

UNIVERSIDAD NACIONAL DEL ALTIPLANO

**FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA,
ELECTRÓNICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS**



Práctica N°3

Docente: Mg. Aldo Hernan Zanabria Galvez

Alumno: Yoel Nhelio Canaza Chagua

Curso:

Programación Orientada a Objetos II

CICLO III – SEMESTRE 2023 – II

PUNO, PERÚ

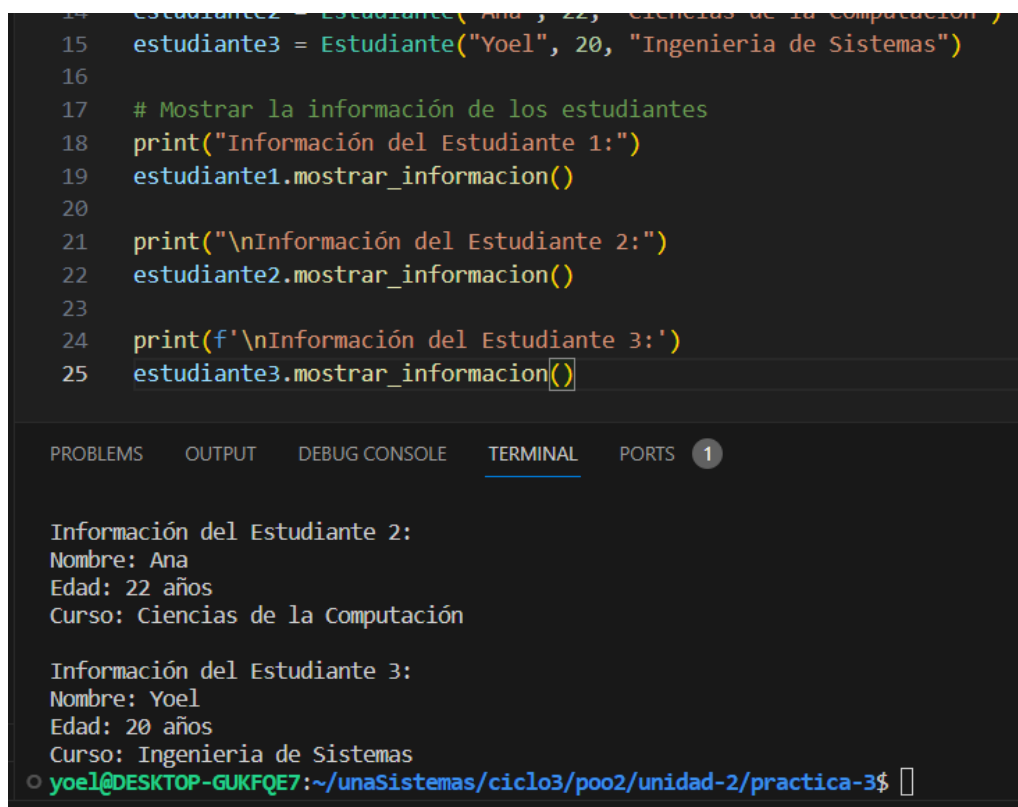
2023

1. Del siguiente código:

```
1
2
3 class Estudiante:
4     def __init__(self, nombre, edad, curso):
5         self.nombre = nombre
6         self.edad = edad
7         self.curso = curso
8
9     def mostrar_informacion(self):
10        print(f"Nombre: {self.nombre}")
11        print(f"Edad: {self.edad} años")
12        print(f"Curso: {self.curso}")
13
14 # Crear instancias de la clase Estudiante
15 estudiante1 = Estudiante("Juan", 20, "Ingeniería")
16 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación")
17
18 # Mostrar la información de los estudiantes
19 print("Información del Estudiante 1:")
20 estudiante1.mostrar_informacion()
21
22 print("\nInformación del Estudiante 2:")
23 estudiante2.mostrar_informacion()
```

Listing 1: Código base sin modificaciones.

- 1.1. Crea una nueva instancia de la clase Estudiante con tus propios datos y muestra su información utilizando el método `mostrar_informacion`.



```
14 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación")
15 estudiante3 = Estudiante("Yoel", 20, "Ingeniería de Sistemas")
16
17 # Mostrar la información de los estudiantes
18 print("Información del Estudiante 1:")
19 estudiante1.mostrar_informacion()
20
21 print("\nInformación del Estudiante 2:")
22 estudiante2.mostrar_informacion()
23
24 print(f'\nInformación del Estudiante 3:')
25 estudiante3.mostrar_informacion()
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS 1

```
Información del Estudiante 2:
Nombre: Ana
Edad: 22 años
Curso: Ciencias de la Computación

Información del Estudiante 3:
Nombre: Yoel
Edad: 20 años
Curso: Ingeniería de Sistemas
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$
```

Figura 1: Creando nueva instancia y mostrando la información de esa instancia

- 1.2. Añade un nuevo método a la clase `Estudiante` llamado `cumpleaños` que incrementa la edad del estudiante en 1 año. Luego, utiliza este método con una de las instancias creadas para simular el paso de un año.

```
11
12     def cumpleaños(self):
13         self.edad += 1
14
15 # Crear instancias de la clase Estudiante
16 estudiante1 = Estudiante("Juan", 20, "Ingeniería")
17 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación")
18 estudiante3 = Estudiante("Yoel", 20, "Ingeniería de Sistemas")
19
20 # Modificando la edad de un estudiante con el método cumpleaños
21 estudiante1.cumpleaños()
22
23 # Mostrar la información de los estudiantes
24 print("Información del Estudiante 1:")
25 estudiante1.mostrar_informacion()
26
27 print("\nInformación del Estudiante 2:")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

Edad: 20 años
Curso: Ingeniería de Sistemas
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3\$ python3 pract

Información del Estudiante 1:
Nombre: Juan
Edad: 21 años
Curso: Ingeniería

Figura 2: Modificando la edad de `estudiante1` con el método `'cumpleaños'`

- 1.3. Crea una clase adicional llamada `Curso` que tenga un atributo para el nombre del curso y un método para mostrar la información del curso. Luego, modifica la clase `Estudiante` para que tenga un atributo de tipo `Curso` y muestre la información del estudiante y el curso al que pertenece.

```
class Curso:
    def __init__(self, nombre_curso):
        self.nombre_curso = nombre_curso

    def mostrar_informacion_curso(self):
        print(f"Nombre del curso: {self.nombre_curso}")

class Estudiante:
    def __init__(self, nombre, edad, escuela, curso):
        self.nombre = nombre
        self.edad = edad
        self.escuela = escuela
        self.curso = curso

    def mostrar_informacion(self):
        print(f"Nombre: {self.nombre}")
        print(f"Edad: {self.edad} años")
        print(f"Escuela profesional: {self.escuela}")
        self.curso.mostrar_informacion_curso()
```

Figura 3: Creando la clase `Curso` y modificando la clase `Estudiante` para que tenga un atributo de tipo `Curso` y su función `'mostrar_informacion'` también permita mostrar el nombre del curso

```
24
25 # Creando cursos
26 poo2 = Curso("Programación Orientada a Objetos 2")
27 idiomas = Curso("Inglés")
28 estructuras_datos = Curso("Estructuras de Datos")
29
30 # Crear instancias de la clase Estudiante
31 estudiante1 = Estudiante("Juan", 20, "Ingeniería", poo2)
32 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación", idiomas)
33 estudiante3 = Estudiante("Yoel", 20, "Ingeniería de Sistemas", estructuras_datos)
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2

Nombre: Juan
Edad: 21 años
Escuela profesional: Ingeniería
Nombre del curso: Programación Orientada a Objetos 2

Información del Estudiante 2:
Nombre: Ana
Edad: 22 años
Escuela profesional: Ciencias de la Computación
Nombre del curso: Inglés

Información del Estudiante 3:
Nombre: Yoel
Edad: 20 años
Escuela profesional: Ingeniería de Sistemas
Nombre del curso: Estructuras de Datos

yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3\$

Figura 4: Creando los cursos para luego pasarlos como argumentos en la inicialización de las instancias de la clase Estudiante

- 1.4. Implementa una clase llamada Universidad que tenga una lista de estudiantes. Agrega métodos para añadir un estudiante nuevo, mostrar la cantidad total de estudiantes y mostrar la información de todos los estudiantes.

```
class Universidad:
    def __init__(self):
        self.estudiantes = []

    def agregar_estudiante(self, estudiante):
        self.estudiantes.append(estudiante)
        print(f"Estudiante {estudiante.nombre} agregado a la universidad")

    def imprimir_numero_estudiantes(self):
        print(f"La cantidad de estudiantes es de {len(self.estudiantes)}")

    def mostrar_informacion_todos_los_estudiantes(self):
        for estudiante in self.estudiantes:
            estudiante.mostrar_informacion()
            print('')
```

Figura 5: Implementando una clase llamada Universidad con una lista de estudiantes, y con métodos para añadir estudiantes nuevos, mostrar la cantidad de estudiantes, y mostrar la información de todos los estudiantes

```

# Creando universidad
una_puno = Universidad()
# Agregándole alumnos a la universidad
una_puno.agregar_estudiante(estudiante1)
una_puno.agregar_estudiante(estudiante2)
una_puno.agregar_estudiante(estudiante3)
print('')
# Mostrando la cantidad de estudiantes en la universidad
una_puno.imprimir_numero_estudiantes()
print('')
# Mostrando la información de todos los estudiantes de la universidad
una_puno.mostrar_informacion_todos_los_estudiantes()

```

Figura 6: Creando una instancia de la clase Universidad, agregándole alumnos y mostrando la información de todos sus estudiantes

```

Estudiante Juan agregado a la universidad
Estudiante Ana agregado a la universidad
Estudiante Yoel agregado a la universidad

La cantidad de estudiantes es de 3

Nombre: Juan
Edad: 21 años
Escuela profesional: Ingeniería
Nombre del curso: Programación Orientada a Objetos 2

Nombre: Ana
Edad: 22 años
Escuela profesional: Ciencias de la Computación
Nombre del curso: Inglés

Nombre: Yoel
Edad: 20 años
Escuela profesional: Ingeniería de Sistemas
Nombre del curso: Estructuras de Datos

```

```

○ yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$

```

Figura 7:

- 1.5. Define un método en la clase `Estudiante` llamado `es_mayor_de_edad` que devuelve `True` si el estudiante tiene 18 años o más, y `False` en caso contrario. Luego, utiliza este método para determinar si un estudiante es mayor de edad.

```
def es_mayor_de_edad(self):  
    if self.edad >= 18:  
        return True  
    else:  
        return False
```

Figura 8: Definiendo el método solicitado

```
84  
85 # Verificando si los estudiantes de la universidad son mayores de edad:  
86 for estudiante in una_puno.estudiantes:  
87     if estudiante.es_mayor_de_edad:  
88         print(f"El estudiante {estudiante.nombre} es mayor de edad")  
89     else:  
90         print(f"El estudiante {estudiante.nombre} no es mayor de edad")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2

```
El estudiante Juan es mayor de edad  
El estudiante Ana es mayor de edad  
El estudiante Yoel es mayor de edad  
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$
```

Figura 9: Utilizando ese método para saber si los estudiantes de la instancia `una_puno` de la clase `Universidad` son mayores de edad

2. Preguntas

2.1. ¿Cuál es la ventaja de utilizar la programación orientada a objetos en comparación con otros enfoques?

La programación orientada a objetos permite, mediante la abstracción, modelar problemas de manera más cercana a cómo se perciben en el mundo real, facilitando la comprensión del código y su mantenimiento.

También facilita la reutilización de código mediante el concepto de clases y objetos, la creación de módulos independientes y bien definidos (cada clase representa un módulo que encapsula su propia lógica y datos),

Además, el encapsulamiento mejora la seguridad y facilita el mantenimiento del código, ya que los cambios internos no afectan al resto del código. La herencia fomenta la reutilización del código. Y el polimorfismo nos permite tratar objetos de clases diferentes de manera uniforme.

2.2. ¿Cuál es la diferencia entre una clase y una instancia en programación orientada a objetos?

Una clase es como un plano o una plantilla que define cómo deben ser los objetos, mientras que una instancia es un objeto específico creado a partir de esa plantilla, con sus propias características y comportamiento. Es decir, la clase proporciona la estructura y el diseño, y las instancias son ejemplos concretos basados en esa estructura.

2.3. ¿Por qué es importante encapsular datos en la programación orientada a objetos?

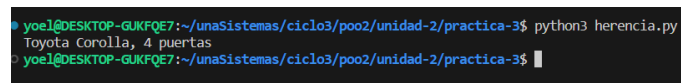
La encapsulación nos ayuda a proteger los datos al limitar el acceso directo a ellos. Solo los métodos definidos en la propia clase pueden manipular los datos internos, evitando modificaciones no deseadas o errores accidentales. Al encapsular datos, se pueden aplicar restricciones y validaciones dentro de los métodos de la clase. Esto asegura que los datos se mantengan en un estado coherente y válido, evitando situaciones inconsistentes.

2.4. Explica el concepto de herencia y proporciona un ejemplo en el contexto de la programación orientada a objetos.

La herencia permite que una clase (llamada clase derivada o subclase) herede atributos y métodos de otra clase (llamada clase base o superclase). La herencia facilita la reutilización de código y la creación de jerarquías de clases, donde las subclases pueden extender o especializar el comportamiento de las superclases.

```
1
2
3 class Vehiculo:
4     def __init__(self, marca, modelo):
5         self.marca = marca
6         self.modelo = modelo
7
8     def obtener_informacion(self):
9         return f"{self.marca} {self.modelo}"
10
11 class Automovil(Vehiculo):
12     def __init__(self, marca, modelo, puertas):
13         # Llamamos al constructor de la clase base para inicializar marca y
14         # modelo
15         super().__init__(marca, modelo)
16         # Atributo específico de la subclase
17         self.puertas = puertas
18
19     def obtener_informacion(self):
20         # Sobreescribimos el método de la clase base para agregar
21         # información específica
22         return f"{self.marca} {self.modelo}, {self.puertas} puertas"
23
24 auto = Automovil("Toyota", "Corolla", 4)
25 print(auto.obtener_informacion())
```

Listing 2: Ejemplo de herencia en Python.



```
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$ python3 herencia.py
Toyota Corolla, 4 puertas
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$
```

Figura 10: La salida del código anterior

2.5. ¿Cuándo deberías usar la programación orientada a objetos en lugar de otros paradigmas de programación?

Cuando necesitemos modelar problemas de una manera más cercana a cómo se perciben en el mundo real, cuando necesitemos reutilizar código con clases y objetos, cuando vemos que hay una jerarquía entre dos objetos que podemos expresar mediante relaciones de herencia, cuando necesitamos ocultar detalles de la implementación para facilitar la implementación de interfaces claras, o cuando tengamos proyectos grandes y complejos que necesiten organizarse y estructurarse de una manera más modular.

Sin embargo, no debemos utilizarla en situaciones donde la simplicidad, la eficiencia o la concisión del código son prioritarias, en esos casos, paradigmas como la programación procedural o la funcional pueden ser más apropiados.

3. Ejercicios Adicionales

3.1. Diseña una clase Libro con atributos como título, autor y año de publicación. Crea instancias de esta clase y muestra la información de los libros.

```
class Libro:
    def __init__(self, titulo, autor, anio_publicacion):
        self.titulo = titulo
        self.autor = autor
        self.anio_publicacion = anio_publicacion

    def mostrar_informacion(self):
        print(f"Nombre del libro: {self.titulo}")
        print(f"Autor: {self.autor}")
        print(f"Año de publicación: {self.anio_publicacion}")
```

Figura 11: Creando la clase Libro

```
103
104 # Creando instancias de la clase Libro y mostrando su información
105 print('-' * 8)
106 libro1 = Libro("The Data Science Handbook", "Carl Shan", 2015)
107 libro2 = Libro("Naked Statistics", "Charles Wheelan", 2014)
108
109 libro1.mostrar_informacion()
110 print('-' * 8)
111 libro2.mostrar_informacion()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 3

```
-----
Nombre del libro: The Data Science Handbook
Autor: Carl Shan
Año de publicación: 2015
-----
Nombre del libro: Naked Statistics
Autor: Charles Wheelan
Año de publicación: 2014
yael@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$
```

Figura 12: Creando dos instancias de la clase Libro y mostrando su información mediante el método 'mostrar_informacion()'

3.2. Modifica la clase Curso para que tenga una lista de estudiantes y métodos para agregar y eliminar estudiantes.

```
class Curso:
    def __init__(self, nombre_curso):
        self.nombre_curso = nombre_curso
        self.estudiantes = []

    def mostrar_informacion_curso(self):
        print(f"Nombre del curso: {self.nombre_curso}")

    def agregar_estudiante(self, estudiante):
        self.estudiantes.append(estudiante)
        print(f"Estudiante {estudiante.nombre} agregado al curso {self.nombre_curso}")

    def eliminar_estudiante(self, estudiante):
        if estudiante in self.estudiantes:
            self.estudiantes.remove(estudiante)
            print(f"Estudiante {estudiante.nombre} eliminado del curso {self.nombre_curso}")
        else:
            print(f"Estudiante {estudiante.nombre} no encontrado en el curso {self.nombre_curso}")
```

Figura 13: Clase 'Curso' modificada para tener una lista de estudiantes y métodos para agregar y eliminar estudiantes

3.3. Crea una clase Profesor que tenga atributos como nombre y especialidad. Relaciona la clase Profesor con la clase Curso.

```
✓ class Curso:
✓     def __init__(self, nombre_curso, docente):
        self.nombre_curso = nombre_curso
        self.docente = docente
        self.estudiantes = []

✓     def mostrar_informacion_curso(self):
        print(f"Nombre del curso: {self.nombre_curso}")

✓     def agregar_estudiante(self, estudiante):
        self.estudiantes.append(estudiante)
        print(f"Estudiante {estudiante.nombre} agregado al curso {self.nombre_curso}")

✓     def eliminar_estudiante(self, estudiante):
✓         if estudiante in self.estudiantes:
            self.estudiantes.remove(estudiante)
            print(f"Estudiante {estudiante.nombre} eliminado del curso {self.nombre_curso}")
✓         else:
            print(f"Estudiante {estudiante.nombre} no encontrado en el curso {self.nombre_curso}")

✓ class Profesor:
✓     def __init__(self, nombre, especialidad):
        self.nombre = nombre
        self.especialidad = especialidad
```

Figura 14: Creando la clase Profesor con atributos de nombre y especialidad y modificando la clase Curso con un atributo adicional que representa al docente del curso

- 3.4. Implementa un programa que simule una situación en la que varios estudiantes se inscriben en diferentes cursos de una universidad ficticia.

```
class Estudiante:
    def __init__(self, nombre, edad, escuela):
        self.nombre = nombre
        self.edad = edad
        self.escuela = escuela
        self.cursos = []

    def mostrar_informacion(self):
        print(f"Nombre: {self.nombre}")
        print(f"Edad: {self.edad} años")
        print(f"Escuela profesional: {self.escuela}")
        print(f"Estudiante matriculado en los siguientes cursos:")
        for curso in self.cursos:
            curso.mostrar_informacion_curso()

    def cumpleaños(self):
        self.edad += 1

    def es_mayor_de_edad(self):
        if self.edad >= 18:
            return True
        else:
            return False

    def matricular_curso(self, curso):
        self.cursos.append(curso)
        print(f"Estudiante {self.nombre} matriculado en el curso {curso.nombre_curso}")

    def desmatricular_curso(self, curso):
        if curso in self.cursos:
            self.cursos.remove(curso)
            print(f"Estudiante {self.nombre} desmatriculado del curso {curso.nombre_curso}")
        else:
            print(f"Estudiante {self.nombre} no está matriculado en el curso {curso.nombre_curso}")
```

Figura 15: Modificando la clase Estudiante para que tenga como atributo una lista de cursos en la que está matriculado y que tenga métodos para matricular y desmatricular de cursos

```

2  # Creando docentes
3  docente1 = Profesor("Guido van Rossum", "Lenguajes de Programación")
4  docente2 = Profesor("Linus Torvalds", "Sistemas Operativos")
5  docente3 = Profesor("Gennady Korotkevich", "Programación Competitiva")
6
7  # Creando cursos
8  poo2 = Curso("Programación Orientada a Objetos 2", docente1)
9  sistemas_operativos = Curso("Sistemas Operativos", docente2)
10 estructuras_datos = Curso("Estructuras de Datos", docente3)
11
12 # Crear instancias de la clase Estudiante
13 estudiante1 = Estudiante("Juan", 20, "Ingeniería")
14 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación")
15 estudiante3 = Estudiante("Yoel", 20, "Ingeniería de Sistemas")
16
17 # Creando universidad
18 u_ficticia = Universidad()
19 # Agregándole alumnos a la universidad
20 u_ficticia.agregar_estudiante(estudiante1)
21 u_ficticia.agregar_estudiante(estudiante2)
22 u_ficticia.agregar_estudiante(estudiante3)
23 #matriculando estudiantes en diferentes cursos:
24 estudiante1.matricular_curso(poo2)
25 poo2.agregar_estudiante(estudiante1)
26 estudiante1.matricular_curso(estructuras_datos)
27 estructuras_datos.agregar_estudiante(estudiante1)
28
29 estudiante2.matricular_curso(sistemas_operativos)
30 sistemas_operativos.agregar_estudiante(estudiante2)
31 estudiante2.matricular_curso(poo2)
32 poo2.agregar_estudiante(estudiante2)
33
34 estudiante3.matricular_curso(sistemas_operativos)
35 sistemas_operativos.agregar_estudiante(estudiante3)
36 estudiante3.matricular_curso(estructuras_datos)
37 estructuras_datos.agregar_estudiante(estudiante3)
38 # mostrando la información de todos los estudiantes
39 u_ficticia.mostrar_informacion_todos_los_estudiantes()

```

Figura 16: Programa que simula estudiantes inscribiéndose en diferentes cursos

```

● yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/unidad-2/practica-3$ python3 practica-3.py
Estudiante Juan agregado a la universidad
Estudiante Ana agregado a la universidad
Estudiante Yoel agregado a la universidad
Estudiante Juan matriculado en el curso Programación Orientada a Objetos 2
Estudiante Juan agregado al curso Programación Orientada a Objetos 2
Estudiante Juan matriculado en el curso Estructuras de Datos
Estudiante Juan agregado al curso Estructuras de Datos
Estudiante Ana matriculado en el curso Sistemas Operativos
Estudiante Ana agregado al curso Sistemas Operativos
Estudiante Ana matriculado en el curso Programación Orientada a Objetos 2
Estudiante Ana agregado al curso Programación Orientada a Objetos 2
Estudiante Yoel matriculado en el curso Sistemas Operativos
Estudiante Yoel agregado al curso Sistemas Operativos
Estudiante Yoel matriculado en el curso Estructuras de Datos
Estudiante Yoel agregado al curso Estructuras de Datos
Nombre: Juan
Edad: 20 años
Escuela profesional: Ingeniería
Estudiante matriculado en los siguientes cursos:
Nombre del curso: Programación Orientada a Objetos 2
Nombre del docente que imparte el curso Guido van Rossum
Nombre del curso: Estructuras de Datos
Nombre del docente que imparte el curso Gennady Korotkevich

Nombre: Ana
Edad: 22 años
Escuela profesional: Ciencias de la Computación
Estudiante matriculado en los siguientes cursos:
Nombre del curso: Sistemas Operativos
Nombre del docente que imparte el curso Linus Torvalds
Nombre del curso: Programación Orientada a Objetos 2
Nombre del docente que imparte el curso Guido van Rossum

Nombre: Yoel
Edad: 20 años
Escuela profesional: Ingenieria de Sistemas
Estudiante matriculado en los siguientes cursos:
Nombre del curso: Sistemas Operativos
Nombre del docente que imparte el curso Linus Torvalds

```

Figura 17: Resultado en pantalla

<https://github.com/YoelCanaza/poo2-practica-3/blob/b446a92eb849b99cd2d16541c51e61478ffca105/practica-3.py>

```

1
2 class Curso:
3     def __init__(self, nombre_curso, docente):
4         self.nombre_curso = nombre_curso
5         self.docente = docente
6         self.estudiantes = []
7
8     def mostrar_informacion_curso(self):
9         print(f"Nombre del curso: {self.nombre_curso}")
10        print(f"Nombre del docente que imparte el curso {self.docente.
11        nombre}")
12
13    def agregar_estudiante(self, estudiante):
14        self.estudiantes.append(estudiante)
15        print(f"Estudiante {estudiante.nombre} agregado al curso {self.
16        nombre_curso}")
17
18    def eliminar_estudiante(self, estudiante):
19        if estudiante in self.estudiantes:

```

```

18         self.estudiantes.remove(estudiante)
19         print(f"Estudiante {estudiante.nombre} eliminado del curso {
self.nombre_curso}")
20     else:
21         print(f"Estudiante {estudiante.nombre} no encontrado en el
curso {self.nombre_curso}")
22
23 class Profesor:
24     def __init__(self, nombre, especialidad):
25         self.nombre = nombre
26         self.especialidad = especialidad
27
28
29 class Estudiante:
30     def __init__(self, nombre, edad, escuela):
31         self.nombre = nombre
32         self.edad = edad
33         self.escuela = escuela
34         self.cursos = []
35
36     def mostrar_informacion(self):
37         print(f"Nombre: {self.nombre}")
38         print(f"Edad: {self.edad} a os ")
39         print(f"Escuela profesional: {self.escuela}")
40         print(f"Estudiante matriculado en los siguientes cursos:")
41         for curso in self.cursos:
42             curso.mostrar_informacion_curso()
43
44     def cumpleaños(self):
45         self.edad += 1
46
47     def es_mayor_de_edad(self):
48         if self.edad >= 18:
49             return True
50         else:
51             return False
52
53     def matricular_curso(self, curso):
54         self.cursos.append(curso)
55         print(f"Estudiante {self.nombre} matriculado en el curso {curso.
nombre_curso}")
56
57     def desmatricular_curso(self, curso):
58         if curso in self.cursos:
59             self.cursos.remove(curso)
60             print(f"Estudiante {self.nombre} desmatriculado del curso {
curso.nombre_curso}")
61         else:
62             print(f"Estudiante {self.nombre} no est matriculado en el
curso {curso.nombre_curso}")
63
64 class Universidad:
65     def __init__(self):
66         self.estudiantes = []
67
68     def agregar_estudiante(self, estudiante):
69         self.estudiantes.append(estudiante)
70         print(f"Estudiante {estudiante.nombre} agregado a la universidad")
71
72     def imprimir_numero_estudiantes(self):
73         print(f"La cantidad de estudiantes es de {len(self.estudiantes)}")
74
75     def mostrar_informacion_todos_los_estudiantes(self):

```

```

76         for estudiante in self.estudiantes:
77             estudiante.mostrar_informacion()
78             print('')
79
80 class Libro:
81     def __init__(self, titulo, autor, anio_publicacion):
82         self.titulo = titulo
83         self.autor = autor
84         self.anio_publicacion = anio_publicacion
85
86     def mostrar_informacion(self):
87         print(f"Nombre del libro: {self.titulo}")
88         print(f"Autor: {self.autor}")
89         print(f"Año de publicación: {self.anio_publicacion}")
90
91
92 # Creando docentes
93 docente1 = Profesor("Guido van Rossum", "Lenguajes de Programación")
94 docente2 = Profesor("Linus Torvalds", "Sistemas Operativos")
95 docente3 = Profesor("Gennady Korotkevich", "Programación Competitiva")
96
97 # Creando cursos
98 poo2 = Curso("Programación Orientada a Objetos 2", docente1)
99 sistemas_operativos = Curso("Sistemas Operativos", docente2)
100 estructuras_datos = Curso("Estructuras de Datos", docente3)
101
102 # Crear instancias de la clase Estudiante
103 estudiante1 = Estudiante("Juan", 20, "Ingeniería")
104 estudiante2 = Estudiante("Ana", 22, "Ciencias de la Computación")
105 estudiante3 = Estudiante("Yoel", 20, "Ingeniería de Sistemas")
106
107 # Creando universidad
108 u_ficticia = Universidad()
109
110 # Agregándole alumnos a la universidad
111 u_ficticia.agregar_estudiante(estudiante1)
112 u_ficticia.agregar_estudiante(estudiante2)
113 u_ficticia.agregar_estudiante(estudiante3)
114
115 #matriculando estudiantes en diferentes cursos:
116 estudiante1.matricular_curso(poo2)
117 poo2.agregar_estudiante(estudiante1)
118 estudiante1.matricular_curso(estructuras_datos)
119 estructuras_datos.agregar_estudiante(estudiante1)
120
121 estudiante2.matricular_curso(sistemas_operativos)
122 sistemas_operativos.agregar_estudiante(estudiante2)
123 estudiante2.matricular_curso(poo2)
124 poo2.agregar_estudiante(estudiante2)
125
126 estudiante3.matricular_curso(sistemas_operativos)
127 sistemas_operativos.agregar_estudiante(estudiante3)
128 estudiante3.matricular_curso(estructuras_datos)
129 estructuras_datos.agregar_estudiante(estudiante3)
130
131 # mostrando la información de todos los estudiantes
132 u_ficticia.mostrar_informacion_todos_los_estudiantes()
133
134
135 # Modificando la edad de un estudiante con el método cumpleaños
136 estudiante1.cumpleaños()
137
138 # Mostrar la información de los estudiantes
139 print("Información del Estudiante 1:")
140 estudiante1.mostrar_informacion()
141
142 print("\nInformación del Estudiante 2:")

```

```

139 estudiante2.mostrar_informacion()
140
141 print(f'\nInformaci n del Estudiante 3:')
142 estudiante3.mostrar_informacion()
143 print('\n')
144
145 # Creando universidad
146 una_puno = Universidad()
147 # Agreg ndole alumnos a la universidad
148 una_puno.agregar_estudiante(estudiante1)
149 una_puno.agregar_estudiante(estudiante2)
150 una_puno.agregar_estudiante(estudiante3)
151 print('')
152 # Mostrando la cantidad de estudiantes en la universidad
153 una_puno.imprimir_numero_estudiantes()
154 print('')
155 # Mostrando la informaci n de todos los estudiantes de la universidad
156 una_puno.mostrar_informacion_todos_los_estudiantes()
157
158
159 # Verificando si los estudiantes de la universidad son mayores de edad:
160 for estudiante in una_puno.estudiantes:
161     if estudiante.es_mayor_de_edad:
162         print(f"El estudiante {estudiante.nombre} es mayor de edad")
163     else:
164         print(f"El estudiante {estudiante.nombre} no es mayor de edad")
165
166 # Creando instancias de la clase Libro y mostrando su informaci n
167 print('-' * 8)
168 libro1 = Libro("The Data Science Handbook", "Carl Shan", 2015)
169 libro2 = Libro("Naked Statistics", "Charles Wheelan", 2014)
170
171 libro1.mostrar_informacion()
172 print('-' * 8)
173 libro2.mostrar_informacion()

```

Listing 3: Código con todas las modificaciones pedidas en los ejercicios