

# Trivia de Números - Proyecto Python

## [Main.py](#)

El archivo main.py es el programa principal que permite al usuario consultar curiosidades sobre un número.

Pregunta al usuario por un número.

Envía ese número a una API local usando la función trivia\_fetch.

Recibe la respuesta de la API y muestra la curiosidad y la categoría si existen.

Si no hay curiosidad para ese número, muestra un mensaje de error.

En resumen: el usuario ingresa un número y el programa le muestra una curiosidad sobre ese número usando una API local.

```
main.py x
main.py > trivia_fetch
1 import requests
2
3 # Función que consulta la API Local y devuelve la curiosidad sobre el número
4 def trivia_fetch(number):
5     url = f"http://127.0.0.1:5000/dato/{number}" # URL de la API Local
6     try:
7         response = requests.get(url) # Realiza la petición GET
8         if response.status_code == 200:
9             return response.json() # Devuelve la respuesta en formato JSON
10        else:
11            # Si la API no responde correctamente, devuelve un mensaje de error
12            return {"number": number, "mensaje": "Error al conectar con la API."}
13        except Exception as e:
14            # Si ocurre una excepción (por ejemplo, la API no está disponible), devuelve el error
15            return {"number": number, "mensaje": f"Excepción: {str(e)}"}
16
17 # Función principal que interactúa con el usuario
18 def main():
19     numero = int(input("Ingresa un número para obtener su curiosidad local: ")) # Solicita un número al usuario
20     resultado = trivia_fetch(numero) # Obtiene la curiosidad usando la función trivia_fetch
21     if "categoria" in resultado:
22         # Si existe la categoría, muestra la curiosidad y la categoría
23         print(f"Número: {resultado['number']}")
24         print(f"Categoría: {resultado['categoria']}")
25         print(f"Curiosidad: {resultado['curiosidad']}")
26     else:
27         # Si no existe la categoría, muestra el mensaje de error
28         print(resultado["mensaje"])
29
30 # Ejecuta la función principal solo si el archivo se ejecuta directamente
31 if __name__ == "__main__":
32     main()
33
```

test\_.py

El archivo test\_.py contiene pruebas automáticas para tu proyecto. Verifica que la función trivia\_fetch devuelve correctamente el número consultado en su respuesta. Si el resultado es correcto para los números 42 y 1000, los tests pasan y tu función funciona bien.

```
test_.py x
test_.py > ...
1  from main import requests, trivia_fetch
2
3
4  # Test 1
5  def test_trivia_42():
6      assert trivia_fetch(42)["number"] == 42
7
8  # Test 2
9  def test_trivia_1000():
10     assert trivia_fetch(1000)["number"] == 1000
```

## [app.py](#)

El archivo app.py crea una API usando Flask.  
Lee curiosidades de números desde el archivo datos.json.  
Cuando alguien visita la ruta /dato/<número>, la API responde con la curiosidad y la categoría de ese número si existe; si no, muestra un mensaje diciendo que no hay datos para ese número.

```
app.py x
app.py > ...
1 from flask import Flask, jsonify
2 import json
3
4 # Crear la aplicación Flask
5 app = Flask(__name__)
6
7 # Cargar los datos curiosos desde el archivo JSON
8 with open('datos.json', 'r', encoding='utf-8') as f:
9     datos = json.load(f)
10
11 # Definir la ruta para obtener la curiosidad de un número
12 @app.route('/dato/<int:numero>', methods=['GET'])
13 def obtener_dato(numero):
14     numero_str = str(numero) # Convertir el número a string para buscar en el diccionario
15     if numero_str in datos:
16         # Si el número existe en los datos, devolver la curiosidad y la categoría
17         return jsonify({
18             "number": numero,
19             "categoria": datos[numero_str]["categoria"],
20             "curiosidad": datos[numero_str]["curiosidad"]
21         })
22     else:
23         # Si el número no existe, devolver un mensaje de error
24         return jsonify({
25             "number": numero,
26             "mensaje": "No hay datos curiosos para este número."
27         })
28
29 # Ejecutar la aplicación si el archivo se ejecuta directamente
30 if __name__ == '__main__':
31     app.run(debug=True)
32
```

datos.json

El archivo datos.json contiene curiosidades sobre varios números. Cada número es una clave y tiene dos datos asociados:

"categoria": el tipo de curiosidad (por ejemplo, "Cultura pop", "Matemáticas").

"curiosidad": una frase con información interesante sobre ese número.

Por ejemplo, para el número 42, se guarda que es famoso en la cultura pop por ser "la respuesta a la vida, el universo y todo".

```
() datos.json x
() datos.json > ...
1 {
2   "7": {
3     "categoria": "cultura general",
4     "curiosidad": "Es considerado un número de la suerte en muchas culturas."
5   },
6   "13": {
7     "categoria": "Superstición",
8     "curiosidad": "Se evita en hoteles y edificios por supersticiones."
9   },
10  "42": {
11    "categoria": "Cultura pop",
12    "curiosidad": "Es la respuesta a la vida, el universo y todo, según Douglas Adams."
13  },
14  "1729": {
15    "categoria": "Matemáticas",
16    "curiosidad": "Es el número de Hardy-Ramanujan, el menor expresable como suma de dos cubos de dos formas distintas."
17  },
18  "1000": {
19    "categoria": "Matemáticas",
20    "curiosidad": "Es un número redondo que representa el milenio, y en notación romana se escribe como 'M'."
21  }
22 }
23
24
```

### Pasos realizados para el proyecto

#### 1. Creación del archivo de datos (datos.json):

Se hizo un archivo con curiosidades y categorías para varios números.

#### 2. Desarrollo de la API local (app.py):

Se creó una API con Flask que lee el archivo datos.json y responde con la curiosidad y categoría de un número cuando se consulta la ruta /dato/<número>.

#### 3. Desarrollo del cliente (main.py):

Se hizo un programa que pide al usuario un número, consulta la API local y muestra la curiosidad y categoría si existen.

#### 4. Implementación de la función trivia\_fetch:

Esta función se encarga de hacer la petición a la API y devolver la información recibida.

#### 5. Creación de pruebas automáticas (test\_.py):

Se escribieron tests para verificar que la función trivia\_fetch devuelve correctamente el número consultado.

#### 6. Instalación de dependencias:

Se instalaron las librerías necesarias (flask y requests) usando pip.

7. Ejecución y prueba del sistema:

Se inició la API local, se ejecutó el cliente y se corrieron los tests para comprobar que todo funciona correctamente.

**Uso de IA en el desarrollo**

Se utilizó IA para investigar sobre APIs públicas de trivia de números, como numbersapi.com, y sobre cómo crear una API propia con Flask. La IA ayudó a identificar buenas prácticas de desarrollo, como el uso de funciones, separación de lógica interactiva y estructuración de pruebas automáticas.

Se empleó IA para generar ejemplos de código, sugerir mejoras y asegurar la compatibilidad con sistemas de evaluación automática.