Relatório 2 - Linguagem de Programação Python (I)

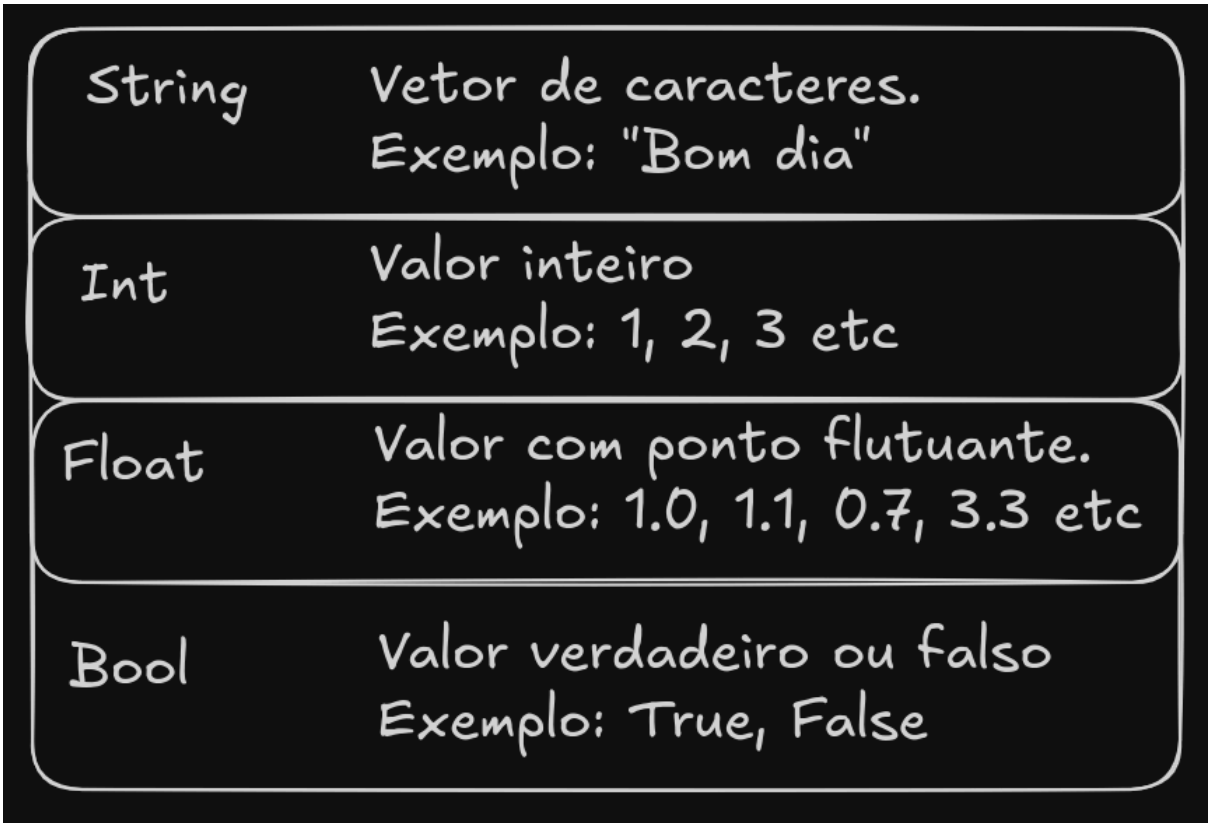
Felipe Fonseca

Descrição da atividade

As duas videoaulas possuíam um conteúdo de introdução para a linguagem de python.

A primeira videoaula começou apresentando comandos básicos como print() e depois apresentou alguns tipos básicos que são utilizados no python.

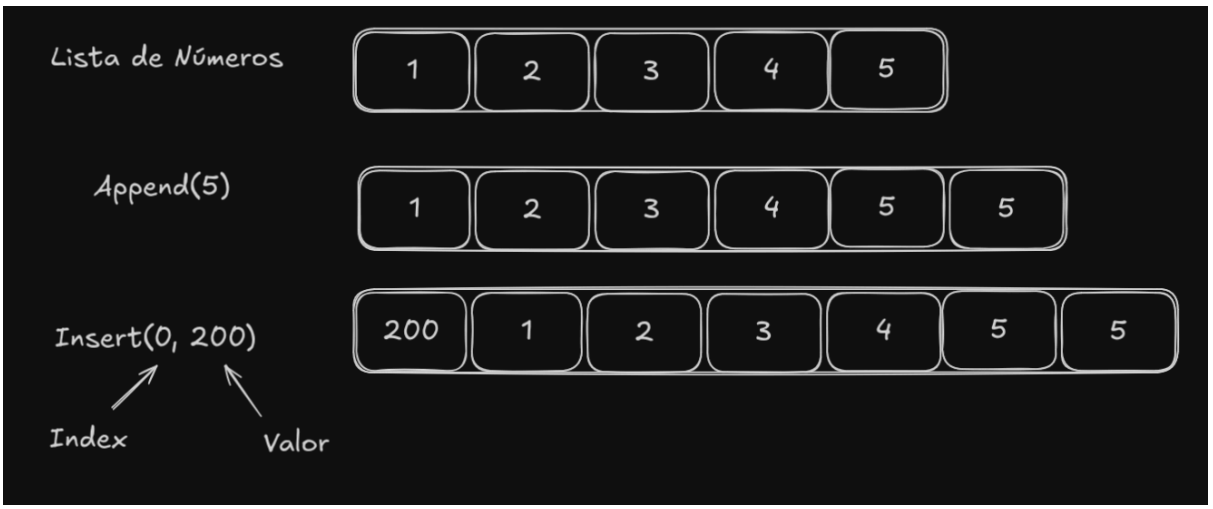
Dentre esses tipos, destacam-se a String, float, e int.



```python
texto = 'Sua idade é...'
idade = 23

#str() transforma uma variável de outro tipo para String
print(texto + ' ' + str(idade))

#FString, que funciona para referenciar outros tipos de valores no Print
print(f'{texto} {idade}')
print(f'{texto} {15+12}')

#Vai mostrar o texto 3 vezes
saudacao = 'Bom dia! '
print(3 * saudacao)

#Conversão de que constantes usam letras maiúsculas
PI = 3.14
```

```python
#Mostra o tipo da variável
print(type(raio))

print(f'A área da circunferência é {area}')


print('\n\n\n')

#Tipos básicos do python
print('#Tipos básicos do python')
print('\n\n\n')

print(type(1))
print(type(1.1))
print(type('texto'))
print(type(True))
print(type(False))
```

Após isso, começaram então outros tipos um pouco mais complexos, como as listas, que são estruturas que armazenam outros valores em uma lista, sem nenhuma regra muito rígida.




```python
nums = [1, 2, 3]
print(type(nums))

#append() adiciona valores na lista
nums.append(3) #Listas podem possuir valores repetidos
nums.append(4)
nums.append(5)

#len() mostra o tamanho da lista
print(len(nums))

#Altera o valor do elemento da lista no indice selecionado
nums[3] = 100
print(nums)

#insert() adiciona um elemento na posição da lista que é desejado e reposiciona o resto
nums.insert( _index: 0, -200)
print(nums)
#printa o elemento do index na lista
print(nums[5])
#printa o elemento no index na lista contando de frente pra tras
print(nums[-1])
```

Depois foi mostrado as tuplas:

```
nomes = ('Ana', 'Bia', 'Gui', 'Leo','Ana')
print(type(nomes))
print(nomes)

print('bia' in nomes) #Mostra se o valor está presente na tupla

print(nomes[0])
print(nomes[1:3])
print(nomes[1:-1])
print(nomes[2:])
print(nomes[:-2])
#Serve para printar de um ponto da tupla (funciona nas listas) até outro
```

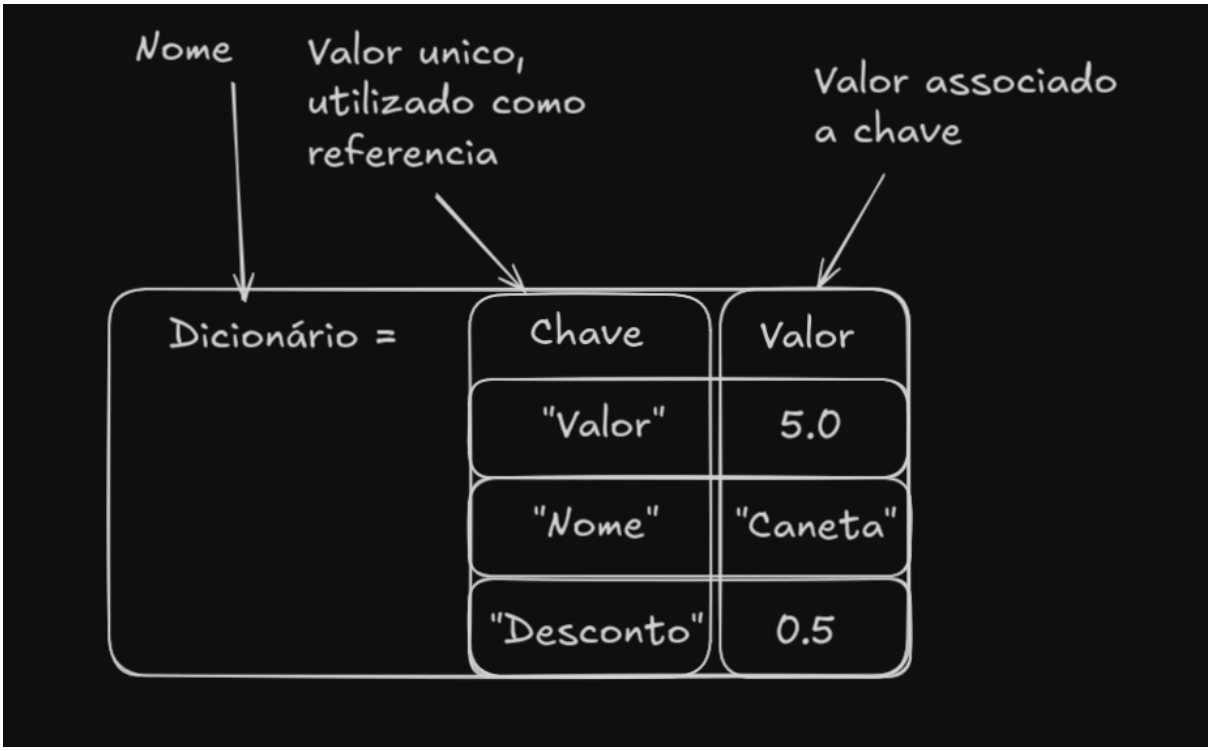
Que diferente da lista, os valores não podem ser alterados.

Nos conjuntos, temos a característica de não permitir repetição de elementos:

```
print('Conjunto')
print('\n\n\n')

print({1,2,3})
print(type({1,2,3}))
conj = {1,2,3,3,3,3,3,3,3,3} #Conjuntos não aceitam valores repetidos

# print(conj[0]) #conjuntos não suportam index

print(conj)
print(len(conj))

print('\n\n\n')
```

Além disso, eles não suportam indexação

Dicionários é uma estrutura que possui chaves e valores, podendo acessar os valores através das chaves e serve muito bem para guardar informações com valores:



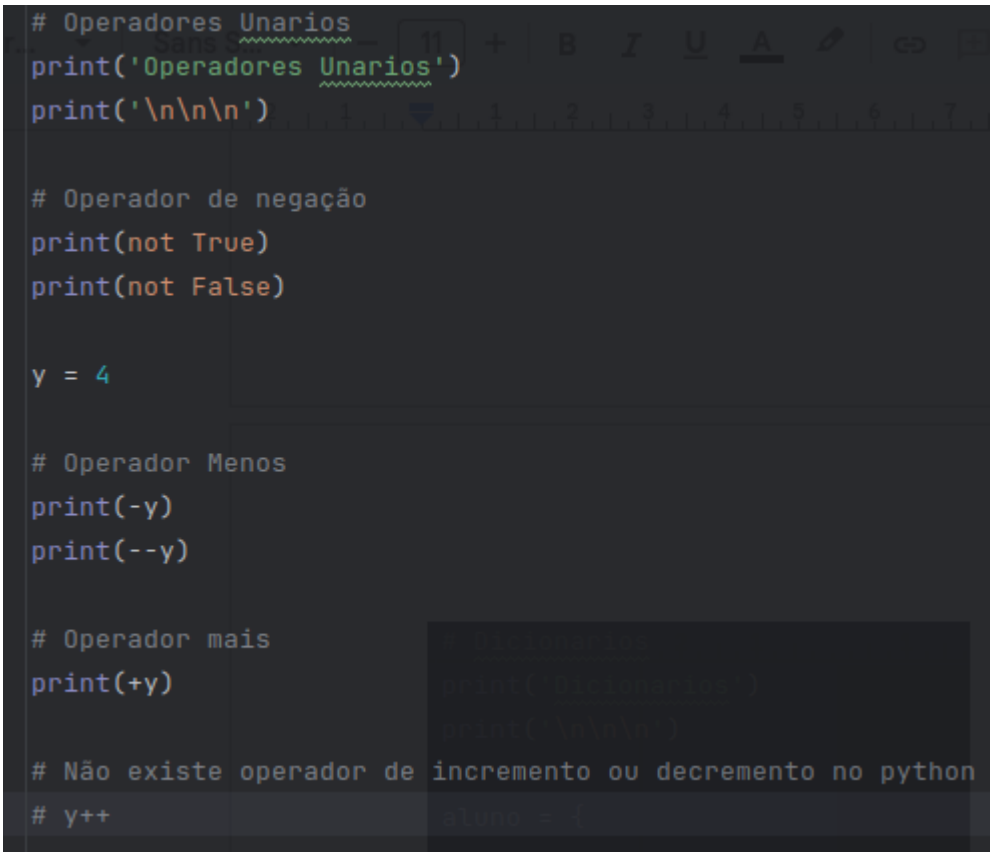

```python
# Dicionarios
print('Dicionarios')
print('\n\n\n')

aluno = {
    'nome': 'Pedro Henrique',
    'nota': 9.2,
    'ativo': True
}

print(type(aluno))

# Acessar pela chave
print(aluno['nome'])
print(aluno['nota'])
print(aluno['ativo'])
```

Depois disso foram mostrado os operadores:

Operadores	
Unários	Operações simples, como negação ou inverter o sinal do número
Aritméticos	Operações matemáticas básicas, como adição, subtração, multiplicação, etc.
Comparação	Comparação entre valores, sempre retorna um valor verdadeiro ou falso para a comparação, que pode ser igual a, menor que, maior que, etc
Lógicos	Operadores que temos na tabela da verdade como Or e And, que servem para trabalhar com condições mais complexas
Ternário	É uma forma resumida de escrever um if else

Depois foi apresentado os operadores unarios e os aritméticos.

Essas são as operações mais simples, como negação.



```
# Operadores Unarios
print('Operadores Unarios')
print('\n\n\n')

# Operador de negação
print(not True)
print(not False)

y = 4

# Operador Menos
print(-y)
print(--y)

# Operador mais
print(+y)

# Não existe operador de incremento ou decremento no python
# y++
```

Aqui tem as operações aritméticas, ou seja, as comuns que são aprendidas na escola por exemplo

```python
# Operadores Aritmeticos
print('Operadores Aritmeticos')
print('\n\n\n')

x = 10
y = 3

print(x + y)
print(x - y)
print(x * y) #Multiplicação
print(x / y)
print(x // y) # no vídeo nao é citado pois ele utiliza a versão 2 do python, mas esse operador mostra apenas
              # o resultado inteiro da divisão
print(x % y) #resto da divisão
```

E aqui tem as comparativas:

```python
# Comparações que retornam verdadeiro ou falso
print(x > y)
print(x >= y)
print(x < y)
print(x <= y)
print(x == y)
print(x != y)
```

Onde pode-se comparar valores que retornam se a operação é verdadeira ou falsa.

Nos operadores lógicos foram apresentados os termos que aprendemos na tabela da verdade:

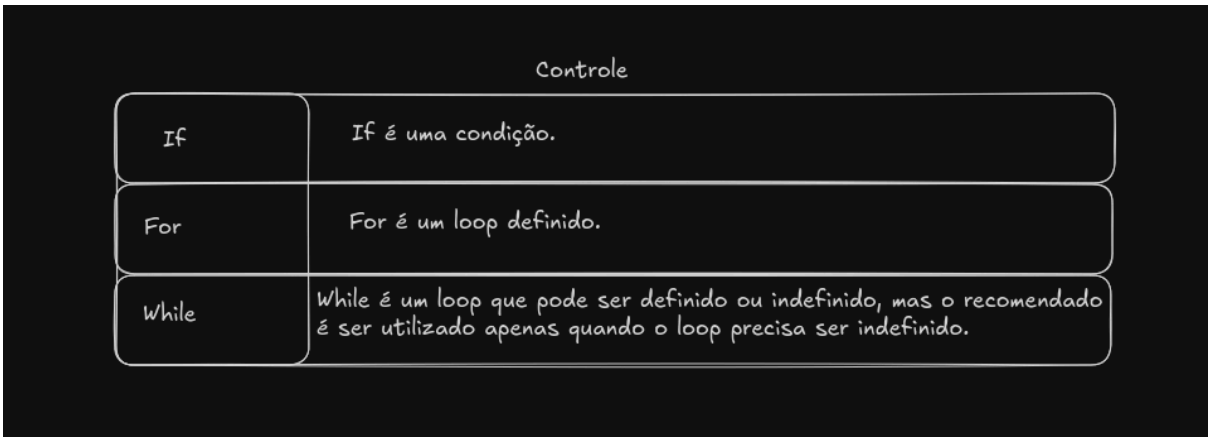
```python
# Operadores logicos
print('Operadores logicos')
print('\n\n\n')

b1 = True
b2 = False
b3 = True

# Operador lógico binário
print(b1 and b2 and b3)
print(b1 or b2 or b3)
print(b1 != b2) #xor
print(not b1)
print(not b2)
```

E podemos utilizar as operações como not, and or etc.

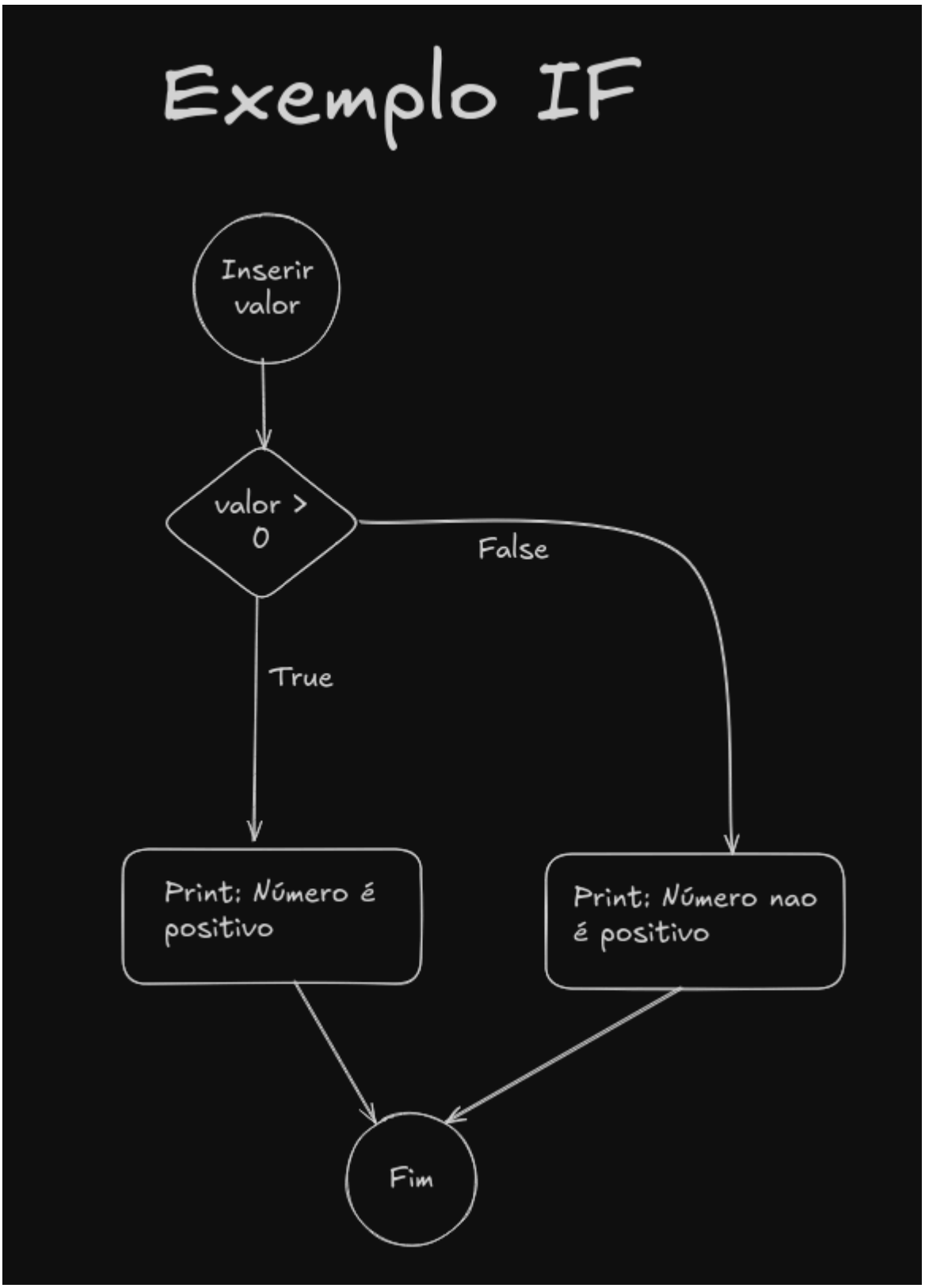
Operador ternário é uma forma fácil de retornar um valor caso verdadeiro ou retornar outra coisa caso seja falsa, determinada uma (ou mais) condição/ções

```python
# Operadores ternarios
print('Operadores ternarios')
print('\n\n\n')

lockdown = False
grana = 30
status = 'Em casa' if lockdown or grana <= 100 else 'Uhuuuuuuuu!'
#If ternário que faz algo se for verdadeiro ou faz outra coisa caso seja falso dada uma condição
print(f'O status é : {status}')

print('\n\n\n')
```

Depois disso foram apresentadas as estruturas de controle: If, For e While.

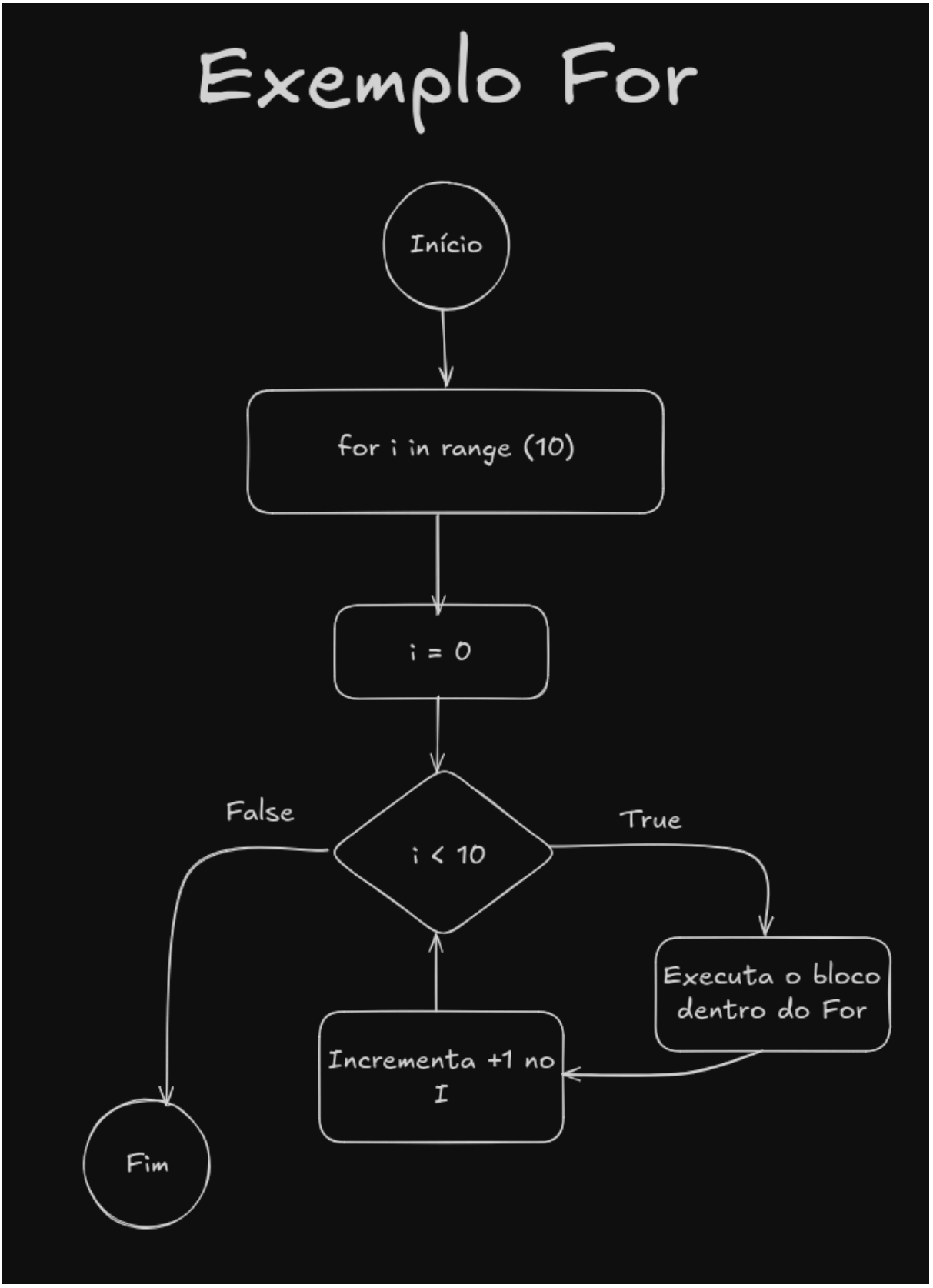


Exemplo de uso do If:



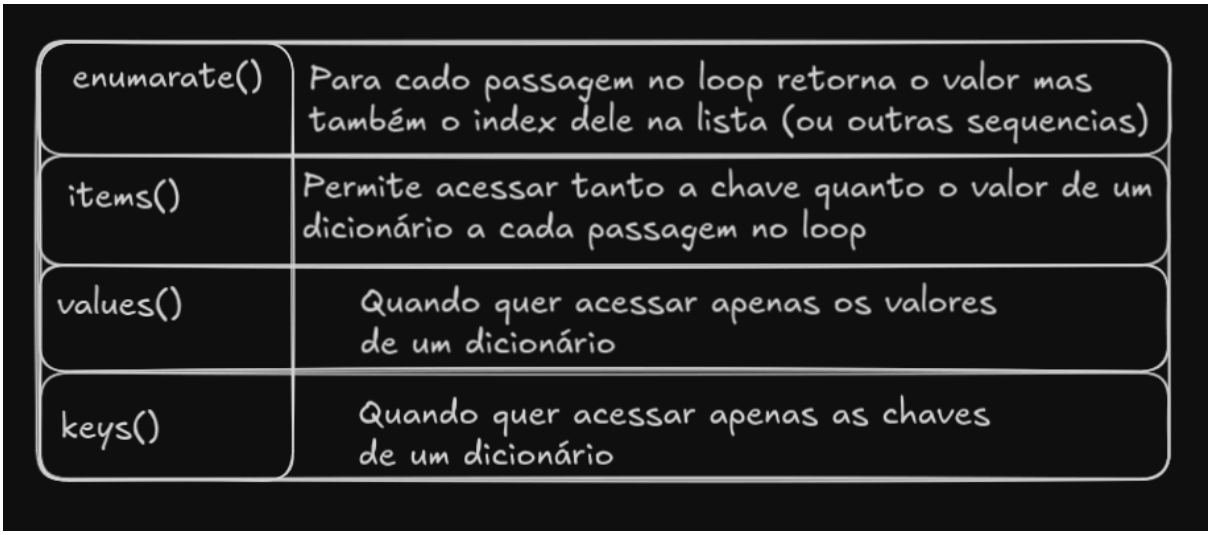

```
if a:
    print('Existe')
else:
    print('Não existe ou zero ou vazio')
```

Exemplo de uso do For:



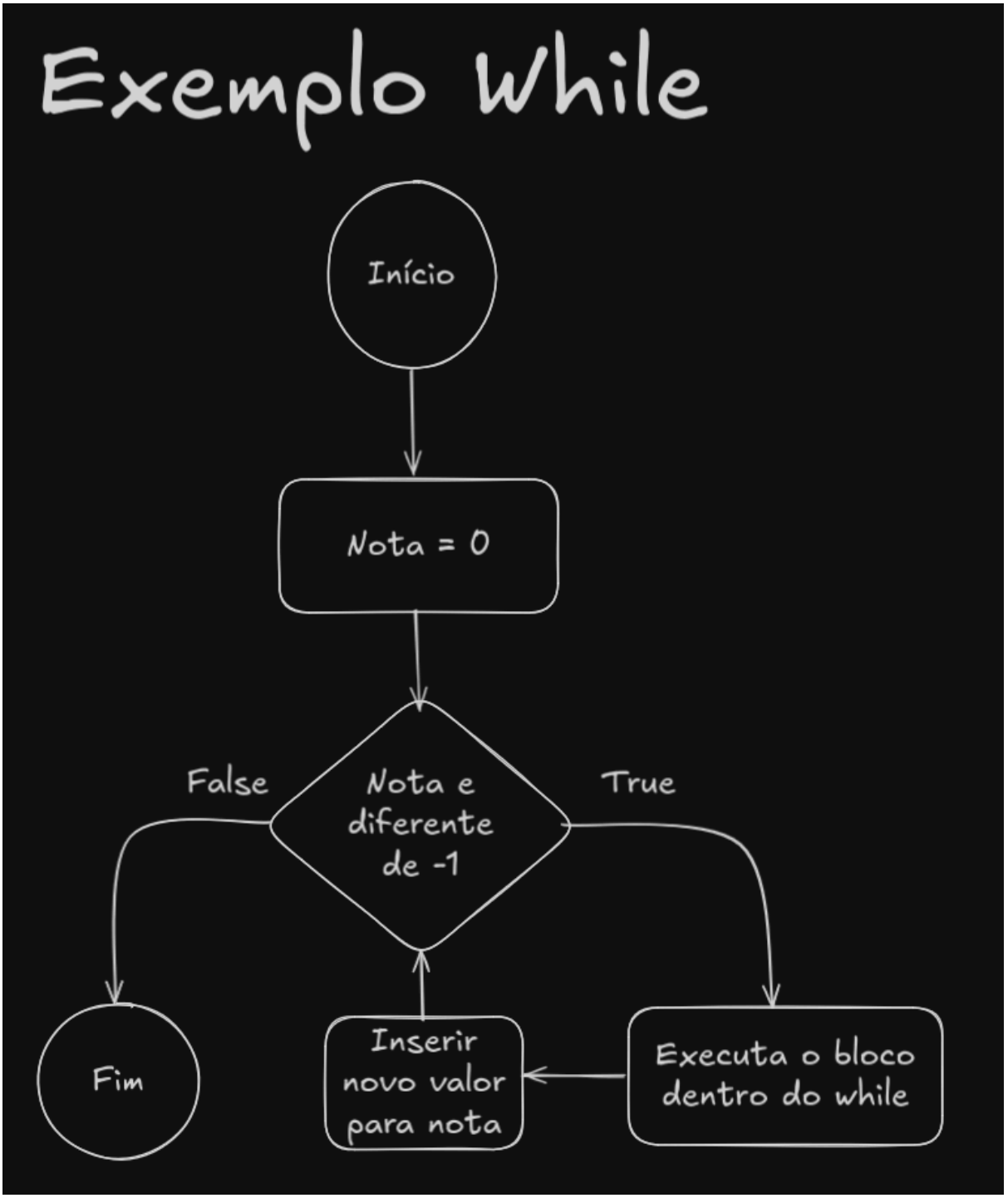
```python
for i in range (10):
    print(i)
```

Além do uso do for in range, também foi mostrado como utilizar o For com vários tipos de variáveis diferentes, como listas ou bibliotecas, utilizando funções como enumerate(), items(), values() e keys().




```python
for atrib in produto: # Pegando os atributos do produto
    print(atrib, '==>', produto[atrib]) # produto[atrib] pega os valores dos atributos

print('\n')


for atrib, valor in produto.items(): # Mesma coisa mas mais simples
    print(atrib, '==>', valor)

print('\n')


for valor in produto.values(): # Percorre apenas os valores
    print(valor, end=' ')

print('\n')

for atrib in produto.keys(): # Percorre apenas as chaves
    print(atrib, end=' ')
```

E aqui um exemplo de uso recomendado do while, quando temos um loop de tamanho indefinido:


Exemplo While
Início
Nota = 0
Nota e diferente de -1
False
True
Fim
Inserir novo valor para nota
Executa o bloco dentro do while

```
# Executa enquanto nota for diferente de -1
while nota != -1:
    nota = float(input('Informe a nota ou -1 para sair: '))
    if nota != -1:
        qtde += 1
        total += nota

print(f'A média da turma é {total / qtde}')
```

Agora no segundo vídeo, o foco ficou em dois pontos principais: Funções e Classes.

Funções é um bloco de código que é feito para ser chamado, é extremamente necessário para qualquer algoritmo que deseja se manter organizado e otimizado.

```
def soma(a, b):  2 usages
    return a + b

def sub(a, b):  1 usage
    return a - b
```

Nas funções você pode passar parâmetros para execução, e elas podem retornar valores para serem usados depois.

O vídeo também mostrou que é possível retornar uma função a uma variável, algo que eu não sabia que era possível.

No foco de funções, também foi mostrado algumas funções como a função lambda, que é uma função anônima definida em uma única linha.

```
somar = lambda a, b: a + b
total = reduce(somar, notas_alunos_aprovados, 0) # Somando as notas
```

Também foi mostrado algumas funções extras, como reduce, que é utilizado para fazer somas consecutivas.

Ou então o Map, que é utilizado para criar uma estrutura semelhante à que foi passada como parâmetro, mas com uma função aplicada a seus valores. Como no exemplo:

```
notas = [6.4, 7.2, 5.8, 8.4]
notas_finais_1 = map(somar_nota(1.5), notas) # Usando map para criar uma nova lista a partir de uma lista
notas_finais_2 = map(somar_nota(2), notas)
```

Foi também apresentado o filter, que pode criar uma nova estrutura utilizando de uma função para filtrar objetos passados pra ela. Nesse caso, a função que filtra é a função lambda, e ela está filtrando uma lista de alunos. Criando assim, uma nova lista apenas com os alunos aprovados.

```python
aluno_aprovado = lambda aluno: aluno['nota'] >= 7 # Recebe um aluno como parametro, retorna verdadeiro ou falso
alunos_aprovados = list(filter(aluno_aprovado, alunos)) # Cria uma nova lista filtrando conforme a outra função retorna
                                                        # verdadeiro/falso
```

E por fim, foi apresentado as classes, que conforme descrito pelo próprio apresentador do vídeo, é um projeto, que pode ser utilizado para instanciar vários objetos.

```python
class Produto:  2 usages
    def __init__(self, nome, preco = 1.99, desc = 0.0): # Método construtor do python
        # Atributos da classe
        self.__nome = nome # Colocar 2 underlines para definir o atributo como privado (Existe como burlar mas é uma convenção)
        self.__preco = preco
        self.desc = desc

    @property # Criando uma nova função para utilizar o nome  2 usages
    def nome(self):
        return self.__nome

    @property  3 usages
    def preco(self):
        return self.__preco

    @preco.setter # Criando um setter com condição, para que não seja manipulado diretamente
    def preco(self, novo_preco):
        if novo_preco > 0:
            self.__preco = novo_preco

    # Função própria da classe
    @property # Decorator que faz chamar a função como se fosse um atributo  2 usages
    def preco_final(self):
        return (1 - self.desc) * self.__preco

p1 = Produto( nome: 'Caneta',  preco: 5.99,  desc: 0.1)
p2 = Produto( nome: 'Caderno',  preco: 12.99,  desc: 0.2)
```

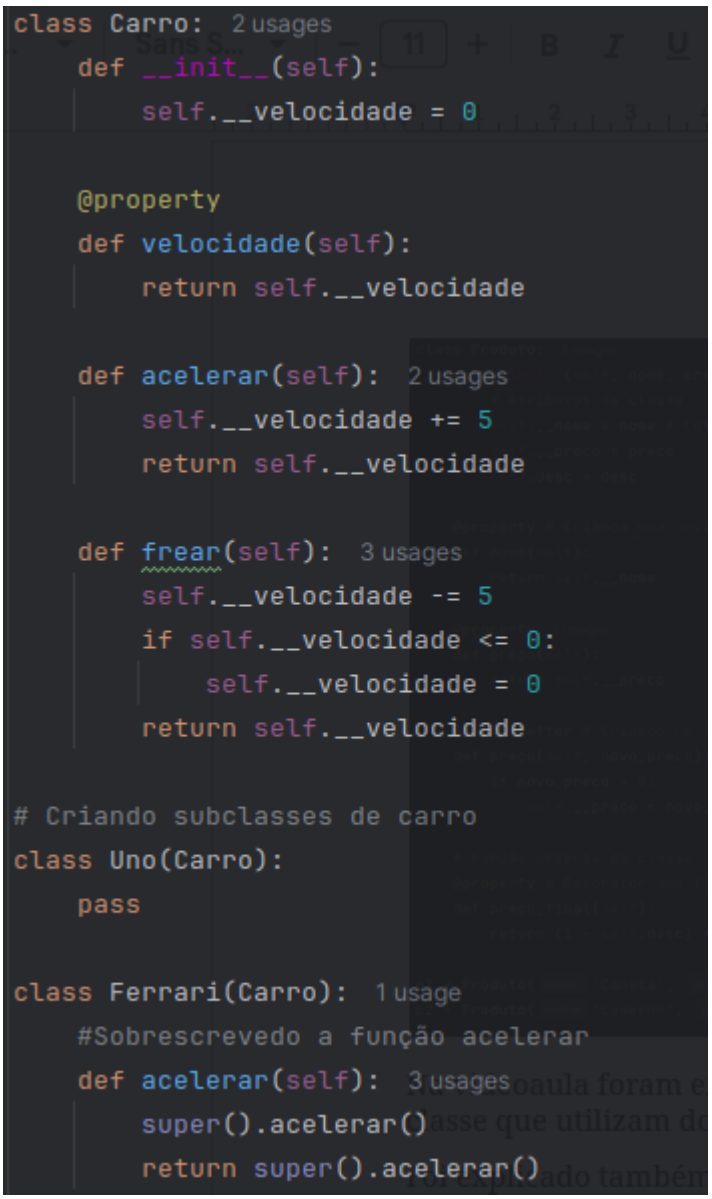
Na videoaula foram explicados o método construtor (def __init__), os atributos da classe que utilizam do "self".

Foi explicado também a convenção de se utilizar 2 underlines para atributos privados de uma classe.

@Property é um decorador, utilizado para que a função seja entendida como um atributo, e o seu valor é o valor que a função retorna.

@...setter é utilizado para criar um setter de condição, para que os valores não sejam manipulados diretamente.

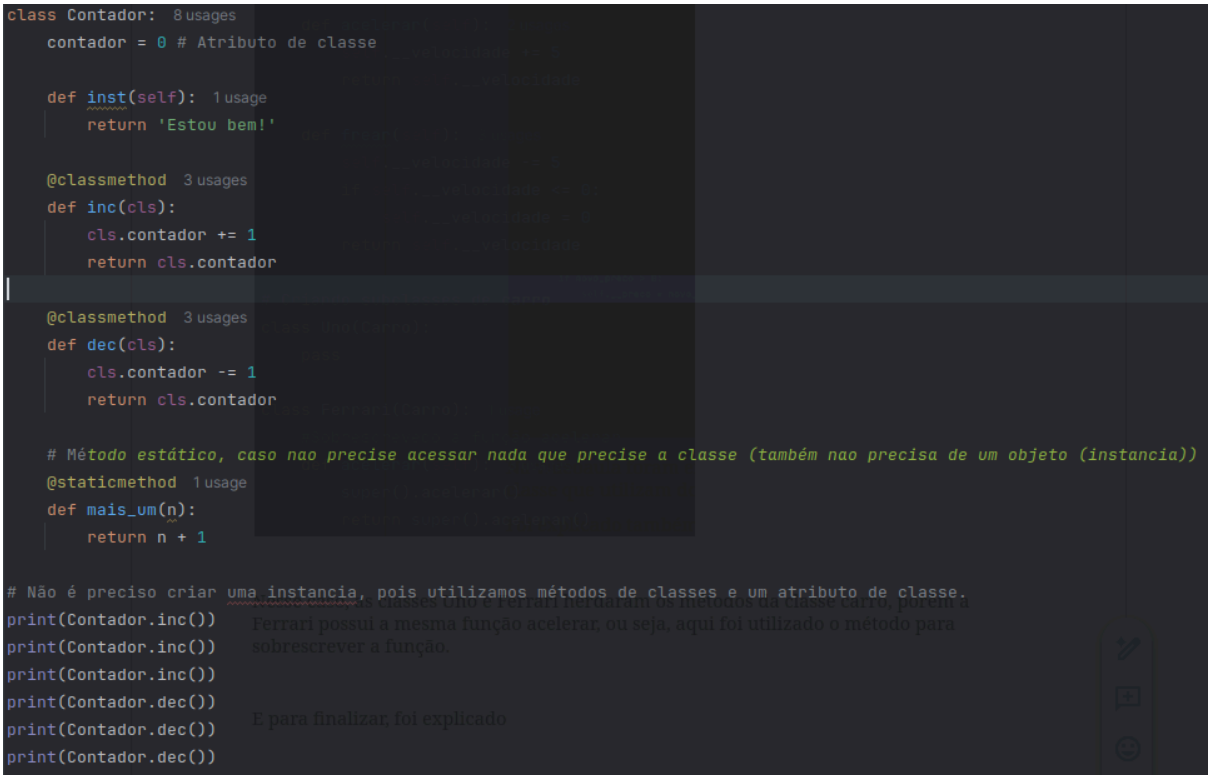
Aqui foi explicado como funciona a herança em python, que é basicamente herdar os atributos e ou métodos da classe superior:



```python
class Carro:  2 usages
    def __init__(self):
        self.__velocidade = 0

    @property
    def velocidade(self):
        return self.__velocidade

    def acelerar(self):  2 usages
        self.__velocidade += 5
        return self.__velocidade

    def frear(self):  3 usages
        self.__velocidade -= 5
        if self.__velocidade <= 0:
            self.__velocidade = 0
        return self.__velocidade

# Criando subclasses de carro
class Uno(Carro):
    pass

class Ferrari(Carro):  1 usage
    #Sobrescrevedo a função acelerar
    def acelerar(self):  3 usages
        super().acelerar()
        return super().acelerar()
```

Nesse caso, as classes Uno e Ferrari herdaram os métodos da classe carro, porém a Ferrari possui a mesma função acelerar, ou seja, aqui foi utilizado o método para sobrescrever a função.

E para finalizar, foi explicado sobre os métodos de classe e os métodos estáticos:



```python
class Contador:  8 usages
    contador = 0 # Atributo de classe

    def inst(self):  1 usage
        return 'Estou bem!'

    @classmethod  3 usages
    def inc(cls):
        cls.contador += 1
        return cls.contador

    @classmethod  3 usages
    def dec(cls):
        cls.contador -= 1
        return cls.contador

    # Método estático, caso nao precise acessar nada que precise a classe (também nao precisa de um objeto (instancia))
    @staticmethod  1 usage
    def mais_um(n):
        return n + 1

# Não é preciso criar uma instancia, pois utilizamos métodos de classes e um atributo de classe.
print(Contador.inc())
print(Contador.inc())
print(Contador.inc())
print(Contador.dec())
print(Contador.dec())
print(Contador.dec())
```

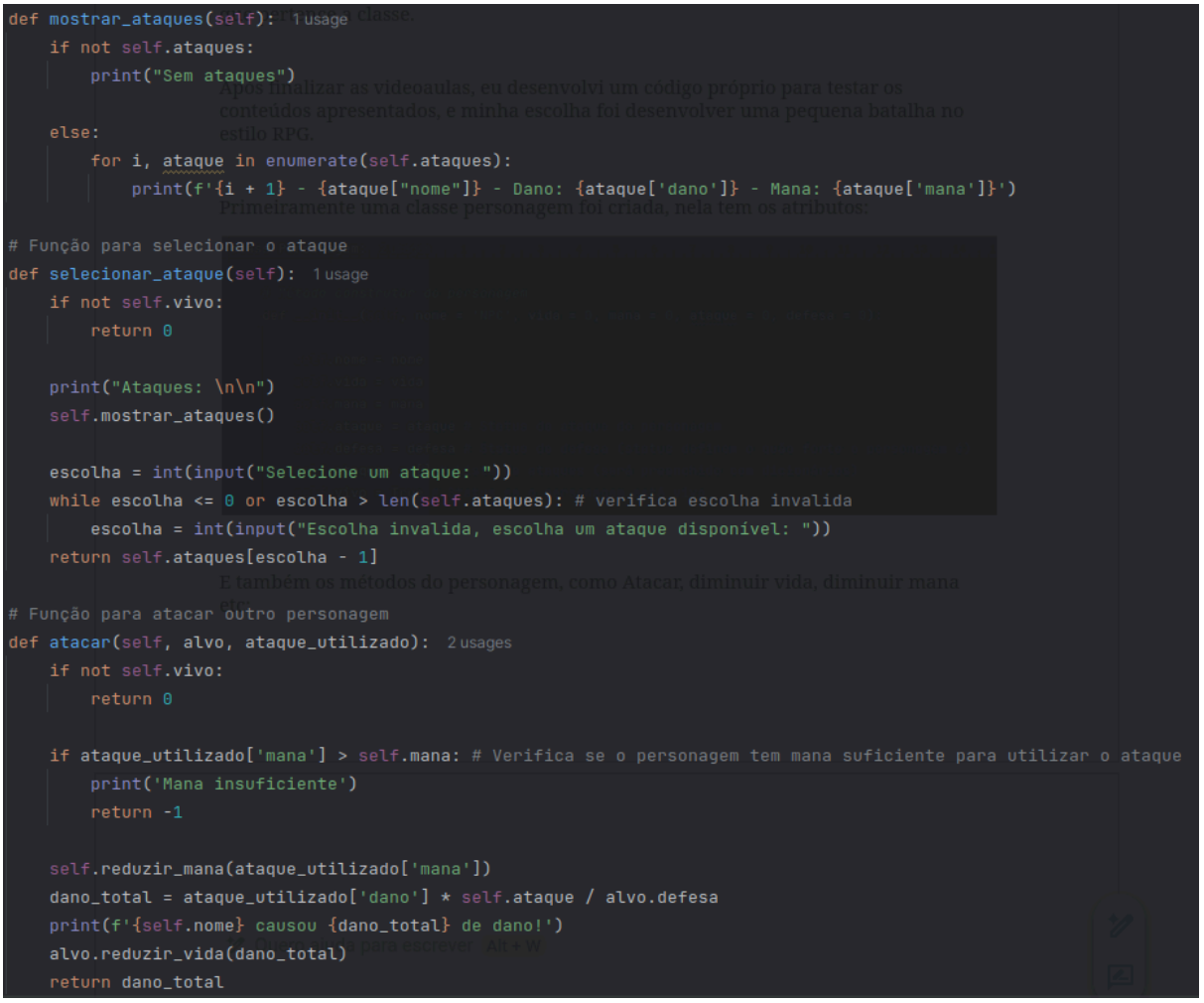
Métodos de classe não dependem de uma instância, são próprios da classe como um todo. Já os estáticos vão além, e devem ser usados quando não precisa acessar NADA que pertence a classe.

Após finalizar as videoaulas, eu desenvolvi um código próprio para testar os conteúdos apresentados, e minha escolha foi desenvolver uma pequena batalha no estilo RPG.

Primeiramente uma classe personagem foi criada, nela tem os atributos:

```python
class Personagem:  2 usages

    # Método construtor do personagem
    def __init__(self, nome = 'NPC', vida = 0, mana = 0, ataque = 0, defesa = 0):

        self.nome = nome
        self.vida = vida
        self.mana = mana
        self.ataque = ataque # Status de ataque do personagem
        self.defesa = defesa # Status de defesa (status definem o quão forte o personagem é)
        self.ataques = [] # Lista de ataques (será preenchido com dicionários)
        self.vivo = True # Status se o personagem está vivo
```

E também os métodos do personagem, como Atacar, diminuir vida, diminuir mana etc:



```python
def mostrar_ataques(self):  1 usage
    if not self.ataques:
        print("Sem ataques")

    else:
        for i, ataque in enumerate(self.ataques):
            print(f'{i + 1} - {ataque["nome"]} - Dano: {ataque['dano']} - Mana: {ataque['mana']}')

# Função para selecionar o ataque
def selecionar_ataque(self):  1 usage
    if not self.vivo:
        return 0

    print("Ataques: \n\n")
    self.mostrar_ataques()

    escolha = int(input("Selecione um ataque: "))
    while escolha <= 0 or escolha > len(self.ataques): # verifica escolha invalida
        escolha = int(input("Escolha invalida, escolha um ataque disponivel: "))
    return self.ataques[escolha - 1]

# Função para atacar outro personagem
def atacar(self, alvo, ataque_utilizado):  2 usages
    if not self.vivo:
        return 0

    if ataque_utilizado['mana'] > self.mana: # Verifica se o personagem tem mana suficiente para utilizar o ataque
        print('Mana insuficiente')
        return -1

    self.reduzir_mana(ataque_utilizado['mana'])
    dano_total = ataque_utilizado['dano'] * self.ataque / alvo.defesa
    print(f'{self.nome} causou {dano_total} de dano!')
    alvo.reduzir_vida(dano_total)
    return dano_total
```

Também criei de forma manual os ataques dos personagens. Essa parte não ficou muito boa e organizada mas não consegui pensar em um jeito melhor de fazer:

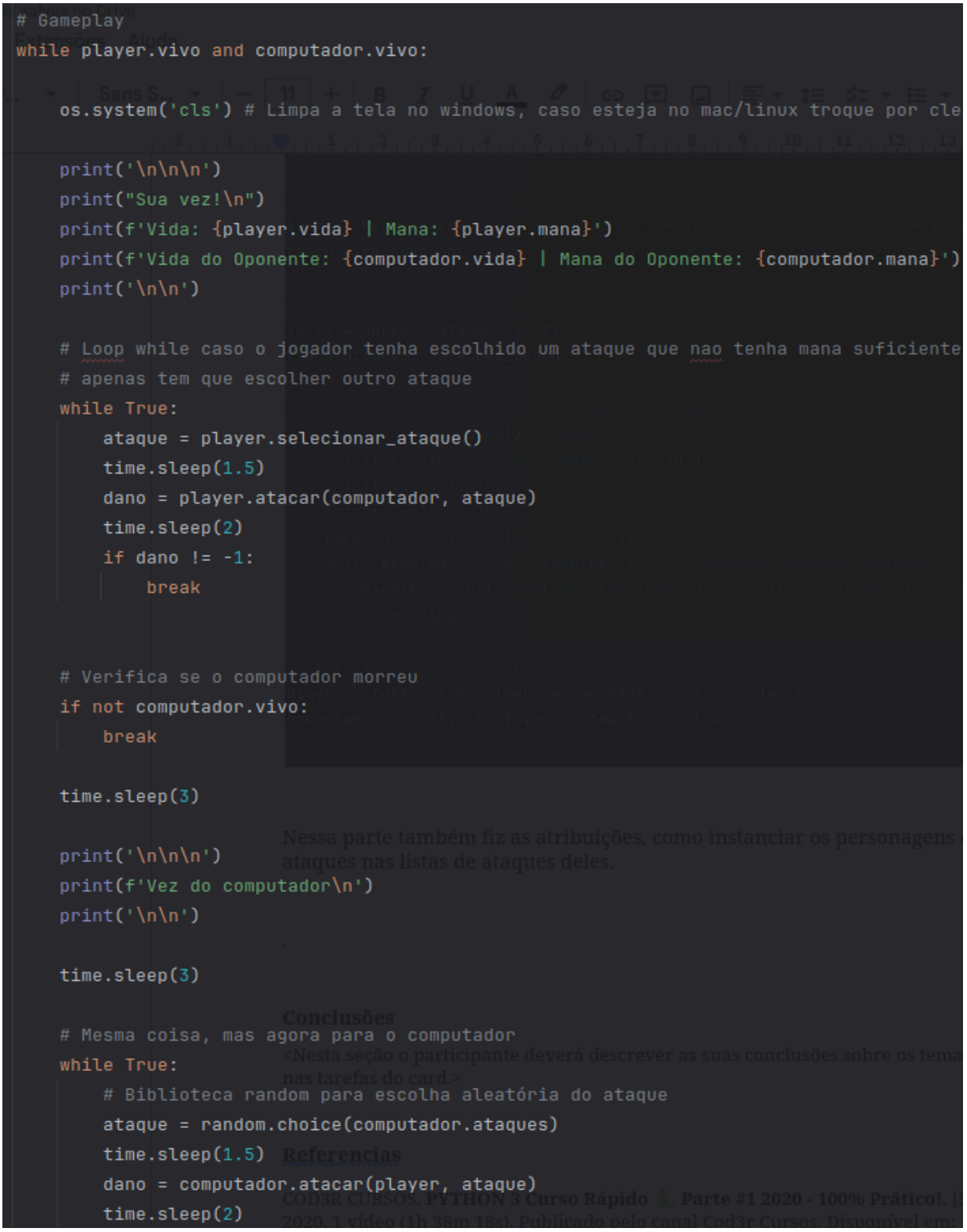
```
#Ataques que eu criei (nao sei como organizar isso de forma legivel)
luffy_ataque_1 = {
    'nome': 'Gomu Gomu no Pistol',
    'dano': 40,
    'mana': 0
}
luffy_ataque_2 = {
    'nome': 'Gomu Gomu no Gatling Gun',
    'dano': 55,
    'mana': 0
}
luffy_ataque_3 = {
    'nome': 'Gomu Gomu no Red Hawk',
    'dano': 80,
    'mana': 40
}
luffy_ataque_4 = {
    'nome': 'Gomu Gomu no Elephant Gun',
    'dano': 115,
    'mana': 65
}
teach_ataque_1 = {
    'nome': 'Soco Sísmico',
    'dano': 45,
    'mana': 0
}
teach_ataque_2 = {
    'nome': 'Liberation',
    'dano': 50,
    'mana': 0
}
teach_ataque_3 = {
    'nome': 'Kurouzu',
    'dano': 75,
    'mana': 35
}
teach_ataque_4 = {
```

Depois fiz uma função para selecionar o personagem que deseja jogar, se é o Luffy ou o Teach:

```python
# Atribuindo os ataques a lista de ataques
ataques_luffy = [luffy_ataque_1, luffy_ataque_2, luffy_ataque_3, luffy_ataque_4]
ataques_teach = [teach_ataque_1, teach_ataque_2, teach_ataque_3, teach_ataque_4]

# Criando os personagens
luffy = Personagem(nome = 'Luffy', vida = 150, mana = 100, ataque = 120, defesa = 80)
teach = Personagem(nome = 'Teach', vida = 250, mana = 200, ataque = 90, defesa = 100)

# Adicionando os ataques
luffy.ataques = ataques_luffy
teach.ataques = ataques_teach

# Função para escolher o personagem para jogar
def escolher_personagem():  1 usage
    print('Escolha seu personagem!!!\n\n\n')
    print('1- Luffy')
    print('2- Teach')
    escolha = int(input('Escolha: '))
    while escolha != 1 and escolha != 2: # Verifica escolha inválida
        escolha = int(input("Escolha invalida, escolha um ataque disponivel: "))
    return escolha

# Atribui o seu personagem e o personagem da cpu
player = luffy if escolher_personagem() == 1 else teach
computador = luffy if player == teach else teach
```

Nessa parte também fiz as atribuições, como instanciar os personagens e adicionar os ataques nas listas de ataques deles.

E por fim uma gameplay simples que consiste apenas em atacar o adversário escolhendo um ataque dos que estão disponíveis, até que você ou o adversário ganhe.



```python
# Gameplay
while player.vivo and computador.vivo:

    os.system('cls') # Limpa a tela no windows, caso esteja no mac/linux troque por cle

    print('\n\n\n')
    print("Sua vez!\n")
    print(f'Vida: {player.vida} | Mana: {player.mana}')
    print(f'Vida do Oponente: {computador.vida} | Mana do Oponente: {computador.mana}')
    print('\n\n')

    # Loop while caso o jogador tenha escolhido um ataque que nao tenha mana suficiente
    # apenas tem que escolher outro ataque
    while True:
        ataque = player.selecionar_ataque()
        time.sleep(1.5)
        dano = player.atacar(computador, ataque)
        time.sleep(2)
        if dano != -1:
            break


    # Verifica se o computador morreu
    if not computador.vivo:
        break

    time.sleep(3)

    print('\n\n\n')
    print(f'Vez do computador\n')
    print('\n\n')

    time.sleep(3)

    # Mesma coisa, mas agora para o computador
    while True:
        # Biblioteca random para escolha aleatória do ataque
        ataque = random.choice(computador.ataques)
        time.sleep(1.5)
        dano = computador.atacar(player, ataque)
        time.sleep(2)
```

Nessa parte utilizei das bibliotecas time para pausar o terminal por alguns segundos e dar mais imersão, além da biblioteca os para limpar a tela (tem que ser ajustado o comando dependendo do seu sistema operacional), e por fim a biblioteca random para escolher de forma aleatória o ataque da cpu.

Conclusões

As videoaulas serviram bem para ampliar meus conhecimentos de programação apresentando estruturas de dados que eu ainda não conhecia. Além disso, as videoaulas foram excelentes para contextualizar, pois o apresentador sempre dava exemplos de onde poderiam ser utilizadas as estruturas, mesmo as mais complexas.

Quanto à atividade de desenvolver meu próprio código também foi útil, pois pude testar os conhecimentos adquiridos e praticar um pouco da minha lógica que estava travada nas últimas semanas.

Referencias

COD3R CURSOS. **PYTHON 3 Curso Rápido 🐍 Parte #1 2020 - 100% Prático!**. [S. l.], 11 maio 2020. 1 vídeo (1h 38m 18s). Publicado pelo canal Cod3r Cursos. Disponível em: https://www.youtube.com/watch?v=iq7JLIH-sV0. Acesso em: 1 ago. 2025.

COD3R CURSOS. **PYTHON 3 Curso Rápido 🐍 Parte #2 2020 - 100% Prático!**. [S. l.], 18 maio 2020. 1 vídeo (1h 41m 13s). Publicado pelo canal Cod3r Cursos. Disponível em: https://www.youtube.com/watch?v=oUrBHiT-lzo. Acesso em: 1 ago. 2025.

KUMAR, Pankaj. **Python time.sleep(): How to Pause Execution in Python Scripts**. DigitalOcean Community, 3 ago. 2022. Disponível em: https://www.digitalocean.com/community/tutorials/python-time-sleep. Acesso em: 1 ago. 2025.