CHECKPOINT6

1.¿Para qué usamos Clases en Python?

- Una clase en Python sirve para definir un conjunto de métodos y atributos que describen un objeto o entidad.

  Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase, se crea un nuevo *tipo* de objeto, permitiendo crear nuevas *instancias* de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

  Para definirlos se utiliza la palabra class delante del nombre que le ponemos a la clase y posterior al nombre ponemos el símbolo : y luego el contenido que se definde dentro de la clase.

  En estos enlaces podemos apreciar más información al respecto.
  https://docs.python.org/es/3/tutorial/classes.html

  https://blog.hubspot.es/website/clases-python#:~:text=Una%20clase%20en%20Python%20es,en%20un%20programa%20de%20computadora.

  Por ejemplo:

  ```
  class Person:

  def __init__(self,name,age):
          self.name=name
          self.age=age


  class car:

  def __init__(self,name,color,motor):
          self.name=name
          self.color=color
          self.motor=motor
  ```

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

- Se inicia el método __init__ para iniciar una instancia. Este método es un método especial que se llama cada vez que se instancia una clase y sirve para inicializar el objeto que se crea. Este método crea los atributos que deben tener todos los objetos de la clase y por tanto contiene los parámetros necesarios para su creación, pero no devuelve nada.

  Los atributos que se crean dentro del método __init__ se conocen como atributos del objeto, mientras que los que se crean fuera de él se conocen como atributos de la clase.

  Aquí podemos ampliar en el link la información y también nos propone dos ejercicios con soluciones para revisarlos.

  https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=44&codigo=44&inicio=30

- Un ejemplo de __init__:

```
class Person:

def __init__(self,name,age):
        self.name=name
        self.age=age
```

3.¿Cuáles son los tres verbos de API?

- Application Programming Interfaces es el significado de las 3 siglas de API. Cuyo significado ampliado lo vemos reflejado en la pregunta 5.

4.¿Es MongoDB una base de datos SQL o NoSQL?

- MongoDB es una base de datos NoSQL de código abierto, almacena objetos de datos en colecciones y documentos. NoSQL es un sistema de base de datos más nuevo que no utiliza un lenguaje de consulta estándar pero emplea documentos JSON para el almacenamiento de datos.

Por ejemplo, en este link se puede apreciar los diferentes métodos para insertar documentos en mongo :

https://www.adrformacion.com/knowledge/programacion/como_insertar_documentos_en_mongodb.html

5.¿Qué es una API?

- Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos.
- Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

  Por ejemplo, con la API Web de Twitter se puede escribir un programa en un lenguaje como Python o Javascript que recoja metadatos sobre twits.

  Aquí podemos ampliar más información al respecto:

  https://www.xataka.com/basics/api-que-sirve

6.¿Qué es Postman?

- Postman es una plataforma de API para crear y utilizar API. Simplifica cada paso del ciclo de vida de la API y agiliza la colaboración para poder crear mejores API, más rápido.

  Postman surgió originariamente como una extensión para el navegador Google Chrome. A día de hoy dispone de aplicaciones nativas para MAC y Windows y están trabajando en una aplicación nativa para Linux.

  Está compuesto por diferentes herramientas y utilidades gratuitas:

  -Creación de peticiones a APIs internas o de terceros.

  -Elaboración de tests para validar el comportamiento de APIs.

  -Posibilidad de crear entornos de trabajo diferentes (con variables globales y locales).

  Todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).

Aquí en este enlace podemos encontrar más información:

https://openwebinars.net/blog/que-es-postman/

7.¿Qué es el polimorfismo?

- El polimorfismo es la capacidad que tienen ciertos lenguajes para hacer que, al enviar el mismo mensaje desde objetos de diferentes clases, cada uno de esos objetos pueda responder a ese mensaje diferente forma.

  Es decir, dos objetos de diferentes clases pueden tener métodos con el mismo nombre, y ambos métodos pueden ser llamados con el mismo código, dando respuestas diferentes.

  Aquí podemos ampliar la información y apreciarlo mejor mediante los ejemplos.

  https://ellibrodepython.com/polimorfismo-en-programacion

8.¿Qué es un método dunder?

- Métodos Dunder o los denominados también métodos Mágicos o especiales. Se utilizan para emular el comportamiento de las funciones integradas.

Sus nombres empiezan y terminan en __ doble barra baja. Por ejemplo init. Se emplean para definir el comportamiento especial de las clases y sus instancias, y son llamados automáticamente por el intérprete de Python en respuesta a ciertas operaciones. Por ejemplo, el método __init__ es un método dunder que se usa para inicializar un objeto cuando se crea una instancia de una clase.

Los métodos dunder pueden utilizarse para implementar características como la inicialización de objetos, la representación en forma de cadena, la sobrecarga de operadores, la comparación, entre otras.

```
class user:
 def __init__(self,age):
        self.age=age
```

Aquí podemos encontrar más información:

https://barcelonageeks.com/dunder-o-metodos-magicos-en-python/#google_vignette

En este otro enlace también podemos ampliar información.

https://blog.hubspot.es/website/clases-python

9.¿Qué es un decorador de python?

- Los decoradores son una caracteristica de Python que permite modificar, reducir las líneas de código duplicadas, hacer nuestro código más legible y fácil de testear. Un decorador en Python es una función que recibe otra función como parámetro, le añade cosas y retorna una función diferente

Lo siguiente es un decorador.

```
def make_upper(func):
    def wrapper():
        return func().upper()

    return wrapper

def python_greeting():
    return 'hi, i\'m a Python developer!'

python_greeting = make_upper(python_greeting)  #This is what decorates the function

print(python_greeting())
```

La función que recibe otra función es `make_upper` que recibe el argumento `func` que es una función. Dentro, se modifica a `func` usando `wrapper`. Y finalmente devuelve `wrapper` que es `func` pero diferente.

La función que se decora es `python_greeting` y la línea de código que lo hace es:

```
python_greeting = make_upper(python_greeting)
```

Aquí podemos ver más ejemplos:
https://ellibrodepython.com/decoradores-python

EJERCICIO

-Cree una clase de Python llamada Usuario que use el método init y cree un nombre de usuario y una contraseña. Crea un objeto usando la clase. (Esta el archivo Python a parte también guardado)

```
class Usuario:

  def __init__(self, username, password):

      self.username = username

      self.password = password

user = Usuario("ainara", "asdfg")

print (user.username, user.password)
```