

Universidad Nacional del Altiplano  
Facultad de Ingeniería Estadística e Informática  
Docente: Leonel Coyla Idme  
Alumno: Clyde Neil Paricahua Pari

## Trabajo Encargado N°2 - Programación Orientada a Objetos

### Pregunta 0: Cálculo de Factorial con POO

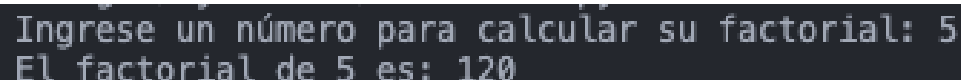
#### Descripción del Problema

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. Se debe crear una clase **Factorial** con un atributo privado para el número, un constructor, un método para calcular el factorial y otro para mostrar el resultado.

#### Código Fuente en Python

```
1 class Factorial:
2     def __init__(self, numero):
3         self.__numero = None
4         self.set_numero(numero)
5
6     def set_numero(self, numero):
7         if isinstance(numero, int) and numero >= 0:
8             self.__numero = numero
9         else:
10            raise ValueError("El n mero debe ser un entero positivo o cero.")
11
12    def get_numero(self):
13        return self.__numero
14
15    def calcular(self):
16        resultado = 1
17        for i in range(2, self.__numero + 1):
18            resultado *= i
19        return resultado
20
21    def mostrar_resultado(self):
22        print(f"El factorial de {self.get_numero()} es: {self.calcular()}")
23
24
25 if __name__ == "__main__":
26     numero = int(input("Ingrese un n mero para calcular su factorial: "))
27     f = Factorial(numero)
28     f.mostrar_resultado()
```

#### Ejecución del Programa



```
Ingrese un número para calcular su factorial: 5
El factorial de 5 es: 120
```

Figura 1:

El factorial de 5 es: 120

## Pregunta 1: Encapsulamiento con Getters y Setters

### Descripción del Problema

Crear una clase `CuentaBancaria` con atributos privados para el titular y el saldo. La clase debe tener un constructor y métodos *getters* y *setters* para acceder y modificar el saldo, con la restricción de no permitir la asignación de saldos negativos. También debe incluir un método para mostrar los datos de la cuenta.

### Código Fuente en Python

```
1 class CuentaBancaria:
2     def __init__(self, titular, saldo):
3         self.__titular = titular
4         self.__saldo = 0
5         self.set_saldo(saldo)
6
7     def get_titular(self):
8         return self.__titular
9
10    def get_saldo(self):
11        return self.__saldo
12
13    def set_saldo(self, saldo):
14        if saldo >= 0:
15            self.__saldo = saldo
16        else:
17            print("El saldo no puede ser negativo.")
18
19    def mostrar_datos(self):
20        print(f"Titular: {self.get_titular()} - Saldo : S/ {self.get_saldo():.2f}")
21
22
23 if __name__ == "__main__":
24     titular = input("Ingrese el nombre del titular: ")
25     saldo_inicial = float(input("Ingrese el saldo inicial: "))
26
27     cuenta = CuentaBancaria(titular, saldo_inicial)
28
29     if cuenta.get_saldo() == 0 and saldo_inicial < 0:
30         print("No se puede crear la cuenta con saldo negativo.")
31     else:
32         while True:
33             print("\nMEN    CUENTA BANCARIA ")
34             print("1. Mostrar datos de la cuenta")
35             print("2. Modificar saldo")
36             print("3. Salir")
37             opcion = input("Seleccione una opción: ")
38
39             if opcion == "1":
40                 cuenta.mostrar_datos()
41             elif opcion == "2":
42                 nuevo_saldo = float(input("Ingrese el nuevo saldo: "))
43                 cuenta.set_saldo(nuevo_saldo)
44             elif opcion == "3":
45                 print("Saliendo del programa...")
46                 break
47             else:
48                 print("Opción no válida. Intente nuevamente.")
```

## Ejecución del Programa

```
Ingrese el nombre del titular: Ana Maria
Ingrese el saldo inicial: 5000

MENÚ CUENTA BANCARIA
1. Mostrar datos de la cuenta
2. Modificar saldo
3. Salir
Seleccione una opción: 2
Ingrese el nuevo saldo: 7000

MENÚ CUENTA BANCARIA
1. Mostrar datos de la cuenta
2. Modificar saldo
3. Salir
Seleccione una opción: 2
Ingrese el nuevo saldo: -100
El saldo no puede ser negativo.

MENÚ CUENTA BANCARIA
1. Mostrar datos de la cuenta
2. Modificar saldo
3. Salir
Seleccione una opción: 3
Saliendo del programa...
```

Figura 2:

## Pregunta 2: Uso de @property para Encapsulamiento Moderno

### Descripción del Problema

Crear una clase `Producto` con atributos privados para el nombre y el precio. Se deben utilizar los decoradores `@property` y `@setter` para acceder y modificar el precio, evitando que se le asigne un valor menor o igual a cero. Además, se requiere un método que muestre la información del producto.

### Código Fuente en Python

```
1 class Producto:
2     def __init__(self, nombre, precio):
3         self.__nombre = nombre
4         self.__precio = 0
```

```
5         self.precio = precio
6
7     @property
8     def precio(self):
9         return self.__precio
10
11     @precio.setter
12     def precio(self, valor):
13         if valor > 0:
14             self.__precio = valor
15         else:
16             print("El precio debe ser mayor a 0.")
17
18
19     def mostrar_producto(self):
20         print(f"Producto: {self.__nombre} - Precio: S/ {self.__precio:.2f}")
21
22
23 if __name__ == "__main__":
24     nombre = input("Ingrese el nombre del producto: ")
25     precio_inicial = float(input("Ingrese el precio del producto: "))
26
27     producto = Producto(nombre, precio_inicial)
28
29     while True:
30         print("\nMEN  PRODUCTO")
31         print("1. Mostrar producto")
32         print("2. Modificar precio")
33         print("3. Salir")
34         opcion = input("Seleccione una opción: ")
35
36         if opcion == "1":
37             producto.mostrar_producto()
38         elif opcion == "2":
39             nuevo_precio = float(input("Ingrese el nuevo precio: "))
40             producto.precio = nuevo_precio
41         elif opcion == "3":
42             print("Saliendo del programa...")
43             break
44         else:
45             print("Opción no válida. Intente nuevamente.")
```



```
2 class TorresDeHanoi:
3     def __init__(self, num_discos):
4         self.num_discos = num_discos
5         self.torres = [list(range(num_discos, 0, -1)), [], []]
6
7     def mover_disco(self, origen, destino):
8         disco = self.torres[origen].pop()
9         self.torres[destino].append(disco)
10        self.mostrar_estado()
11        input("\nPresione Enter para continuar...\n")
12
13    def resolver(self, n=None, origen=0, destino=2, auxiliar=1):
14        if n is None:
15            n = self.num_discos
16        if n == 1:
17            print(f"Mover disco 1 de Torre {origen + 1} a Torre {destino + 1}")
18            self.mover_disco(origen, destino)
19        else:
20            self.resolver(n - 1, origen, auxiliar, destino)
21            print(f"Mover disco {n} de Torre {origen + 1} a Torre {destino + 1}")
22            self.mover_disco(origen, destino)
23            self.resolver(n - 1, auxiliar, destino, origen)
24
25    def mostrar_estado(self):
26        print("\nEstado de las torres:")
27        for i, torre in enumerate(self.torres):
28            print(f"Torre {i + 1}: {torre}")
29
30
31 if __name__ == "__main__":
32     num_discos = int(input("Ingrese el n mero de discos: "))
33     juego = TorresDeHanoi(num_discos)
34     print("Estado inicial:")
35     juego.mostrar_estado()
36     input("\nPresione Enter para comenzar la resoluci n...\n")
37     juego.resolver()
38     print("\n Problema resuelto!")
```

## Ejecución del Programa

```
Ingrese el número de discos: 4
Estado inicial:

Estado de las torres:
Torre 1: [4, 3, 2, 1]
Torre 2: []
Torre 3: []

Presione Enter para comenzar la resolución...

Mover disco 1 de Torre 1 a Torre 2

Estado de las torres:
Torre 1: [4, 3, 2]
Torre 2: [1]
Torre 3: []

Presione Enter para continuar...

Mover disco 2 de Torre 1 a Torre 3

Estado de las torres:
Torre 1: [4, 3]
Torre 2: [1]
Torre 3: [2]

Presione Enter para continuar...

Mover disco 1 de Torre 2 a Torre 3

Estado de las torres:
Torre 1: [4, 3]
Torre 2: []
Torre 3: [2, 1]

Presione Enter para continuar...

Mover disco 3 de Torre 1 a Torre 2

Estado de las torres:
Torre 1: [4]
Torre 2: [3]
Torre 3: [2, 1]

Presione Enter para continuar...

Mover disco 1 de Torre 3 a Torre 1

Estado de las torres:
Torre 1: [4, 1]
Torre 2: [3]
Torre 3: [2]

Presione Enter para continuar...

Mover disco 2 de Torre 3 a Torre 2
```

Figura 4:

```
Estado de las torres:
Torre 1: [4, 1]
Torre 2: [3, 2]
Torre 3: []

Presione Enter para continuar...

Mover disco 1 de Torre 1 a Torre 2

Estado de las torres:
Torre 1: [4]
Torre 2: [3, 2, 1]
Torre 3: []

Presione Enter para continuar...

Mover disco 4 de Torre 1 a Torre 3

Estado de las torres:
Torre 1: []
Torre 2: [3, 2, 1]
Torre 3: [4]

Presione Enter para continuar...

Mover disco 1 de Torre 2 a Torre 3

Estado de las torres:
Torre 1: []
Torre 2: [3, 2]
Torre 3: [4, 1]

Presione Enter para continuar...

Mover disco 2 de Torre 2 a Torre 1

Estado de las torres:
Torre 1: [2]
Torre 2: [3]
Torre 3: [4, 1]

Presione Enter para continuar...

Mover disco 1 de Torre 3 a Torre 1

Estado de las torres:
Torre 1: [2, 1]
Torre 2: [3]
Torre 3: [4]

Presione Enter para continuar...
```



```
Presione Enter para continuar...

Mover disco 1 de Torre 1 a Torre 2

Estado de las torres:
Torre 1: [2]
Torre 2: [1]
Torre 3: [4, 3]

Presione Enter para continuar...

Mover disco 2 de Torre 1 a Torre 3

Estado de las torres:
Torre 1: []
Torre 2: [1]
Torre 3: [4, 3, 2]

Presione Enter para continuar...

Mover disco 1 de Torre 2 a Torre 3

Estado de las torres:
Torre 1: []
Torre 2: []
Torre 3: [4, 3, 2, 1]

Presione Enter para continuar...

¡Problema resuelto!
```

Figura 6:

## Pregunta 4: Torres de Hanoi con Interfaz Grafica

### Código Fuente en Python

```
1
2 import tkinter as tk
3
4 class TorresDeHanoi:
5     def __init__(self, num_discos, canvas):
6         self.num_discos = num_discos
7         self.torres = [list(range(num_discos, 0, -1)), [], []]
8         self.canvas = canvas
9         self.ancha_torre = 20
10        self.ancha_disco = 30
11        self.alto_disco = 20
12        self.margen = 100
13        self.posiciones_torres = [
14            self.margen,
15            self.canvas.winfo_width() // 2,
16            self.canvas.winfo_width() - self.margen
17        ]
18        self.init_grafico()
19
20    def init_grafico(self):
21        self.canvas.delete("all")
22        for i, x in enumerate(self.posiciones_torres):
23            self.canvas.create_rectangle(
24                x - self.ancha_torre//2,
25                self.canvas.winfo_height() - (self.num_discos + 1) * self.alto_disco,
26                x + self.ancha_torre//2,
27                self.canvas.winfo_height(),
28                fill="brown"
29            )
30            self.canvas.create_text(x, self.canvas.winfo_height() - 10, text=str(i+1),
31                                   font=("Arial", 14))
32        self.actualizar_grafico()
33
34    def dibujar_disco(self, disco, torre, nivel):
35        x = self.posiciones_torres[torre]
36        y = self.canvas.winfo_height() - (nivel + 1) * self.alto_disco
37        ancho = self.ancha_disco + disco * 10
38        color = f"#{(disco*40)%256:02x}{(100 + disco*20)%256:02x}{(200 - disco*20)%256:02x}"
39        self.canvas.create_rectangle(
40            x - ancho//2, y - self.alto_disco, x + ancho//2, y,
41            fill=color, outline="black"
42        )
43
44    def actualizar_grafico(self):
45        self.canvas.delete("all")
46        for i, x in enumerate(self.posiciones_torres):
47            self.canvas.create_rectangle(
48                x - self.ancha_torre//2,
49                self.canvas.winfo_height() - (self.num_discos + 1) * self.alto_disco,
50                x + self.ancha_torre//2,
51                self.canvas.winfo_height(),
52                fill="brown"
53            )
54            self.canvas.create_text(x, self.canvas.winfo_height() - 10, text=str(i+1),
55                                   font=("Arial", 14))
56        for torre_index, torre in enumerate(self.torres):
```

```
55         for nivel, disco in enumerate(torre):
56             self.dibujar_disco(disco, torre_index, nivel)
57     self.canvas.update()
58
59     def mover_disco(self, origen, destino):
60         disco = self.torres[origen].pop()
61         self.torres[destino].append(disco)
62         self.actualizar_grafico()
63         input(f"Mover disco {disco} de Torre {origen+1} a Torre {destino+1}. Presione
64             Enter para continuar...")
65
66     def resolver(self, n=None, origen=0, destino=2, auxiliar=1):
67         if n is None:
68             n = self.num_discos
69         if n == 1:
70             self.mover_disco(origen, destino)
71         else:
72             self.resolver(n-1, origen, auxiliar, destino)
73             self.mover_disco(origen, destino)
74             self.resolver(n-1, auxiliar, destino, origen)
75
76 if __name__ == "__main__":
77     num_discos = 4
78     root = tk.Tk()
79     root.title("Torres de Hanoi")
80     canvas = tk.Canvas(root, width=600, height=400, bg="white")
81     canvas.pack()
82     root.update()
83
84     juego = TorresDeHanoi(num_discos, canvas)
85     input("Presione Enter para iniciar la animaci n")
86     juego.resolver()
87     print(" Problema  resuelto!")
88     root.mainloop()
```

## Ejecución del Programa

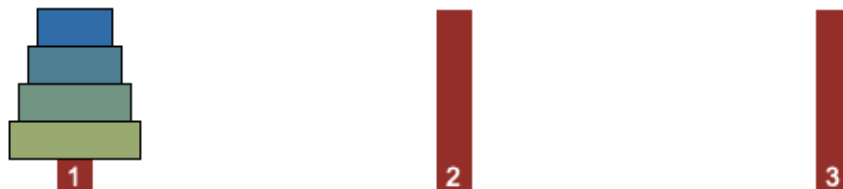


Figura 7:

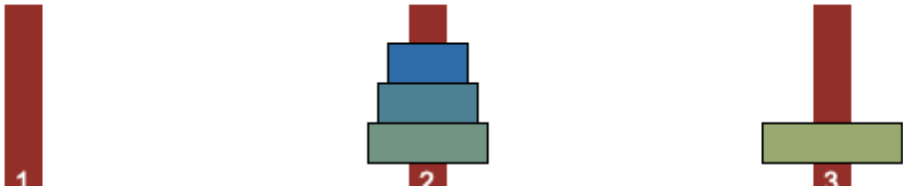


Figura 8:



Figura 9: