Introdução à Linguagem Python

Aula 1: O que é Python?

O que é Python?

- Uma linguagem de programação de **alto nível**, interpretada.
- **Multiparadigma**: Suporta programação imperativa, orientada a objetos e funcional.
- **Fácil de aprender** e ler, com uma sintaxe clara e acessível.
- Extremamente **versátil**: usada para desenvolvimento web, automação, ciência de dados, inteligência artificial, e muito mais.

Aula 1: História do Python

- Criada por **Guido van Rossum** no final dos anos 80 e lançada em 1991.
- Nomeada em homenagem ao grupo de comédia britânico **"Monty Python"**.
- **Python 2.0** foi lançado em 2000 com melhorias significativas.
- **Python 3.0** foi lançado em 2008, trazendo várias mudanças importantes.

Aula 1: Por que Python?

Simplicidade:

- Sintaxe limpa e de fácil compreensão.
- Ideal para iniciantes e projetos complexos.

Popularidade:

- Python é amplamente adotada, com uma enorme comunidade de desenvolvedores.

Biblioteca Rica:

- Mais de 300 mil bibliotecas disponíveis (Pandas, NumPy, TensorFlow, Flask, Django).

Aula 1: Aplicações do Python

1. **Desenvolvimento Web**:
    1. Frameworks como Django e Flask.
2. **Ciência de Dados e Machine Learning**:
    1. Bibliotecas: Pandas, NumPy, Scikit-learn, TensorFlow.
3. **Automação de Tarefas**:
    1. Scripts para automatizar processos repetitivos.
4. **Desenvolvimento de Jogos**:
    1. Frameworks como Pygame.
5. **Internet das Coisas (IoT)**:
    1. Python em dispositivos embarcados como Raspberry Pi.

Aula 1: Sintaxe Simples e Clara

Exemplo de um código simples em Python:

python Copiar código

```python
# Exemplo: Olá, Mundo
print("Olá, Mundo!")
```

Exemplo de uma função simples:

python Copiar código

```python
def soma(a, b):
    return a + b

resultado = soma(5, 3)
print(resultado)  # Saída: 8
```

Aula 1: Características do Python

1. **Simples e Intuitiva**: Código fácil de ler e escrever.
2. **Interpretada**: O código é executado linha por linha, sem necessidade de compilação.
3. **Tipagem Dinâmica**: Não é necessário declarar o tipo das variáveis.
4. **Extensível**: Integra-se bem com outras linguagens como C e C++.
5. **Grande comunidade**: Milhões de desenvolvedores ao redor do mundo.

Internet

Software de Desenvolvimento

Existem no mercado diversos softwares de desenvolvimento de Web, ou também conhecidos como editores de HTML. Porém estas ferramentas tem foco em desenvolvimento, alguns apenas para desenvolvimento em HTML, outros especialista em desenvolvimento de Blogs, etc.

Adobe Dreamweaver é um software de desenvolvimento para Web por suportar várias línguas de programação e conexão com banco de dados, dentre elas estão: HTML, ActionScript, CSS, XML, ASP, ColdFusion, JSP e PHP.

O melhor IDE abrangente para desenvolvedores .NET e C++ no Windows. Totalmente empacotado com uma bela matriz de ferramentas e recursos para elevar e aprimorar cada estágio de desenvolvimento de software.

Internet

Software de Desenvolvimento - online

Replit é a melhor maneira de programar e lançar projetos de verdade, aplicativos, jogos e muito mais diretamente do seu telefone. Com o Replit, você pode programar qualquer coisa, em qualquer lugar. Oferecemos suporte a centenas de linguagens de programação e frameworks com configuração zero.

Aula 1: Ferramentas e Ambientes de Desenvolvimento

IDEs Populares para Python:

1. **PyCharm**: IDE completa com ferramentas avançadas.
2. **Visual Studio Code**: Editor leve e altamente personalizável.
3. **Jupyter Notebook**: Ideal para ciência de dados e aprendizado de máquina.
4. **IDLE**: O editor padrão do Python, simples e fácil para iniciantes

Aula 1: Ferramentas e Ambientes de Desenvolvimento

Instalação do python → https://www.python.org/downloads/

Instalação do VSC → https://code.visualstudio.com/

Código do Visual Studio

Aula 1: Bibliotecas e Frameworks Populares

1. **Desenvolvimento Web**: Django, Flask
2. **Ciência de Dados**: Pandas, NumPy, Matplotlib
3. **Machine Learning**: Scikit-learn, TensorFlow, Keras
4. **Automação**: Selenium, OpenPyXL
5. **Desenvolvimento de Jogos**: Pygame

Aula 1: A Comunidade Python

- **Fóruns e Grupos**: Stack Overflow, Reddit (r/learnpython), Discord.
- **Eventos e Conferências**: PyCon, DjangoCon.
- **Repositórios de Código**: GitHub, PyPI.

Aula 1: Caminhos para Aprender Python

1. **Recursos Online**:
    1. Codecademy, Coursera, Udemy, YouTube.
2. **Livros**:
    1. "Python Crash Course" de Eric Matthes.
    2. "Automate the Boring Stuff with Python" de Al Sweigart.
3. **Prática**:
    1. Sites de desafios: HackerRank, LeetCode.
    2. Projetos pessoais para aplicar o aprendizado

Aula 1: Vantagens de Usar Python

- **Curva de aprendizado baixa**: Ótimo para iniciantes.
- **Versatilidade**: Usado em várias áreas da tecnologia.
- **Produtividade**: Desenvolvedores podem criar soluções rápidas e eficientes.
- **Grande demanda no mercado**: Habilidades em Python são valorizadas em diversas indústrias

Aula 1: Desafios e Limitações

- **Performance**: Python não é tão rápido quanto linguagens como C ou C++, especialmente em operações de baixo nível.
- **Multithreading**: Python tem algumas limitações com processamento paralelo (GIL - Global Interpreter Lock).

Aula 1: Demonstração ao Vivo (Opcional)

1. Escreva um código Python simples durante a apresentação.
2. Exemplo de calculadora ou script de automação

https://www.python.org/downloads/

Aula 1: Caminhos para Aprender Python

- Python é uma linguagem **poderosa**, **simples** e **versátil**.
- Pode ser usada por iniciantes e especialistas em diversas áreas.

Aula 1: Atalhos e comandos

Clt + K + C. → comenta

Clt + K + U → tira comentário

””” “”” -→ comenta múltiplas linhas

→ comenta única linha

pip --version → Verifica a versão do Python

python --version → Verifica a versão do Python

python → entra na pasta do python

Help (‘modules’) → lista os complementos

Ctrl + Z + enter → retorna a pasta do projeto

Principais comandos

Comando de entrada:

input()

Recebe dados do usuário (sempre como texto).

Comando de saída:

print()

Exibe informações na tela.

Aula 1: Operadores e Expressões

Uma variável é um container para um valor, como um número que podemos usar em uma operação de adição, ou uma sequência de texto que possamos usar como parte de uma oração. Mas uma coisa especial a respeito das variáveis é que seu conteúdo pode mudar.

Aula 1: Operadores e Expressões

Em programação, **atribuição** é o ato de **dar um valor a uma variável**.
É como colocar uma etiqueta em uma caixa para saber o que tem dentro, e depois poder trocar o conteúdo dessa caixa quando quiser.

A **variável** é o espaço na memória que guarda um valor.

O **operador de atribuição** mais comum é o sinal de igual 

Quando você faz x = 10, está dizendo: "Guarde o valor **10** dentro da variável **x**."

Aula 1: Operadores e Expressões

Exemplos de atribuições

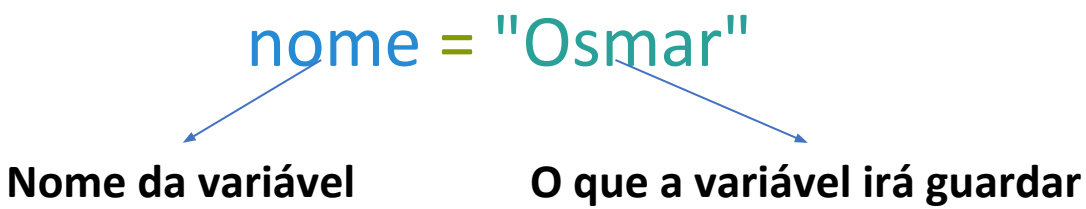
```
x = 10

x += 5    # x = 15
x -= 3    # x = 12
x *= 2    # x = 24
x /= 4    # x = 6.0
x %= 4    # x = 2.0
x **= 3   # x = 8.0
```

Atribuição **não é** a mesma coisa que igualdade matemática.

- Em matemática, x = 10 significa que x **é** igual a 10.
- Em programação, x = 10 significa **coloque** o valor 10 dentro de x

Aula 1: Operadores e Expressões

Exemplos de atribuições




```
nome = "Osmar"   # atribuição de string
idade = 35       # atribuição de número

idade = idade + 1  # atribuição usando o valor atual da variável

print(idade)  # agora exibe 36
```

Aula 1: Variáveis

Os **tipos de dados padrão** do Python são:

1. Inteiro (int);
   Exemplo: 1

2. Ponto Flutuante ou Decimal (float);
   Exemplo: 1.1

3. Tipo Complexo (complexo);
   Exemplo: 8j

4. String (str);
   Exemplo: hello

5. Boolean (bool);
   Exemplo: true / false

6. List (list);
   Exemplo: ['Mônica', 'Ana', 'Bruno', 'Alice']

7. Tuple;
   Exemplo: (90, 79, 54, 32, 21)

8. Dictionary (dic);
   Exemplo: {'Camila': 1.65, 'Larissa': 1.60, 'Guilherme': 1.70}

Chamando executável

PS C:\Users\d315561\Desktop\meu_python> python exerc_01.py

PS significa **PowerShell**

disco principal do Windows

pasta do seu usuário

chama o interpretador Python instalado no seu computador

nome do arquivo Python que você quer executar (um script)

Aula 1: Variáveis

```
idade = 20
ano = 2010

print(type(idade))
print(type(ano))
```

01) Tipo Inteiro (int)

É um tipo usado para um **número** que pode ser escrito **sem um componente decimal,** podendo ser **positivo ou negativo**.

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr
<type 'int'>
<type 'int'>
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 1: Variáveis

```
altura = 1.73
peso = 78.500

print(type(peso))
print(type(altura))
```

02) Ponto Flutuante ou Decimal (float)

É um tipo composto por **números decimais**. O **float** é usado para números racionais (que podem ser representados por uma fração), informalmente conhecidos como "números quebrados".

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/
<type 'float'>
<type 'float'>
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 1: Variáveis

```
a = 10+16j
b = 3+80j

print(type(a))
print(type(b))
```

03) Complexo (complex)

Tipo de dado usado para representar **números complexos**, normalmente em **cálculos geométricos e científicos.**

Um tipo complexo contém **duas partes**: a parte **real** e a parte **imaginária**, sendo que a imaginária contém um j no sufixo.

Com a função complex() podemos **converter** reais em números complexos. Ela traz dois argumentos, sendo o primeiro deles um número real, correspondente à parte real do número complexo, e o outro, um argumento opcional, que representa a parte imaginária. Por exemplo: z = complex(x,y).

```
iMac-de-Osmar:meu_python osmarzafalon$ /us
<type 'complex'>
<type 'complex'>
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 1: Variáveis

04) String (str)

É um **conjunto de caracteres** geralmente utilizados para representar palavras, frases ou textos.

Temos como exemplo as variáveis nome e profissão, com os dados Guilherme e Engenheiro de Software atribuídos a elas.

```
nome = "Osmar"
profissao = "Trabalhando com Python"

print(type(profissao ))
print(type(nome))
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /us
<type 'str'>
<type 'str'>
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 1: Variáveis

```
sexta_feira = True
feriado = False

print(type(sexta_feira)
print(type(feriado))
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /us
<type 'bool'>
<type 'bool'>
iMac-de-Osmar:meu_python osmarzafalon$
```

05) Boolean (bool)
Tipo de dado **lógico** que pode representar apenas **dois** valores: **falso ou verdadeiro** (False ou True, em Python).
Na lógica computacional, podem ser considerados como **0** ou **1**.
Como exemplo, temos as variáveis: sexta_feira e feriado, com os dados True e False atribuídos a elas.

Aula 1: Variáveis

06) Listas (list)

As listas agrupam um **conjunto de elementos variados,** podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se **colchetes** para delimitar a lista e **vírgulas** para separar os elementos. Já existe um artigo sobre listas em Python: operações básicas na plataforma, caso você queira se aprofundar no tema.

No código abaixo, por exemplo, vemos as variáveis alunos e notas, com os dados: 'Mônica', 'Ana', 'Bruno', 'Alice' e 10, 8.5, 7.8, 8.0 atribuídos a elas.

```
alunos = ["Monica", "Ana", "Osmar", "Alice"]
notas = [10, 8.5, 7.8, 8.0]

print(type(alunos))
print(type(notas))
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
<type 'list'>
<type 'list'>
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Operadores e Expressões

Operadores aritméticos:

Operador	Operação
+	Adição
-	Subtração
/	Divisão
//	Divisão descartando parte fracionária
*	Multiplicação
%	Resto
-variável	Negação
**	Potência

Concatenar em Python

Método	Exemplo	Saída	Observações
+ (soma de strings)	"Osmar" + " " + "Zafalon"	Osmar Zafalon	Simples, mas só funciona com strings.
F-strings *(Python 3.6+)*	f"{nome} tem {idade} anos"	Osmar tem 35 anos	Mais moderno e legível; aceita variáveis e expressões.
.format()	"{} tem {} anos".format(nome, idade)	Osmar tem 35 anos	Flexível, funciona em versões antigas do Python.
Concatenação com join()	" ".join(["Osmar", "Zafalon"])	Osmar Zafalon	Ideal para listas ou várias strings.
Com números (str())	"Idade: " + str(idade)	Idade: 35	Precisa converter número para string.
Repetição de string	"Oi! " * 3	Oi! Oi! Oi!	Multiplica strings.

Aula 2: operações matemáticas

```
a = 200
b = 4

soma = a + b
sub = a - b
mult = a * b
div = a / b

#linha de print
print("Soma: " + str(soma)) # Concatena o texto "Soma: " com a soma
print("Subtracao: " + str(sub)) # Concatena o texto "Subtracao: " com a subtracao
print("Multiplicacao: " + str(mult)) # Concatena o texto "Multiplicacao: " com a multiplicacao
print("Divisao: " + str(div)) # Concatena o texto "Divisao: " com a divisao
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Soma: 204
Subtracao: 196
Multiplicacao: 800
Divisao: 50
iMac-de-Osmar:meu_python osmarzafalon$
```

```
print("Soma: " + str(soma),"Subtracao: " + str(sub), "Multiplicacao: " + str(mult), "Divisao: " + str(div))

# Adicionando quebras de linha usando \n
print("Soma: " + str(soma) + "\n" +
"Subtração: " + str(sub) + "\n" +
"Multiplicação: " + str(mult) + "\n" +
"Divisão: " + str(div))
```

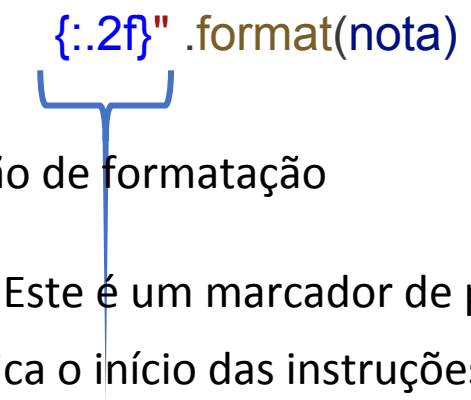
Aula 2: Porcentagem

```
# Definindo o funcionario, salario atual e a porcentagem de aumento
funcionario = "Osmar Zafalon"
salario_atual = 3000.00
aumento_porcentagem = 20 # 20%

# Calculo do aumento
aumento = (salario_atual / 100) * aumento_porcentagem # Aumenta o salario indireto

# Calculando o novo salario
novo_salario = salario_atual + aumento

# Exibindo os resultados
print("Funcionario:", str(funcionario))
print("Salario Atual: R$ {:.2f}".format(salario_atual))
print("Aumento 20%: R$ {:.2f}".format(aumento))
print("Novo Salario: R$ {:.2f}".format(novo_salario))
```

instrução de formatação

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
('Funcionario:', 'Osmar Zafalon')
Salario Atual: R$ 3000.00
Aumento 20%: R$ 600.00
Novo Salario: R$ 3600.00
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Formatação casa decimais



{:.2f}. Este é um marcador de posição usado dentro da string, onde o número será inserido.

: Indica o início das instruções de formatação.

.2f Significa "formato decimal com duas casas após a vírgula".

F Especifica que o valor deve ser formatado como um número de ponto flutuante (decimal).

2 Define que serão mostradas duas casas decimais após o ponto.

.format(salario_atual): O método format substitui o marcador **{:.2f}** pelo valor de salario_atual, formatado de acordo com a especificação (com duas casas decimais).

Aula 2: Calculo de media (simples)

```
# Definicao da variaveis (media)
nota1 = 6
nota2 = 8
nota3 = 9
nota4 = 5

#Calculando a media
media = (nota1 + nota2 + nota3 + nota4) / 4


# Exibindo os resultados
print ("Suas medias sao: {:.2f}" .format(nota1) +
" / {:.2f}" .format(nota2) +
" / {:.2f}" .format(nota3) +
" / {:.2f}" .format(nota4))
print("Sua nota Atual e: {:.2f}".format(media))
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/
Suas medias sao: 6.00 / 8.00 / 9.00 / 5.00
Sua nota Atual e: 7.00
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Função float( )

- Quando você recebe dados de uma fonte externa, como a entrada do usuário ou dados de arquivos, eles geralmente vêm como strings.
- O float() garante que você possa realizar cálculos matemáticos corretamente, convertendo essas strings em números decimais.

- Também é útil quando você quer realizar operações aritméticas que envolvem números com casas decimais, em vez de apenas números inteiros.

Se a string não puder ser convertida em um número válido (como tentar converter uma palavra para float), o Python lançará um erro ValueError.

Aula 2: Função input( )

É utilizada para **receber dados inseridos pelo usuário** durante a execução do programa.
Ela exibe uma mensagem (ou prompt) para o usuário, espera que ele insira um valor e retorna esse valor como uma **string** (sequência de caracteres).

```
variavel = input("Mensagem para o usuário: ")
```

"Mensagem para o usuário": O texto que aparece na tela pedindo que o usuário insira algum dado.
variavel: A variável que armazenará o valor digitado pelo usuário.

Aula 2: Função input( )

Exemplos

```
nome = input("Digite seu nome: ")
print("Ola, " + nome + "!")
```

```
  nome = input("Digite seu nome: ")
File "<string>", line 1
  Osmar Zafalon
```

```
idade = int(input("Digite sua idade: "))
print("Sua idade e:", idade)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Digite sua idade: 35
('Sua idade e:', 35)
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Calculo de media com float( ), input

```
# Funcao para calcular a mdia de quatro notas
def calcular_media(nota1, nota2, nota3, nota4):
# Calcula a media
media = (nota1 + nota2 + nota3 + nota4) / 4
return media

# Solicita as quatro notas ao usuario
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))
nota3 = float(input("Digite a terceira nota: "))
nota4 = float(input("Digite a quarta nota: "))

# Calcula a madia usando a funcao
media_final = calcular_media(nota1, nota2, nota3, nota4)

# Exibe o resultado usando .format()
print("A media das quatro notas e: {:.2f}".format(media_final))
```

instrução de formatação

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Digite a primeira nota: 6
Digite a segunda nota: 7
Digite a terceira nota: 9
Digite a quarta nota: 4
A media das quatro notas e: 6.50
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Atividade programada

Exercício: Calcular o valor total de uma compra

O objetivo deste exercício é criar um programa que:

1. Solicite o **preço unitário** de um produto.
2. Solicite a **quantidade** de produtos que o cliente deseja comprar.
3. Calcule e exiba o **valor total** da compra

Aula 2: Atividade programada

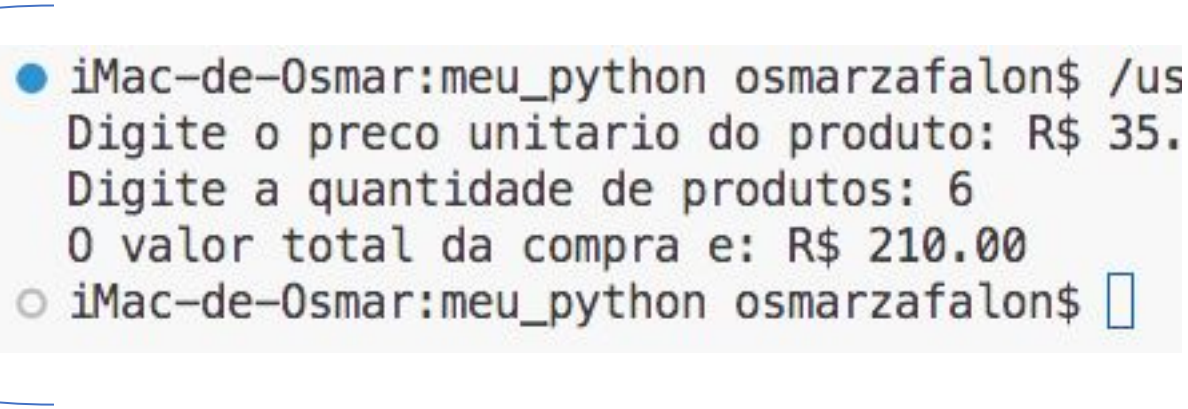
Exercício: Calcular o valor total de uma compra

```
# Solicita o preco unitario do produto
preco_unitario = float(input("Digite o preco unitario do produto: R$ "))

# Solicita a quantidade de produtos que o cliente deseja comprar
quantidade = int(input("Digite a quantidade de produtos: "))

# Calcula o valor total da compra
valor_total = preco_unitario * quantidade

# Exibe o valor total formatado com duas casas decimais
print("O valor total da compra e: R$ {:.2f}".format(valor_total))
```

Aula 2: Atividade programada

Exercício: Calcular o valor total de uma compra

Resultado



Aula 2: Operadores de comparação

Operador	Comparação
=	Igual a
<>	Diferente de
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a

Aula 2: Operadores de comparação

```
# Solicita que o usuario insira dois numeros
numero1 = float(input("Digite o primeiro numero: "))
numero2 = float(input("Digite o segundo numero: "))

# Comparaco10es usando operadores
print("Comparando os numeros:")
print(numero1, "==", numero2, ":", numero1 == numero2) # Igual
print(numero1, "!=", numero2, ":", numero1 != numero2) # Diferente
print(numero1, ">", numero2, ":", numero1 > numero2) # Maior que
print(numero1, "<", numero2, ":", numero1 < numero2) # Menor que
print(numero1, ">=", numero2, ":", numero1 >= numero2) # Maior ou igual a
print(numero1, "<=", numero2, ":", numero1 <= numero2) # Menor ou igual a
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /
Digite o primeiro numero: 20
Digite o segundo numero: 7
Comparando os numeros:
(20.0, '==', 7.0, ':', False)
(20.0, '!=', 7.0, ':', True)
(20.0, '>', 7.0, ':', True)
(20.0, '<', 7.0, ':', False)
(20.0, '>=', 7.0, ':', True)
(20.0, '<=', 7.0, ':', False)
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Operadores de comparação

Atividade: **Operadores de comparação:**
Enunciado

atividade simples que utiliza operadores de comparação em Python. A atividade é projetada para comparar a idade de duas pessoas e determinar quem é mais velho, quem tem a mesma idade, ou se uma pessoa é mais nova.

- Solicite a idade de duas pessoas.
- Compare as idades usando operadores de comparação (>,<, `==, etc.).
- Mostrar

Aula 2: Operadores de comparação

Atividade: **Operadores de comparação:**

```
# Solicita a idade de duas pessoas
idade_pessoa1 = int(input("Digite a idade da primeira pessoa: "))
idade_pessoa2 = int(input("Digite a idade da segunda pessoa: "))

# Operadores de comparacao
mais_velha = idade_pessoa1 > idade_pessoa2 # Pessoa 1 e mais velha que Pessoa 2
mesma_idade = idade_pessoa1 == idade_pessoa2 # Pessoa 1 tem a mesma idade que Pessoa 2
mais_nova = idade_pessoa1 < idade_pessoa2 # Pessoa 1 e mais nova que Pessoa 2

# Exibe os resultados das comparacaes
print("A primeira pessoa e mais velha que a segunda?", mais_velha)
print("As duas pessoas tem a mesma idade?", mesma_idade)
print("A primeira pessoa e mais nova que a segunda?", mais_nova)
```

resultado

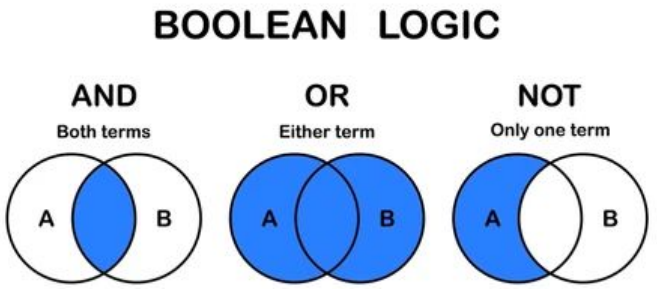
Aula 2: Operadores de comparação

Atividade: **Operadores de comparação:**

resultado

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/bin/python /
Digite a idade da primeira pessoa: 30
Digite a idade da segunda pessoa: 40
('A primeira pessoa e mais velha que a segunda?', False)
('As duas pessoas tem a mesma idade?', False)
('A primeira pessoa e mais nova que a segunda?', True)
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 2: Operadores e Expressões



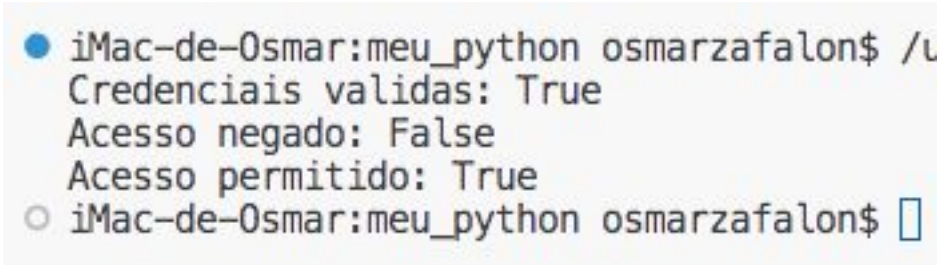
Operadores lógicos:

Lógica	Python
Não	not
E	and
Ou	or

Aula 2: Operadores e Expressões

```
# Definindo variaveis para as credenciais
usuario = "admin" # Nome de usuario
senha = "12345" # Senha do usuario

# Definindo as credenciais corretas
usuario_correto = "admin"
senha_correta = "12345"

# Verificando se as credenciais estao corretas
credenciais_validas = (usuario == usuario_correto) and (senha == senha_correta)

# Usando OR para verificar se o acesso deve ser negado
acesso_negado = (usuario != usuario_correto) or (senha != senha_correta)

# Usando NOT para verificar se o acesso e permitido
acesso_permitido = not acesso_negado

# Exibindo os resultados
print("Credenciais validas: {}".format(credenciais_validas)) # True se credenciais e corretas
print("Acesso negado: {}".format(acesso_negado)) # True se credenciais incorretas
print("Acesso permitido: {}".format(acesso_permitido)) # True se credenciais corretas
```

Operadores lógicos



string que contém um texto fixo e um **placeholder {** verdadeiro/ flaso**}**

Aula 2: Operadores e Expressões

Atividade com Operadores and, ou or e not

Enunciado

1. Verifique se um usuário é maior de idade (idade >= 18) **e** se tem habilitação
2. Caso o usuário tenha pelo menos uma dessas condições (idade >= 18 **ou** tenha habil
3. Se o usuário **não** atender a nenhuma delas

Aula 2: Operadores e Expressões

```
# Solicita a idade do usuario
idade = int(input("Digite sua idade: "))

# Pergunta se o usuario tem habilitacao
tem_habilitacao = input("Voce possui habilitacao? (sim/nao): ").strip().lower() == "sim"

# Operadores logicos para definir as mensagens
pode_dirigir = idade >= 18 and tem_habilitacao
pode_participar_sorteio = idade >= 18 or tem_habilitacao
nao_pode_participar = not pode_participar_sorteio

# Exibe as mensagens
print("Pode dirigir:", pode_dirigir)
print("Pode participar do sorteio:", pode_participar_sorteio)
print("Nao pode participar:", nao_pode_participar)
```

Esse comando garante que **qualquer variação de "Sim" digitada pelo usuário** (maiúsculas, minúsculas, com espaços, etc.) seja interpretada como "sim"

resultado

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/b
Digite sua idade: 21
Voce possui habilitacao? (sim/nao): "sim"
('Pode dirigir:', True)
('Pode participar do sorteio:', True)
('Nao pode participar:', False)
iMac-de-Osmar:meu_python osmarzafalon$
```

Remove espaços em branco extras no **início** e no **fim** da string.

Converte toda a string para **letras minúsculas**.

Aula – **bibliotecas**

Coleção de Módulos: Uma biblioteca pode conter um ou mais **módulos**, que são arquivos Python com extensão .py que definem funções, classes e variáveis.

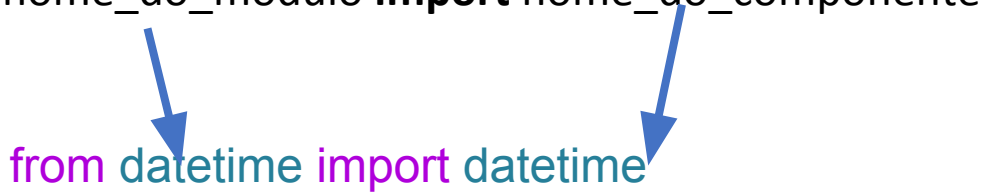
Funcionalidade Pronta: Elas fornecem soluções já testadas e documentadas para tarefas específicas, como manipulação de dados, cálculos matemáticos, desenvolvimento web, etc.

Aula – **import / from**

import em Python é usado para **importar módulos inteiros** no seu código, permitindo que você acesse todas as funções, classes e variáveis definidas naquele módulo.

from em Python é usado para **importar partes específicas de um módulo** ou pacote, em vez de importar o módulo inteiro. Ele permite que você traga apenas os componentes que você precisa, o que pode simplificar o código e melhorar a legibilidade.

from nome_do_modulo **import** nome_do_componente



Aula 3 - **datetime**

Link para estudo data e hora

https://www.w3schools.com/python/python_datetime.asp

Aula 3 - **datetime**

O módulo **datetime** é parte da biblioteca padrão do Python e oferece várias ferramentas para trabalhar com datas e horas.

data e hora combinadas (ou seja, tanto a data quanto a hora juntas).

Aula 3

```
from datetime import datetime

current_dateTime = datetime.now()

print(current_dateTime)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/
py
2024-10-18 11:01:06.459370
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 3

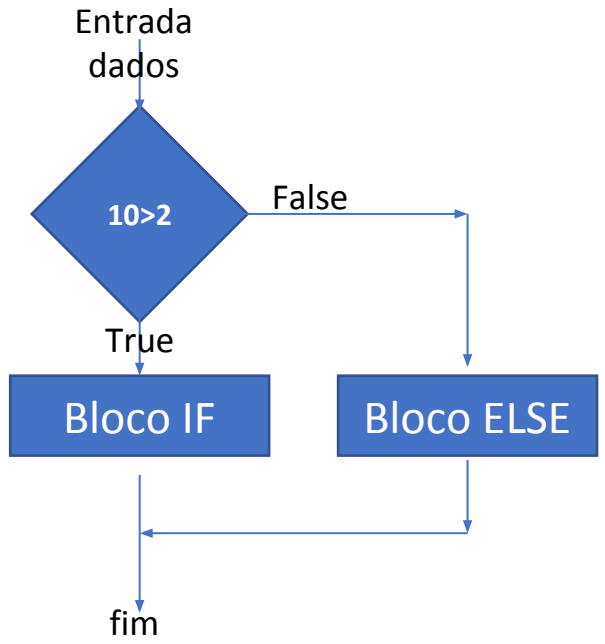
Conteúdo:

- Estrutura básica do if, else,elif.
- Condição
- Uso

Aula 3: Estrutura do if, else, elif.

Estrutura condicional

if: Avalia uma condição. Se a condição for verdadeira (True), o bloco de código dentro do if é executado.
elif: Significa "else if" e permite testar outras condições caso a primeira if seja falsa. Você pode ter quantos elif forem necessários.
else: Se todas as condições anteriores forem falsas, o código dentro do else será executado. O else é opcional.



Aula 3: Estrutura do if, else, elif.

Estrutura condicional

```
# Solicitando uma nota do usuario
nota = float(input("Digite a sua nota: "))

# Estrutura condicional
if nota >= 7.0:
        print("Aprovado")
elif nota >= 5.0:
        print("Recuperacao")
else:
        print("Reprovado")
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /us
Digite a sua nota: 8
Aprovado
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 3: Estrutura do if, else, elif.

Estrutura condicional

```
idade = int(input("Digite sua idade: "))

if idade < 12:
        print("Crianca")
elif idade < 18:
        print("Adolescente")
elif idade < 60:
        print("Adulto")
else:
        print("Idoso")
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Digite sua idade: 25
Adulto
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 4 - **Estruturas de Repetição**

Conteúdo:

- Loop for: iterar sobre listas e intervalos.
- Loop while: execute até uma condição ser falsa.
- Controle de fluxo: break, continue.

Aula 3: Estrutura do if, else, elif.

O **while True** garante que o programa continuará pedindo ao usuário um número até que ele insira um valor correto.

O **try** tenta converter a entrada em um número inteiro.
Se a conversão falhar (por exemplo, se o usuário digitar texto em vez de números),

o **except** captura o erro e exibe uma mensagem de erro, mas o programa continua rodando até que uma entrada válida seja recebida.
O loop é interrompido apenas quando o usuário insere um número válido (usando o break).

Aula 3: Estrutura do if, else, elif.

- O while True: mantém o programa perguntando sem parar.
- O try: tenta executar o código.
- Se houver erro, o except trata e o loop continua.
- Se der certo, usamos break para sair do loop.

Aula 3: Estrutura do if, else, elif.

```
while True:
    try:
        idade = int(input("Digite sua idade: ")) # Tenta converter a entrada para inteiro
        if 0 <= idade <= 120: # Verifica se a idade esta no intervalo valido
            print("Idade valida: {} anos".format(idade))
            break
        else:
            print("Idade invalida. Insira uma idade entre 0 e 120.")
    except ValueError:
        print("Entrada invalida. Por favor, insira um numero inteiro.")
```

exerc_039.py > ...

```
while True:
    try:
        idade = int(input("Digite sua idade: "))  # Tenta converter a
        if 0 <= idade <= 120:  # Verifica se a idade está no interval
            print("Idade válida: {} anos".format(idade))
            break  # sai do while se a idade for válida
        else:
            print("Idade inválida. Insira uma idade entre 0 e 120.")
    except ValueError:
        print("Entrada inválida. Por favor, insira um número inteiro.
```

```
Digite sua idade: /usr/bin/python /Users/o
Entrada invalida. Por favor, insira um num
Digite sua idade: 30
Idade valida: 30 anos
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 4 - **Estruturas de Repetição**

O comando **for** em Python é uma estrutura de controle usada para **repetir** um bloco de código um determinado número de vezes ou **iterar** sobre uma sequência (como listas, tuplas, dicionários, conjuntos ou strings).
Ele é útil quando você sabe antecipadamente quantas vezes deseja executar um bloco de código ou quando você deseja iterar sobre todos os elementos de uma sequência.

```
nomes = ["Angelina", "Marcia", "Osmar"]

for nome in nomes:
print(nome)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Angelina
Marcia
Osmar
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 4 - **Estruturas de Repetição**

comando **while**

O comando **while** em Python é uma estrutura de controle que permite a execução repetitiva de um bloco de código enquanto uma condição especificada for verdadeira. É uma forma de criar loops que continuam a executar até que uma condição seja atendida.

```
contador = 0

while contador < 10:
        contador += 1
        if contador % 2 == 0:
                continue # Pula numeros pares
        print(contador)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
1
3
5
7
9
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 4 - **Estruturas de Repetição**

comando **break**

O comando **break** em Python é usado para sair de um loop imediatamente, interrompendo a execução do bloco de código dentro do loop, mesmo que a condição do loop ainda seja verdadeira.
Ele é comumente utilizado em loops for e while quando você quer interromper o ciclo antes que a condição do loop seja completamente atendida.

Aula 4 - **Estruturas de Repetição**

```python
#----- Loop for com range ----------------------------------------------------
for i in range(1, 5):
    print(i)
#------ Loop while -----------------------------------------------------------
contador = 0
while contador < 5:
    print("Contador: {}".format(contador)) # Usando format() no lugar de f-string
    contador += 1
#------- Controle de fluxo: break e continue-----------------------------------
for i in range(5):
    if i == 10:
        break # Sai do loop quando i for 5
    if i % 2 == 0:
        continue # Pula os numeros pares
    print(i)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
1
2
3
4
Contador: 0
Contador: 1
Contador: 2
Contador: 3
Contador: 4
1
3
iMac-de-Osmar:meu_python osmarzafalon$
```

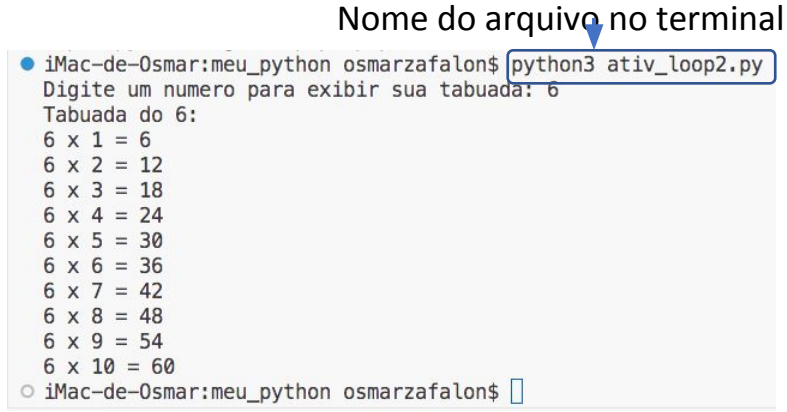
Aula 4 - **Estruturas de Repetição**

Atividade:

Crie um programa que exiba uma tabela de um número.

```python
# Função para exibir a tabuada de um número
def exibir_tabuada(numero):
    print(f"Tabuada do {numero}:")
    for i in range(1, 11):  # Exibe a multiplicação de 1 a 10
        resultado = numero * i
        print(f"{numero} x {i} = {resultado}")

# Solicita o número do usuário
numero = int(input("Digite um número para exibir sua tabuada: " ))

# Exibe a tabuada
exibir_tabuada(numero)
```

Nome do arquivo no terminal



Aula 4 - **Estruturas de Repetição**

Crie um programa que calcula a soma dos números em um intervalo.

Atividade:

```python
# Função para calcular a soma dos números em um intervalo
def soma_intervalo(inicio, fim):
    soma = 0
    for num in range(inicio, fim + 1): # +1 para incluir o número final
        soma += num
    return soma

# Solicita os valores do intervalo ao usuário
inicio = int(input("Digite o valor inicial do intervalo: " ))
fim = int(input("Digite o valor final do intervalo: " ))
# Calcula a soma usando a função

resultado = soma_intervalo(inicio, fim)

# Exibe o resultado
print("A soma dos números de  {} até {} é: {}".format(inicio, fim, resultado))
```

Soma o intervalo

1 + 2 + 3 + 4 + 5 = 15

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr
Digite o valor inicial do intervalo: 2
Digite o valor final do intervalo: 11
A soma dos numeros de 2 ate 11 e: 65
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 5 - **Funções em Python**

Conteúdo:

1. Definição de funções com def.
2. Parâmetros e argumentos.
3. Valores de retorno com return.

Aula 5 - **Funções em Python**

Vantagens de Usar Funções

1. **Reutilização**: Funções permitem reutilizar o mesmo código várias vezes sem a necessidade de duplicação.
2. **Modularidade**: O código se torna mais modular e organizado, facilitando sua manutenção e depuração.
3. **Abstração**: Ao usar funções, você pode criar blocos de código reutilizáveis, abstraindo a lógica interna e simplificando o restante do código.

Aula 5 - **Funções em Python**

Função **def**

No Python, a definição de funções é feita com a *palavra-chave* **def**.
Uma função é um bloco de código reutilizável que pode ser chamado para executar uma tarefa específica.

def: A palavra-chave usada para definir uma função.
nome_da_funcao: O nome da função, que deve ser significativo e seguir as regras de nomenclatura do Python.

parametros: (Opcional) Os valores que a função pode receber quando chamada.
return: (Opcional) Usado para retornar um valor quando a função é chamada.

Aula 5 - **Funções em Python**

Função **def**

```
# Definindo uma função
def nome_da_funcao(parametro1, parametro2):
    # Bloco de código que executa a tarefa
    resultado = parametro1 + parametro2

    # Retorna o valor
    return resultado


#Exemplo
soma = nome_da_funcao(8, 7)
print(soma)  # Saída: 15
```

Aula 5 - **Funções em Python**

Função **def**

```
# Definindo uma função que saúda o usuário
def saudacao(nome):
    print("Olá, {}! Seja bem-vindo!".format(nome))

# Chamando a função
saudacao("Osmar")
```

```
iMac-de-Osmar:meu_python osmarzafalon$ python3 funcao_def.py
Olá, Osmar! Seja bem-vindo!
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 5 - **Funções em Python**

Função **def**

Funções com Parâmetros Padrão (Default)

Podemos definir valores padrão para os parâmetros de uma função.
Se nenhum valor for passado para esses parâmetros ao chamar a função, o valor padrão será utilizado.

Aula 5 - **Funções em Python**

Função **def**

Funções com Parâmetros Padrão (Default)

Se não passar nenhum argumento para a função cumprimentar, ela usará "amigo" como valor padrão para o parâmetro nome.

```
def cumprimentar(nome="amigo"):
    print("Olá, {}!".format(nome))

# Chamadas da função
cumprimentar()      # Usa o valor padrão "amigo"
cumprimentar("Osmar")  # Usa o valor fornecido "Osmar"
```

```
iMac-de-Osmar:meu_python osmarzafalon$
def_3.py
Olá, amigo!
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 5 - **Funções em Python**

Função **def**

Funções com Vários Argumentos

Podemos querer definir uma função que aceite uma quantidade **indeterminada de argumentos**.

- Isso pode ser feito com o operador
:

- ***args** (para argumentos posicionais)
- ****kwargs** (para argumentos nomeados)

O operador * na frente de um nome de parâmetro, como *args, permite que uma função aceite uma quantidade **variável de argumentos posicionais**. Esses argumentos são passados para a função como uma **tupla**.

O operador ** na frente de um nome de parâmetro, como **kwargs, permite que uma função aceite uma quantidade **variável de argumentos nomeados** (ou seja, argumentos passados na forma de pares chave-valor). Esses argumentos são passados para a função como um **dicionário**.

Aula 5 - **Funções em Python**

Função **def**

Funções com Vários Argumentos - *args

```
iMac-de-Osmar:meu_python osmarzafalon$
def_args.py
A soma total é: 10
iMac-de-Osmar:meu_python osmarzafalon$
```

```
def somar_todos(*numeros):
    total = sum(numeros)
    return total

resultado = somar_todos(1, 2, 3, 4)
print("A soma total é:", resultado)
```

O **numeros** permite que a função aceite uma quantidade arbitrária de números. Dentro da função, **numeros** será uma **tupla** contendo todos os argumentos passados.

Aula 5 - **Funções em Python**

Função **def**

Funções com Argumentos Nomeados (kwargs)**

Pode usar ****kwargs** para passar argumentos nomeados (como pares de chave-valor) para uma função

Aula 5 - **Funções em Python**

Função **def**

Funções com Argumentos Nomeados (kwargs)**

```
def exibir_informacoes(**info):
    for chave, valor in info.items():
        print(f"{chave}: {valor}")

# Chamando a função (fora do for)
exibir_informacoes(nome="Osmar", idade=30, cidade="São
Paulo")
```

```
iMac-de-Osmar:meu_python osmarzafalon$
ef_kwargs.py
nome: Osmar
idade: 30
cidade: São Paulo
iMac-de-Osmar:meu_python osmarzafalon$
```

****info** coleta os argumentos nomeados e os armazena como um dicionário.
.items() este método permite iterar pelas chaves e valores.

Aula 5 - **Funções em Python**

Função **def**

```
def calcular_total(preco, quantidade):
    total = preco * quantidade
    return total

# Chamando a função com argumentos
preco_unitario = 25.50
quantidade_comprada = 4
total_compra = calcular_total(preco_unitario, quantidade_comprada)

# Exibindo o resultado
print("O total da compra é: R$ {:.2f}".format(total_compra))
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr
O total da compra e: R$ 102.00
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 5 - **Funções em Python**

Atividade - Função **def**

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
Digite seu peso (kg): 73
Digite sua altura (m): 1.70
Seu IMC e: 25.26
Classificacao: Sobrepeso
iMac-de-Osmar:meu_python osmarzafalon$
```

```python
# Definindo a função que calcula o IMC
def calcular_imc(peso, altura):
    imc = peso / (altura ** 2)
    return imc

# Solicitando os dados do usuário
peso = float(input("Digite seu peso (kg): " ))
altura = float(input("Digite sua altura (m): " ))

# Calculando o IMC usando a função
imc = calcular_imc(peso, altura)

# Exibindo o resultado
print("Seu IMC é: {:.2f}".format(imc))

# Determinando a categoria do IMC
if imc < 18.5:
    print("Classificação: Abaixo do peso" )
elif 18.5 <= imc < 24.9:
    print("Classificação: Peso normal" )
elif 25 <= imc < 29.9:
    print("Classificação: Sobrepeso" )
else:
    print("Classificação: Obesidade" )
```

Aula 6 - **Listas e Tuplas**

Conteúdo:

1. Definir e acessar elementos de listas.
2. Métodos de lista: append(), remove(), insert(), pop().
3. Tuplas: definição e imutabilidade.

Aula 6 - **Listas e Tuplas**

São dois tipos de estruturas de dados amplamente usadas em Python para armazenar coleções de elementos.

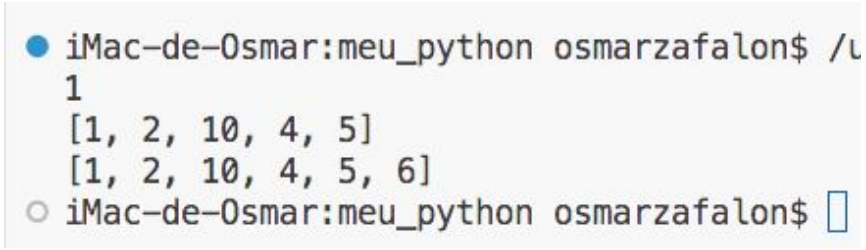
Listas

- As listas são mutáveis, ou seja, você pode modificar seus elementos (adicionar, remover ou alterar).
- São criadas usando colchetes [ ].
- Podem armazenar diferentes tipos de dados (inteiros, strings, etc.).
- Permitem várias operações, como adicionar e remover elementos.

Aula 6 - **Listas e Tuplas**

Exemplo de Listas

```
# Criando uma lista de numeros
numeros = [1, 2, 3, 4, 5]

# Acessando um elemento
print(numeros[0]) # Saida: 1

# Modificando um elemento
numeros[2] = 10
print(numeros) # Saida: [1, 2, 10, 4, 5]

# Adicionando um elemento
numeros.append(6)
print(numeros) # Saida: [1, 2, 10, 4, 5, 6]
```



Aula 6 - **Listas e Tuplas**

tuplas

São imutáveis, ou seja, uma vez criadas, seus elementos não podem ser alterados.
São criadas usando parênteses ( ).
Também podem armazenar diferentes tipos de dados.
Úteis para armazenar dados que não devem ser modificados.

Aula 6 - **Listas e Tuplas**

Exemplo de Tuplas

```
# Criando uma tupla de cores
cores = ("vermelho", "azul", "verde")

# Acessando um elemento
print(cores[1]) # Saida: azul

# Nao e possivel modificar os elementos de uma tupla
# cores[1] = "amarelo" # Isso causaria um erro!
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
azul
iMac-de-Osmar:meu_python osmarzafalon$
```

Obs. **Lista**: Mutável (pode ser modificada).
Tupla: Imutável (não pode ser modificada).

Aula 6 - **Listas e Tuplas**

Tuplas

Não é possível adicionar novos elementos.
Não é possível remover elementos.
Não é possível modificar os valores dos elementos existentes.

```
# Criando uma tupla
tupla = (1, 2, 3)

# Tentando modificar um elemento
tupla[0] = 10 # Isso causará um erro!
```

Aula 6 - **Listas e Tuplas**

Tuplas

Pode fazer:

- **Acessar** elementos (como com listas).
- **Iterar** sobre seus elementos usando loops.
- **Concatenar** duas tuplas para formar uma nova tupla.
- Verificar se um valor está presente na tupla (usando in).

```
# Criando duas tuplas
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)

# Concatenando tuplas (gera uma nova tupla)
tupla3 = tupla1 + tupla2
print(tupla3) # Saida: (1, 2, 3, 4, 5, 6)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /u
(1, 2, 3, 4, 5, 6)
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 6 - **Listas e Tuplas**

Exercício

Gerenciar uma lista de alunos e suas notas

O comando **len**(notas) retorna o **número de elementos** presentes no objeto notas. No seu código, notas é uma **tupla** que contém três valores (as três notas de cada aluno), então len(notas) vai retornar 3, já que há três notas armazenadas

```python
# Inicializa as listas para armazenar os alunos e as notas
alunos = []
notas_alunos = []

# Função para calcular a média de uma tupla de notas
def calcular_media(notas):
    return sum(notas) / len(notas)

# Solicita o nome e as notas de 5 alunos
for i in range(5):
    nome = input(f"Digite o nome do {i+1}º aluno: ")
    alunos.append(nome)  # Adiciona o nome na lista de alunos

    # Solicita as 3 notas e armazena em uma tupla
    nota1 = float(input(f"Digite a 1ª nota de {nome}: "))
    nota2 = float(input(f"Digite a 2ª nota de {nome}: "))
    nota3 = float(input(f"Digite a 3ª nota de {nome}: "))

    notas = (nota1, nota2, nota3)  # Cria uma tupla com as notas
    notas_alunos.append(notas)  # Armazena a tupla na lista de notas

# Exibe o resultado para cada aluno
print("\nResultados:")
for i in range(len(alunos)):  # Usa o tamanho da lista 'alunos' para garantir índice válido
    media = calcular_media(notas_alunos[i])
    situacao = "Aprovado" if media >= 6.0 else "Reprovado"
    print(f"Aluno: {alunos[i]} | Notas: {notas_alunos[i]} | Média: {media:.2f} | Situação: {situacao}")
```

Aula 6 - **Listas e Tuplas**

Exercício

Gerenciar uma lista de alunos e suas notas

notas = (8.0, 7.5, 9.0)
print(len(notas)) # Saída: 3

listas_tuplas2.py > ...

```python
# Inicializa as listas para armazenar os alunos e as notas
alunos = []
notas_alunos = []

# Função para calcular a média de uma tupla de notas
def calcular_media(notas):
    return sum(notas) / len(notas)

# Solicita o nome e as notas de 5 alunos
for i in range(5):
    nome = input(f"Digite o nome do {i+1}º aluno: ")
    alunos.append(nome)  # Adiciona o nome na lista de alunos

    # Solicita as 3 notas e armazena em uma tupla
    nota1 = float(input(f"Digite a 1ª nota de {nome}: "))
    nota2 = float(input(f"Digite a 2ª nota de {nome}: "))
    nota3 = float(input(f"Digite a 3ª nota de {nome}: "))
    notas = (nota1, nota2, nota3)  # Cria uma tupla com as notas
    notas_alunos.append(notas)  # Armazena a tupla na lista de notas

# Exibe o resultado para cada aluno
print("\nResultados:")
for i in range(len(alunos)):  # Use o tamanho da lista 'alunos' para garantir que o índice seja válido
    media = calcular_media(notas_alunos[i])
    situacao = "Aprovado" if media >= 6.0 else "Reprovado"
    print(f"Aluno: {alunos[i]} | Notas: {notas_alunos[i]} | Média: {media:.2f} | Situação: {situacao}")

```


```python
# Inicializa as listas para armazenar os alunos e as notas
alunos = []
notas_alunos = []

# Função para calcular a média de uma tupla de notas
def calcular_media(notas):
    return sum(notas) / len(notas)

# Solicita o nome e as notas de 5 alunos
for i in range(5):
    nome = input(f"Digite o nome do {i+1}º aluno: ")
    alunos.append(nome)  # Adiciona o nome na lista de alunos

    # Solicita as 3 notas e armazena em uma tupla
    nota1 = float(input(f"Digite a 1ª nota de {nome}: "))
    nota2 = float(input(f"Digite a 2ª nota de {nome}: "))
    nota3 = float(input(f"Digite a 3ª nota de {nome}: "))

    notas = (nota1, nota2, nota3)  # Cria uma tupla com as notas
    notas_alunos.append(notas)    # Armazena a tupla na lista de notas

# Exibe o resultado para cada aluno
print("\nResultados:")
for i in range(len(alunos)):  # Usa o tamanho da lista 'alunos' para garantir que o índice seja válido
    media = calcular_media(notas_alunos[i])
    situacao = "Aprovado" if media >= 6.0 else "Reprovado"
    print(f"Aluno: {alunos[i]} | Notas: {notas_alunos[i]} | Média: {media:.2f} | Situação: {situacao}")
```

Aula 6 - **Listas e Tuplas**

Exercício

Resultado

```
iMac-de-Osmar:meu_python osmarzafalon$ python3 ativ_lista_tupla.py
Digite o nome do 1º aluno: Osmar
Digite a 1ª nota de Osmar: 6
Digite a 2ª nota de Osmar: 8
Digite a 3ª nota de Osmar: 7
Digite o nome do 2º aluno: Zafalon
Digite a 1ª nota de Zafalon: 3
Digite a 2ª nota de Zafalon: 6
Digite a 3ª nota de Zafalon: 5
Digite o nome do 3º aluno: Angelina
Digite a 1ª nota de Angelina: 6
Digite a 2ª nota de Angelina: 9
Digite a 3ª nota de Angelina: 10
Digite o nome do 4º aluno: Joaquina
Digite a 1ª nota de Joaquina: 8
Digite a 2ª nota de Joaquina: 6
Digite a 3ª nota de Joaquina: 3
Digite o nome do 5º aluno: Paula
Digite a 1ª nota de Paula: 5
Digite a 2ª nota de Paula: 7
Digite a 3ª nota de Paula: 9

Resultados:
Aluno: Osmar | Notas: (6.0, 8.0, 7.0) | Média: 7.00 | Situação: Aprovado
Aluno: Zafalon | Notas: (3.0, 6.0, 5.0) | Média: 4.67 | Situação: Reprovado
Aluno: Angelina | Notas: (6.0, 9.0, 10.0) | Média: 8.33 | Situação: Aprovado
Aluno: Joaquina | Notas: (8.0, 6.0, 3.0) | Média: 5.67 | Situação: Reprovado
Aluno: Paula | Notas: (5.0, 7.0, 9.0) | Média: 7.00 | Situação: Aprovado
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 6 - **Listas e Tuplas**

Exercício

Gerenciar uma lista de alunos e suas notas

Outra situação

```
Digite o nome do 1º aluno: Osmar
Digite a 1ª nota de Osmar: 8
Digite a 2ª nota de Osmar: 8
Digite a 3ª nota de Osmar: 8
Digite o nome do 2º aluno: Claudia
Digite a 1ª nota de Claudia: 9
Digite a 2ª nota de Claudia: 9
Digite a 3ª nota de Claudia: 9
Digite o nome do 3º aluno: Cris
Digite a 1ª nota de Cris: 8
Digite a 2ª nota de Cris: 9
Digite a 3ª nota de Cris: 8
Digite o nome do 4º aluno: Kelly
Digite a 1ª nota de Kelly: 6
Digite a 2ª nota de Kelly: 7
Digite a 3ª nota de Kelly: 6
Digite o nome do 5º aluno: 7
Digite a 1ª nota de 7: Marcos
Por favor, insira um número válido para as notas.
Digite a 1ª nota de 7: 6
Digite a 2ª nota de 7: 9
Digite a 3ª nota de 7: 8

Resultados:
Aluno: Osmar | Notas: (8.0, 8.0, 8.0) | Média: 8.00 | Situação: Aprovado
Aluno: Claudia | Notas: (9.0, 9.0, 9.0) | Média: 9.00 | Situação: Aprovado
Aluno: Cris | Notas: (8.0, 9.0, 8.0) | Média: 8.33 | Situação: Aprovado
Aluno: Kelly | Notas: (6.0, 7.0, 6.0) | Média: 6.33 | Situação: Aprovado
Aluno: 7 | Notas: (6.0, 9.0, 8.0) | Média: 7.67 | Situação: Aprovado
iMac-de-Osmar:meu_python osmarzafalon$
```

```python
# Inicializa as listas para armazenar os alunos e as notas
alunos = []
notas_alunos = []

# Função para calcular a média de uma tupla de notas
def calcular_media(notas):
    return sum(notas) / len(notas)

# Solicita o nome e as notas de 5 alunos
for i in range(5):
    nome = input(f"Digite o nome do {i+1}º aluno: ")
    alunos.append(nome)  # Adiciona o nome na lista de alunos

    # Solicita as 3 notas e garante que são números válidos
    while True:
        try:
            nota1 = float(input(f"Digite a 1ª nota de {nome}: "))
            nota2 = float(input(f"Digite a 2ª nota de {nome}: "))
            nota3 = float(input(f"Digite a 3ª nota de {nome}: "))
            break  # Sai do loop se todas as notas forem válidas
        except ValueError:
            print("Por favor, insira um número válido para as notas.")

    # Cria uma tupla com as notas e armazena na lista de notas
    notas = (nota1, nota2, nota3)
    notas_alunos.append(notas)

# Exibe o resultado para cada aluno
print("\nResultados:")
for i in range(5):
    media = calcular_media(notas_alunos[i])
    situacao = "Aprovado" if media >= 6.0 else "Reprovado"
    print(f"Aluno: {alunos[i]} | Notas: {notas_alunos[i]} | Média: {media:.2f} | Situação: {situacao}")
```

Usando Dicionario

```python
# Função para calcular a média
def calcular_media(notas):
    return sum(notas) / len(notas)

# Lista de alunos (cada aluno será um dicionário)
alunos = []

# Solicita os dados de 5 alunos
for i in range(5):
    nome = input(f"Digite o nome do {i+1}º aluno: ")

    notas = []
    for j in range(3):
        nota = float(input(f"Digite a {j+1}ª nota de {nome}: "))
        notas.append(nota)

    aluno = {
        "nome": nome,
        "notas": notas,
        "media": calcular_media(notas),
    }

    # Define a situação do aluno
    aluno["situacao"] = "Aprovado" if aluno["media"] >= 6.0 else "Reprovado"

    alunos.append(aluno)

# Exibe o resultado
print("\nResultados:")
for aluno in alunos:
    print(f"Aluno: {aluno['nome']} | Notas: {aluno['notas']} | "
          f"Média: {aluno['media']:.2f} | Situação: {aluno['situacao']}")
```

Aula 6 - **Listas e Tuplas**

Exercício: Gerenciar uma lista de produtos e seus preços

O objetivo deste exercício é criar um programa que:

- Solicite o nome e o preço de **cinco produtos**.
- Armazene os nomes dos produtos em uma **lista**.
- Armazene os preços dos produtos em uma **tupla**.
- Exiba uma tabela com o nome do produto e o preço de cada um.
- Calcule e exiba o preço total dos produtos.

Aula 6 - **Listas e Tuplas**

Exercício:

Gerenciar uma lista de produtos e seus preços

```python
# Inicializa a lista para armazenar os produtos e os preços
produtos = []
precos = []

# Solicita o nome e o preço de 5 produtos
for i in range(5):
    produto = input(f"Digite o nome do {i+1}º produto: ")
    produtos.append(produto)  # Adiciona o nome do produto na lista

    preco = float(input(f"Digite o preço de {produto}: R$ "))
    precos.append(preco)  # Adiciona o preço na lista de preços

# Exibe uma tabela com os produtos e seus preços
print("\nTabela de Produtos:")
for i in range(5):
    print(f"{produtos[i]}: R$ {precos[i]:.2f}")

# Calcula o preço total dos produtos
preco_total = sum(precos)

# Exibe o preço total
print(f"\nPreço Total: R$ {preco_total:.2f}")
```

Aula 6 - **Listas e Tuplas**

Exercício:

Gerenciar uma lista de produtos e seus preços

parar quando digitar "fim":

```python
# Inicializa a lista para armazenar os produtos e os preços
produtos = []
precos = []

print("=== Cadastro de Produtos ===")
print("Digite 'fim' para encerrar.\n")

# Solicita o nome e o preço dos produtos até o usuário digitar 'fim'
while True:
    produto = input("Digite o nome do produto: ")
    if produto.strip().lower() == "fim":
        break  # Sai do loop quando o usuário digita 'fim'

    produtos.append(produto)  # Adiciona o nome do produto na lista

    preco = float(input(f"Digite o preço de {produto}: R$ "))
    precos.append(preco)  # Adiciona o preço na lista de preços

# Exibe uma tabela com os produtos e seus preços
print("\n=== Tabela de Produtos ===")
for i in range(len(produtos)):
    print(f"{produtos[i]}: R$ {precos[i]:.2f}")

# Calcula o preço total dos produtos
preco_total = sum(precos)

# Exibe o preço total
print(f"\nPreço Total: R$ {preco_total:.2f}")
```

Aula 6 - **Listas e Tuplas**

Exercício:

Gerenciar uma lista de produtos e seus preços

Resultado

```
iMac-de-Osmar:meu_python osmarzafalon$ python3 ativ_lista_tupla2.py
Digite o nome do 1º produto: Arroz
Digite o preço de Arroz: R$ 43.00
Digite o nome do 2º produto: Feijão
Digite o preço de Feijão: R$ 21.00
Digite o nome do 3º produto: Óleo
Digite o preço de Óleo: R$ 12.00
Digite o nome do 4º produto: Macarrão
Digite o preço de Macarrão: R$ 6.80
Digite o nome do 5º produto: Açúcar
Digite o preço de Açúcar: R$ 5.40

Tabela de Produtos:
Arroz: R$ 43.00
Feijão: R$ 21.00
Óleo: R$ 12.00
Macarrão: R$ 6.80
Açúcar: R$ 5.40

Preco Total: R$ 88.20
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 7 - **Dicionários e Conjuntos**

Conteúdo:

1. Definir e acessar dicionários.
2. Métodos de dicionários: keys(), values(), items().
3. Conjuntos: características e métodos

Aula 7 - **Dicionários e Conjuntos**

Definir e Acessar Dicionários

Os dicionários em Python são coleções não ordenadas de **pares chave-valor**, onde cada chave é única. Eles são definidos usando **{} (chaves)** e podem ser acessados através das chaves.

Aula 7 - **Dicionários e Conjuntos**

Definir e Acessar Dicionários

Definindo um dicionário

```
# Definindo um dicionário com informações de um aluno
aluno = {
"nome": "Osmar",
"idade": 22,
"curso": "Python",
"notas": [8.5, 7.3, 9.0]
}
```

```
iMac-de-Osmar:meu_python osmarzafalon$
nario.py
Nome: Osmar
Idade: 22
Curso: Python
Notas: [8.5, 7.3, 9.0]
iMac-de-Osmar:meu_python osmarzafalon$
```

Acessando informações do dicionário

```
# Acessando informações do dicionário
print("Nome:", aluno["nome"]) # Saída: Nome: Carlos
print("Idade:", aluno["idade"]) # Saída: Idade: 22
print("Curso:", aluno["curso"]) # Saída: Curso: Engenharia

# Acessando a lista de notas
print("Notas:", aluno["notas"]) # Saída: Notas: [8.5, 7.3, 9.0]
```

Aula 7 - **Dicionários e Conjuntos**

Adicionando ou Modificando Valores:

```
# Definindo um dicionário com informações de um aluno
aluno = {
"nome": "Osmar",
"idade": 22,
"curso": "Python",
"notas": [8.5, 7.3, 9.0]
}

# Adicionando um novo campo 'cidade'
aluno["cidade"] = "São Paulo"

# Modificando o valor da chave 'curso'
aluno["curso"] = "JavaScrypt"

# Exibindo o dicionário atualizado
print(aluno)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/local/bin/python3 /Users/osmarzafalon/Desktop/meu_python/dicio
nario2.py
{'nome': 'Osmar', 'idade': 22, 'curso': 'JavaScrypt', 'notas': [8.5, 7.3, 9.0], 'cidade': 'São Paulo'}
iMac-de-Osmar:meu_python osmarzafalon$
```

Novo valor

Valor acrescentado

Aula 7 - **Dicionários e Conjuntos**

Verificando se uma Chave Existe: com operador “ in “

Podemos usar o operador in para verificar se uma chave está presente no dicionário.

```
# Definindo um dicionário com informações de um aluno
aluno = {
"nome": "Osmar",
"idade": 22,
"curso": "Python",
"notas": [8.5, 7.3, 9.0]
}

if "nome" in aluno:
print("A chave 'nome' existe no dicionário.")
```

```
iMac-de-Osmar:meu_python osmarzafalon$
nario2.py
A chave 'nome' existe no dicionário.
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 7 - **Dicionários e Conjuntos**

Iterando sobre um Dicionário:

Para percorrer as chaves e valores de um dicionário, podemos usar um laço **for.**

```
# Definindo um dicionário com informações de um aluno
aluno = {
"nome": "Osmar",
"idade": 22,
"curso": "Python",
"notas": [8.5, 7.3, 9.0]
}

# Iterando pelas chaves e valores
for chave, valor in aluno.items():
print(f"{chave}: {valor}")
```

```
iMac-de-Osmar:meu_python osmarzafalon$
nario2.py
nome: Osmar
idade: 22
curso: Python
notas: [8.5, 7.3, 9.0]
iMac-de-Osmar:meu_python osmarzafalon$
```

f-string é uma maneira prática e eficiente de **inserir valores de variáveis dentro de uma string**.

Aula 7 - **Dicionários e Conjuntos**

Métodos de dicionários: keys(), values(), items().

keys() retorna uma visão de todas as chaves presentes no dicionário.

```
dados = {"nome": "Osmar", "idade": 25, "cidade": "São Paulo"}

# Exibindo as chaves
print(dados.keys())
```

```
iMac-de-Osmar:meu_python osmarzafalon$
nario2.py
dict_keys(['nome', 'idade', 'cidade'])
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 7 - **Dicionários e Conjuntos**

Métodos de dicionários: keys(), values(), items().

values() retorna uma visão de todos os **valores** presentes no dicionário.

```
dados = {"nome": "Osmar", "idade": 25, "cidade": "São Paulo"}

# Exibindo os valores
print(dados.values())
```

```
iMac-de-Osmar:meu_python osmarzafalon$
nario2.py
dict_values(['Osmar', 25, 'São Paulo'])
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 7 - **Dicionários e Conjuntos**

Métodos de dicionários: keys(), values(), items().

items() retorna uma visão de todos os pares **chave-valor** presentes no dicionário. Cada par é representado como uma tupla.

```
dados = {"nome": "Osmar", "idade": 25, "cidade": "São Paulo"}

# Exibindo os pares chave-valor
print(dados.items())
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/local/bin/python3 /Users/osm
nario2.py
dict_items([('nome', 'Osmar'), ('idade', 25), ('cidade', 'São Paulo')])
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 7 - **Dicionários e Conjuntos**

Conjuntos: características e métodos

Conjuntos (ou **sets**) no Python são coleções **não ordenadas** e **não indexadas** de elementos **únicos**, ou seja, não permitem duplicatas. Eles são úteis para armazenar elementos onde a ordem não importa e as duplicatas precisam ser eliminadas automaticamente.

Características de Conjuntos:

- **Elementos únicos** : Não aceitam duplicatas.
- **Não ordenado** : A ordem dos elementos não é garantida, e eles não podem ser acessados por índice.
- **Mutável**: Você pode adicionar ou remover elementos de um conjunto.
- **Imutabilidade dos elementos** : Todos os elementos dentro de um conjunto devem ser de tipos imutáveis (por exemplo, inteiros, strings, tuplas)

Aula 7 - **Dicionários e Conjuntos**

Conjuntos: características e métodos

Métodos Comuns de Conjuntos:

- **add()**: Adiciona um elemento ao conjunto.
- **remove()**: Remove um elemento específico do conjunto. Se o elemento não estiver presente, gera um erro.
- **discard()**: Remove um elemento do conjunto. Se o elemento não estiver presente, não gera erro.
- **clear()**: Remove todos os elementos do conjunto.
- **union()**: Retorna a união de dois conjuntos (elementos presentes em um ou em ambos).
- **intersection()**: Retorna a interseção de dois conjuntos (elementos presentes em ambos os conjuntos).
- **difference()**: Retorna a diferença entre dois conjuntos (elementos presentes no primeiro conjunto, mas não no segundo).
- **issubset()**: Verifica se um conjunto é subconjunto de outro.
- **issuperset()**: Verifica se um conjunto é superconjunto de outro.

Aula 7 - **Dicionários e Conjuntos**

Conjuntos: características e métodos

Métodos Comuns de Conjuntos:

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/local/bin/
ntos.py
{'ônibus', 'carro', 'bicicleta', 'moto'}
{'ônibus', 'carro', 'bicicleta', 'moto'}
{'ônibus', 'carro', 'bicicleta'}
{'caminhão', 'carro'}
{'ônibus', 'moto', 'bicicleta', 'caminhão', 'carro'}
iMac-de-Osmar:meu_python osmarzafalon$
```

```python
# Definindo um conjunto de veículos
veiculos = {"carro", "moto", "bicicleta"}

# Adicionando um elemento
veiculos.add("ônibus")
print(veiculos) # Saída: {"carro", "moto", "bicicleta", "ônibus"}

# Tentando adicionar um elemento duplicado
veiculos.add("carro") # Elementos duplicados são ignorados
print(veiculos) # Saída: {"carro", "moto", "bicicleta", "ônibus"}

# Removendo um elemento
veiculos.remove("moto")
print(veiculos) # Saída: {"carro", "bicicleta", "ônibus"}

# Diferença entre dois conjuntos de veículos
veiculos_a = {"carro", "moto", "caminhão"}
veiculos_b = {"moto", "bicicleta", "ônibus"}

diferenca = veiculos_a.difference(veiculos_b)
print(diferenca) # Saída: {'carro', 'caminhão'}

# União de dois conjuntos
uniao = veiculos_a.union(veiculos_b)
print(uniao) # Saída: {'carro', 'moto', 'caminhão', 'bicicleta', 'ônibus'}
```

Aula 7 - **Dicionários e Conjuntos**

Conjuntos: características e métodos

Métodos Comuns de Conjuntos: .clear()

```
# Definindo um conjunto de veículos
veiculos = {"carro", "moto", "bicicleta"}

print("Conjunto original:", veiculos) # Saída: {"carro", "moto", "bicicleta"}

# Limpando o conjunto usando clear()
veiculos.clear()

print("Conjunto após clear():", veiculos) # Saída: set() (um conjunto vazio)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/local/
ntos_crear.py
Conjunto original: {'bicicleta', 'carro', 'moto'}
Conjunto após clear(): set()
iMac-de-Osmar:meu_python osmarzafalon$
```

Atividade - Sujestão

```
# Programa para cadastro de notas de alunos

# Desenvolver código para uso em sala de aula
# que tenha Turma
# nome do professor
# nome do aluno
# quatro notas (1º, 2º, 3º, 4º bimestres)
# média do aluno
# situação aprovado >= 6 / reprovado
# utilizando loop while
# except para retorno de erro na nota (0 a 10)
# para sair do loop (fim)
```

Atividade - Resultado

Duas possíveis opções
(mais completa,
estruturada com dicionários)

(mais simples, usando
lista dentro de lista)

```
            break  # Sai do loop

    notas = []
    bimestres = ["1º Bimestre", "2º Bimestre", "3º Bimestre", "4º Bimestre"]

    # Captura das 4 notas com validação
    for b in bimestres:
        while True:
            try:
                nota = float(input(f"Digite a nota do {b} de {nome}: "))
                if 0 <= nota <= 10:
                    notas.append(nota)
                    break  # sai do loop interno se a nota for válida
                else:
                    print(" A nota deve estar entre 0 e 10.")
            except ValueError:
                print(" Entrada inválida! Digite um número.")

    # Calcula média e situação
    media = sum(notas) / len(notas)
    situacao = "Aprovado" if media >= 6 else "Reprovado"

    # Armazena os dados do aluno
    aluno = {
        "nome": nome,
        "notas": notas,
        "media": media,
        "situacao": situacao
    }
    alunos.append(aluno)

# Exibição dos resultados
print("\n=== RESULTADOS DA TURMA ===")
print(f"Turma: {turma}")
print(f"Professor: {professor}\n")

for aluno in alunos:
    print(f"Aluno: {aluno['nome']} | Notas: {aluno['notas']} | "
          f"Média: {aluno['media']:.2f} | Situação: {aluno['situacao']}")
```

Aula 8 - **Manipulação de Arquivos**

Conteúdo:

1. Abertura de arquivos com open().
2. Leitura de arquivos (read(), readline()).
3. Escrita em arquivos (write()).

Aula 8 - **Manipulação de Arquivos**

O comando open()

O comando **open()** é usado para abrir arquivos no sistema.
Com ele, é possível ler, escrever ou manipular arquivos.
O comando retorna um objeto de arquivo que pode ser manipulado.

```
arquivo = open("nome_do_arquivo.txt", "modo")
```

- `arquivo` — Nome da varável
- `open` — comando
- `"nome_do_arquivo.txt"` — Caminho e nome do arquivo
- `"modo"` — “r”, “w”, “a”, “b”

Aula 8 - **Manipulação de Arquivos**

O comando open()

Modos de abertura de arquivos:

- "r": Leitura (modo padrão). O arquivo deve existir.
- "w": Escrita. Cria um arquivo novo ou sobrescreve o existente.
- "a": Acrescenta (append). Abre o arquivo para adicionar conteúdo no final, sem apagar o conteúdo existente.
- "b": Modo binário. Usado para abrir arquivos em formato binário (imagens, vídeos, etc.), geralmente combinado com "r", "w" ou "a".

Aula 8 - **Manipulação de Arquivos**

O comando open()

Quando usamos a função open() para abrir um arquivo, especificamos um **modo de abertura** por meio de uma letra (ou combinação de letras). Essas letras indicam o que queremos fazer com o arquivo.

Modos de abertura de arquivos:

- **"r"**: **read** (leitura)
  Abre o arquivo para leitura. Este é o modo padrão. Se o arquivo não existir, gera um erro.
- **"w"**: **write** (escrita)
  Abre o arquivo para escrita. Se o arquivo já existir, seu conteúdo será **sobrescrito**. Se o arquivo não existir, ele será criado.
- **"a"**: **append** (adicionar)
  Abre o arquivo para adição de conteúdo. O texto será adicionado ao final do arquivo, sem apagar o conteúdo existente.
- **"x"**: **exclusive creation**
  Cria um novo arquivo. Se o arquivo já existir, gera um erro.
- **"b"**: **binary** (binário)
  Abre o arquivo em modo binário. Pode ser combinado com outros modos, como "rb" para leitura binária ou "wb" para escrita binária. Usado para arquivos como imagens, vídeos, etc.
- **"t"**: **text** (texto)
  Abre o arquivo no modo texto. Este é o modo padrão para leitura e escrita de arquivos de texto. Pode ser combinado com outros modos, como "rt" para leitura de texto.

Aula 8 - **Manipulação de Arquivos**

O comando open()

Modos de abertura de arquivos:

- "r": Leitura (modo padrão). O arquivo deve existir.
- "w": Escrita. Cria um arquivo novo ou sobrescreve o existente.
- "a": Acrescenta (append). Abre o arquivo para adicionar conteúdo no final, sem apagar o conteúdo existente.
- "b": Modo binário. Usado para abrir arquivos em formato binário (imagens, vídeos, etc.), geralmente combinado com "r", "w" ou "a".

Aula 8 - **Manipulação de Arquivos**

O comando open()

Abrindo e lendo um arquivo de texto com "r"

```
# Abrindo o arquivo para leitura
arquivo = open("meu_txt.txt", "r", encoding="utf-8")

# Lendo o conteúdo
conteudo = arquivo.read()
print(conteudo)

# Fechando o arquivo
arquivo.close()
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /usr/local/bin/python3 /Users/osmarzafalon/Desktop/meu_p
py

O que é o Lorem Ipsum?
O Lorem Ipsum é um texto modelo da indústria tipográfica e de impressão.
O Lorem Ipsum tem vindo a ser o texto padrão usado por estas indústrias desde o ano de 1500,
quando uma misturou os caracteres de um texto para criar um espécime de livro.
Este texto não só sobreviveu 5 séculos, mas também o salto para a tipografia electrónica,
mantendo-se essencialmente inalterada. Foi popularizada nos anos 60 com a
disponibilização das folhas de Letraset, que continham passagens com Lorem Ipsum,
e mais recentemente com os programas de publicação como o Aldus PageMaker que incluem
versões do Lorem Ipsum.

iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 8 - **Manipulação de Arquivos**

O comando open()

Criando e escrevendo em um arquivo com "w"

```
# Abrindo um arquivo para escrita (ou criando se não existir)
arquivo = open("novo_arquivo.txt", "w")

# Escrevendo no arquivo
arquivo.write("Este é um novo arquivo criado com Python!\n Osmar Zafalon")

# Fechando o arquivo
arquivo.close()
```

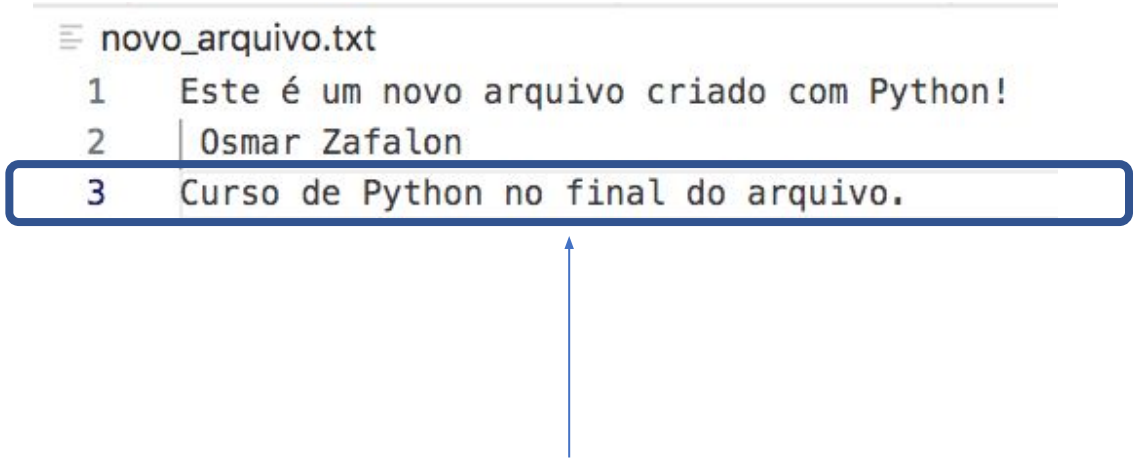
```
iMac-de-Osmar:meu_python osmarzafalon$ do_escr.py
iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 8 - **Manipulação de Arquivos**

O comando open()

Criando e escrevendo em um arquivo com "a":

```
# Abrindo o arquivo em modo de adição (append)
arquivo = open("novo_arquivo.txt", "a")

# Acrescentando mais conteúdo
arquivo.write("\nCurso de Python no final do arquivo.")

# Fechando o arquivo
arquivo.close()
```



Conteúdo acrescentado ao arquivo

Aula 8 - **Manipulação de Arquivos**

O comando open()

Abrindo um arquivo de texto com "with"

```
# Abrindo um arquivo usando 'with' (fechamento automático)
with open("novo_arquivo.txt", "r") as arquivo:
conteudo = arquivo.read()
print(conteudo)
# O arquivo será fechado automaticamente ao final do bloco 'with'
```

O método **with** é uma maneira mais segura de abrir arquivos, pois garante que o arquivo seja fechado corretamente, mesmo que ocorra algum erro no código

Aula 8 - **Manipulação de Arquivos**

O comando open()

atividade

```
# Abrindo o arquivo para escrita (modo texto "t")
with open("arquivo_exemplo.txt", "wt") as arquivo:
    # Escrevendo no arquivo
    arquivo.write("Este é um exemplo de texto.\n")
    arquivo.write("Estamos escrevendo no modo texto ('t').\n")
    arquivo.write("Python facilita o manuseio de arquivos.\n")

# Abrindo o arquivo para leitura (modo texto "t")
with open("arquivo_exemplo.txt", "rt") as arquivo:
    # Lendo e exibindo o conteúdo do arquivo
    conteudo = arquivo.read()
    print("Conteúdo do arquivo:\n")
    print(conteudo)
```

```
iMac-de-Osmar:meu_python osmarzafalon$ /us
t.py
Conteúdo do arquivo:

Este é um exemplo de texto.
Estamos escrevendo no modo texto ('t').
Python facilita o manuseio de arquivos.

iMac-de-Osmar:meu_python osmarzafalon$
```

Aula 9 - **Módulos e Bibliotecas**

Biblioteca para abrir planilha do excel

```
import pandas as pd

# Lendo o arquivo Excel
df =
pd.read_excel("Plan_resultados_2025_HTML_tarde.
xlsx", sheet_name="HTML_CSS_Tarde")

# Mostrando as 5 primeiras linhas
print(df.head())
```

pip install pandas openpyxl

Aula 9 - **Módulos e Bibliotecas**

Biblioteca para abrir planilha do excel

pip install pandas openpyxl

Workbook (arquivo inteiro) e **Worksheet** (a folha ativa).

```python
import openpyxl
from openpyxl import Workbook
import os

# Nome do arquivo Excel
arquivo = "dados.xlsx"

# Se o arquivo não existir, cria um novo
if not os.path.exists(arquivo):
    wb = Workbook()
    ws = wb.active
    ws.title = "Alunos"
    # Cabeçalho da planilha
    ws.append(["Nome", "Idade", "Nota"])
    wb.save(arquivo)

# Abrindo o arquivo existente
wb = openpyxl.load_workbook(arquivo)
ws = wb["Alunos"]

# Adicionando informações
alunos = [
    ["João", 15, 8.5],
    ["Maria", 16, 9.2],
    ["Pedro", 14, 7.0]
]

for aluno in alunos:
    ws.append(aluno)

# Salvando as alterações
wb.save(arquivo)

print("Dados adicionados com sucesso em" , arquivo)
```

Aula 9 - **Módulos e Bibliotecas**

Biblioteca para abrir
planilha do excel e gerar pdf

pip install pandas openpyxl

pip install openpyxl reportlab

```
import openpyxl
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas

# Abrir o arquivo Excel
wb = openpyxl.load_workbook("dados.xlsx")
ws = wb.active  # Pega a primeira planilha

# Criar o arquivo PDF
pdf = canvas.Canvas("saida.pdf", pagesize=A4)
largura, altura = A4

# Título
pdf.setFont("Helvetica-Bold", 14)
pdf.drawString(50, altura - 50, "Relatório gerado a partir do Excel")

# Posição inicial para os dados
y = altura - 100
pdf.setFont("Helvetica", 12)

# Ler os dados do Excel e escrever no PDF
for linha in ws.iter_rows(values_only=True):
    texto = " | ".join([str(celula) if celula is not None else "" for celula in linha])
    pdf.drawString(50, y, texto)
    y -= 20  # Desce a linha no PDF

    if y < 50:  # Se chegar no fim da página, cria nova página
        pdf.showPage()
        pdf.setFont("Helvetica", 12)
        y = altura - 50

# Salvar o PDF
pdf.save()

print("PDF gerado com sucesso: saida.pdf")
```

Aula 9 - **Módulos e Bibliotecas**

Conteúdo:

1. Importação de módulos com import.
2. Usar bibliotecas padrão do Python (math, random, etc.).

Aula 9 - **Módulos e Bibliotecas**

O comando pip

O **PIP** (Python Package Installer) é a ferramenta padrão para instalar e gerenciar pacotes Python.

É um gerenciador de pacotes do Python que permite instalar, remover, listar e atualizar pacotes de software escritos em Python.
O pip é similar aos gerenciadores de pacotes npm e composer (php).
A maioria das distribuições do Python vem com o pip pré-instalado. Para verificar se o pip está instalado, você pode pesquisar por "cmd" no menu iniciar e clicar duas vezes para abri-lo.
Para que o pip possa ser executado diretamente do prompt de comando e do terminal do editor, é necessário configurar o Python. Caso não seja configurado corretamente, o erro "pip não é reconhecido como um comando interno" irá aparecer.
O pip foi introduzido em 2008 por Ian Bicking, criador do pacote virtualenv.
O nome pip é um acrônimo para "Pip Install Packages"

Aula 9 - **Módulos e Bibliotecas**

O comando pip

Instalação do Pip no Windows:

1. Durante a instalação, marque a opção “Add Python 3. x to PATH” e prossiga com a instalação.
2. Abra o prompt de comando (**cmd**) como administrador.
3. Digite python -m ensurepip --default-**pip** e pressione Enter para **instalar** o **Pip**.

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

- **pip install**: Instala um pacote
- **pip install --upgrade**: Atualiza um pacote
- **pip freeze**: Lista os pacotes instalados e suas versões
- **pip uninstall**: Remove um pacote
- **python -m pip**: Executa o pip quando ele não é reconhecido

O pip é um gerenciador de pacotes do Python que permite instalar, desinstalar e atualizar bibliotecas.
Para instalar o pip, é possível:

1. Baixar o arquivo get-pip.py
2. Salvar o arquivo no diretório desejado
3. Executar o arquivo get-pip.py usando o Python através do terminal ou cmd

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Instalar Pacotes

pip install nome_do_pacote

Para instalar um pacote:

pip install nome_do_pacote==versao

Para instalar uma versão específica de um pacote

pip install -r requirements.txt

Para instalar múltiplos pacotes de um arquivo

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Listar Pacotes Instalados

pip list
Para listar todos os pacotes instalados no ambiente:

pip list --outdated
Para listar pacotes desatualizados:

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Atualizar Pacotes

pip install --upgrade nome_do_pacote
Para atualizar um pacote

pip install --upgrade $(pip list --outdated | grep -o '^[a-zA-Z0-9_\-]*')
Para atualizar todos os pacotes desatualizados

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Desinstalar Pacotes

pip uninstall nome_do_pacote
Para desinstalar um pacote:

pip uninstall -r requirements.txt
Para desinstalar múltiplos pacotes de um arquivo

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Exibir Informações sobre um Pacote

pip show nome_do_pacote
Para exibir informações detalhadas sobre um pacote instalado

Buscar Pacotes

pip search nome_do_pacote
Para pesquisar pacotes disponíveis no PyPI

Aula 9 - **Módulos e Bibliotecas**

Alguns comandos do pip para Python são:

Verificar Dependências de Segurança

pip check	Para verificar vulnerabilidades de segurança em pacotes instalados:

Ajuda e Versão

pip --version	Para verificar a versão do PIP instalada:
pip --help	Para obter ajuda sobre os comandos do PIP:

Aula 9 - **Módulos e Bibliotecas**

PS C:\Users\d315561\Desktop\meu_python> **python**

Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> **help('modules')**

Lista todos os módulos do python

Aula 9 - **Módulos e Bibliotecas**

Biblioteca em qualquer linguagem de programação

é um conjunto de:

- **funções**
- **Classes**
- **módulos**

previamente escritos que oferecem funcionalidades específicas para facilitar o desenvolvimento de programas. Essas bibliotecas ajudam os programadores a não precisar reescrever o código básico ou repetitivo, permitindo que eles foquem em resolver problemas mais complexos.

Funções: Blocos de código reutilizáveis que realizam tarefas específicas.
Classes: Estruturas que definem objetos com atributos e métodos.
Módulos: Arquivos que agrupam funções e classes para reutilização em diferentes partes de um programa.

Aula 9 - **Módulos e Bibliotecas**

O Python vem com uma série de bibliotecas padrão, que são módulos e pacotes que fazem parte da distribuição padrão do Python.

os: Interação com o sistema operacional (manipulação de arquivos e diretórios).
sys: Manipulação de parâmetros e funções que interagem com o interpretador Python.
math: Funções matemáticas básicas e constantes.
random: Geração de números aleatórios.
datetime: Manipulação de datas e horas.
json: Manipulação de dados no formato JSON.
csv: Leitura e escrita de arquivos CSV.
re: Expressões regulares para busca e manipulação de strings.
urllib: Manipulação de URLs e requisições HTTP.
requests: (não é padrão, mas muito comum e amplamente usada) para fazer requisições HTTP de forma simples.
sqlite3: Interface para banco de dados SQLite.
collections: Contêineres de dados especializados (como listas, dicionários, conjuntos).
itertools: Funções para criar iteradores eficientes.
functools: Funções para manipulação de funções e métodos.
threading: Suporte a programação multithread.