Roteiro de Atividade Prática

Nome: Turma: .

**Atividade 1: Classe Simples - Lampada**

**Objetivos**

Antes de resolver este exercício, é importante entender o que é uma classe e como criar objetos a partir dela. Uma classe é definida usando a palavra-chave class, seguida do nome da classe e dois-pontos. Dentro da classe, podemos definir métodos (funções) e atributos (dados).

**Descrição do exercício:**

Ulizando a linguagem de programação Python, crie uma classe Lampada com um atributo estado (ligado/desligado) e um método para alterar seu estado.

**Tempo estimado:** 15 minutos

**Lista de materiais**

* Computador com internet;
* Caderno para anotações;
* 1 caneta.

**Procedimento experimental**

1. Analise o exemplo-base para criação do código:

class Lampada:

def \_\_init\_\_(/self):

self.estado = False # Começa desligada

# Uso da classe

lampada = Lampada()

print(lampada.alterar\_estado()) # Liga a lâmpada

print(lampada.alterar\_estado()) # Desliga a lâmpada

1. Agora, crie uma definição para alterar o seu estado entre ligada e desligada de acordo com o exemplo disponibilizado.
2. Anote o código desenvolvido nas linhas seguintes e envie por meio do AVA.

|  |
| --- |
|  |
|  |
| Def alterar\_estado(self):  self.estado = not self.estado  return “Ligada” if self.estado else “Desligada” |
|  |
|  |

**Atividade 2: Encapsulamento com Atributos Privados**

**Objetivos**

Entenda que encapsulamento em Python é mais uma convenção do que uma restrição técnica. Atributos privados são definidos com um sublinhado duplo (\_\_). Eles não são acessíveis diretamente de fora da classe, promovendo o encapsulamento.

**Descrição do exercício:**

Crie uma classe Contador que mantenha um valor interno privado e tenha métodos para incrementar, decrementar e obter esse valor.

**Tempo estimado:** 15 minutos

**Procedimento experimental**

1. Analise o exemplo base para criação do código:

class Contador:

def \_\_init\_\_(self):

self.\_\_valor = 0

# Uso da classe

contador = Contador()  
contador.incrementar()

contador.incrementar()

print(contador.get\_valor()) # Deve mostrar 2

contador.decrementar()

print(contador.get\_valor()) # Deve mostrar 1

1. Agora, a partir do código analisado, crie as definições para incrementar, decrementar e obter esse valor.
2. Anote o código desenvolvido nas linhas seguintes e envie por meio do AVA.

|  |
| --- |
|  |
| Class contador:  def \_\_init\_\_(self):  self.\_\_valor = 0  def incrementar(self):  self.\_\_valor += 1  def decremetar(self):  if self.\_\_valor > 0:  self.\_\_valor -= 1  def get\_valor(self):  return self.\_\_valor |
|  |
|  |
|  |

**Atividade 3: Uso de Getters e Setters**

**Objetivos**

Getters e Setters são métodos usados para obter e definir o valor de atributos privados. Eles são úteis para adicionar lógica adicional durante a obtenção ou a definição de um valor, como validações.

**Descrição do exercício:**

Crie uma classe Termometro que armazene a temperatura em graus Celsius, mas permita definir e obter a temperatura em Fahrenheit.

**Tempo estimado:** 10 minutos

**Procedimento experimental**

1. Analise o exemplo-base para criação do código:

class Termometro:

def \_\_init\_\_(self, temperatura=0):

self.\_\_temperatura\_celsius = temperatura

def get\_temperatura\_fahrenheit(self):

return (self.\_\_temperatura\_celsius \* 9/5) + 32

# Uso da classe

termometro = Termometro()

termometro.set\_temperatura\_fahrenheit(68)

print(termometro.get\_temperatura\_fahrenheit()) # Deve mostrar 68

1. Agora, a partir do código analisado, crie as definições para que a temperatura também possa ser obtida em Celsius.
2. Anote o código desenvolvido nas linhas seguintes e envie por meio do AVA.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |