TECNOLOGÍAS DE INFORMACIÓN Y CIBERSEGURIDAD

Inacap

PROGRAMACIÓN ORIENTADA A OBJETO (POO)

Integración con Servicios Externos

JSON

Un archivo JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos que es fácilmente legible tanto para humanos como para máquinas. Se utiliza ampliamente para representar estructuras de datos simples y complejas.


```json
{
  "empleados": [
    {
      "id": 1,
      "nombre": "Juan",
      "puesto": "Desarrollador",
      "edad": 30,
      "salario": 45000.00
    },
    {
      "id": 2,
      "nombre": "María",
      "puesto": "Diseñador",
      "edad": 28,
      "salario": 38000.00
    },
    {
      "id": 3,
      "nombre": "Pedro",
      "puesto": "Gerente",
      "edad": 35,
      "salario": 60000.00
    }
  ]
}
```

JSON

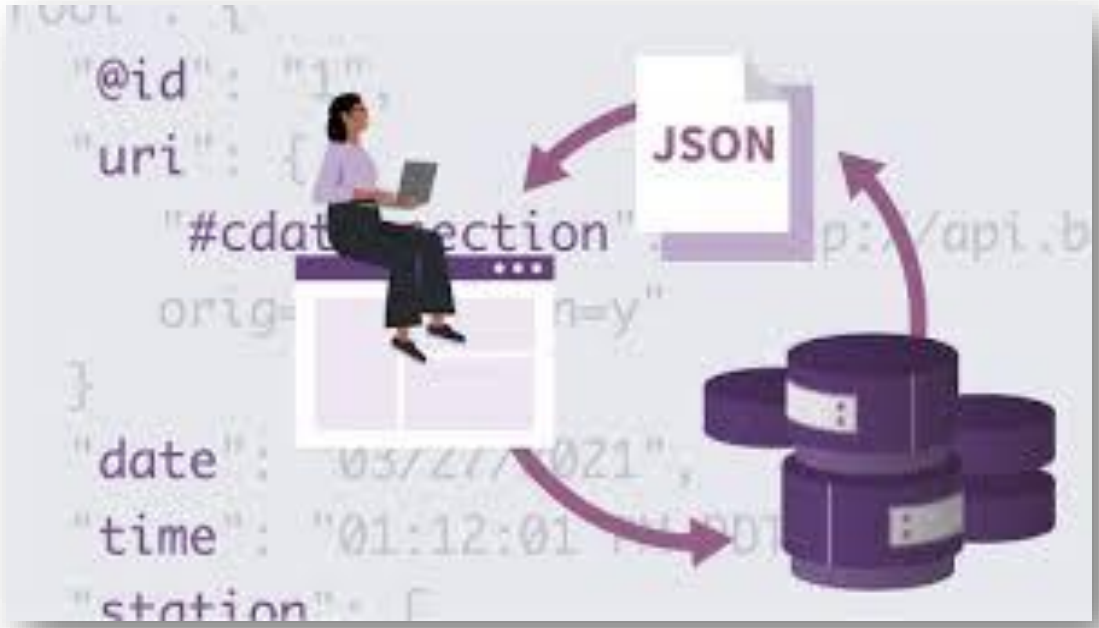
En este ejemplo, el archivo JSON contiene un objeto raíz que tiene una clave "empleados" cuyo valor es una lista de objetos. Cada objeto representa un empleado y tiene las siguientes propiedades: "id", "nombre", "puesto", "edad" y "salario".

```json
{
  "empleados": [
    {
      "id": 1,
      "nombre": "Juan",
      "puesto": "Desarrollador",
      "edad": 30,
      "salario": 45000.00
    },
    {
      "id": 2,
      "nombre": "María",
      "puesto": "Diseñador",
      "edad": 28,
      "salario": 38000.00
    },
    {
      "id": 3,
      "nombre": "Pedro",
      "puesto": "Gerente",
      "edad": 35,
      "salario": 60000.00
    }
  ]
}
```

JSON

- Es importante destacar que en JSON, los nombres de las claves (propiedades) y los valores pueden estar en mayúsculas o minúsculas, pero es una práctica común utilizar minúsculas para las claves y mantener los valores según corresponda.

- Este archivo JSON puede ser utilizado para almacenar y transportar información de empleados en una variedad de aplicaciones, y es ampliamente utilizado en el intercambio de datos entre sistemas.



MÉTODOS CON JSON: DUMPS Y LOADS

En Python, existen dos métodos principales para trabajar con JSON: **dumps()** y **loads()**. Estos métodos se encuentran en el módulo **json** de la biblioteca estándar de Python y permiten convertir datos entre formato JSON y objetos Python (diccionarios, listas, etc.).

dumps(): Este método se utiliza para convertir un objeto Python en una cadena JSON **(serialización).**

```
#Importamos el módulo json
import json

data = {
    "nombre": "Juan",
    "edad": 30,
    "puesto": "Desarrollador"
}

json_string = json.dumps(data)
print(json_string)  # Salida: '{"nombre": "Juan", "edad": 30, "puesto":
"Desarrollador"}'
print(type(json_string)) #Podemos verificar el tipo de la variable
```

MÉTODOS CON JSON: DUMPS Y LOADS

- **loads()**: Este método se utiliza para convertir una cadena JSON en un objeto Python **(deserialización)**.

```
#Importamos el módulo json
import json

json_string = '{"nombre": "Juan", "edad": 30, "puesto":
"Desarrollador"}'

data = json.loads(json_string)
print(data)  # Salida: {'nombre': 'Juan', 'edad': 30, 'puesto':
'Desarrollador'}
print(type(data)) #Podemos verificar el tipo de la variable
```

MÉTODOS CON JSON: DUMP Y LOAD

- Además de estos métodos, el módulo **json** también proporciona otros métodos útiles, como **dump()** y **load()**, que se utilizan para trabajar con archivos JSON.

- **dump()**: Este método se utiliza para escribir un objeto Python en un archivo JSON.

```
#Importamos el módulo json
import json

data = {
    "nombre": "Juan",
    "edad": 30,
    "puesto": "Desarrollador"
}

#Se crea un archivo datos.json y los datos de la variable data
with open("datos.json", "w") as archivo_json:
    json.dump(data, archivo_json)
```

MÉTODOS CON JSON: DUMP Y LOAD

- **Load()**: Este método se utiliza para leer datos desde un archivo JSON y convertirlos en un objeto Python.

```
#Importamos el módulo json
import json

with open("datos.json", "r") as archivo_json:
    data = json.load(archivo_json)

print(data)  # Salida: {'nombre': 'Juan', 'edad': 30, 'puesto':
'Desarrollador'}
```

COMPARATIVA DUMP Y DUMPS

Función	Propósito	Argumento	Valor Retornado
dump	Escribe un objeto Python en un archivo en formato JSON	Objeto Python y Archivo (objeto de tipo archivo abierto)	No hay valor retornado. Los datos se escriben en el archivo en formato JSON.
dumps	Convierte un objeto Python en una cadena de texto en formato JSON	Objeto Python	Cadena de texto (str) que contiene el objeto Python convertido en formato JSON.

COMPARATIVA LOAD Y LOADS

Función	Propósito	Argumento	Valor Retornado
load	Carga un JSON desde un archivo	Archivo (objeto de tipo archivo abierto)	Objeto Python con los datos cargados desde el archivo.
loads	Carga un JSON desde una cadena de texto	Cadena de texto (str)	Objeto Python con los datos cargados desde la cadena de texto.

CONVERSIÓN DE TIPOS DE DATOS JSON Y PYTHON

Tipo JSON (Formato JSON)	Tipo Python (Objeto Python)
object	dict (diccionario)
array	list (lista)
string	str (cadena de caracteres)
Number (int o float)	int o float
true	True
false	False
null	None

JSON: USANDO OBJETO REQUEST

Para trabajar con JSON y hacer **solicitudes HTTP** utilizando el módulo **requests** en Python, necesitaremos importar ambas bibliotecas. El módulo **requests** se utiliza para realizar **solicitudes HTTP**, mientras que el módulo **json** se utiliza para trabajar con el formato JSON. Asegúrate de tener instalado **requests** utilizando **pip install requests**.

```
#Importamos los módulos necesarios
import requests
import json
def obtener_datos_desde_api():
    url = "https://raw.githubusercontent.com/jclavijod/inacap/main/empleados.json"  # Reemplaza esto con la URL de
la API que deseas consultar
    try:
        response = requests.get(url)
        response.raise_for_status()  # Comprobar si la solicitud fue exitosa (código de estado 200)
        data = response.json()  # Convertir la respuesta JSON en un objeto Python (diccionario o lista)
        return data
    except requests.exceptions.RequestException as e:
        print("Error al hacer la solicitud:", e)
        return None
def main():
    data = obtener_datos_desde_api()

    if data is not None:
        # Trabajar con los datos en formato JSON
        for item in data["empleados"]:#Indicamos el nombre de la colección
            print("ID:", item["id"]) #Indicamos la key del valor a mostrar
            print("Nombre:", item["nombre"])
            print("Edad:", item["edad"])
            print("Puesto:", item["puesto"])
            print("---")
if __name__ == "__main__":
    main()
```

CREACIÓN DE DATOS EN LA BD A PARTIR DE UN JSON

Necesitamos tener una estructura adecuada para representar los datos en el JSON, utilizar el módulo **json** para leer el JSON y el módulo de **conexión** a la base de datos correspondiente para insertar los datos en la base de datos.

```python
import json
import mysql.connector
def conectar():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="tu_usuario",
            password="tu_contraseña",
            database="tu_base_de_datos"
        )
        return conn
    except mysql.connector.Error as e:
        print("Error al conectar a la base de datos:", e)
        return None
def insertar_datos(conn, empleado):
    cursor = conn.cursor()
    consulta = "INSERT INTO empleados (nombre, puesto, edad, salario) VALUES (%s, %s, %s, %s)"
    datos = (empleado["nombre"], empleado["puesto"], empleado["edad"], empleado["salario"])
    cursor.execute(consulta, datos)
    conn.commit()
    cursor.close()
def main():
    with open("empleados.json", "r") as archivo_json:
        data = json.load(archivo_json)
    conn = conectar()
    if conn is None:
        return
    for empleado in data["empleados"]:
        insertar_datos(conn, empleado)
    conn.close()
    print("Datos creados exitosamente en la base de datos.")
if __name__ == "__main__":
    main()
```