# Preguntas

## ¿Para qué usamos Clases en Python?

Las clases sirven para empaquetar funcionalidades y crear objetos sin tener que repetir código. Si tuviéramos una receta de distintos pasteles, cada uno tendría sus propias características, como por ejemplo sabor, pero tendría elementos comunes. Así, a partir de una misma receta podríamos crear distintos pasteles, aunque cambiando ingredientes. Pasando a código esto sería algo así:

```
class Pastel:
  def __init__(self, ingrediente):
    self.ingrediente = ingrediente
```

Con esto tenemos la receta, vamos a crear los objetos/ingredientes:

```
pastel_chocolate = Pastel("chocolate")
pastel_fresa = Pastel("fresa")
```

Solo con esto me ahorro repetir código, y además, si quiero cambiar algo, solo tengo que cambiarlo en la propia clase, sin necesidad de tocar el resto del código. Así, puedo crear cuantos objetos quiera a partir de unas pocas líneas de código, sin tener que escribir manualmente cada uno.

## ¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?

El método que se ejecuta al crear una clase es el método __init__(), también llamado constructor. Este método se ejecuta automáticamente cada vez que se crea una instancia de la clase. Su sintaxis es como sigue:

```
class NombreClase:
  def __init__(self, parámetros):
    self.parámetros = parámetros
```

Y tras esto procederíamos a crear cuantos objetos sean necesarios mediante las instancias de clase.

## ¿Cuáles son los tres verbos de API?

Los verbos API, también llamados verbos HTTP (aunque no siempre son verbos), sirven para indicar a la API qué datos queremos obtener, y cómo queremos obtenerlos.

Los más importantes son GET, POST y PUT. El método GET solicita una representación de un recurso específico (“pide” datos al servidor), el método POST crea un nuevo recurso y el método PUT actualiza dicho recurso.

## ¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es una base NoSQL (Not Only SQL) que almacena datos fuera de las estructuras relacionales tradicionales, y almacena estructuras de datos usando pares key-value y colecciones. En lugar de la típica estructura tabular de una base de datos relacional, las bases de datos NoSQL alojan datos dentro de una estructura de datos, como un documento JSON. Dado que este diseño de base de datos no relacional no requiere un esquema, ofrece una rápida escalabilidad para gestionar grandes conjuntos de datos normalmente no estructurados. En cambio, una base SQL trabaja con datos estructurados.

## ¿Qué es una API?

Una API (Application Programming Interface) es una interfaz de programación que define la manera en la que un conjunto de aplicaciones y servicios intercambian información. Una API es una interfaz que permite a los desarrolladores de software comunicarse entre sí y con los sistemas externos. Es una especie de "puente" entre el código del desarrollador y el código de los sistemas externos. Por ejemplo, un programa podría comunicarse con una API de un instituto meteorológico para obtener información sobre el clima actual.

## ¿Qué es Postman?

Postman es una interfaz gráfica para crear y probar APIs, así como para enviar peticiones HTTP a servidores web. Estas peticiones descargan datos que luego se pueden tratar o filtrar con líneas de código.

## ¿Qué es el polimorfismo?

El polimorfismo es, en programación orientada a objetos, la capacidad de un objeto de tener diferentes formas o representaciones dependiendo del contexto. En código podríamos tener lo siguiente:

```
class Gato():
  def sonido(self):
    return "miau"


class Perro():
  def sonido(self):
    return "guau"


gato = Gato()
perro = Perro()


print(perro.sonido()) # imprime guau porque hace referencia al objeto perro
print(gato.sonido()) # imprime miau porque hace referencia al objeto gato
```

**¿Qué es un método dunder?**

# Un método dunder es un método especial de Python que se utiliza para implementar operaciones en una clase y se llaman automáticamente dentro de esta. Se llaman así porque tienen dos guiones bajos al inicio y al final de su nombre. __init__ es un ejemplo.

## ¿Qué es un decorador de Python?

Un decorador es una función que toma una función como argumento y devuelve otra función con más funcionalidades sin modificar la función previa.

```
def decorador(funcion):
    def wrapper():
        print("Antes de ejeutar.")
        funcion()
        print("Tras ejecutar")
    return wrapper
```

Para utilizar un decorador solo hay que poner una @ delante:

```
@decorador
def saludar():
    print("Hola")
```

Y si imprimiéramos esto obtendríamos:

```
Antes de ejecutar
Hola
Tras ejecutar
```