¿Para qué usamos Classes en Python?

Las clases son bloques de código que contienen datos y funciones que describen objetos, son un aspecto fundamental en Python, ya que este es un lenguaje de programación orientado a objetos.

Las clases, una vez definidas, pueden utilizar o llamar tantas veces como sea necesario para crear distintos objetos dependiendo de las variables o atributos de estos.

Se podría decir que las clases son como el plano, por ejemplo, de una casa; en estos planos se definen unas propiedades que son iguales para todas las casas, que tengan cuatro paredes y un tejado, pero, además, se definen también unos atributos que cambian según la casa, el color de las paredes, el número de ventanas que tiene, etc. Cada vez que se utilice la clase se creará una casa (objeto) que tendrá la misma estructura, pero el diseño será diferente según sus atributos.

```
class Casa:
     tejado = 1
     paredes = 4
```

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

Al crear una instancia de clase automáticamente se ejecuta la función __init__(); esta función asigna valores a los atributos del objeto que se va a crear. Estos atributos representan características o propiedades del objeto, pueden ser diferentes tipos de datos: enteros, cadenas, listas, etc.

```
class Casa:
     tejado = 1
     paredes = 4
     def __init__(self, color, ventanas):
          self.color = color
          self.ventanas = ventanas
```

¿Qué es un método dunder?

Estos métodos reciben ese nombre porque comienzan y terminan con doble guion bajo (double underscore - dunder), se utilizan para definir un comportamiento especial de la clase, los más comunes son estos:

- __init__(): como ya se ha explicado antes, se utiliza para asignar loa valores iniciales a los atributos de la clase.
- __str__(): convierte el objeto a una cadena de caracteres legible para el usuario.
- __repr__(): convierte el objeto a una cadena de caracteres que recrea al objeto.
- Operadores matemáticos: __add__ - suma, __sub__ - resta, __mul__ - multiplicación, __truediv__ - división.

```
class Casa:
      tejado = 1
      paredes = 4
      def __init__(self, color, ventanas):
            self.color = color
            self.ventanas = ventanas
      def __str__(self):
            return f'La casa es de color {self.color} y tiene {self.ventanas}
            ventanas'

casa_azul = Casa("azul", 2)
print(str(casa_azul)) #'La casa es de color azul y tiene 2 ventanas'
```

¿Qué es un decorador de Python?

Los decoradores son herramientas que permiten modificar el comportamiento de una función o clase sin cambiar su estructura.

```
class Casa:
      tejado = 1
      paredes = 4
      def __init__(self, color, ventanas):
            self._color = color
            self.ventanas = ventanas
      def __str__(self):
            return f'La casa es de color {self._color} y tiene
            {self.ventanas} ventanas'
      @property
            def color(self):
                return self._color
      @color.setter
            def color(self, color):
                 self._color = color

casa_azul = Casa("azul", 2)
print(str(casa_azul)) #'La casa es de color azul y tiene 2 ventanas'

casa_azul.color = 'azul oscuro'
print(casa_azul.color) #'azul oscuro'
print(str(casa_azul)) #'La casa es de color azul oscuro y tiene 2 ventanas'
```

¿Qué es el polimorfismo?

El polimorfismo permite que una misma función se ejecute en diferentes clases con distinto resultado en cada una de estas, ya sean independientes o dependientes la una de la otra (padre e hijo).

```
class Casa:
     tejado = 1
     paredes = 4
     def __init__(self, color, ventanas):
          self.color = color
          self.ventanas = ventanas
     def construir(self):
          return f'{Casa.paredes} paredes, {Casa.tejado} tejado, pintura
          {self.color} y {self.ventanas} ventanas'
casa_azul = Casa("azul", 2)

class Duplex:
     tejado = 1
     paredes = 8
     escalera = 1
     def __init__(self, color, ventanas):
          self.color = color
          self.ventanas = ventanas
     def construir(self):
          return f'{Duplex.paredes} paredes, {Duplex.escalera} escalera,
          {Duplex.tejado} tejado, pintura {self.color} y {self.ventanas}
          ventanas'

casa_azul = Casa("azul", 2)
duplex_verde = Duplex("verde", 6)

print(casa_azul.construir()) #4 paredes, 1 tejado, pintura azul y 2 ventanas
print(duplex_verde.construir()) #8 paredes, 1 escalera, 1 tejado, pintura
verde y 6 ventanas
```

¿Qué es una API?

API, en español, significa interfaz de programación de aplicaciones (Application Programming Interfaces, en inglés). Una API es un conjunto de reglas definidas que permite que dos aplicaciones se comuniquen entre sí y compartan información y funcionalidades.

A pesar de que las APIs no son visibles en webs y aplicaciones, están presentes en la gran mayoría de estas, algunos de los ejemplos más comunes que utilizamos casi a diario son los siguientes:

- Autentificación: algunas aplicaciones y páginas web permiten utilizar las cuentas de Facebook, Google, GitHub, etc. para iniciar sesión sin necesidad de registrarse gracias a las APIs.

- Comercio electrónico: al realizar compras on-line en algunas páginas la pasarela de pago de Paypal, por ejemplo, está integrada e iniciando sesión en la cuenta de Paypal se realiza el pago directamente.
- Comparadores de precios: hay páginas que son comparadores de vuelos, hoteles, seguros, etc. y toda esta información la obtienen y actualizan utilizando las APIs de los proveedores de estos servicios.

¿Qué es Postman?

Postman es una aplicación que permite construir, probar y utilizar APIs.

Entre sus funcionalidades se encuentran las solicitudes HTTP, guardar entornos para su posterior uso y la conversión de APIs a otros lenguajes de programación.

¿Cuáles son los tres verbos de API?

Los verbos de API o métodos HTTP se utilizan para indicar la acción que un cliente API quiere realizar en el recurso indicado. Los más comunes son los siguientes:

- GET: devuelve uno o todos los recursos.
- POST: crea nuevos recursos.
- PUT: actualiza o reemplaza uno o varios recursos.
- DELETE: elimina uno o vario recursos.

¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es un sistema de base de datos NoSQL o no relacional de tipo documental.

Al ser una base de datos de tipo documental, MongoDB almacena los datos en documentos similares a objetos JSON, cada uno de estos documentos contiene pares de campos y valores. Los valores que se introducen pueden ser de distintos tipos: cadenas, números, valores booleanos, matrices u objetos, entre otros.

A diferencia de las bases de datos relacionales o SQL, MongoDB permite almacenar y gestionar grandes cantidades de datos estructurados, semi-estructurados y no estructurados, dando así una gran flexibilidad a la hora de trabajar con los datos, ya que cada uno de los documentos que se introducen en la base de datos puede contener distintos campos y, dentro de estos, diferentes tipos de valores.

Más información sobre BBDD NoSQL: https://rb.gy/twblg3

Create a Python class called User that uses the init method and creates a username and a password. Create an object using the class.

Solución: https://replit.com/@alazneportal/Chechpoint-6#main.py