7. Clases

💡 Apuntes Detallados: Clases y Expresiones Regulares

---

1. 🏛️ Clases y Programación Orientada a Objetos (OOP)

La OOP es una filosofía de programación que organiza tu código en **objetos**, que son entidades que agrupan datos (atributos) y comportamientos (métodos).

A. Anatomía de una Clase

Una **Clase** es el **molde** (el plano de una casa). Un **Objeto** es la **instancia** (la casa construida).

Elemento	Descripción	Clave
`__init__`	El **Constructor**. Define los valores iniciales de los atributos del objeto al crearse.	Se usa `def __init__(self, ...):`.
`self`	Representa al objeto **actual** que se está manejando. Permite acceder a los atributos del objeto (ej. `self.edad`).	Siempre es el primer argumento de los métodos.
Método	Son las **funciones** internas de la clase (comportamientos).	Se usa `def nombre_metodo(self, ...):`.

Ejemplo Práctico: Inventario

Python

```
class Producto:
    def __init__(self, nombre, precio, stock):
        self.nombre = nombre
        self.precio = precio
        self.stock = stock

    def vender(self, cantidad):
        # El método modifica un atributo del objeto
        if self.stock >= cantidad:
            self.stock -= cantidad
```

```
            return f"Vendidos {cantidad} de {self.nombre}. Stock restante:
{self.stock}"
        return "Stock insuficiente."

# 📦 Creación de un Objeto (Instancia)
manzana = Producto("Manzana", 0.5, 100)

# Acceder y usar métodos:
print(manzana.vender(10))  # Output: Vendidos 10 de Manzana. Stock restante:
90
print(f"Precio: {manzana.precio}") # Output: Precio: 0.5
```

B. Herencia (Compartir Código)

La **Herencia** te permite crear una **Clase Hija** que toma todas las propiedades (atributos) y métodos de una **Clase Madre**.

- **Clave**: En el constructor de la hija, usas `super().__init__()` para pasar los atributos a la madre y así evitar reescribir código.

Python

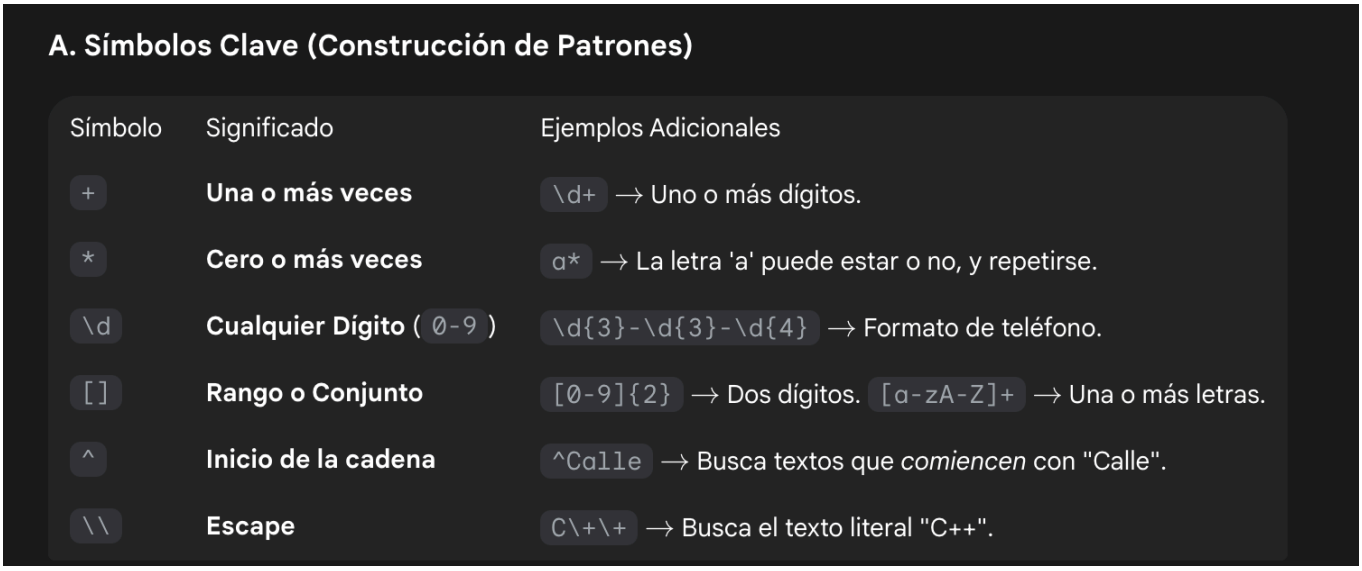
```
class Electronica(Producto): # Clase Hija hereda de Producto
    def __init__(self, nombre, precio, stock, garantia_meses):
        # Llama al constructor de la clase Producto
        super().__init__(nombre, precio, stock)
        self.garantia = garantia_meses # Atributo propio

    def chequear_garantia(self):
        return f"El producto {self.nombre} tiene {self.garantia} meses de
garantía."

# 🖥️ Creación de Objeto Heredado
tv = Electronica("SmartTV", 500, 5, 24)

print(tv.vender(1))          # Método heredado de Producto
print(tv.chequear_garantia()) # Método propio
```

---

2. 🔎 Expresiones Regulares (Regex)

Regex es un **lenguaje de patrones** para **buscar y manipular texto** de forma avanzada, ideal para la limpieza y extracción de datos. Se usa con el módulo `re` (`import re`).

A. Símbolos Clave (Construcción de Patrones)



B. Métodos Clave de la Librería `re`

Método	Función	Usado para...
`re.findall()`	Devuelve una **lista** con **TODAS** las coincidencias en el texto.	Extracción de todos los correos o fechas de un documento.
`re.search()`	Busca la **primera** coincidencia en **TODO** el texto.	Saber si un patrón existe, sin importar su ubicación.
`re.sub()`	Reemplaza todas las coincidencias del patrón con otro texto y devuelve el nuevo *string*.	Limpieza de datos (ej. quitar símbolos o espacios extra).

Ejemplo Práctico: Extracción y Limpieza

Python

```
import re
texto = "Tienes 23 items y el precio es de 50€, tienes que pagar 500€ en
total."

# 1. Extraer todos los números (incluyendo el €)
numeros_raw = re.findall(r"\d+", texto)
print(f"Números encontrados (raw): {numeros_raw}") # Output: ['23', '50',
'500']

# 2. Limpieza: Eliminar el símbolo de euro (€) usando re.sub()
texto_sin_euro = re.sub(r"€", "", texto)
```

```
print(f"Texto sin €: {texto_sin_euro}")
# Output: Texto sin €: Tienes 23 items y el precio es de 50, tienes que
pagar 500 en total.

# 3. re.match() (Ejemplo de Fallo)
# Intenta buscar 'tienes' al principio del texto
match_inicio = re.match(r"tienes", texto)
print(f"Resultado match: {match_inicio}") # Output: None
# Falla porque 'tienes' no está en la primera posición
```