



Programá  
tu futuro



Municipalidad de  
Tres de Febrero

[www.tresdefebrero.gov.ar/tecno3f](http://www.tresdefebrero.gov.ar/tecno3f)

# #ProgramáTuFuturo



Programá  
tu futuro



Municipalidad de  
Tres de Febrero



Municipalidad de  
Tres de Febrero



Programá  
tu futuro

# PYTHON INTERMEDIO

¡Les damos la bienvenida!

</>





Municipalidad de  
Tres de Febrero



Programá  
tu futuro

# PROGRAMACIÓN ORIENTADA A OBJETOS I

</>

CLASE 4

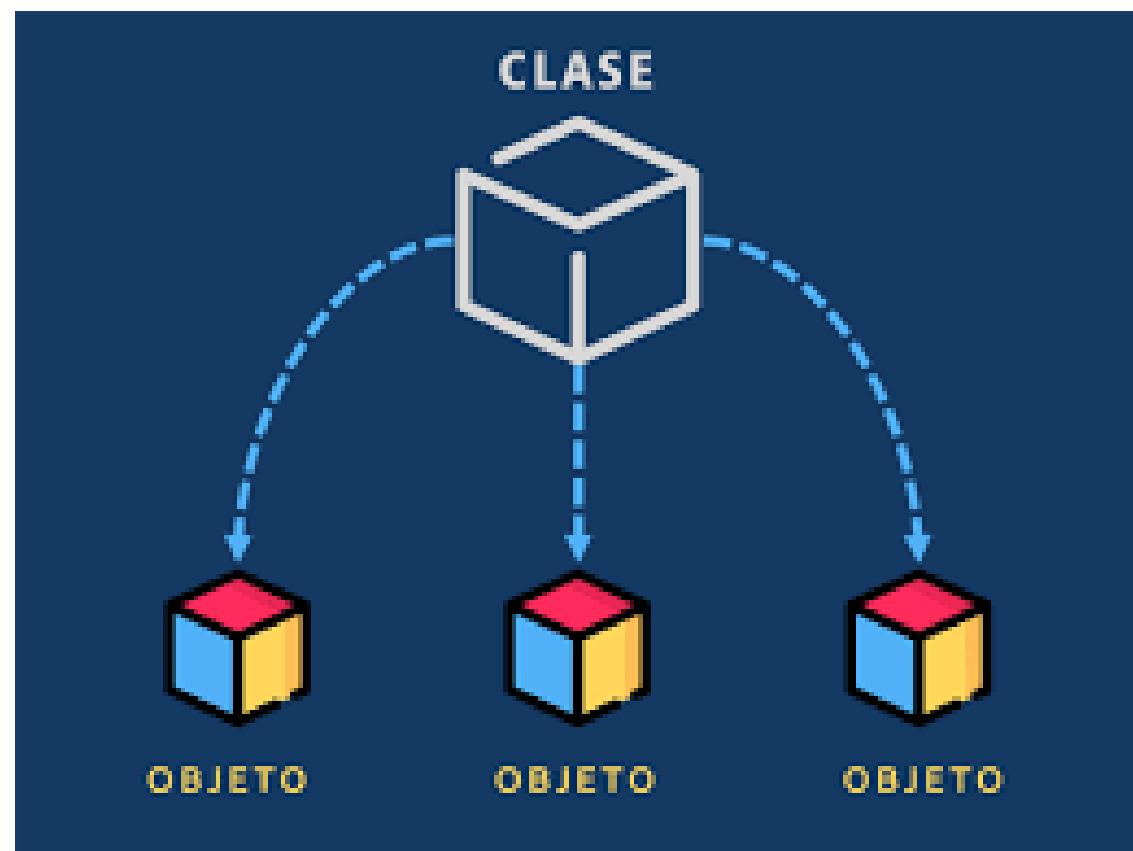


# ¿QUÉ ES?

La POO, o Programación Orientada a Objetos, es un paradigma de programación que se basa en la creación de objetos.

Estos objetos son unidades que encapsulan información y comportamiento, y que se pueden reutilizar en diferentes programas.

Los objetos se definen a partir de **clases**, que son como plantillas que establecen las características y el comportamiento de los objetos que se crearán a partir de ellas. Una clase define los **atributos** (las características) y los **métodos** (el comportamiento) de los objetos.



# Ventajas

**Reutilización de código:** Los objetos se pueden reutilizar en diferentes programas, lo que ahorra tiempo y esfuerzo a los programadores.

**Mantenimiento del código:** El código escrito en POO es más fácil de mantener y actualizar que el código escrito en otros paradigmas de programación.

**Modularidad:** La POO permite dividir el código en módulos independientes, lo que facilita su comprensión y mantenimiento.

**Extensibilidad:** La POO facilita la extensión de los programas con nuevas funcionalidades.

# Conceptos

- **Clase:** Plantilla que define las características y el comportamiento de los objetos.
- **Objeto:** Unidad que encapsula información y comportamiento.
- **Atributo:** Característica de un objeto.
- **Método:** Comportamiento de un objeto.
- **Herencia:** Mecanismo que permite a una clase heredar las características y el comportamiento de otra clase.
- **Polimorfismo:** Mecanismo que permite a un objeto responder de diferentes maneras a un mismo mensaje.



# Pregunta

¿Aparte de Python, que lenguaje conocen o les parece que tambien puede usar POO?

Dejen su respuesta por el chat del  
**ZOOM**



# LOS 4 PILARES



Programá  
tu futuro



Municipalidad de  
Tres de Febrero

# Encapsulamiento

El encapsulamiento consiste en ocultar los detalles de implementación de un objeto. Esto significa que los usuarios del objeto no necesitan saber cómo funciona internamente, solo necesitan saber cómo usarlo. El encapsulamiento se logra mediante el uso de atributos privados y métodos públicos.

## Ventajas

- **Seguridad:** El encapsulamiento ayuda a proteger los datos internos del objeto de accesos no autorizados.
- **Mantenimiento:** El encapsulamiento facilita el mantenimiento del código, ya que los cambios en la implementación interna del objeto no afectan a los usuarios del objeto.
- **Flexibilidad:** El encapsulamiento permite modificar la implementación interna del objeto sin necesidad de modificar el código de los usuarios del objeto.

# Herencia

La herencia es un mecanismo que permite a una clase heredar las características y el comportamiento de otra clase. Esto significa que una clase puede reutilizar el código de otra clase, lo que ahorra tiempo y esfuerzo a los programadores.

## Ventajas

- **Reutilización de código:** La herencia permite reutilizar el código de una clase en otras clases, lo que ahorra tiempo y esfuerzo a los programadores.
- **Mantenimiento del código:** La herencia facilita el mantenimiento del código, ya que los cambios en la clase padre se propagan automáticamente a las clases hijas.
- **Extensibilidad:** La herencia permite extender las funcionalidades de una clase mediante la creación de clases hijas.

# Polimorfismo

El polimorfismo es un mecanismo que permite a un objeto responder de diferentes maneras a un mismo mensaje. Esto significa que el mismo método puede tener diferentes implementaciones en diferentes clases.

## Ventajas

- **Flexibilidad:** El polimorfismo permite escribir código más flexible y adaptable a diferentes situaciones.
- **Reutilización de código:** El polimorfismo permite reutilizar el código de una clase en otras clases, incluso si las clases tienen diferentes implementaciones del mismo método.
- **Extensibilidad:** El polimorfismo permite extender las funcionalidades de una clase mediante la creación de clases hijas con diferentes implementaciones del mismo método.

# Abstracción

La abstracción es un proceso que consiste en identificar las características esenciales de un objeto y ocultar los detalles irrelevantes. Esto permite crear objetos que sean fáciles de entender y usar.

## Ventajas

- **Simplicidad:** La abstracción ayuda a simplificar el código, ya que los usuarios del objeto no necesitan saber cómo funciona internamente.
- **Mantenimiento:** La abstracción facilita el mantenimiento del código, ya que los cambios en la implementación interna del objeto no afectan a los usuarios del objeto.
- **Reutilización de código:** La abstracción permite reutilizar el código de una clase en otras clases, incluso si las clases tienen diferentes implementaciones del mismo método.

# TEST



Programá  
tu futuro



Municipalidad de  
Tres de Febrero

## ¿A que Pilar se hace referencia?

Imagina una clase Coche que tiene un **atributo privado** `_velocidad` que almacena la velocidad actual del coche.

La clase también tiene un **método público** `acelerar()` que aumenta la velocidad del coche.

Los usuarios de la clase Coche **no necesitan saber** cómo se implementa el método `acelerar()`. Solo necesitan saber que el método aumenta la velocidad del coche.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

## ¿A que Pilar se hace referencia?

Imagina una clase Coche que tiene un **atributo privado** `_velocidad` que almacena la velocidad actual del coche.

La clase también tiene un **método público** `acelerar()` que aumenta la velocidad del coche.

Los usuarios de la clase Coche **no necesitan saber** cómo se implementa el método `acelerar()`. Solo necesitan saber que el método aumenta la velocidad del coche.

**Polimorfismo**

**Encapsulamiento**

**Abstracción**

**Herencia**

## ¿A que Pilar se hace referencia?

Imagina una función imprimirVelocidad() que recibe un objeto de la **clase** Coche o de una **clase hija** de Coche como parámetro.

La función imprimirVelocidad() imprime la velocidad del coche.

La función imprimirVelocidad() puede usarse con cualquier objeto de la **clase** Coche o de una **clase hija** de Coche, sin necesidad de **modificar la función**.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

## ¿A que Pilar se hace referencia?

Imagina una función imprimirVelocidad() que recibe un objeto de la **clase** Coche o de una **clase hija** de Coche como parámetro.

La función imprimirVelocidad() imprime la velocidad del coche.

La función imprimirVelocidad() puede usarse con cualquier objeto de la **clase** Coche o de una **clase hija** de Coche, sin necesidad de **modificar la función**.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

## ¿A que Pilar se hace referencia?

Imagina una **clase** Camión que hereda de la **clase** Coche. La **clase** Camión tiene un **atributo privado** `_capacidadCarga` que almacena la capacidad de carga del camión.

La **clase** Camión también tiene un **método público** `frenar()` que reduce la velocidad del camión.

Este **método es una implementación específica del método** `frenar()` de la **clase** Coche, que se adapta a las características específicas del camión.

**Polimorfismo**

**Abstracción**

**Encapsulamiento**

**Herencia**

## ¿A que Pilar se hace referencia?

Imagina una **clase** Camión que hereda de la **clase** Coche. La **clase** Camión tiene un **atributo privado** `_capacidadCarga` que almacena la capacidad de carga del camión.

La **clase** Camión también tiene un **método público** `frenar()` que reduce la velocidad del camión.

Este **método es una implementación específica del método** `frenar()` de la **clase** Coche, que se adapta a las características específicas del camión.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

## ¿A que Pilar se hace referencia?

Imagina una clase Animal que tiene un **método abstracto** comer().

El **método** comer() no tiene una implementación en la **clase** Animal.

Las **clases hijas** de Animal, como Perro o Gato, deben proporcionar una implementación específica del **método** comer() que se adapte a las características específicas del animal.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

## ¿A que Pilar se hace referencia?

Imagina una clase Animal que tiene un **método abstracto** comer().

El **método** comer() no tiene una implementación en la **clase** Animal.

Las **clases hijas** de Animal, como Perro o Gato, deben proporcionar una implementación específica del **método** comer() que se adapte a las características específicas del animal.

Polimorfismo

Abstracción

Encapsulamiento

Herencia

# CREANDO NUESTRO PRIMER OBJETO



Programá  
tu futuro



Municipalidad de  
Tres de Febrero

# Creando la Clase y el Constructor

```
1 class Coche:  
2     pass
```

Con la palabra reservada **Class** , creamos la clase y le asignamos un nombre

Luego procedemos a crear su **Constructor**

```
class Coche:  
    def __init__(self, marca, modelo, color):  
        self.marca = marca  
        self.modelo = modelo  
        self.color = color
```

## ¿Que es un Constructor?

Es un método que se encarga de inicializar un objeto de la clase, es decir, de asignarle valores a sus **atributos**.

En este caso

- self es una referencia al objeto que se está creando.
- marca, modelo y color son los parámetros del método constructor.  
Estos parámetros se utilizan para inicializar los atributos del objeto.

Hagamos una prueba **Instanciemos** la clase **Coche** y creamos dos objetos

```
coche = Coche("Fiat", "Duna", "Rojo")
coche2 = Coche("Renault", 12, "Azul Metalizado")
```

# Leyendo Valores

Bien ahora que ya instanciamos la clase Coche, y creamos dos objetos, podemos acceder y ver sus valores

```
print(coche.marca)
print(coche.modelo)
print(coche.color)
print("\n")
print(coche2.marca)
print(coche2.modelo)
print(coche2.color)
```

Como se ve, pudimos crear 2 objetos diferentes a partir de la misma clase

S-2024/Clase 4/01  
Fiat  
Duna  
Rojo  
  
Renault  
12  
Azul Metalizado

# Agregando acciones

Venimos excelente, ahora agreguemos algunas acciones

```
class Coche:  
    def __init__(self, marca, modelo, color):  
        self.marca = marca  
        self.modelo = modelo  
        self.color = color  
        self.encendido = False  
  
    def encender(self):  
        self.encendido = True  
        print(f"El coche {self.marca} {self.modelo} está encendido.")  
  
    def apagar(self):  
        self.encendido = False  
        print(f"El coche {self.marca} {self.modelo} está apagado.")
```

Bien, como primera medida, agregaremos un nuevo atributo llamado encendido y creamos las acciones de encender y apagar

# ¿Que hacen estas acciones?

En este caso las acciones modificaran el atributo **encendido** cambiando su estado.

```
def encender(self):
    self.encendido = True
    print(f"El coche {self.marca} {self.modelo} está encendido.")
```

## Prueba

```
print(coche.encendido) #estado del coche antes de la accion
coche.encender() #se realia la accion
print(coche.encendido) #estado del coche luego de la accion
print("\n")
print(coche2.encendido) #estado del coche2
```

## Resultado

```
False
El coche Fiat Duna está encendido.
True

False
garcia-pc@garcon-MS-7052 ~ /Documentos /
```

# ¡MUCHAS GRACIAS!



Programá  
tu futuro



Municipalidad de  
Tres de Febrero