



# Programación Orientada a Objetos



**Facultad:**

Ingenierías y Arquitectura



**Carrera:**

Redes y Analítica de Datos



**Autor:**

Ramiro Leonardo Ramírez Coronel

**Universidad Técnica Particular de Loja**

**Programación Orientada a Objetos**

Guía didáctica

Ramiro Leonardo Ramírez Coronel

**Diagramación y diseño digital:**

Ediloja Cía. Ltda.

Marcelino Champagnat s/n y París

[edilojacialtda@ediloja.com.ec](mailto:edilojacialtda@ediloja.com.ec)

[www.ediloja.com.ec](http://www.ediloja.com.ec)

**ISBN digital** - 978-9942-47-491-9

**Año de edición:**



# Índice

<b>1. Datos de información .....</b>	<b>7</b>
1.1. Presentación de la asignatura .....	7
1.2. Competencias genéricas de la UTPL.....	7
1.3. Competencias dmpe	7

<b>Semana 4 .....</b>	<b>35</b>
1.7. Modelos para diseño .....	35
1.8. Diagramas de casos de uso.....	37
1.9.	

Semana 8 .....	81
Actividades finales del bimestre.....	81
<b>Segundo bimestre .....</b>	<b>83</b>
Resultado de aprendizaje 2	

Semana 13 .....	118
<b>Unidad 4. Modelado y aplicaciones avanzadas .....</b>	<b>118</b>
4.1. Diseño de clases y relaciones avanzadas.....	118
Actividad de aprendizaje recomendada	



---

## 1. Datos de información

---

### 1.1. Presentación de la asignatura



### 1.2. Competencias genéricas de la UTPL

- Comportamiento Ético.

### 1.3. Competencias

## 1.4. Problemática que aborda la asignatura

La asignatura Programación Orientada a Objetos en la carrera de Redes y Analítica de Datos es la continuación de la materia de la Programación avanzada a Objetos, donde potencian las habilidades en el desarrollo. La asignatura surge como una solución a diversas problemáticas presentes en los paradigmas de programación estructurada. Algunas de

herencia, polimorfismo y el modelado mediante UML. Las sesiones síncronas brindan acompañamiento docente para aclarar dudas, realizar demostraciones, fomentando la participación. Adicionalmente, las actividades asíncronas permiten reforzar el aprendizaje con autoevaluación, evaluaciones en línea, foros de discusión y actividades práctico-experimentales.

Este modelo garantiza un aprendizaje flexible y efectivo, combinando



### 3. Orientaciones didácticas por resultados de aprendizaje

#### Resultado de aprendizaje 1



##### Primer bimestre

- Aplica los conceptos y fundamentos de la Programación Orientada a Objetos (POO)

índice

I Bimestre

II Bimestre

Solucionario

Referencias

competencias fundamentales para su desarrollo profesional. Como su profesor, estaré aquí para guiarle en este proceso. Con dedicación y esfuerzo, estos conocimientos se convertirán en una base sólida para su éxito académico y profesional. ¡Avancemos juntos en este aprendizaje!

## Contenidos, recursos y actividades de aprendizaje recomendadas





Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 1

¡Bienvenidos a la primera semana del curso de Programación Orientada a Objetos! Durante esta unidad inicial vamos a sumergirnos en los paradigmas de la Programación Orientada a Objetos: clases, objetos, atributos. También vamos a ver la introducción al lenguaje UML que nos ayudará a diseñar soluciones software.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

Piense en un vehículo. Tiene atributos como el color, la marca y la velocidad máxima. También tiene acciones que puede realizar, como arrancar, frenar o acelerar. Dentro de la Programación Orientada a Objetos, esto se traduce en clases y objetos. La clase es el plano que define cómo es un vehículo en términos generales, mientras que el objeto es el vehículo real que usted puede ver y usar. Cada vehículo es una

índice

I Bimestre

II Bimestre

Solucionario

Referencias

que otorga flexibilidad al permitir que un mismo método se comporte de distintas maneras según el contexto.

Cuando domine estos principios, verá la programación desde una perspectiva completamente nueva. Ya no solo escribirá código, sino que diseñará sistemas más eficientes, escalables y elegantes. La

índice

I Bimestre

II Bimestre

Solucionario

Referencias

a utilizar. En el lenguaje unificado de modelado sucede algo similar. Hay diagramas que representan la estructura de un sistema, otros que muestran su comportamiento y otros que describen la interacción entre sus partes.

Uno de los más conocidos es el diagrama de clases, que permite organizar los elementos de un sistema en términos de sus atributos y comportamientos. También están los diagramas de secuencia, que

### 1.3. Paradigma de la Programación Orientada a Objetos

Para entender el Paradigma de la Programación Orientada a Objetos, imagine que debe organizar su hogar de la manera más eficiente posible. En lugar de tener todos sus objetos regados por la casa sin orden ni propósito, decide agruparlos según su función. En la cocina coloca los utensilios y electrodomésticos, en la sala deja los muebles y la

índice

I Bimestre

II Bimestre

Solucionario

Referencias

A medida que se adentre en este enfoque, notará que la Programación Orientada a Objetos no es solo una forma de escribir código, sino una manera de pensar en soluciones estructuradas y eficientes. Su software dejará de ser un conjunto desordenado de instrucciones y se convertirá en un sistema donde cada componente tiene un propósito claro y una

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 2**

Descripción de los elementos de la Programación Orientada a Objetos



Nota. Ramírez, R., 2025.

¡Muy bien, finalizamos la semana 1! Ahora le invito a participar en el siguiente quiz para poner en práctica los conocimientos adquiridos.

[Programación Orientada a Objetos \(POO\) y UML](#)

Reconocer los fundamentos del Paradigma de la Programación

proyectos. Este conocimiento fortalecerá su capacidad para transformar ideas en sistemas eficientes y bien planificados.



### Actividad de aprendizaje recomendada

¡Bienvenidos a las actividades recomendadas de aprendizaje de la semana 1! La siguiente actividad está diseñada para que alcance el resultado de aprendizaje, en ellas tendrá que entender, comprender y analizar los términos que existen en lo que es la Programación Orientada

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Tenga un registro de los puntos clave que detecte en esta semana para mayor comprensión del contenido.



Sem 1

Sem 2

Sem 3

Sem 4

Sem 5

Sem 6

Sem 7

Sem 8



## Semana 2

Imagine construir un programa como si fuera un mundo en miniatura, donde cada elemento tiene su propia identidad, responsabilidades y la capacidad de interactuar con los demás. Así funciona la Programación Orientada a Objetos, un paradigma que organiza el código en objetos que representan entidades de la vida real, haciendo el desarrollo más

índice

I Bimestre

II Bimestre

Solucionario

Referencias

Piense en una aplicación que gestiona una tienda en línea. Podría tener objetos como productos, clientes y pedidos. Cada producto tiene un nombre, un precio y una cantidad en stock. Cada cliente tiene un nombre, un correo electrónico y un historial de compras. Cada pedido tiene una lista de productos, una fecha de compra y un método de pago. En un lenguaje orientado a objetos, cada uno de estos

prácticos y fáciles de seguir. Es ideal tanto para aprender desde cero como para reforzar conocimientos. ¡Descubra los beneficios de la POO y comience a aprender ahora!

## 1.5. Ventajas de la POO

La Programación Orientada a Objetos (POO) en Python ofrece múltiples ventajas que facilitan el desarrollo de software

índice

I Bimestre

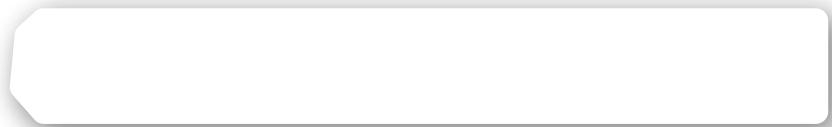
II Bimestre

Solucionario

Referencias

### Figura 3

Principales ventajas de la POO



índice

I Bimestre

II Bimestre

Solucionario

Referencias

Reconocer cómo funcionan los lenguajes orientados a objetos y las ventajas que aporta la POO le permitirá organizar el código de forma más clara, reutilizable y eficiente.



### Actividad de aprendizaje recomendada

¡Bienvenidos a las actividades recomendadas de aprendizaje de la semana 2! La siguiente actividad está diseñada para que alcance el resultado de aprendizaje en ellas tendrá comprender cuáles son los

[Sem 1](#)[Sem 2](#)[Sem 3](#)[Sem 4](#)[Sem 5](#)[Sem 6](#)[Sem 7](#)[Sem 8](#)

## Semana 3

La Programación Orientada a Objetos es como armar un rompecabezas donde cada pieza es una entidad independiente pero conectada con las demás. En este paradigma, todo gira en torno a clases, plantillas que definen las características como los atributos y comportamientos, como los métodos de lo que queremos modelar. Cuando una clase

[Índice](#)[I Bimestre](#)[II Bimestre](#)[Solucionario](#)[Referencias](#)

Un objeto es, por tanto, la representación en un programa de un concepto, que incluye toda la información necesaria para abstraerlo: datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos. La siguiente figura presenta un ejemplo simplificado en el que un automóvil consolida sus características (atributos) y las acciones que puede realizar (métodos).

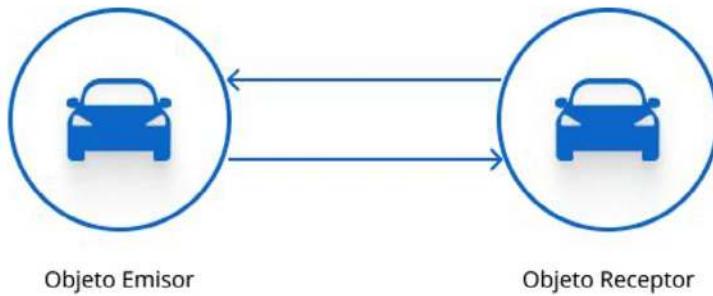
**Figura 4**

*Consolidado de los atributos y métodos de una clase*

compartir información. La figura 5 muestra este proceso, donde un objeto emisor solicita a otro objeto (receptor) la realización de una acción específica, enviando, si es necesario, parámetros como parte de la comunicación.

**Figura 5**

Ejemplo de envío de mensajes de un objeto



Nota.

3. **Constructor (`__init__` en Python):** método especial que se ejecuta al crear un objeto y sirve para inicializar sus atributos.

**Figura 6**

Representación de un objeto y clase



Nota. Ramírez, R., 2025.

### Características de una clase

- 

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

La herencia es un pilar clave en la POO, ya que permite construir sistemas modulares, escalables y fáciles de mantener.

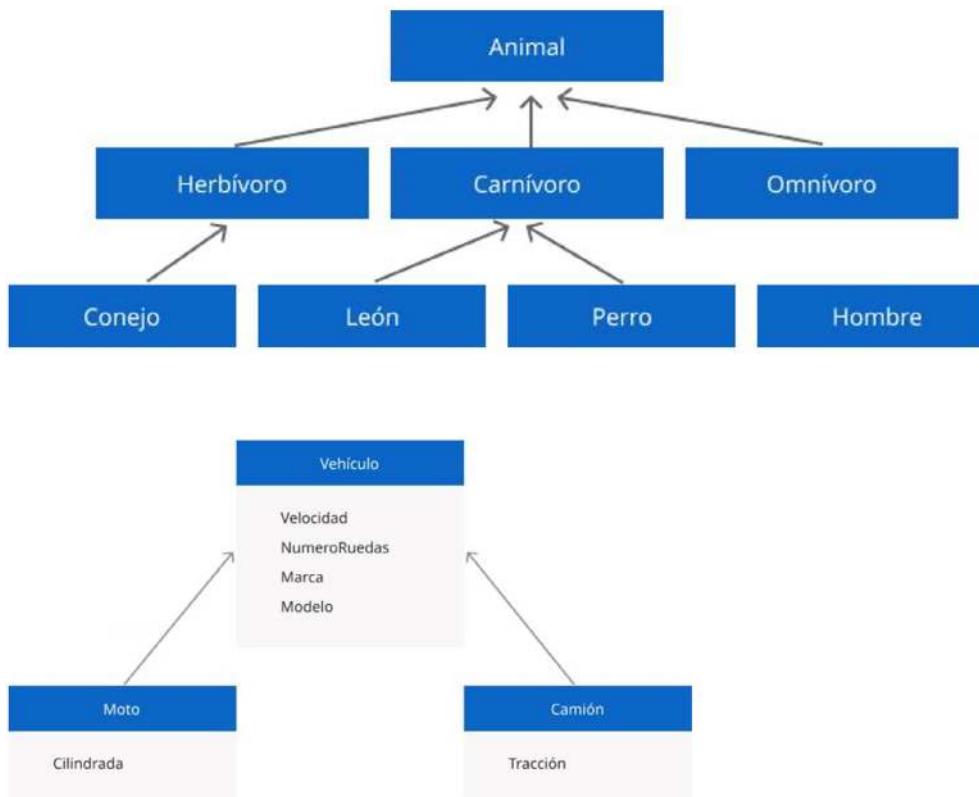
### Tipos de herencia

1. **Herencia simple:** una subclase hereda de una sola clase padre.
2. **Herencia múltiple:** una subclase hereda de múltiples clases padres.
3. **Herencia multinivel:**

## Ejemplos de herencia

**Figura 8**

Representación de una clase principal y clases derivadas



Nota. Ejemplos de clases. Ramírez, R., 2025.

**¿Le gustaría entender la Programación Orientada a Objetos de una manera visual y práctica?** Le invito a explorar el siguiente ejemplo en [Lucidchart](#) donde encontrará ejemplos claros de

### 1.6.5. Polimorfismo

El polimorfismo es un concepto clave en la Programación Orientada a Objetos que permite que un mismo método o función pueda comportarse de diferentes maneras dependiendo del objeto que lo use.

En términos simples, el polimorfismo permite usar una misma interfaz para diferentes tipos de datos, facilitando la reutilización y extensibilidad

índice

I Bimestre

II Bimestre

Solucionario

Referencias

### 1.6.6. Encapsulamiento

El encapsulamiento es un principio de la Programación Orientada a Objetos que consiste en restringir el acceso a los atributos y métodos de una clase, protegiendo los datos y asegurando que solo puedan ser modificados de manera controlada.

En términos simples, el encapsulamiento oculta los detalles internos

índice

I Bimestre

II Bimestre

Solucionario

Referencias

computacionales. Se recomienda realizar una lectura detallada del material y sintetizar los conceptos clave mediante mapas conceptuales o esquemas que ayuden a visualizar la estructura de una clase y sus componentes. Además, se sugiere aplicar lo aprendido en la implementación de nuevos ejemplos, enfocándose en la herencia y polimorfismo.

- Finalmente, reflexionar sobre la importancia de un buen análisis y



Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 4

¿Alguna vez ha querido visualizar cómo interactúan los usuarios con un sistema o cómo se relacionan los componentes de tu **software**?

Los diagramas de casos de uso son como mapas que nos ayudan a entender las funcionalidades clave desde la perspectiva del usuario,

índice

I Bimestre

II Bimestre

Solucionario

Referencias

su carácter semiformal, incluye una notación gráfica como parte fundamental.

El Lenguaje de Modelado Unificado (UML, Unified Modeling Language) surgió con el propósito de estandarizar las diversas notaciones utilizadas previamente.

Su enfoque principal es analizar y comprender tanto la aplicación como el dominio en el que opera. El punto de partida es la formulación del

índice

I Bimestre

II Bimestre

Solucionario

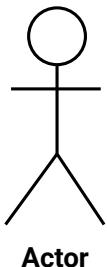
Referencias

## 1.8. Diagramas de casos de uso

Imagina que estás diseñando una aplicación y quieres visualizar cómo interactuarán los usuarios con el sistema. Aquí es donde entra en juego el diagrama de casos de uso, una herramienta gráfica de UML (Unified Modeling Language) que permite representar las funcionalidades principales del sistema y sus interacciones con los actores externos.

**Figura 11**

Representación de una clase principal y polimorfismo en clases derivadas

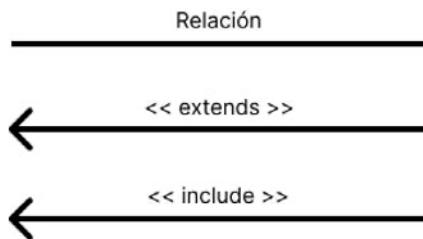
**Actor**

Nota. Ramírez, R., 2025.

■

**Figura 13**

Representación de una clase principal y polimorfismo en clases derivadas



Nota. Ramírez, R., 2025.

- **Sistema** → Se representa como un rectángulo que agrupa los casos de uso dentro de sus límites.

**Casos de uso:**

- Buscar un libro.
- Prestar un libro.
- Devolver un libro.
- Registrar un usuario.

**Relaciones:**

índice

I Bimestre

II Bimestre

Solucionario

Referencias

Para aplicar los conocimientos adquiridos en este tema, le invito a participar en el siguiente quiz:

### Elementos clave de un Diagrama de Caso de Uso

Comprender los elementos clave de un diagrama de caso de uso, actores, casos de uso, relaciones y sistema le permitirá interpretar y diseñar mejor cómo interactúan los usuarios con una aplicación.

## 1.10. Diagramas de clases

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Facilitar la comunicación entre desarrolladores y analistas.
- Reutilizar código mediante herencia y asociaciones.

Se usa en la fase de análisis y diseño, asegurando que la arquitectura del sistema sea clara y bien organizada.

Sirve para ver las relaciones entre las clases que componen el sistema de forma asociativa, herencia o de uso.

Para entenderlo mejor, veamos sus componentes principales de un

## 1.12. Relaciones

Una relación representa una dependencia, la misma que puede ser entre dos o más clases. Las relaciones se representan con una línea que une las clases, la misma que variará dependiendo del tipo de relación.

Las relaciones poseen varias propiedades que, dependiendo de la profundidad que se quiera dar al diagrama, se representarán o no. A continuación, se listan:

- **Multiplicidad:**

**Figura 18**

Representación de una relación de 1 a n



Nota. Ramírez, R., 2025.

- **Herencia** (generalización): una clase hija hereda de una clase padre. Ejemplo: "Círculo" y "Cuadrado" heredan de "Figura".

**Figura 19**

Representación de herencia entre clases

**Figura 20**

Representación de agregación entre clases



Nota. Ramírez, R., 2025.

- **Composición:** relación en la que una clase es parte esencial de otra. Ejemplo: un "Edificio" tiene "Habitaciones" y no puede existir sin ellas.

**Figura 21**

## Ejemplo de diagrama de clases

Imaginemos que se requiere diseñar un sistema de biblioteca. Algunas clases podrían ser:

- Estudiante (con atributos como "nombre" y "ID").
- Libro (con atributos como "título" y "autor").
- 

índice

I Bimestre

II Bimestre

Solucionario

Referencias

necesario declarar las variables explícitamente antes de usarlas, sino que se crean automáticamente y adoptan un tipo de contenido ... Al asignarles un valor por primera vez." (p. 50)

Para poner en práctica los conocimientos adquiridos en este tema, le invito a participar en el siguiente juego de arrastrar y soltar:

### Relaciones

¡Excelente trabajo! Ha logrado relacionar correctamente los distintos tipos de dependencias entre clases y su representación en UML.



## Actividades de aprendizaje recomendadas

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 4! Las siguientes actividades están diseñadas para que alcance el resultado de aprendizaje, en ellas podrá diseñar los diferentes diagramas que intervienen en la Programación Orientada a Objetos.

1. ¿Le resulta complicado entender los diagramas de clases UML? ¡No

índice

I Bimestre

II Bimestre

Solucionario

Referencias

Previo a realizar la autoevaluación, se recomienda seguir las siguientes estrategias de estudio:

- Consultar los recursos educativos (documentos, ejemplos de código y bibliografía recomendada) para reforzar sus conocimientos.
- Revisar cuidadosamente el contenido de cada tema, prestando especial atención a la estructura de las



## Autoevaluación 1

### 1. ¿Qué es una clase en Python?

- a. Una función que retorna valores.
- b. Un "molde" para crear objetos con atributos y métodos.
- c. Un tipo de dato como int o str.
- d.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

5. ¿Qué relación representa una «parte esencial» de un todo, donde la vida del objeto depende del todo?

- a. Asociación.
- b. Agregación.
- c. Composición.
- d. Herencia.

6.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**10. Empareja el concepto de POO con su definición:**

Concepto

---

---

---

índice

I Bimestre

II Bimestre

Solucionario

Referencias



Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 5

**¡Aprender Programación Orientada a Objetos en Python es como darle superpoderes a su código!** Python, con su sintaxis clara y amigable, es el lenguaje perfecto para dominar este paradigma que organiza el software

índice

I Bimestre

II Bimestre

Solucionario

Referencias

## 2.2. Objetos

- Los objetos son o representan cosas.
- Los objetos pueden ser simples o complejos.
- Los objetos pueden ser reales o imaginarios.

## 2.3. Atributos

índice

I Bimestre

II Bimestre

Solucionario

Referencias

## 2.5. Clases

Imaginemos que una clase es como un plano arquitectónico: no es la casa en sí, sino la descripción de cómo debe ser.

Cuando se crea una instancia (objeto) de una clase, estás construyendo una casa basada en ese plano.

- Representan un tipo particular de objetos.
-

**Figura 23**

Código fuente de una clase en Python

```
persona.py
1 #nombre de la clase Persona
2 class Persona:
3
4     #Es el constructor. Se ejecuta automáticamente al crear un nuevo objeto.
5     def __init__(self, nombre, edad):
6
7         #Se refiere al objeto actual.
8         self.nombre = nombre    # atributo
9         self.edad = edad        # atributo
10
11     # Es un método, una acción que puede hacer el objeto.
12     def saludar(self):
13         print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")
```

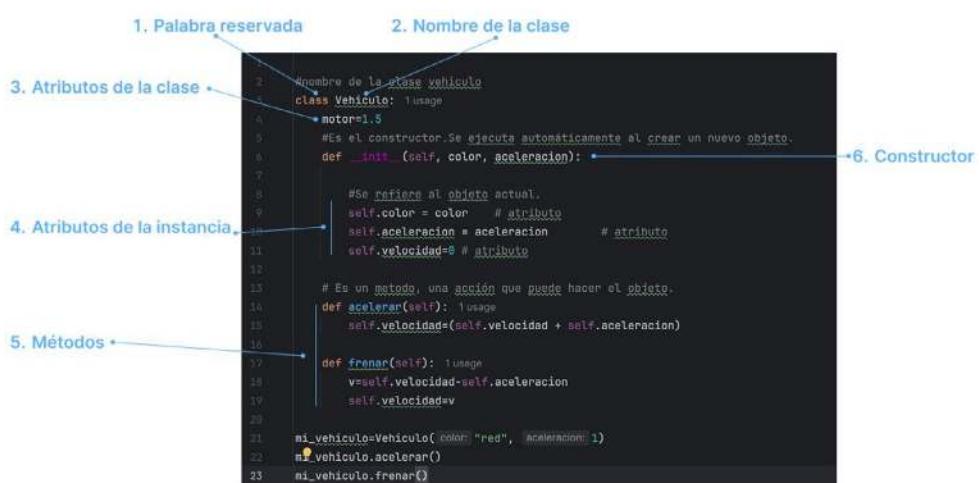
Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**Explicación del código:**

- class Persona: → Define una clase llamada **Persona**.

**Figura 24**

Representación de las partes de una clase



Nota. Este código muestra las partes de una clase en Python como el nombre, los atributos, métodos y constructor. Ramírez, R., 2025.

La clase Vehículo es un ejemplo de programación orientada a objetos en Python. Vamos a analizar cada uno de sus componentes:

1. Declaración de la clase

`class Vehiculo:`

Es un atributo compartido por todas las instancias de la clase.

- Todos los vehículos creados tendrán este valor por defecto para motor (1.5).
  - Se puede acceder directamente con Vehiculo.motor sin necesidad de instanciar la clase.
4. Método Constructor (`__init__`)
- 5.

## Métodos de instancia

Vamos a codificar dos métodos, acelerar y frenar. Los métodos no recibirán ningún parámetro.

### Método acelerar()

```
def acelerar(self):  
    self.velocidad = (self.velocidad + self.aceleracion)
```

Aumenta la velocidad actual sumando el valor de aceleración.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

### Características Importantes:

- self: siempre es el primer parámetro en métodos de instancia y representa al objeto actual.
- Atributos: pueden ser de clase (compartidos) o de instancia (específicos).
- Métodos: definen el comportamiento del objeto.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

permitirá modelar problemas de forma clara, reutilizable y organizada, construyendo sistemas más eficientes y fáciles de mantener.



## Actividades de aprendizaje recomendadas

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 5! Las siguientes actividades están diseñadas para que alcance el resultado de aprendizaje en ellas tendrá que aplicar la forma de

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Instale Python 3 para poder ejecutar el código de la actividad.

¿Te gustaría aprender a trabajar con clases y objetos en Python de una forma práctica y sin complicaciones?

2. Le invito a explorar el repositorio interactivo de [LearnPython.org](#) sobre [Clases y Objetos](#), donde encontrarás ejemplos claros



Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 6

¡Los métodos en Python son como los superpoderes de tus objetos! En la Programación Orientada a Objetos, los métodos son las acciones que sus objetos pueden realizar, desde algo simple como calcular un total hasta complejo como conectarse a una base de datos. En Python, definir

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 26**

Código fuente de métodos de una clase en Python

```
1  class Clase:
2      def metodo(self):
3          return 'Método normal', self
4
5      @classmethod
6      def metododeclase(cls):
7          return 'Método de clase', cls
8
9      @staticmethod
10     def metodoestatico():
11         return "Método estático"
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

### 2.6.1. Métodos de instancia

Los métodos de instancia son los métodos normales. Reciben como parámetro de entrada **self**

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 27**

Código fuente de una clase son *self* en Python

```
1  class Clase: 1 usage
2      def metodo(self, arg1, arg2): 1 usage
3          return 'Método normal', self
4
5  #creación del objeto objeto_clase
6  objeto_clase = Clase()
7  objeto_clase.metodo(arg1: "a", arg2: "b") #envio de argumentos al método
8
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

### 2.6.2. Métodos de clase @classmethod

Los métodos de clase reciben como argumento *cls*, que hace referencia

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

### 2.6.3. Métodos estáticos `@staticmethod`

Los métodos estáticos se pueden definir con el decorador `@staticmethod` y no aceptan como parámetro ni la instancia ni la clase. Por ende, no pueden modificar el estado ni de la clase ni de la instancia y sí pueden aceptar parámetros de entrada.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

## 2.7. Getters y setters

En POO aplicado en Python, los getters y setters son métodos utilizados para acceder (get) y modificar (set) los atributos de una clase de manera controlada, respetando el principio de encapsulamiento (ocultar detalles internos y proteger la integridad de los datos).

Ejemplo de implementación de getter y setters representado por la clase Persona

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 30**

Código fuente de una clase con los métodos get y set en Python

```
1  class Persona: 1 usage
2      def __init__(self, nombre, edad):
3          self._nombre = nombre
4          self._edad = edad
5
6      # Getter como propiedad
7      @property 2 usages
8      def nombre(self):
9          return self._nombre
10
11     # Setter asociado a la propiedad
12     @nombre.setter
13     def nombre(self, nuevo_nombre):
14         self._nombre = nuevo_nombre
15
16     @property 1 usage
17     def edad(self):
18         return self._edad
19
20     @edad.setter 2 usages
21     def edad(self, nueva_edad):
22         if nueva_edad >= 0:
23             self._edad = nueva_edad
24         else:
25             raise ValueError("La edad no puede ser negativa")
26
27     # Uso de getter y setter por medio de los objetos
28 persona = Persona(nombre="Luis", edad=28)
29 print(persona.nombre) # Output: Luis (usa el getter como atributo)
30 persona.edad = 30      # Usa el setter (válido)
31 persona.edad = -3     # Lanza ValueError
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**Ventajas:**

- Sintaxis más limpia, ya que permite el acceso directo a atributos.
-

## ¿Cuándo usar getters/setters?

- **Validación:** asegurar que los valores asignados cumplan reglas.  
Ejemplo: edad > 0.
- **Derivar datos:** calcular un valor al acceder (ejemplo: `@property def nombre_completo`).

índice

I Bimestre

II Bimestre

Solucionario

Referencias

y setters en Python. Cada ejemplo está comentado para que entienda el "por qué" detrás de cada línea.

**Perfecto para:**

- Quienes aprenden mejor viendo código funcional.
- Refrescar conceptos como clases, atributos, métodos, objetos.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

### Estrategia de trabajo:

- Detecte los puntos clave que se mencionan en este recurso, estos son temas claves que tiene que saber y comprender en la POO.
- Tenga un registro de los puntos clave que detecte en esta semana para mayor comprensión del contenido.

índice

I Bimestre

II Bimestre

Solucionario

Referencias



Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 7

¡La separación de comportamientos y su implementación es cómo organizar una cocina profesional! En la Programación Orientada a Objetos, no basta con saber qué hace un objeto y sus comportamientos, sino también cómo

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 31**

Código fuente de una clase abstracta en Python

```
1  from abc import ABC, abstractmethod
2
3  class Forma(ABC): # Comportamiento definido (¿qué hace?)
4      @abstractmethod
5      def area(self) -> float:
6          pass
7
8      @abstractmethod
9      def perimetro(self) -> float:
10         pass
11
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

## 2.9. Implementación

Como puede observarse en la siguiente figura:

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 32**

Código fuente de una clase separada los métodos en Python

```
1  class Cuadrado(Forma):
2      def __init__(self, lado: float):
3          self.lado = lado
4
5      def area(self) -> float: # Implementación concreta
6          return self.lado ** 2
7
8      def perimetro(self) -> float:
9          return 4 * self.lado
10
11 class Circulo(Forma):
12     def __init__(self, radio: float):
13         self.radio = radio
14
15     def area(self) -> float:
16         return 3.1416 * self.radio ** 2
17
18     def perimetro(self) -> float:
19         return 2 * 3.1416 * self.radio
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**Beneficios de esta separación**

- **Flexibilidad:** puede cambiar la implementación sin afectar el

Ejemplo de uso, como se muestra en la siguiente figura:

### Figura 33

Código fuente de ejemplo de una clase con el método en Python

```
21 def imprimir_info(forma: Forma): # Acepta cualquier objeto que cumpla con Forma 2 usages
22     print(f"Área: {forma.area()}, Perímetro: {forma.perimetro()}")
23
24 cuadrado = Cuadrado(Lado=5)
25 circulo = Circulo(radio=3)
26
27 imprimir_info(cuadrado) # Área: 25, Perímetro: 20
28 imprimir_info(circulo) # Área: 28.2744, Perímetro: 18.8496
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

Separar comportamientos (interfaces) de implementaciones (clases)

Índice

I Bimestre

II Bimestre

Solucionario

Referencias



## Actividades de aprendizaje recomendadas

¡Bienvenidos a las actividades recomendadas de aprendizaje de la semana 7! Las siguientes actividades están diseñadas para que alcance el resultado de aprendizaje, en ellas tendrá que aplicar cómo separar los comportamientos de las clases en Programación Orientada a Objetos. A continuación, le planteo unas actividades para repasar los contenidos de esta semana.

1. Te invito a explorar el repositorio en

índice

I Bimestre

II Bimestre

Solucionario

Referencias

2. Estimado estudiante, ha llegado el momento de evaluar su comprensión sobre los temas abordados en la Unidad 2. Programación Orientada a Objetos, los cuales incluyen elementos básicos de la programación en POO, manejo de clases y atributos, métodos en lenguaje de programación Python. Esta autoevaluación le permitirá identificar qué conceptos domina y en



## Autoevaluación 2

1. ¿Qué decorador se usa para definir un método estático en Python?

- a. @classmethod
- b. @classmethod

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**4. ¿Qué tipo de método se puede llamar sin instanciar un objeto de la clase?**

- a. Método de instancia.
- b. Método estático.
- c. Método privado.
- d. Método público.

**5.**

índice

I Bimestre

II Bimestre

Solucionario

Referencias

10. Empareja el tipo de método con su descripción:

Tipo de método	Descripción
A) Método de clase	1) Accede solo a parámetros generales de la clase.



Sem 1 Sem 2 Sem 3 Sem 4 Sem 5 Sem 6 Sem 7 Sem 8



## Semana 8



### Actividades finales del bimestre

La semana 8 es un período importante para que usted, estimado estudiante, pueda volver a revisar los temas estudiados en el primer bimestre y reforzar aquellos conceptos que requieran mayor comprensión. Este tiempo está destinado a consolidar su aprendizaje y

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Consultar en los recursos de aprendizaje recomendados en la planificación de la asignatura.
- Practicar con ejercicios similares a los presentados en la guía didáctica para reforzar la resolución de problemas.
- Repasar las preguntas de autoevaluación para evaluar su

## Resultado de aprendizaje 2



### Segundo bimestre

- Diseña modelos avanzados con Programación Orientada a Objetos (POO) desarrollando soluciones innovadoras que optimicen la visualización de datos y la adaptación en infraestructuras tecnológicas

En la asignatura de Programación Orientada a Objetos, para poder

## Contenidos, recursos y actividades de aprendizaje recomendadas

índice

I Bimestre

II Bimestre

Solucionario

Referencias



Sem 9

Sem 10

Sem 11

Sem 12

Sem 13

Sem 14

Sem 15

Sem 16



## Semana 9

**¿Se ha preguntado cómo los programas pueden organizarse de forma parecida a un árbol genealógico?**

Imagine que está construyendo una familia de clases en Python. Con la **herencia**, puede crear una clase padre (como Animal) que comparta

índice

I Bimestre

II Bimestre

Solucionario

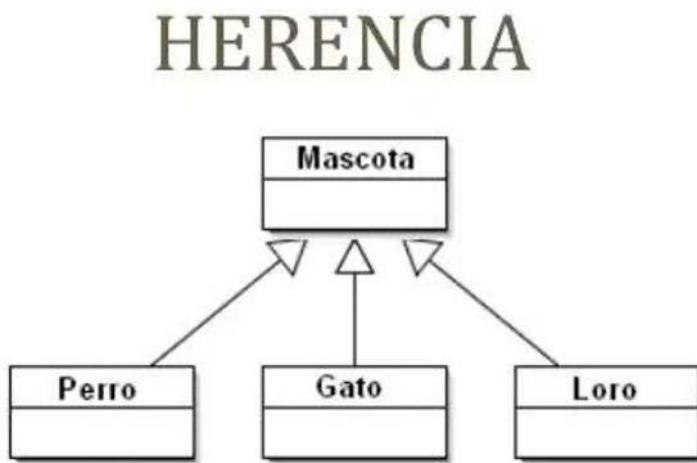
Referencias

También, una clase hija puede sobreescribir los métodos o atributos, o incluso definir unos nuevos.

Se puede crear una clase hija con tan solo pasar como parámetro la clase de la que queremos heredar. En el siguiente ejemplo vemos cómo se puede usar la herencia en Python, con la clase Perro que hereda de Mascota. Así de fácil.

**Figura 34**

Representación UML de herencia



Nota.

**Figura 35**

Código fuente de una clase con herencia en Python

```
1  @ class Mascota: # Clase padre 1 usage
2      def __init__(self, nombre):
3          self.nombre = nombre
4
5  @     def hacer_sonido(self):
6      return "¡Sonido genérico!"
7
8  class Perro(Mascota): # Clase hija 1 usage
9  @     def hacer_sonido(self): # Sobrescritura de método 1 usage
10        return "¡Guau guau!"
11
12 mi_perro = Perro("Firulais")
13 print(mi_perro.nombre)           # Salida: "Firulais" (heredado)
14 print(mi_perro.hacer_sonido())   # Salida: "¡Guau guau!" (sobrescrito)
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

La clase padre **Mascota**, que generaliza las características y funcionalidades que toda mascota puede tener. Ahora creamos una

## Figura 36

Código fuente de una clase con herencia múltiple en Python

```
1 @+ class Nadador: 1 usage
2     def nadar(self): 1 usage
3         return "Nadando"
4
5 @+ class Volador: 1 usage
6     def volar(self): 1 usage
7         return "Volando"
8
9 class Pato(Nadador, Volador): # Hereda de dos clases 1 usage
10    pass
11
12 #creacion de objetos
13 pato = Pato()
14 print(pato.nadar()) # Output: "Nadando"
15 print(pato.volar()) # Output: "Volando"
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

La clase hija Pato hereda de la clase Nadador y Volador, como se muestra en la línea 9.

Creamos el objeto pato en la línea de código 13 para poder acceder a los métodos nadar en la línea 14 de la clase Nadador y colar en la línea

**Figura 37**

Código fuente de una clase con el método overriding en Python

```
1 @ class Vehiculo: 1 usage
2 @     def arrancar(self):
3         return "Motor encendido"
4
5 class CocheElectricos(Vehiculo): 1 usage
6 @     def arrancar(self): # Sobrescribe el método 1 usage
7         return "Bateria activada"
8
9 auto = CocheElectricos()
10 print(auto.arrancar()) # Output: "Bateria activada" (no "Motor encendido")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

En la línea 10 ejecutamos el método arrancar() y se muestra el método de la clase hija y no el método heredado de la clase padre.

**Extensión de métodos (super())**

Se puede **añadir funcionalidad**

**Figura 38**

Código fuente de una clase con el método super en Python

```
1 @l class Persona: 1 usage
2 @l     def __init__(self, nombre):
3         self.nombre = nombre
4
5 class Estudiante(Persona): 1 usage
6     def __init__(self, nombre, carrera):
7         super().__init__(nombre) # Llama al __init__ de Persona
8         self.carrera = carrera
9
10 estudiante = Estudiante(nombre: "Ana", carrera: "Ingeniería")
11 print(estudiante.nombre) # Output: "Ana" (heredado)
12 print(estudiante.carrera) # Output: "Ingeniería" (nuevo)
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

En el ejemplo, la línea 7 llamamos al constructor de la clase padre y enviamos el atributo nombre.

Por medio de las líneas 11 y 12, creamos los objetos para acceder al atributo

índice

I Bimestre

II Bimestre

Solucionario

Referencias

### 3.5. Herencia multinivel

Una clase hereda de otra que a su vez hereda de una tercera. Para este ejemplo se registra la clase **Hijo** que hereda de la clase **Padre** y a su vez hereda métodos de la clase **Abuelo**.

Por medio del objeto hijo creado en la línea 12, se puede acceder a los

Concluido el quiz, puede establecerse que la herencia en Python favorece la reutilización de código, la extensión de funcionalidades y el acceso jerárquico a métodos mediante super() y estructuras multinivel.



## Actividades de aprendizaje recomendadas

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 9! Las siguientes actividades están diseñadas para que alcance el resultado de aprendizaje, en ellas tendrá que entender, comprender y

2. Leer y analizar la sección "Herencia" del capítulo cinco, "Programación Orientada a Objetos", del texto "Python 3: Curso Práctico", donde se explican los conceptos y ejercicios aplicando Herencia. Esta actividad permitirá al estudiante comprender cómo diseñar con UML y posteriormente desarrollar en Python las clases

índice

I Bimestre

II Bimestre

Solucionario

Referencias



Sem 9

Sem 10

Sem 11

Sem 12

Sem 13

Sem 14

Sem 15

Sem 16



## **Semana 10**

En esta semana veremos en detalle el polimorfismo que permite que un mismo método, como `hacer_sonido()` se comporte de manera diferente según la clase que lo use. Por ejemplo: un perro ladra, un gato maúlla, y un pájaro canta, pero todos responden al mismo

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 40**

Código fuente de una clase con polimorfismo en Python

```
1 @+ class Animal: 2 usages
2 @+     def hacer_sonido(self):
3         return "Sonido genérico"
4
5 class Perro(Animal): 1 usage
6 @+     def hacer_sonido(self): # Sobrescritura 1 usage
7         return ";Guau guau!"
8
9 class Gato(Animal): 1 usage
10 @+    def hacer_sonido(self): # Sobrescritura 1 usage
11        return ";Miau miau!"
12
13 # Uso del polimorfismo
14 miPerro=Perro()
15 miGato=Gato()
16 animales = [miPerro, miGato]
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

- **Polimorfismo por sobrecarga por**

**Figura 41**

Código fuente de una clase con polimorfismo Duck en Python

```
1  class Pato: 1 usage
2      def graznar(self): 1 usage (1 dynamic)
3          return "¡Cuack!"
4
5  class Persona: 1 usage
6      def graznar(self): 1 usage (1 dynamic)
7          return ";¡Imito a un pato!"
8
9  def hacer_sonido(objeto): 2 usages
10     print(objeto.graznar())
11
12 pato = Pato()
13 persona = Persona()
14
15 hacer_sonido(pato)    # Output: "¡Cuack!"
16 hacer_sonido(persona) # Output: ";¡Imito a un pato!"
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

¿Le resulta complicado entender la herencia en Programación Orientada a Objetos? ¡No se preocupe! En este video:

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 42**

Código fuente de una clase con polimorfismo con interfaces en Python

```
1  from abc import ABC, abstractmethod
2
3  class Forma(ABC): 2 usages
4      @abstractmethod
5  def area(self):
6      pass
7
8  class Cuadrado(Forma): 1 usage
9      def __init__(self, lado):
10         self.lado = lado
11
12 def area(self): 1 usage
13     return self.lado ** 2
14
15 class Circulo(Forma): 1 usage
16     def __init__(self, radio):
17         self.radio = radio
18
19 def area(self): 1 usage
20     return 3.1416 * self.radio ** 2
21
22 formas = [Cuadrado(4), Circulo(3)]
23 for forma in formas:
24     print(f"Área: {forma.area()}")
25
26 #Área: 16
27 #Área: 28.2744
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

Para comprobar sus conocimientos sobre este tema, le invito a participar en el siguiente juego de arrastrar y soltar:

Concluida la experiencia, puede establecerse que el polimorfismo en Python favorece la reutilización y flexibilidad del código; finalmente, se propone un quiz para reforzar los conocimientos adquiridos.

### Polimorfismo en Python

El polimorfismo en Python es un principio clave de la POO que aporta

### Estrategia de trabajo:

- Utilice un IDE de desarrollo para poder ejecutar el código del repositorio.
- Configure su ordenador con Git para poder acceder de mejor manera al repositorio.
-



Sem 9 Sem 10 Sem 11 Sem 12 Sem 13 Sem 14 Sem 15 Sem 16



## Semana 11

**¿Se ha preguntado cómo proteger la información valiosa para que solo pueda ser usada de la forma correcta?**

Imagine que sus clases son pequeñas cajas mágicas: los atributos son sus tesoros internos y los métodos son las llaves que controlan quién

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Ejemplo 1:** como se observa en la siguiente figura.

### Figura 43

Código fuente de una clase con encapsulación en Python

```
1  class Persona:
2      def __init__(self, nombre, edad):
3          self.__nombre = nombre      # Atributo privado
4          self.__edad = edad         # Atributo privado
5
6      def get_nombre(self):
7          return self.__nombre
8
9      def set_nombre(self, nuevo_nombre):
10         self.__nombre = nuevo_nombre
11
12     def get_edad(self):
13         return self.__edad
14
15     def set_edad(self, nueva_edad):
16         if nueva_edad > 0:
17             self.__edad = nueva_edad
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

En el ejemplo, específicamente en la línea de código 3 y 4,

**Figura 44**

Código fuente de un ejemplo de encapsulamiento en Python

```
1  class CuentaBancaria: 1 usage
2      def __init__(self, titular, saldo):
3          self.__titular = titular # Atributo privado
4          self.__saldo = saldo # Atributo privado
5
6      # Getter para saldo
7      def get_saldo(self): 3 usages
8          return self.__saldo
9
10     # Setter para saldo con validación
11     def set_saldo(self, cantidad):
12         if cantidad >= 0:
13             self.__saldo = cantidad
14         else:
15             print("El saldo no puede ser negativo")
16
17     # Método para depositar dinero
18     def depositar(self, cantidad): 1 usage
19         if cantidad > 0:
20             self.__saldo += cantidad
21         else:
22             print("La cantidad a depositar debe ser positiva")
23
24     # Método para retirar dinero
25     def retirar(self, cantidad): 1 usage
26         if 0 < cantidad <= self.__saldo:
27             self.__saldo -= cantidad
28         else:
29             print("Cantidad no válida o saldo insuficiente")
30
31     # Uso de la clase
32     cuenta = CuentaBancaria(titular="Juan Pérez", saldo=1000)
33     print(cuenta.get_saldo()) # 1000
34     cuenta.depositar(500)
35     print(cuenta.get_saldo()) # 1500
36     cuenta.retirar(200)
37     print(cuenta.get_saldo()) # 1300
38
39     # Esto no funcionará (atributo privado)
40     # print(cuenta.__saldo) # Error!
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

En las líneas 3 y 4 se crean los atributos privados de la clase CuentaBancaria, los métodos getter y setter en la línea 7 y 11 respectivamente, para poder acceder de manera segura.

Los métodos retirar y depositar, cada una de ellas esperan un parámetro que es la cantidad.

Ya en el uso y la creación de objetos de la línea 32 podemos enviar los datos respectivos por medio del constructor.

### Ejemplo en la vida real:

Cuando conduces un vehículo, usas el volante, freno y acelerador, sin saber cómo funciona el motor internamente. Eso es abstracción.

**Ejemplo 1:** como se observa en la siguiente figura.

#### Figura 45

Código fuente de encapsulamiento en Python

```
1  from abc import ABC, abstractmethod
2
3 @class Animal(ABC): # Clase abstracta 2 usages
4     @abstractmethod
5     def hacer_sonido(self):
6         pass
7
8     class Perro(Animal):
9         def hacer_sonido(self):
10            print("Guau!")
11
12     class Gato(Animal):
13         def hacer_sonido(self):
14            print("Miau!")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a

**Figura 46**

Código fuente de métodos abstractos en Python

```

1  from abc import ABC, abstractmethod
2
3
4  class FiguraGeometrica(ABC): 2 usages
5      @abstractmethod
6      def area(self):
7          pass
8
9      @abstractmethod
10     def perimetro(self):
11         pass
12
13
14  class Rectangulo(FiguraGeometrica): 1 usage
15      def __init__(self, base, altura):
16          self.base = base
17          self.altura = altura
18
19      def area(self): 1 usage
20          return self.base * self.altura
21
22      def perimetro(self): 1 usage
23          return 2 * (self.base + self.altura)
24
25
26  class Circulo(FiguraGeometrica): 1 usage
27      def __init__(self, radio):
28          self.radio = radio
29
30      def area(self): 1 usage
31          return 3.1416 * self.radio ** 2
32
33      def perimetro(self): 1 usage
34          return 2 * 3.1416 * self.radio
35
36
37 # Uso de las clases
38 rect = Rectangulo(base=5, altura=3)
39 print(f"Área del rectángulo: {rect.area()}")
40 print(f"Perímetro del rectángulo: {rect.perimetro()}")
41
42 circ = Circulo(4)
43 print(f"Área del círculo: {circ.area():.2f}")
44 print(f"Perímetro del círculo: {circ.perimetro():.2f}")
45
46
47 # Esto dará error (no se puede instanciar una clase abstracta)
48 # figura = FiguraGeometrica()

```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

## Resumen de la encapsulación y la abstracción

- **Encapsulación:** se trata de ocultar los detalles de implementación y proteger los datos.
- **Abstracción:** se trata de simplificar la complejidad mostrando solo

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Perfecto para:**

- Quienes aprenden mejor viendo código funcional.
- Refrescar conceptos como clases, atributos, métodos, objetos.

*“¿Le quedan dudas después de revisarlo? ¡Aquí estoy para explicarle lo que necesite! ”*

**Estrategia de trabajo:**

- Utilice una libreta de anotaciones para que registre los temas importantes y relevantes que los detecte.
- Puede socializar las anotaciones en el foro de la materia.



Sem 9

Sem 10

Sem 11

Sem 12

Sem 13

Sem 14

Sem 15

Sem 16



## Semana 12

**¡El manejo de excepciones en Python es como tener un paracaídas para cuando tu código decide hacer acrobacias peligrosas!** En la Programación Orientada a Objetos, los errores no son el fin del mundo, sino oportunidades para recuperarse con elegancia. Imagina que tus

índice

I Bimestre

II Bimestre

Solucionario

Referencias

de sus programas y facilitar la depuración del código. ¡Continúe practicando para fortalecer sus habilidades!

Como se puede observar en la siguiente figura, la estructura básica (try-except)

### Figura 47

Código fuente de try-except en Python

```
1  try:
2      # Código que puede fallar
3      resultado = 10 / 0
4  except ZeroDivisionError:
5      # Manejo del error
6      print("¡Error: División por cero!")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a

**Figura 48**

Código fuente de excepción personalizada en Python

```
1  class SaldoInsuficienteError(Exception): 2 usages
2      """Excepción personalizada para saldo insuficiente."""
3      pass
4
5  class CuentaBancaria: 1 usage
6      def __init__(self, saldo_inicial):
7          self.saldo = saldo_inicial
8
9      def retirar(self, monto): 1 usage
10         if monto > self.saldo:
11             raise SaldoInsuficienteError("Saldo insuficiente.")
12         self.saldo -= monto
13         return self.saldo
14
15     # Uso
16     try:
17         cuenta = CuentaBancaria(100)
18         cuenta.retirar(200)
19     except SaldoInsuficienteError as e:
20         print(f"Error: {e}")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

Como podemos observar en la siguiente figura, en el ejemplo 2: Múltiples excepciones:

**Figura 49**

Código fuente de múltiples excepciones en Python

```
1  try:
2      numero = int(input("Ingrese un número: "))
3      resultado = 10 / numero
4      lista = [1, 2, 3]
5      print(lista[10])
6  except ValueError:
7      print("¡Debe ingresar un número válido!")
8  except ZeroDivisionError:
9      print("¡No se puede dividir por cero!")
10 except IndexError:
11     print("¡Índice fuera de rango!")
12 except Exception as e: # Captura cualquier otro error
13     print(f"Error inesperado: {e}")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**Bloque else y finally**

- **else:**

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

**Figura 50**

Código fuente de lectura de archivos en Python

```
1  try:
2      archivo = open("datos.txt", "r")
3      contenido = archivo.read()
4  except FileNotFoundError:
5      print("¡Archivo no encontrado!")
6  else:
7      print("Lectura exitosa.")
8      print(contenido)
9  finally:
10     archivo.close() if 'archivo' in locals() else None
11     print("Proceso finalizado.")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**¿Le resulta complicado entender la Encapsulación en Programación Orientada a Objetos? ¡No se preocupe! En el video**

3. Evita try-except demasiado amplios, solo captura lo necesario.
4. Usa finally para liberar recursos, archivos, conexiones a BD.



### Actividades de aprendizaje recomendadas

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 12! Las siguientes actividades están diseñadas para que

índice

I Bimestre

II Bimestre

Solucionario

Referencias

Programación Orientada a Objetos, Herencia y polimorfismo en Python. Esta autoevaluación le permitirá identificar qué conceptos domina y en cuáles debería reforzar su aprendizaje.

Previo a realizar la autoevaluación, se recomienda seguir las siguientes estrategias de estudio:

-



## Autoevaluación 3

1. ¿Qué palabra clave se utiliza en Python para establecer una relación de herencia entre dos clases?
  - a. extends.
  - b. base.
  - c. inherits.
  - d.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

5. ¿Qué estructura permite manejar una excepción en Python?

- a. if-else
- b. try-catch
- c. try-except
- d. raise-rescue

6. ( )

índice

I Bimestre

II Bimestre

Solucionario

Referencias

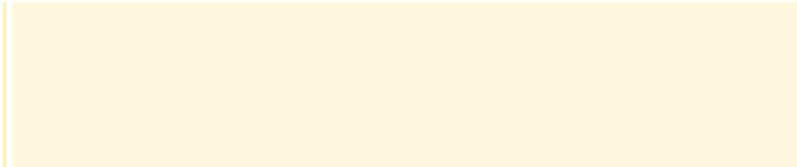


Sem 9 Sem 10 Sem 11 Sem 12 Sem 13 Sem 14 Sem 15 Sem 16



## Semana 13

¡Diseñar clases en Python con relaciones avanzadas es como ser un arquitecto digital! En la Programación Orientada a Objetos, no solo creamos clases básicas, sino que también definimos cómo interactúan



índice

I Bimestre

II Bimestre

Solucionario

Referencias

Se usa cuando hay una relación “**tiene un**”,

**Ejemplo:** un Auto **tiene un** Motor. Como se puede observar en la siguiente figura:

### Tabla 1

Herencia y Composición



**Figura 51**

Código fuente de SOLID en Python

```
1 ##### Mal diseño
2 class Reporte:
3     def calcular_datos(self): ...
4     def imprimir_pdf(self): ...
5
6 ### Buen diseño
7 class CalculadorReporte:
8     def calcular_datos(self): ...
9
10 class Exportador:
11     def imprimir
```



Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

Los principios SOLID tienen las siguientes características:

- **SRP:** una clase, una responsabilidad.

**Composición:** más fuerte que la agregación; la clase contenida no puede existir sin la clase principal.

### Figura 52

Código fuente composición en Python

```
1  class Profesor:
2      def __init__(self, nombre):
3          self.nombre = nombre
4
5  class Universidad:
6      def __init__(self):
7          self.profesores = [] # Agregación
8
9      def agregar_profesor(self, profesor):
10         self.profesores.append(profesor)
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

Una **clase abstracta** no se puede instanciar y puede contener métodos abstractos que las clases hijas deben implementar.

### Figura 53

Código fuente de clases abstractas en Python

```
1  from abc import ABC, abstractmethod
2
3  class Vehiculo(ABC):
4      @abstractmethod
5      def acelerar(self):
6          pass
7
8  class Coche(Vehiculo):
9      def acelerar(self):
10         print("El coche acelera")
11
12 class Bicicleta(Vehiculo):
13     def acelerar(self):
14         print("La bicicleta acelera")
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.



## Actividad de aprendizaje recomendada

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 13! La siguiente actividad está diseñada para que alcance el resultado de aprendizaje, en ellas tendrá que aplicar los conocimientos de programar POO con interfaces y clases abstractas.

1. Lo invito a seguir explorando el repositorio en [GitHub](#) donde



Sem 9 Sem 10 Sem 11 Sem 12 Sem 13 Sem 14 Sem 15 Sem 16



## Semana 14

¡Los patrones de diseño son como los trucos de magia que todo desarrollador experto guarda en su manga! Y cuando se combinan con aplicaciones avanzadas de POO, se convierten en superpoderes para crear software robusto, flexible y elegante. Los patrones de diseño

índice

I Bimestre

II Bimestre

Solucionario

Referencias

### a. Singleton

Asegura que una clase tenga **una única instancia** y proporcione un punto global de acceso.

**Figura 54**

Código fuente de singleton en Python

```
1 class Singleton: __usage__
2     _instance = None
3
4     def __new__(cls):
5         if cls._instance is None:
6             cls._instance = super(Singleton, cls).__new__(cls)
7         return cls._instance
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

### c. Builder

Separa la construcción de un objeto complejo de su representación.

**Figura 56**

Código fuente de builder en Python

```
1  class PizzaBuilder:
2      def __init__(self):
3          self.pizza = {}
4
5      def add_queso(self):
6          self.pizza["queso"] = True
7
8      def add_pepperoni(self):
9          self.pizza["pepperoni"] = True
10
11     def build(self):
12         return self.pizza
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

#### 4.2.2.

## Figura 57

Código fuente de adapter en Python

```
1  class EnchufeEuropeo:
2      def conectar(self): 1 usage (1 dynamic)
3          return "220V"
4
5  class Adaptador:
6      def __init__(self, enchufe):
7          self.enchufe = enchufe
8
9      def conectar(self): 1 usage (1 dynamic)
10         return f"Convertido: {self.enchufe.conectar()} a 110V"
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

## b. Decorator

Agrega funcionalidades a objetos de forma dinámica.

## Figura 58

Código fuente de un decorador en Python

```
1  def log(func):
2      def wrapper():
3          print("Llamando a", func.__name__)
4          return func()
5      return wrapper
6
7  @log
8  def saludar():
9      print("Hola")
```

### c. Facade

Proporciona una interfaz unificada a un conjunto de interfaces más complejas.

**Figura 59**

Código fuente de facade en Python

```
1 class Motor: 1 usage
2     def encender(self): print("Motor encendido") 1 usage
3
4 class Luces: 1 usage
5     def encender(self): print("Luces encendidas") 1 usage
6
7 class Auto:
8     def __init__(self):
9         self.motor = Motor()
10        self.luces = Luces()
11
12    def arrancar(self):
13        self.motor.encender()
14        self.luces.encender()
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**a. Observer**

Un objeto notifica a sus observadores cuando cambia su estado.

**Figura 60**

Código fuente de observar en Python

```
1  class Sujeto:
2      def __init__(self):
3          self._observadores = []
4
5      def agregar(self, obs):
6          self._observadores.append(obs)
7
8      def notificar(self, mensaje):
9          for obs in self._observadores:
10              obs.actualizar(mensaje)
11
12  class Observador:
13      def actualizar(self, mensaje): 1 usage (1 dynamic)
14          print("Notificado con:", mensaje)
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**b.**

### c. Command

Encapsula una solicitud como un objeto.

**Figura 62**

Código fuente de command en Python

```
1  class EncenderLuz:
2      def ejecutar(self):  1 usage (1 dynar)
3          print("Luz encendida")
4
5  class Interruptor:
6      def __init__(self, comando):
7          self.comando = comando
8
9      def presionar(self):
10         self.comando.ejecutar()
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**¿Le resulta complicado entender los patrones de diseño?**

## 4.3. Aplicaciones avanzadas de POO

### 1. Persistencia de objetos (Serialización con pickle y json)

La persistencia de objetos se refiere a la capacidad de guardar el

índice

I Bimestre

II Bimestre

Solucionario

Referencias

## 2. Conexión a bases de datos (ORM con SQLAlchemy)

Como podemos observar en la siguiente figura:

**Figura 64**

Código fuente de una conexión a una db en Python

```
1  from sqlalchemy import Column, Integer, String, create_engine
2  from sqlalchemy.orm import declarative_base, sessionmaker
3
4  Base = declarative_base()
5
6  class Usuario(Base):
7      __tablename__ = 'usuarios'
8      id = Column(Integer, primary_key=True)
9      nombre = Column(String)
10
11  engine = create_engine('sqlite:///mi_db.db')
12  Base.metadata.create_all(engine)
13  Session = sessionmaker(bind=engine)
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

## 3.

#### 4. Programación de Interfaces gráficas (POO en tkinter o PyQt)

De acuerdo con lo representado en la siguiente figura:

**Figura 66**

Código fuente de *interfaces en Python*

índice

I Bimestre

II Bimestre

Solucionario

Referencias

y codificar aplicando técnicas avanzadas de desarrollo en la Programación Orientada a Objetos.

1. Lo invito a seguir explorando el repositorio en [GitHub](#) donde encontrará ejemplos claros y prácticos de Manejos de Excepciones en Python. Podrá ver cómo se estructura la separación de comportamientos en Python. Cada ejemplo está comentado para



Sem 9 Sem 10 Sem 11 Sem 12 Sem 13 Sem 14 Sem 15 Sem 16



## Semana 15

¡Las buenas prácticas en POO son como las reglas de tránsito para su código: hacen que todo fluya sin choques ni atascos! Cuando domina la Programación Orientada a Objetos, no solo se trata de que funcione, sino de que sea limpio, eficiente y fácil de mantener, incluso cuando

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- **Uso de properties (@property, @setter)**

### Figura 68

Código fuente del uso de properties en Python

```
1  class Producto:
2      def __init__(self, precio):
3          self._precio = precio
4
5      @property 1 usage
6      def precio(self):
7          return self._precio
8
9      @precio.setter
10     def precio(self, nuevo):
11         if nuevo >= 0:
12             self._precio = nuevo
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

- **Inyección de d**

Índice

I Bimestre

II Bimestre

Solucionario

Referencias

- **Testing en POO** (Pruebas unitarias con unittest y pytest)

### Figura 70

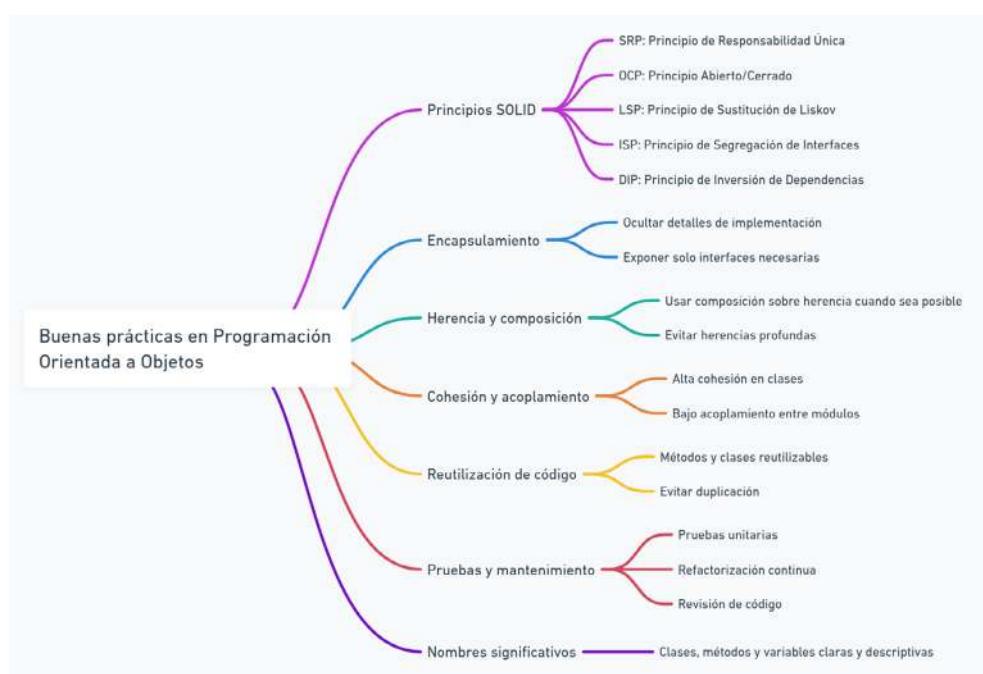
Código fuente de testing en Python

```
1 import unittest  
2  
3 ▷ class PruebasBasicas(unittest.TestCase):  
4 ▷     def test_suma(self):  
5         self.assertEqual(2 + 2, second: 4)  
6  
7 ▷     if __name__ == "__main__":  
8         unittest.main()
```

Nota. Este código fuente de Python se utiliza en la Programación Orientada a Objetos. Ramírez, R., 2025.

**Figura 71**

Mapa mental de las buenas prácticas de la POO



Nota. Este mapa mental muestra la clasificación de las buenas prácticas de la Programación Orientada a Objetos. Ramírez, R., 2025.

Creado por Ramírez, R. (2025) con la asistencia de OpenAI y Whimsical.

¡Felicitaciones, concluimos la unidad 4! Para poner en práctica los conceptos estudiados esta semana, le invito a s 3.himsical.



## Actividades de aprendizaje recomendadas

¡Bienvenido a las actividades recomendadas de aprendizaje de la semana 15! Las siguientes actividades están diseñadas para que alcance el resultado de aprendizaje, en ellas tendrá que aplicar buenas prácticas de codificación en Python en la Programación Orientada a Objetos.

1.

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Practicar con ejercicios sobre creación de clases, atributos, métodos y el consumo de métodos.

Recuerde que cada pregunta tiene una única respuesta correcta.

Lea con atención cada enunciado y seleccione la opción que usted considere adecuada. ¡Mucho éxito en esta autoevaluación!



## Autoevaluación 4

1. ¿Cuál es el principio del SOLID que sugiere que una clase debe tener solo una razón para cambiar?

- a. Principio de Inversión de dependencias.
- b. Principio de Responsabilidad única.
- c. Principio de Sustitución de Liskov..

índice

I Bimestre

II Bimestre

Solucionario

Referencias

**5. ¿Cuál de los siguientes es un patrón de comportamiento?**

- a. Adapter.
- b. Observer.
- c. Facade.
- d. Singleton.

**6. ¿Qué**

índice

I Bimestre

II Bimestre

Solucionario

Referencias



Sem 9 Sem 10 Sem 11 Sem 12 Sem 13 Sem 14 Sem 15 Sem 16



## **Semana 16**



### **Actividades finales del bimestre**

La semana 16 es un período para que usted, estimado estudiante, tenga la oportunidad de volver a revisar los temas estudiados en el segundo bimestre y reforzar aquellos conceptos que requieran mayor comprensión. Este tiempo está destinado a consolidar su aprendizaje y

índice

I Bimestre

II Bimestre

Solucionario

Referencias

- Revisar las actividades realizadas en semanas anteriores y, cuando sea necesario, recrear los ejercicios desde cero para afianzar el manejo de herramientas y conceptos.
- Consultar en los recursos de aprendizaje recomendados en la planificación de la asignatura.



## 4. Solucionario



## Autoevaluación 1

Pregunta      Respuesta

---

---

---

índice

I Bimestre

II Bimestre

Solucionario

Referencias





---

---

---

---

---

índice

I Bimestre

II Bimestre

Solucionario

Referencias





---

---

---

---

---

índice

I Bimestre

II Bimestre

Solucionario

Referencias



### Autoevaluación 3

Pregunta | Respuesta | Retroalimentación

10

---

índice

I Bimestre

II Bimestre

Solucionario

Referencias



índice

I Bimestre

II Bimestre

Solucionario

Referencias



---

---

---

---

---

---





---

## 5. Referencias bibliográficas

---

Cuevas Álvarez, A. (2016). *Python 3: curso práctico*: ( ed.). RA-MA Editorial.  
<https://elibro.net/es/ereader/bibliotecaupl/106404?page=216>

Ramírez Marín, J. H. (2019). *Fundamentos iniciales de lógica de*

Ayala San Martín, G. (2020). *Algoritmos y programación: mejores prácticas*: ( ed.). Fundación Universidad de las Américas Puebla (UDLAP). <https://elibro.net/es/ereader/bibliotecaupl/180290>

Fritelli, V. Guzman, A. & Tymoschuk, J. (2020). *Algoritmos y estructuras de datos*: (2 ed.). Jorge Sarmiento Editor - Universitas. <https://elibro>.

Cuevas Álvarez, A. (2016). *Python 3: curso práctico:* ( ed.). RA-MA Editorial.  
<https://elibro.net/es/ereader/bibliotecautpl/106404>

Moreno Pérez, J. C. (2015). *Programación:* ( ed.). RA-MA Editorial. <https://elibro.net/es/ereader/bibliotecautpl/62476>

Arteaga Martínez, M. M. (2023). *Lógica de programación con Pseint:*