Trabajo 01: Principios de POO en Python

Alumno: Puma Huanca Anthony Rusbel
Docente: Ing. Coyla Idme Leonel
Lenguajes de programación II – FINESI
Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Pregunta 0: Aplicar los principios básicos de la programación orientada a objetos (POO) en Python para resolver un problema matemático (el factorial) utilizando clases, atributos, métodos y objetos.

Instrucciones:

1. Crea una clase llamada `Factorial`.
2. Define un atributo privado `__numero` que almacene el número del cual se calculará el factorial.
3. Crea un método constructor (`__init__`) que reciba el número y lo asigne al atributo.
4. Implementa un método público llamado `calcular()` que:
5. Calcule el factorial del número almacenado.
6. Devuelva el resultado.
7. Crea otro método llamado `mostrar_resultado()` que imprima el número y su factorial.
8. Finalmente, crea un objeto de la clase y muestra el resultado.

Resultado esperado:

```
El factorial de 5 es 120
```

Codigo Python:

```python
class Factorial:
    def __init__(self, numero):
        self.__numero = numero

```

```
    def calcular(self):
        if self.__numero < 0:
            return "El factorial no est  definido para n meros
               negativos"
        elif self.__numero == 0 or self.__numero == 1:
            return 1
        else:
            factorial = 1
            for i in range(2, self.__numero + 1):
                factorial *= i
            return factorial

    def mostrar_resultado(self):
        resultado = self.calcular()
        print(f"El factorial de {self.__numero} es: {resultado}")

if __name__ == "__main__":
    factorial_obj = Factorial(5)
    factorial_obj.mostrar_resultado()
```

Listing 1: Código Pregunta 0

Pregunta 1: Atributos privados y métodos getters/setters

Enunciado:

Crea una clase llamada `CuentaBancaria` que tenga los siguientes atributos privados:

`__titular` (nombre del titular)

`__saldo` (cantidad de dinero en la cuenta)

La clase debe cumplir con lo siguiente:

1. Tener un constructor que reciba el nombre del titular y el saldo inicial.
2. Implementar métodos `getters` y `setters` para acceder y modificar el saldo, asegurando que:
3. No se permita asignar un saldo negativo.
4. Incluir un método público `mostrar_datos()` que muestre el nombre del titular y el saldo actual.

Resultado esperado:

```
Titular: Ana López
Saldo actual: 1500.0
```

Codigo Python:

```python
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.__titular = titular
        self.__saldo = saldo if saldo >= 0 else 0

    def get_titular(self):
        return self.__titular

    def set_titular(self, titular):
        self.__titular = titular

    def get_saldo(self):
        return self.__saldo

    def set_saldo(self, saldo):
        if saldo >= 0:
            self.__saldo = saldo
        else:
            print("Error: No se puede asignar un saldo negativo.")

    def mostrar_datos(self):
        print(f"Titular: {self.__titular}")
        print(f"Saldo: S/. {self.__saldo:.2f}")

if __name__ == "__main__":
    cuenta1 = CuentaBancaria("Ana Perez", 5000)
    cuenta1.mostrar_datos()
    cuenta1.set_saldo(7000)
    cuenta1.mostrar_datos()
    cuenta1.set_saldo(-100)
```

Listing 2: Código Pregunta 1

Pregunta 2: Uso de `@property` para encapsulamiento moderno

Enunciado:

Crea una clase llamada `Producto` con los atributos privados:

`__nombre`

`__precio`

La clase debe:

1. Definir propiedades con `@property` y `@setter` para acceder y modificar el atributo `precio`.

2. Evitar que se establezca un precio menor o igual a 0.

3. Tener un método `mostrar_producto()` que muestre el nombre y el precio.

Resultado esperado:

```
Producto: Laptop
Precio: $1200.00
```

Codigo Python:

```python
class Producto:
    def __init__(self, nombre, precio):
        self.__nombre = nombre
        self.__precio = precio if precio > 0 else 1.0

    @property
    def nombre(self):
        return self.__nombre

    @nombre.setter
    def nombre(self, nombre):
        self.__nombre = nombre

    @property
    def precio(self):
        return self.__precio

    @precio.setter
    def precio(self, precio):
        if precio > 0:
            self.__precio = precio
        else:
            print("Error: El precio debe ser mayor a 0455.")

    def mostrar_producto(self):
        print(f"Producto: {self.__nombre}")
        print(f"Precio: S/. {self.__precio:.2f}")

if __name__ == "__main__":
    producto1 = Producto("Laptop", 2500.00)
    producto1.mostrar_producto()
    producto1.precio = 3000.00
    producto1.mostrar_producto()
    producto1.precio = -500
```

Listing 3: Código Pregunta 2

Pregunta 3: Implementar la solución al problema clásico de las Torres de Hanoi utilizando programación orientada a objetos en Python y mostrar gráficamente el proceso de resolución usando la biblioteca `Tkinter`.

Enunciado:

1. Crea una clase llamada `TorresDeHanoi` que tenga los siguientes atributos:
2. `num_discos`: número de discos.
3. `torres`: estructura para representar las tres torres (por ejemplo, listas que contienen discos).
4. La clase debe incluir los siguientes métodos:
5. `__init__(self, num_discos)`: constructor que inicialice las torres con todos los discos en la torre 1 (de mayor a menor).
6. `mover_disco(self, origen, destino)`: método para mover el disco superior de una torre a otra, actualizando la estructura interna.
7. `resolver(self, n, origen, destino, auxiliar)`: método recursivo que resuelva el problema para n discos, moviendo discos entre torres.
8. `mostrar_estado(self)`: método para mostrar el estado actual de las torres (puede imprimir en consola o actualizar la interfaz gráfica).
9. Métodos necesarios para integrar con `Tkinter` y actualizar la vista gráfica tras cada movimiento.
10. Crea una interfaz gráfica con `Tkinter` que visualice:
11. Las tres torres.
12. Los discos apilados en las torres.
13. Animación o actualización visual en cada movimiento del algoritmo.
14. Finalmente, crea un objeto de la clase `TorresDeHanoi` con un número de discos definido (por ejemplo, 4), y ejecuta el método para resolver el problema mostrando la animación gráfica paso a paso.

Codigo Python:

```
1 import tkinter as tk
2 from tkinter import messagebox
3 import time
4
```

```
5
6  class TorresDeHanoi:
7      """
8      Clase para resolver el problema de las Torres de Hanoi
9      con visualizaci n  gr fica usando Tkinter
10     """
11
12     def __init__(self, num_discos, canvas, ventana):
13         """
14         Constructor que inicializa las torres y la interfaz gr fica
15
16         Args:
17             num_discos (int): N mero de discos
18             canvas (tk.Canvas): Canvas de Tkinter para dibujar
19             ventana (tk.Tk): Ventana principal de Tkinter
20         """
21         self.num_discos = num_discos
22         self.canvas = canvas
23         self.ventana = ventana
24
25         # Inicializar las tres torres (listas)
26         # Torre 1 tiene todos los discos (de mayor a menor)
27         self.torres = {
28             1: list(range(num_discos, 0, -1)),
29             2: [],
30             3: []
31         }
32
33         # Configuraci n  gr fica
34         self.ancho_canvas = 800
35         self.alto_canvas = 400
36         self.base_y = 350
37         self.torre_x = {1: 150, 2: 400, 3: 650}
38         self.altura_disco = 20
39         self.ancho_disco_base = 150
40         self.movimientos = 0
41
42         # Dibujar estado inicial
43         self.dibujar_torres()
44
45     def dibujar_torres(self):
46         """
47         Dibuja el estado actual de las torres en el canvas
48         """
49         self.canvas.delete("all")
50
51         # Dibujar bases y postes
52         for i in range(1, 4):
```

```
            x = self.torre_x[i]
            # Poste
            self.canvas.create_rectangle(
                x - 5, self.base_y - 200, x + 5, self.base_y,
                fill="brown", outline="black"
            )
            # Base
            self.canvas.create_rectangle(
                x - 100, self.base_y, x + 100, self.base_y + 10,
                fill="saddlebrown", outline="black"
            )
            # Etiqueta de torre
            self.canvas.create_text(
                x, self.base_y + 30,
                text=f"Torre {i}",
                font=("Arial", 14, "bold")
            )

        # Dibujar discos
        colores = ["red", "orange", "yellow", "green", "blue", "
           purple", "pink", "cyan"]

        for torre_num, discos in self.torres.items():
            x_torre = self.torre_x[torre_num]
            for idx, disco in enumerate(discos):
                # Calcular ancho del disco proporcionalmente
                ancho = (disco * self.ancho_disco_base) // self.
                   num_discos
                y_pos = self.base_y - (idx + 1) * self.altura_disco

                color = colores[(disco - 1) % len(colores)]

                self.canvas.create_rectangle(
                    x_torre - ancho // 2, y_pos,
                    x_torre + ancho // 2, y_pos + self.altura_disco,
                    fill=color, outline="black", width=2
                )

                # N mero del disco
                self.canvas.create_text(
                    x_torre, y_pos + self.altura_disco // 2,
                    text=str(disco),
                    font=("Arial", 10, "bold"),
                    fill="white"
                )

    def mover_disco(self, origen, destino):
        """
```

```
99          Mueve el disco superior de una torre a otra
100
101         Args:
102             origen (int): Torre de origen (1, 2 o 3)
103             destino (int): Torre de destino (1, 2 o 3)
104         """
105         if self.torres[origen]:
106             disco = self.torres[origen].pop()
107             self.torres[destino].append(disco)
108             self.movimientos += 1
109
110     def mostrar_estado(self):
111         """
112         Muestra el estado actual de las torres en consola
113         """
114         print(f"\nMovimiento #{self.movimientos}")
115         print(f"Torre 1: {self.torres[1]}")
116         print(f"Torre 2: {self.torres[2]}")
117         print(f"Torre 3: {self.torres[3]}")
118
119     def resolver(self, n, origen, destino, auxiliar):
120         """
121         M todo recursivo que resuelve el problema de las Torres de
               Hanoi
122
123         Args:
124             n (int): N mero de discos a mover
125             origen (int): Torre de origen
126             destino (int): Torre de destino
127             auxiliar (int): Torre auxiliar
128         """
129         if n > 0:
130             # Mover n-1 discos de origen a auxiliar usando destino
131             self.resolver(n - 1, origen, auxiliar, destino)
132
133             # Mover el disco m s grande de origen a destino
134             self.mover_disco(origen, destino)
135             self.mostrar_estado()
136             self.dibujar_torres()
137             self.ventana.update()
138             time.sleep(0.5)  # Pausa para visualizar el movimiento
139
140             # Mover n-1 discos de auxiliar a destino usando origen
141             self.resolver(n - 1, auxiliar, destino, origen)
142
143     def iniciar_resolucion(self):
144         """
145         Inicia el proceso de resoluci n del problema
```

```
146         """
147         self.movimientos = 0
148         print(f"\n{'=' * 50}")
149         print(f"Resolviendo Torres de Hanoi con {self.num_discos}
                discos")
150         print(f"{'=' * 50}")
151         self.resolver(self.num_discos, 1, 3, 2)
152         messagebox.showinfo(
153             "  Completado  !",
154             f"Torres de Hanoi resueltas en {self.movimientos}
                movimientos"
155         )
156
157
158 def crear_interfaz():
159     """
160     Crea la interfaz gr fica con Tkinter
161     """
162     ventana = tk.Tk()
163     ventana.title("Torres de Hanoi - Visualizaci n")
164     ventana.geometry("820x550")
165     ventana.resizable(False, False)
166
167     # Frame superior para controles
168     frame_control = tk.Frame(ventana, bg="lightgray", height=80)
169     frame_control.pack(fill=tk.X, padx=10, pady=10)
170
171     # Etiqueta
172     label = tk.Label(
173         frame_control,
174         text="Torres de Hanoi - Soluci n Recursiva",
175         font=("Arial", 16, "bold"),
176         bg="lightgray"
177     )
178     label.pack(pady=5)
179
180     # Canvas para dibujar las torres
181     canvas = tk.Canvas(ventana, width=800, height=450, bg="white")
182     canvas.pack(padx=10, pady=5)
183
184     # Variable para almacenar el objeto TorresDeHanoi
185     torres_obj = None
186
187     def iniciar():
188         """Funci n para iniciar la resoluci n"""
189         nonlocal torres_obj
190         try:
191             num_discos = int(entry_discos.get())
```

```
192             if num_discos < 1 or num_discos > 8:
193                 messagebox.showerror("Error", "Ingrese un n mero
                        entre 1 y 8")
194                 return
195 
196             btn_iniciar.config(state=tk.DISABLED)
197             entry_discos.config(state=tk.DISABLED)
198 
199             # Crear objeto y resolver
200             torres_obj = TorresDeHanoi(num_discos, canvas, ventana)
201             torres_obj.iniciar_resolucion()
202 
203             btn_iniciar.config(state=tk.NORMAL)
204             entry_discos.config(state=tk.NORMAL)
205 
206         except ValueError:
207             messagebox.showerror("Error", "Ingrese un n mero
                    v lido")
208 
209     # Controles
210     frame_input = tk.Frame(frame_control, bg="lightgray")
211     frame_input.pack()
212 
213     tk.Label(
214         frame_input,
215         text="N mero de discos:",
216         font=("Arial", 12),
217         bg="lightgray"
218     ).pack(side=tk.LEFT, padx=5)
219 
220     entry_discos = tk.Entry(frame_input, width=5, font=("Arial", 12)
            )
221     entry_discos.insert(0, "4")
222     entry_discos.pack(side=tk.LEFT, padx=5)
223 
224     btn_iniciar = tk.Button(
225         frame_input,
226         text="Resolver",
227         command=iniciar,
228         font=("Arial", 12, "bold"),
229         bg="green",
230         fg="white",
231         padx=20
232     )
233     btn_iniciar.pack(side=tk.LEFT, padx=10)
234 
235     # Dibujar estado inicial con 4 discos
236     torres_inicial = TorresDeHanoi(4, canvas, ventana)
```

```
237
238     ventana.mainloop()
239
240
241 # Ejecutar la aplicaci n
242 if __name__ == "__main__":
243     crear_interfaz()
```

Listing 4: Código Pregunta 3