📝 Apuntes de Programación: Abstracción y Clases Abstractas en Python

La **Abstracción** es uno de los pilares de la Programación Orientada a Objetos (POO). Consiste en mostrar solo la información esencial y ocultar los detalles de implementación al usuario. En Python, esto se implementa principalmente a través de las **Clases Abstractas**.

1. Clases Abstractas: La Plantilla Obligatoria

Una clase abstracta es una clase que **no puede ser instanciada** (no puedes crear objetos directamente de ella), sino que sirve como plantilla (o contrato) para otras clases. Su objetivo es asegurar que las clases hijas implementen ciertos métodos.

A. Implementación con el Módulo `abc`

Python usa el módulo `abc` (Abstract Base Classes) para definir clases abstractas, ya que no las tiene por defecto.

```
from abc import abstractmethod, ABCMeta

#* Definición de la clase abstracta
class Abstracta(metaclass=ABCMeta):
    # ...
```

- `metaclass=ABCMeta` **:** Indica a Python que esta clase es una Clase Base Abstracta. Sin esto, la clase sería una clase normal.

B. El Método Abstracto (`@abstractmethod`)

Un **Método Abstracto** es un método declarado en la clase abstracta pero que **no tiene implementación** (solo la firma, usando `pass`).

- **Obligatoriedad:** Cualquier clase hija que herede de la clase abstracta está **obligada** a sobrescribir e implementar este método. Si no lo hace, Python lanza un error al intentar instanciar la clase hija.

```
@abstractmethod #* Tiene que llevar siempre este decorador
def metodoAbstracto(self):
    pass # No tiene lógica aquí
```

C. Métodos Concretos (Normales)

Una clase abstracta puede contener métodos con implementación, llamados **Métodos Concretos**. Estos son heredados y utilizados directamente por las clases hijas.

```
def metodoNormal(self):
    print("Hola mundo") # Método con implementación
```

2. Herencia e Implementación

Para utilizar la clase abstracta, una clase hija debe heredar de ella e implementar **todos** los métodos abstractos.

```
class hija(Abstracta): # Hereda de Abstracta
    def metodoAbstracto(self): # Obligada a implementar este método
        print("Adios")
```

A. Uso Correcto

La única forma de usar la funcionalidad es a través de una clase concreta (no abstracta) que ha implementado los métodos necesarios.

```
elemento = hija()
elemento.metodoNormal() # Usa el método heredado
elemento.metodoAbstracto() # Usa la implementación propia de la clase hija
```

3. Consejos para Mejorar la Programación con Abstracciones

La abstracción es una herramienta de diseño poderosa.

🎯 Principio 1: Definir Contratos (API)

- Utiliza clases abstractas para definir una **Interfaz** o **Contrato**. Por ejemplo, si estás diseñando un sistema de pagos, puedes crear una clase abstracta `MetodoDePago` con un método abstracto `procesar_pago()`.
- **Beneficio:** Garantizas que `Visa`, `PayPal` y `Bitcoin` (las clases hijas) tengan todas el método `procesar_pago()`, lo que permite que el resto del sistema interactúe con ellas de manera uniforme sin saber los detalles internos.

🎯 Principio 2: Usar Abstracción en Frameworks y Plugins

- Los *frameworks* de software a menudo utilizan clases abstractas para definir qué métodos debes implementar si quieres crear un *plugin* o módulo compatible.
- **Ejemplo:** En un juego, podrías tener una clase abstracta `Personaje` con métodos abstractos como `atacar()` y `moverse()`. Todas las criaturas del juego (dragones, héroes, zombies) deben implementar esos métodos, asegurando que todos puedan participar en la acción.

🎯 Principio 3: Pensar en la Arquitectura

- Antes de escribir el código de las clases concretas, define las responsabilidades clave en una clase abstracta. Esto obliga a mantener una estructura de código limpia y a pensar en el diseño desde arriba hacia abajo. La abstracción es la base de un buen diseño de software, facilitando el mantenimiento y la escalabilidad.