interagir com um banco de dados SQLite, o primeiro passo é estabelecer uma conexão.
- A função `sqlite3.connect()` é usada para isso. Você pode passar o nome de um arquivo `.db` (que será criado se não existir) ou `:memory:` para um banco de dados em memória (que será apagado ao fechar a conexão).
- **Cursor:** Após a conexão, é necessário criar um objeto cursor. O cursor é usado para executar comandos SQL no banco de dados.
- `connection.commit()`: Em SQLite, as alterações (INSERT, UPDATE, DELETE) não são salvas permanentemente no arquivo do banco de dados até que `connection.commit()` seja chamado. É crucial chamar este método após operações de modificação de dados.
- `connection.close()`: É importante fechar a conexão com o banco de dados quando não for mais necessária para liberar recursos e garantir que as alterações sejam salvas.

### 3.3. Demonstração prática (30 minutos)

Nesta seção, o professor irá demonstrar as operações de integração com SQLite utilizando o Google Colab.

- **Apresentação (utilizando o Colab):** O professor apresentará os seguintes exemplos, explicando cada um.

- **Passo 1: Importar a biblioteca sqlite3**

- **Explicação:** O módulo sqlite3 já vem com o Python, então não precisamos instalar nada. A primeira coisa a fazer em qualquer script que interaja com SQLite é importá-lo.

- **Código no Colab:**

```
import sqlite3
```

- **Passo 2: Verificar a versão do Python (opcional, mas útil)**

- **Explicação:** É uma boa prática verificar a versão do Python para garantir compatibilidade, embora para sqlite3 isso raramente seja um problema.

- **Código no Colab:**

```
from platform import python_version
```

```
print(python_version())
```

- **Passo 3: Criar ou conectar-se ao banco de dados SQLite**

- **Explicação:** Usamos sqlite3.connect() para criar um arquivo de banco de dados (Banco\_Dados no exemplo do anexo) ou conectar-se a um existente. Se o arquivo não existir, ele será criado.

- **Código no Colab:**

```
# Com esse comando o colab vai criar nosso banco de dados 'Banco_Dados.db'
```

```
bd = sqlite3.connect('Banco_Dados.db')
```

- **Passo 4: Criar um objeto Cursor**

- **Explicação:** O cursor é a ferramenta que usamos para enviar comandos SQL para o banco de dados e receber os resultados.

- **Código no Colab:**

```
# Agora nós precisamos apontar para o banco de dados
```

```
# Toda vez que eu for enviar uma informação para o meu bd nós vamos utilizar essa variavel
```

```
Cursor = bd.cursor()
```

- **Passo 5: Criar a tabela Aluno e popular com dados iniciais**

- **Explicação:** Usaremos o comando CREATE TABLE para definir a estrutura da nossa tabela Aluno. Incluiremos Codigo como INTEGER PRIMARY KEY (que também serve como auto-incremento em SQLite), Nome, Semestre e Curso. Em seguida, usaremos INSERT INTO para adicionar os dados iniciais. O executescrypt é útil para executar múltiplos comandos SQL de uma vez.

- **Código no Colab:**

```
# Script SQL para criar e popular a tabela Aluno
```

```
sql_script_aluno = ""
```

```
DROP TABLE IF EXISTS Aluno;
```

```
CREATE TABLE Aluno
```

```
(
```

```
    Codigo INTEGER PRIMARY KEY,
```

```
    Nome CHAR(15),
```

```
    Semestre INTEGER,
```

```
    Curso CHAR(10)
```

```
);
```

```
INSERT INTO Aluno VALUES (1, 'AMANDO DIAS', 3, 'CC');
INSERT INTO Aluno VALUES (2, 'DANILA CARMO', 4, 'SI');
INSERT INTO Aluno VALUES (3, 'ISABELO PAIVA', 2, 'CC');
INSERT INTO Aluno VALUES (4, 'GUSTAVA MARTINS', 1, 'SI');
INSERT INTO Aluno VALUES (5, 'ORLANDA MOURA', 2, 'SI');
INSERT INTO Aluno VALUES (6, 'REGINO SILVA', 2, 'CC');
INSERT INTO Aluno VALUES (7, 'GILFREDO SANTOS', 8, 'CC');
INSERT INTO Aluno VALUES (8, 'ABDUL SATO', 4, 'CC');
""
```

```
# Executa o script SQL
```

```
Cursor.executescript(sql_script_aluno)
```

```
bd.commit() # Confirma as alterações
```

```
print("Tabela 'Aluno' criada e populada com sucesso!")
```

#### ▪ **Passo 6: Leitura de Dados (SELECT) da tabela Aluno**

- **Explicação:** O comando `SELECT * FROM Aluno` busca todas as colunas e todas as linhas da tabela Aluno. `fetchall()` recupera todos os resultados.

- **Código no Colab:**

```
# Consulta todos os dados da tabela Aluno
```

```
Consulta = Cursor.execute('SELECT * FROM Aluno').fetchall()
```

```
print("\n--- Dados da tabela 'Aluno' ---")
```

```
for Linha in Consulta:
```

```
    print(Linha)
```

- **Passo 7: Inserção de Dados (INSERT) na tabela Aluno**

- **Explicação:** Usaremos INSERT INTO para adicionar um novo registro. O ? é o placeholder padrão para SQLite.

- **Código no Colab:**

```
# Insere um novo aluno
Cursor.execute('INSERT INTO Aluno VALUES (?, ?, ?, ?)', (9, 'NOVO ALUNO', 1, 'SI'))
bd.commit() # Confirma a transação

print("\nNovo aluno inserido. Verificando:")
Consulta = Cursor.execute('SELECT * FROM Aluno WHERE Codigo = 9').fetchall()
for Linha in Consulta:
    print(Linha)
```

- **Passo 8: Alteração de Dados (UPDATE) na tabela Aluno**

- **Explicação:** O comando UPDATE é usado para modificar registros existentes. Usaremos a cláusula WHERE para especificar qual registro será atualizado.

- **Código no Colab:**

```
# Altera o semestre do aluno com Código 1 para 5
Cursor.execute('UPDATE Aluno SET Semestre = ? WHERE Codigo = ?', (5, 1))
bd.commit() # Confirma a transação

print("\nAluno com Código 1 atualizado. Verificando:")
Consulta = Cursor.execute('SELECT * FROM Aluno WHERE Codigo = 1').fetchall()
for Linha in Consulta:
    print(Linha)
```

- **Passo 9: Fechando a Conexão**

- **Explicação:** É fundamental fechar o cursor e a conexão para liberar recursos e garantir que todas as alterações sejam salvas.

- **Código no Colab:**

```
# Fechar o cursor e a conexão com o banco de dados
Cursor.close()
bd.close()
print("\nConexão com o banco de dados 'Banco_Dados.db' encerrada.")
```

### 3.4. Outros Comandos SQL (Baseado no SQL\_COM\_PYTHON\_PARA\_ANALISE\_DE\_DADOS.ipynb)

O professor pode apresentar brevemente outros comandos SQL que estão no anexo, explicando seus conceitos e mostrando exemplos de execução.

- **CREATE TABLE para Tab\_Vendas e Tab\_Cadastro\_Vendedor:**

- **Explicação:** Demonstra a criação de mais tabelas para ilustrar JOINS.

- **Código no Colab:** (Professor pode copiar e colar do anexo)

```
# Criar tabelas para JOINS
Cursor.execute("""
    CREATE TABLE Tab_Vendas (id real, valor real)
""")
bd.commit()

Cursor.execute("""
    CREATE TABLE Tab_Cadastro_Vendedor (id real, nome text)
""")
bd.commit()
```

- **INSERT para Tab\_Vendas e Tab\_Cadastro\_Vendedor:**

- **Explicação:** Popula as novas tabelas com dados.
- **Código no Colab:** (Professor pode copiar e colar do anexo)

# Inserir dados nas tabelas de vendas e cadastro

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (1, 100)')
```

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (1, 200)')
```

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (1, 150)')
```

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (2, 50)')
```

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (2, 200)')
```

```
Cursor.execute('INSERT INTO Tab_Vendas VALUES (2, 900)')
```

```
bd.commit()
```

```
Cursor.execute('INSERT INTO Tab_Cadastro_Vendedor VALUES (1, "Odemir Depieri Jr")')
```

```
Cursor.execute('INSERT INTO Tab_Cadastro_Vendedor VALUES (2, "Lucas Calmon")')
```

```
bd.commit()
```

- **SELECT com WHERE (Operadores de comparação, BETWEEN, LIKE, IN, AND, OR, NOT)**

- **Explicação:** Mostrar como filtrar dados usando diferentes condições.
- **Código no Colab:** (Professor pode copiar e colar exemplos do anexo)

# Exemplo de SELECT com WHERE

```
Consulta = Cursor.execute("""
```

```
    SELECT * FROM Aluno
```

```
    WHERE Semestre > 2
```

```
""").fetchall()
```

```
print("\nAlunos com Semestre > 2:")
```

```
for Linha in Consulta:
```

```
    print(Linha)
```

- **ORDER BY**

- **Explicação:** Classificar os resultados de uma consulta.
- **Código no Colab:** (Professor pode copiar e colar exemplos do anexo)

# Exemplo de SELECT com ORDER BY

```
Consulta = Cursor.execute("""
    SELECT * FROM Aluno
    ORDER BY Nome ASC
""").fetchall()
print("\nAlunos ordenados por Nome:")
for Linha in Consulta:
    print(Linha)
```

- **NULL VALUES e UPDATE para NULL**

- **Explicação:** Como lidar com valores nulos e atualizá-los.
- **Código no Colab:** (Professor pode adaptar exemplos do anexo para a tabela Aluno ou criar uma nova tabela para demonstrar)

# Exemplo de INSERT com NULL (se a coluna permitir)

# Primeiro, vamos adicionar uma coluna que pode ser nula na tabela Aluno para demonstração

# Isso é um ALTER TABLE, que não está no roteiro original, mas é útil para demonstrar NULL

```
# Cursor.execute('ALTER TABLE Aluno ADD COLUMN Nota REAL;')
# bd.commit()
# Cursor.execute('INSERT INTO Aluno (Codigo, Nome, Semestre, Curso, Nota) VALUES (?,
?, ?, ?, NULL)', (10, 'ALUNO TESTE', 3, 'CC'))
# bd.commit()
```

# Exemplo de UPDATE para NULL (se houver valores nulos)

```
# Cursor.execute('UPDATE Aluno SET Nota = ? WHERE Nota IS NULL', (7.5,))
# bd.commit()
```



- **DELETE**

- **Explicação:** Remover registros da tabela.

- **Código no Colab:** (Professor pode copiar e colar exemplos do anexo)

# Exemplo de DELETE

```
Cursor.execute('DELETE FROM Aluno WHERE Codigo = ?', (9,))
```

```
bd.commit()
```

```
print("\nAluno com Código 9 removido. Verificando:")
```

```
listar_alunos() # Chama a função de listar alunos para verificar
```

#### 4. Atividade prática (40 minutos)

Os alunos deverão aplicar os conhecimentos adquiridos para resolver os desafios propostos. Escreva seu código nas células abaixo de cada desafio.

**Importante:** Para os desafios abaixo, você deve usar o banco de dados SQLite Banco\_Dados.db que será criado e populado no início da aula. Lembre-se de reabrir a conexão (`sqlite3.connect()`) para cada desafio, se necessário, ou gerenciar uma única conexão.

##### 4.1 Desafio inicial

- **Descrição:** Crie um programa Python que se conecte ao banco de dados SQLite Banco\_Dados.db. Em seguida, liste todos os alunos da tabela Aluno, exibindo Codigo, Nome, Semestre e Curso de forma organizada.

**Seu código aqui:**

## 4.2 Ampliação do desafio

- **Descrição:** Desenvolva um sistema básico de gerenciamento de alunos para a tabela Aluno. O programa deve permitir ao usuário realizar as seguintes operações:

1. **Listar todos os alunos:** Exiba todos os alunos da tabela Aluno.
  2. **Adicionar um novo aluno:** Permita ao usuário inserir Código, Nome, Semestre e Curso para um novo aluno.
  3. **Atualizar o semestre de um aluno:** Permita ao usuário informar o Código de um aluno e um novo Semestre para ele.
  4. **Remover um aluno:** Permita ao usuário informar o Código de um aluno para removê-lo da tabela.
  5. **Buscar aluno por nome:** Permita ao usuário digitar parte de um nome e liste todos os alunos que contenham essa parte no nome.
  6. **Sair:** Encerra o programa.
- **Requisitos:**
    - Implemente um menu interativo para o usuário escolher as opções.
    - Utilize as operações SELECT, INSERT, UPDATE e DELETE do SQL.
    - O programa deve continuar executando até que o usuário escolha sair.
    - Para a busca por nome, use a cláusula LIKE no SQL (ex.: SELECT \* FROM Aluno WHERE Nome LIKE ?).

Seu código aqui:

## 5. Encerramento e orientações finais (20 minutos)

- **Discussão:** Abrir para discussão sobre as principais dificuldades enfrentadas pelos alunos durante a prática dos desafios, especialmente com a conexão ao SQLite, a execução do script SQL e a lógica das operações de banco de dados.
- **Revisão das soluções:** Se o tempo permitir, revisar as soluções dos desafios, destacando diferentes abordagens e boas práticas de código.
- **Orientação sobre o relatório final:** Explicar detalhadamente a estrutura e os requisitos para a entrega do relatório final, ressaltando a importância de cada seção para a avaliação.

## 6. Orientações para o relatório final

Cada aluno deve produzir um relatório curto (1 a 2 páginas) contendo:

- **Resumo teórico:** Uma explicação clara e concisa sobre o que é um banco de dados SQLite, suas características (embarcado, sem servidor, arquivo único), a importância do módulo `sqlite3` em Python, e os conceitos das operações SQL básicas (SELECT, INSERT, UPDATE, DELETE) aplicadas ao SQLite.
- **Código-fonte comentado:** O código-fonte completo das soluções desenvolvidas para o "Desafio inicial" (Listagem de Alunos) e a "Ampliação do desafio" (Sistema de Gerenciamento de Alunos). Cada trecho de código deve ser acompanhado de comentários explicativos detalhados sobre seu funcionamento, o propósito de cada linha ou bloco, e como a interação com o banco de dados foi implementada.


## 7. Critérios de avaliação

Critério	Peso	Descrição
Qualidade do Resumo Teórico	2,0	Clareza, objetividade e demonstração de entendimento sobre a teoria abordada na aula (SQLite, módulo sqlite3, operações SQL).
Estrutura e Organização do Código	3,0	Legibilidade, uso de comentários, nomes de variáveis e funções significativos e organização lógica do código.
Funcionamento da Solução	3,0	Os códigos do 'Desafio inicial' e da 'Ampliação do desafio' devem funcionar corretamente, realizando as operações esperadas no banco de dados SQLite Banco_Dados.db e na tabela Aluno.
Criatividade e Aprimoramentos	2,0	Inclusão de melhorias que demonstrem domínio do conteúdo, como validação de entrada robusta, tratamento de casos especiais, ou apresentação mais elaborada dos resultados.

**Nota Final:** Será a soma dos valores obtidos em cada critério. Alunos ou equipes que não cumprirem os requisitos mínimos de funcionamento do código ou não entregarem o relatório dentro do prazo terão sua nota diminuída proporcionalmente.

## 8. Conclusão

Ao final desta aula, o aluno terá um entendimento fundamental sobre como Python pode interagir com um banco de dados SQLite, sendo capaz de estabelecer conexões, executar queries SQL para ler, inserir, atualizar e remover dados na tabela Aluno, o que é uma habilidade essencial para o desenvolvimento de aplicações que precisam armazenar e gerenciar informações de forma persistente. O SQLite é amplamente utilizado em diversas aplicações, incluindo aplicativos móveis (muitos aplicativos Android e iOS usam SQLite para armazenamento de dados local), navegadores web (Google Chrome, Firefox e Safari usam SQLite para gerenciar histórico, favoritos e cookies), dispositivos embarcados (ideal para dispositivos com recursos limitados), testes e prototipagem (ótimo para desenvolver e testar aplicações sem a complexidade de um servidor de banco de dados completo), e aplicações desktop (pode ser usado como o banco de dados principal para softwares de desktop). O domínio da integração com bancos de dados, seja SQLite ou outros, é crucial para os próximos



tópicos da disciplina, sendo um pré-requisito para o desenvolvimento de aplicações mais complexas, sistemas de gerenciamento de dados e para a compreensão de arquiteturas de software que dependem de persistência de dados. Uma base sólida neste conceito permitirá que o aluno construa programas cada vez mais robustos, eficientes e escaláveis.

**Bom estudo e boa prática!**

