



ESCUELA DE  
INGENIERÍA EN CIENCIAS Y SISTEMAS  
FACULTAD DE INGENIERÍA  
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



**Día, Fecha:**

Jueves, 03/10/2024

**Hora de inicio:**

10:40 - 12:20

# Introducción a la Programación y Computación 2 [P]

Denilson Florentín de León Aguilar

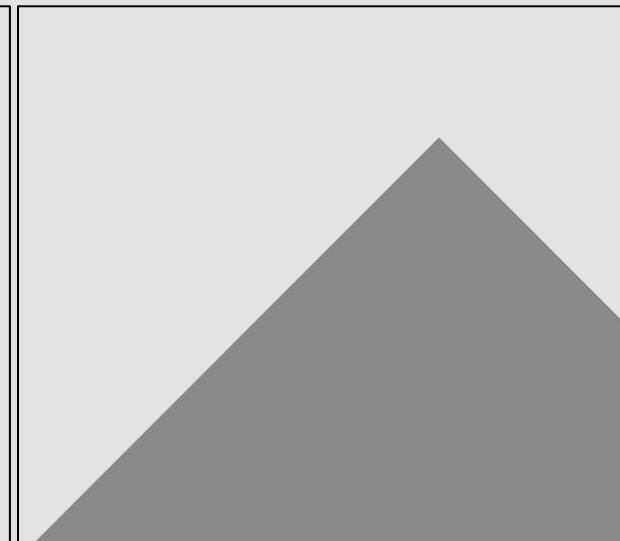


Nombre de la actividad:	Calificación DTT - DSI
Cantidad de participantes:	4
Doy fe que esta actividad está planificada en dtt (Sí/No):	<input checked="" type="checkbox"/> Si

Hora de inicio:	10:50
Hora de fin:	11:00
Duración (min):	10

**Participantes: llenar las siguientes cajas de texto (tomar información del chat del meet)**

ad



# Recordatorio Grabar Clase





## Contenido clase 10

- 8. Acceso a Datos Web:
  - 8.4. Peticiones HTTP en Python
  - 8.5. Librería en Python para realizar peticiones HTTP
  - 8.6. Lectura de archivos binarios mediante peticiones HTTP
  - 8.7. Análisis de XML en Peticiones
- 7. DJANGO
  - instalación
- EJEMPLO:
  - Uso de Django

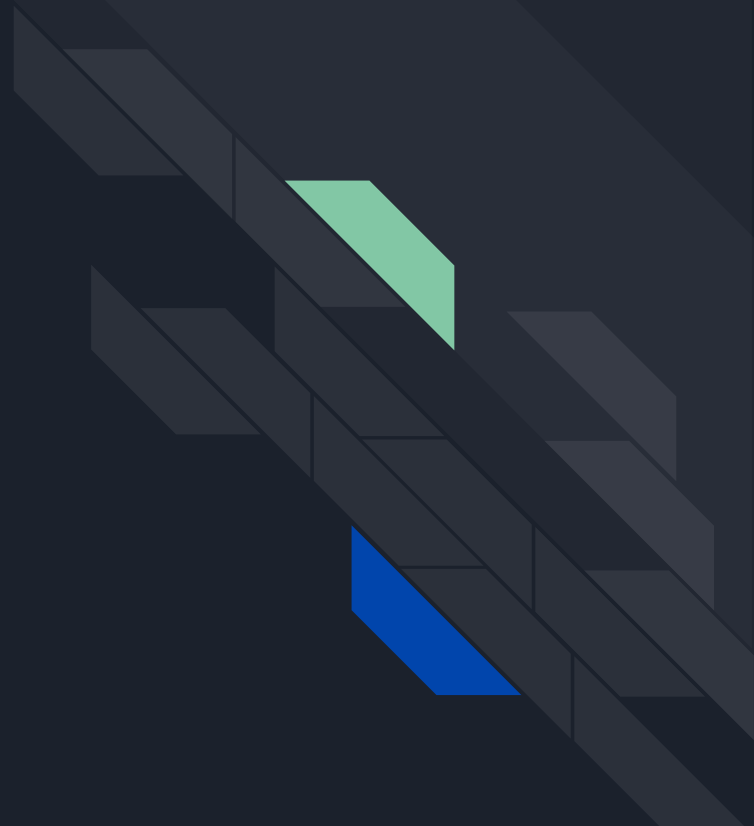


## Contenido clase 10

- Lectura proyecto 3
- Lectura práctica 3
- Calificación Docente N.2 DSI
- JavaScript Object Notation
  - Sintaxis de JSON
  - JSON vs XML
  - Tipos de datos en JSON
  - JSON Parse
  - JSON Stringify
  - JSON Objects
  - JSON Arrays
  - JSON Parse en Python
  - JSON Stringify en Python
- Expresiones Regulares en python

# Lectura proyecto 3 y práctica 3

## 7. JavaScript Object Notation (JSON)





## 7.1 Sintaxis de JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y legible por humanos. Su sintaxis se compone principalmente de pares clave-valor encerrados entre llaves { clave : valor, ....., clave\_n : valor\_n }.

Cada par clave-valor se separa por dos puntos : y los pares se separan entre sí por comas. La clave debe estar entre comillas dobles ", mientras que el valor puede ser una cadena de texto, un número, un booleano, un array, un objeto o null.

```
{  
  "clave1": "valor1",  
  ... ,  
  "claveN": "valorN",  
}
```



## 7.1 Sintaxis de JSON - Ejemplo

Ejemplo de JSON válido:

En este ejemplo, "nombre", "edad", "ciudad", "casado", "hobbies" y "direccion" son claves, y sus valores correspondientes pueden ser cadenas de texto, números, booleanos, arrays u objetos.

```
{  
  "nombre": "Juan",  
  "edad": 30,  
  "ciudad": "Madrid",  
  "casado": false,  
  "hobbies": ["fútbol", "lectura",  
  "viajes"],  
  "direccion": {  
    "calle": "Calle Mayor",  
    "numero": 123  
  }  
}
```

## 7.2 JSON vs XML

JSON y XML son formatos de intercambio de datos comunes. Ambos son legibles por humanos y máquinas, pero difieren en su estructura y uso.

- JSON tiende a ser más ligero y más fácil de leer y escribir para los humanos debido a su sintaxis más simple y concisa.
- Por otro lado, XML es más estructurado y versátil, lo que lo hace más adecuado para representar datos complejos y documentos con una estructura jerárquica definida.

```
{  
  "clave1": "valor1",  
  ...,  
  "claveN": "valorN",  
}
```

```
<root>  
  <child> </child>  
</root>
```



## 7.2 JSON vs XML

Aunque JSON ha ganado popularidad en los últimos años debido a su simplicidad y facilidad de uso, XML sigue siendo ampliamente utilizado en muchas aplicaciones y entornos, especialmente en aquellos donde se requiere una estructura más compleja y una validación rigurosa de los datos.

```
{  
  "clave1": "valor1",  
  ...,  
  "claveN": "valorN",  
}
```

```
<root>  
  <child> </child>  
</root>
```



## 7.3. Tipos de datos en JSON

JSON es un formato de intercambio de datos versátil que admite varios tipos de datos. Aquí te presentamos una explicación detallada de los tipos de datos admitidos en JSON, junto con ejemplos para cada uno:

1. Cadenas de texto
2. Números
3. Booleanos
4. Arrays
5. Objects

```
{  
  "clave1": "valor1",  
  ...,  
  "claveN": "valorN",  
}
```

## 7.3. Tipos de datos en JSON - Cadenas de texto

### Cadenas de texto:

Las cadenas de texto en JSON deben estar entre comillas dobles ("). Pueden contener cualquier combinación de caracteres Unicode y se utilizan para representar texto.

```
{  
  "nombre": "Juan"  
}
```

## 7.3. Tipos de datos en JSON - Números

### Números:

Los números en JSON pueden ser enteros o flotantes y se escriben sin comillas

```
{  
  "edad": 30  
}
```



## 7.3. Tipos de datos en JSON - Booleanos

### Booleanos:

Los booleanos en JSON representan valores verdaderos (true) o falsos (false).

```
{  
  "casado": false  
}
```



## 7.3. Tipos de datos en JSON - Arrays

### Arrays:

Los arrays en JSON son colecciones ordenadas de valores, separados por comas y encerrados entre corchetes ([]). Pueden contener cualquier tipo de dato JSON, incluyendo otros arrays u objetos.

```
{  
  "hobbies": ["fútbol", "lectura", "viajes"]  
}
```



## 7.3. Tipos de datos en JSON - Objects

### Objects:

Los objetos en JSON son conjuntos no ordenados de pares clave-valor, separados por comas y encerrados entre llaves ({}). Las claves deben ser cadenas de texto y los valores pueden ser de cualquier tipo de datos JSON, incluyendo otros objetos.

```
{  
  "direccion": {  
    "calle": "Calle Mayor",  
    "numero": 123  
  }  
}
```



## 7.3. Tipos de datos en JSON

Estos son los tipos de datos básicos en JSON que te permiten representar una amplia variedad de datos estructurados de manera clara y legible.

Es posible utilizar diferentes combinaciones y estructuras de datos según sea necesario.



## 7.4. JSON Parse

La función `JSON.parse()` en JavaScript te permite convertir una cadena JSON en un objeto JavaScript. Esto es útil cuando recibes datos JSON desde un servidor o los lees de un archivo y necesitas manipularlos como objetos en tu código JavaScript.

Ejemplo de uso:

Supongamos que tienes la siguiente cadena JSON:

```
var jsonString = '{"nombre": "Juan", "edad": 30, "casado": false}';
```



## 7.4. JSON Parse

Puedes usar `JSON.parse()` para convertir esta cadena en un objeto JavaScript:

```
var jsonString = '{"nombre": "Juan", "edad": 30, "casado": false}';
```

```
var objeto = JSON.parse(jsonString);  
  
console.log(objeto.nombre); // Salida: Juan  
console.log(objeto.edad); // Salida: 300  
console.log(objeto.casado); // Salida: false
```



## 7.5. JSON Stringify

La función `JSON.stringify()` en JavaScript te permite convertir un objeto JavaScript en una cadena JSON. Esto es útil cuando necesitas enviar datos estructurados a un servidor o almacenarlos en un archivo en formato JSON.

Ejemplo de uso:

Supongamos que tienes un objeto JavaScript como este:

```
var persona = {  
  nombre: "María",  
  edad: 25,  
  casada: true  
};
```



## 7.5. JSON Stringify

Puedes usar `JSON.stringify()` para convertir este objeto en una cadena JSON:

```
var persona = {  
  nombre: "María",  
  edad: 25,  
  casada: true  
};
```

```
var jsonString = JSON.stringify(persona);  
  
console.log(jsonString); // Salida: {"nombre":"María","edad":25,"casada":true}
```

## 7.8 JSON Parse y Stringify en Python?

En Python también existen funciones equivalentes a `JSON.stringify()` y `JSON.parse()` de JavaScript para manipular datos JSON.

Estos son `json.dumps()` y `json.loads()`

```
import json

data = {"name": "John", "age": 30}
json_string = json.dumps(data)
print(json_string)
```

```
import json

json_string = '{"name": "John", "age": 30}'
python_object = json.loads(json_string)
print(python_object)
```

## 7.8 JSON Parse y Stringify en Python?

### `json.dumps()`:

Esta función se utiliza para convertir un objeto de Python en una cadena JSON. Es similar a `JSON.stringify()` en JavaScript. Por ejemplo:

```
import json

data = {"name": "John", "age": 30}
json_string = json.dumps(data)
print(json_string)
```





## 7.8 JSON Parse y Stringify en Python?

### `json.loads():`

Esta función se utiliza para convertir una cadena JSON en un objeto de Python. Es similar a `JSON.parse()` en JavaScript. Por ejemplo:

```
import json

json_string = '{"name": "John", "age": 30}'
python_object = json.loads(json_string)
print(python_object)
```

## 7.8 JSON Parse y Stringify en Python?

Estas funciones son parte del módulo json de Python, que proporciona soporte para codificar y decodificar datos JSON.

```
import json

data = {"name": "John", "age": 30}
json_string = json.dumps(data)
print(json_string)
```

```
import json

json_string = '{"name": "John", "age": 30}'
python_object = json.loads(json_string)
print(python_object)
```

## 7.8 JSON Parse y Stringify en Python?

Salidas de ambos ejemplos:

En el primer ejemplo, `json.dumps()` convierte un objeto Python en una cadena JSON, mientras que en el segundo ejemplo, `json.loads()` convierte una cadena JSON en un objeto Python. La diferencia es que el resultado de `json.dumps()` es una cadena de texto JSON, mientras que el resultado de `json.loads()` es un diccionario Python (objeto).

Para `json.dumps()`:

```
{"name": "John", "age": 30}
```

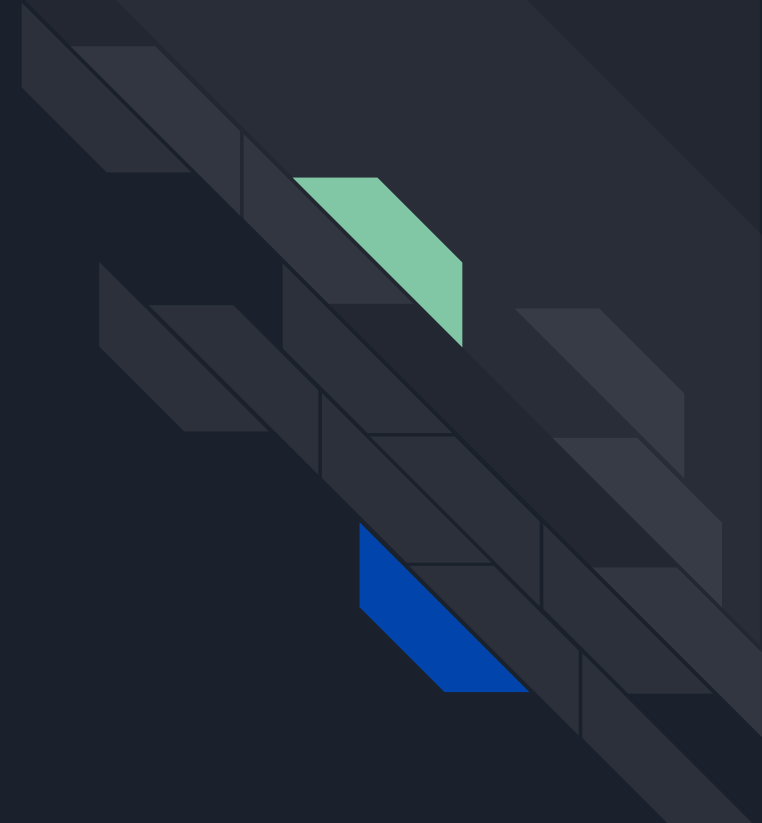
Para `json.loads()`:

```
{'name': 'John', 'age': 30}
```

# Recordatorio Captura de pantalla



# Expresiones Regulares en python





# Expresiones Regulares en python - Manuales

En Python, las expresiones regulares (regex) son un módulo incorporado que te permite buscar patrones dentro de cadenas de texto.

¿Cómo funcionan algunos de los componentes básicos de las expresiones regulares en Python?

cómo funcionan algunos de los componentes básicos de las expresiones regulares en Python:

[0-9]: Esto coincide con cualquier dígito del 0 al 9.

[a-zA-Z]: Esto coincide con cualquier letra, tanto en minúsculas como en mayúsculas.

[0-9a-zA-Z]: Esto coincide con cualquier carácter alfanumérico (letras o números).

\*: Se utiliza para indicar que el patrón anterior puede aparecer 0 o más veces.

+: Se utiliza para indicar que el patrón anterior puede aparecer 1 o más veces.

{n, m}: Se utiliza para indicar que el patrón anterior debe aparecer al menos n veces y como máximo m veces.

\b: Esto coincide con un límite de palabra (es decir, el inicio o el final de una palabra).

# Expresiones Regulares en python - Manuales

## Ejemplos:

```
import re

# Buscar cualquier dígito en una cadena
resultado = re.search(r'[0-9]', 'abc123xyz')
print(resultado.group()) # Salida: 1 (el primer dígito encontrado)

# Buscar palabras que comienzan con "h" seguidas de letras minúsculas
resultado = re.findall(r'\bh[a-z]+\b', 'Hello, how are you? I hope you have a happy day.')
print(resultado) # Salida: ['how', 'have', 'happy']

# Buscar números de 9 dígitos en una cadena
resultado = re.findall(r'\b\d{9}\b', 'Mi carnet es 123456789 y mi número de teléfono es 9876543210.')
print(resultado) # Salida: ['123456789']
```

# Expresiones Regulares en python - Manuales


Ejemplos:

```
import re

# Verificar si una cadena contiene solo letras y números
if re.match(r'^[a-zA-Z0-9]+$', 'abc123'):
    print("La cadena contiene solo letras y números")
else:
    print("La cadena contiene otros caracteres además de letras y números")

# Para numeros funcionan \d o [0-9] Buscar digitos usando 'd'
resultado = re.findall(r'\d', 'abc123xyz')
print(resultado) # Salida: ['1', '2', '3']
# Buscar digitos usando [0-9]
resultado = re.findall(r'[0-9]', 'abc123xyz')
print(resultado) # Salida: ['1', '2', '3']
```






# Expresiones Regulares en python - Secuencias de Escape

Python ofrece varios "alias" o secuencias de escape predefinidas que pueden resultar útiles al escribir expresiones regulares. Algunos de los más comunes son:

```
\d: Representa cualquier dígito decimal (equivalente a [0-9]).  
\w: Representa cualquier carácter alfanumérico (letras, números y guiones bajos) (equivalente a [a-zA-Z0-9_]).  
\s: Representa cualquier carácter de espacio en blanco (espacios, tabulaciones, saltos de línea, etc.).  
\b: Representa un límite de palabra (el inicio o el final de una palabra).  
\D: Representa cualquier carácter que no sea un dígito decimal (equivalente a [^0-9]).  
\W: Representa cualquier carácter que no sea alfanumérico.  
\S: Representa cualquier carácter que no sea un espacio en blanco.
```



# Expresiones Regulares en python - Secuencias de Escape

Ejemplos:

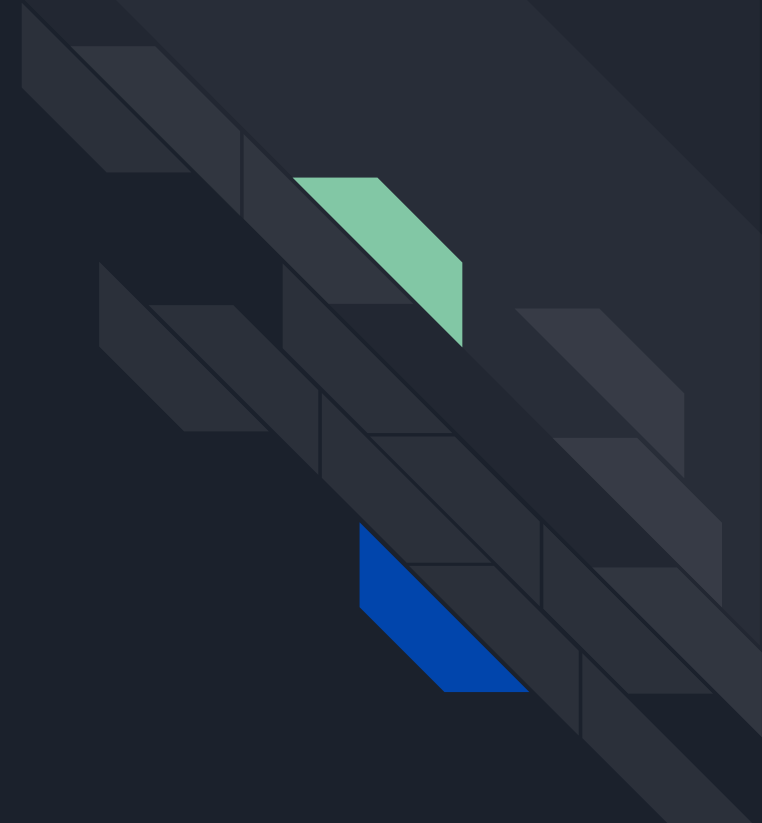
```
import re

# Buscar cualquier dígito decimal usando '\d'
resultado = re.findall(r'\d', 'abc123xyz')
print(resultado) # Salida: ['1', '2', '3']

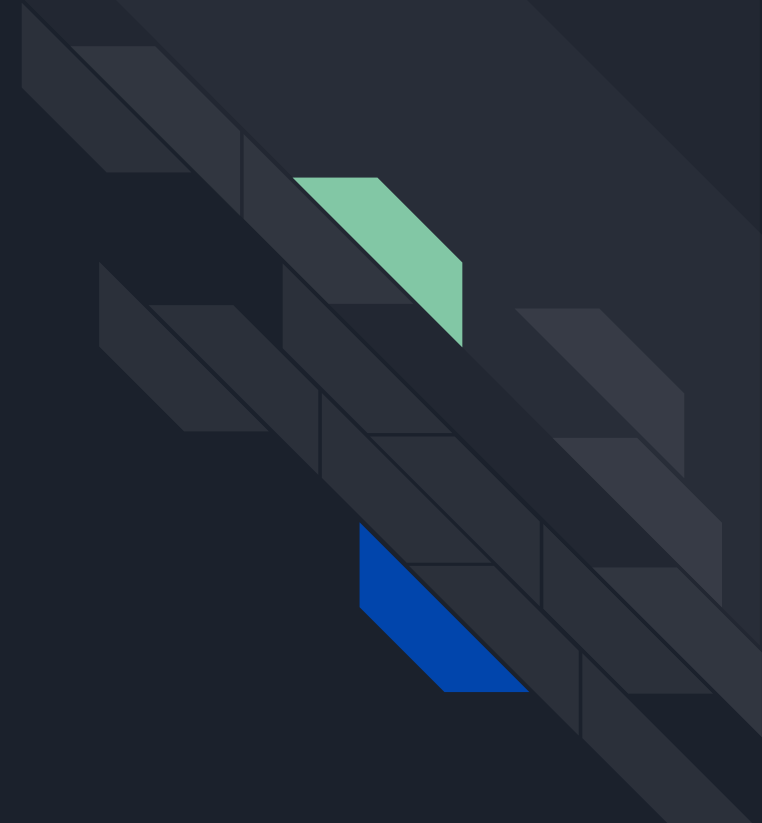
# Buscar cualquier carácter alfanumérico usando '\w'
resultado = re.findall(r'\w', 'abc123xyz')
print(resultado) # Salida: ['a', 'b', 'c', '1', '2', '3', 'x', 'y', 'z']

# Buscar cualquier carácter de espacio en blanco usando '\s'
resultado = re.findall(r'\s', 'Hola Mundo')
print(resultado) # Salida: [' ']
```

## 8. Acceso a datos web



# Conexión HTTP






## 8.4. Peticiones HTTP en Python

¿Cómo podemos utilizar la biblioteca requests para enviar solicitudes HTTP a servidores remotos y manejar las respuestas recibidas?

Discutiremos los diferentes tipos de solicitudes HTTP, como GET, POST, PUT y DELETE, y cómo podemos usar estos métodos para interactuar con recursos en la web.

```
import requests
```



## 8.5. Librería en Python para realizar peticiones HTTP

requests posee diversas características y funcionalidades que nos ofrece, como la gestión de sesiones, el manejo de cookies y encabezados personalizados, entre otros.

```
import requests

class HTTPRequests:
    def __init__(self, url):
        self.url = url

    def make_get_request(self):
        response = requests.get(self.url)
        return response.text

    def make_post_request(self, data):
        response = requests.post(self.url, data=data)
        return response.text
```



## 8.6. Lectura de archivos binarios mediante peticiones HTTP

En esta sección, nos adentraremos en cómo podemos descargar y leer archivos binarios desde servidores remotos utilizando Python y la biblioteca requests. Veremos cómo enviar solicitudes HTTP para descargar archivos binarios, como imágenes, archivos de audio o documentos, y cómo podemos guardar y manipular estos archivos binarios en nuestro sistema local.

```
import requests
```

```
def download_binary_file(self, filepath):  
    response = requests.get(self.url)  
    with open(filepath, 'wb') as file:  
        file.write(response.content)  
    return f"Archivo binario descargado en {filepath}"
```



## 8.6. Lectura de archivos binarios mediante peticiones HTTP

Ejemplo de uso: En este ejemplo, se ha agregado un nuevo método `download_binary_file` que permite descargar un archivo binario desde una URL y guardarlo en el sistema de archivos local.

```
# Ejemplo de uso
url = 'https://example.com/data/sample.bin'
http_client = HTTPRequests(url)

# Realizar descarga de archivo binario
filepath = 'sample.bin'
result = http_client.download_binary_file(filepath)
print(result)
```



## 8.7. Análisis de XML en Peticiones HTTP

¿Cómo analizar y procesar respuestas XML obtenidas a través de solicitudes HTTP en Python?

Utilizaremos la biblioteca estándar `xml.etree.ElementTree` para parsear documentos XML y extraer información útil de ellos. Manejaremos diferentes estructuras y formatos de datos presentes en respuestas XML de servidores remotos, y cómo podemos utilizar esta información en nuestras aplicaciones Python.

```
import requests
import xml.etree.ElementTree as ET

class HTTPRequests:
    // demas codigo

    def parse_xml_response(self):
        response = requests.get(self.url)
        root = ET.fromstring(response.text)
        return root
```

# Ejemplos Flask y Node



# Comunicación entre servicios

<https://genderize.io/documentation#localization>

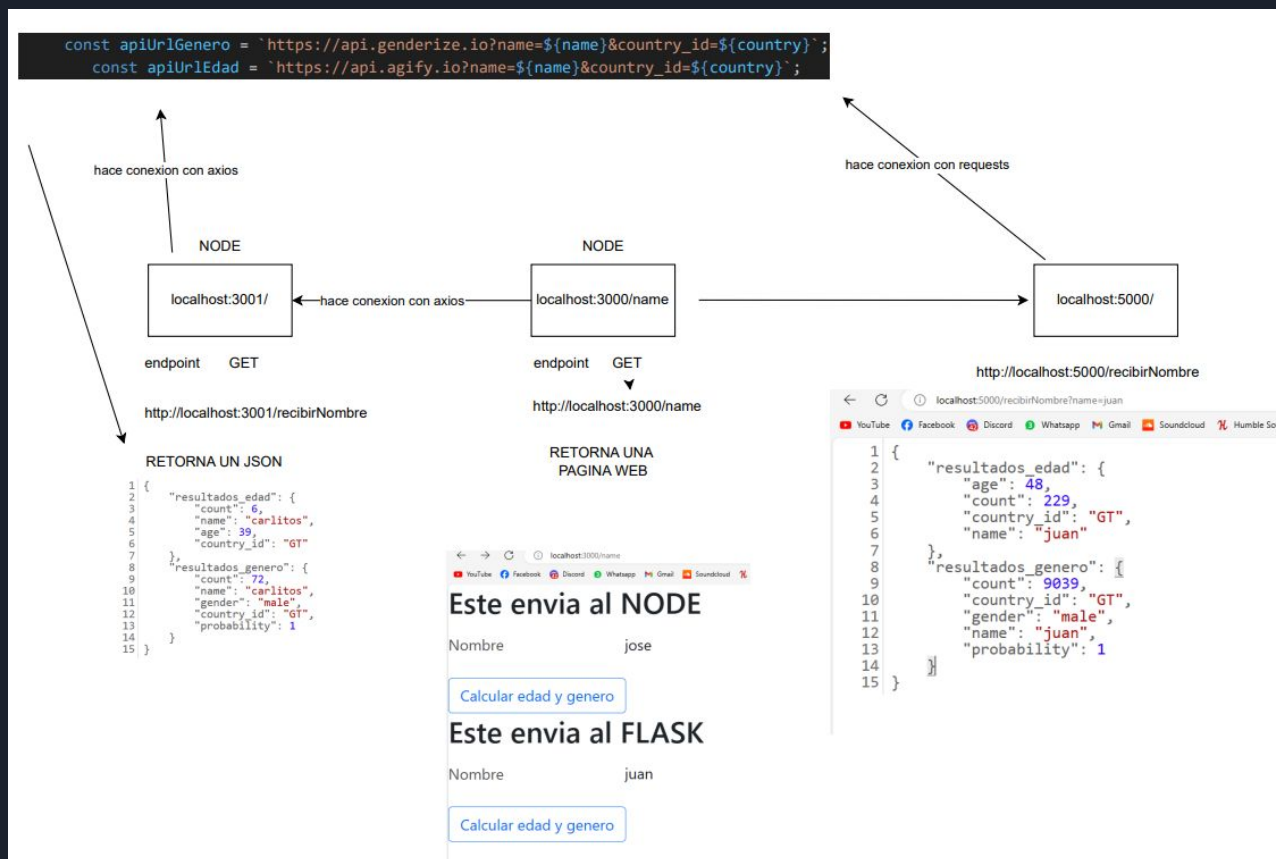
<https://agify.io/documentation#localization>

# Flujo - Salida de las API de flask y node

retornan un json

```
1  {  
2    "resultados_edad": {  
3      "age": 48,  
4      "count": 229,  
5      "country_id": "GT",  
6      "name": "juan"  
7    },  
8    "resultados_genero": {  
9      "count": 9039,  
10     "country_id": "GT",  
11     "gender": "male",  
12     "name": "juan",  
13     "probability": 1  
14   }  
15 }
```

# Flujo final



# Ejemplos Django Configuration

The background of the slide features a series of dark gray, three-dimensional rectangular planes that recede into the distance, creating a sense of depth. A bright green parallelogram is positioned on one of the upper planes, and a bright blue parallelogram is on a lower plane, both adding a pop of color to the monochromatic scheme.

## 7. DJANGO - Instalación y Creación Proyecto

Preparar virtual environment (venv)

```
python -m venv myVenv
```

Ejecutar en linux y windows

```
# En Linux  
source myVenv/Scripts/activate
```

```
# En windows  
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass  
  
.\myVenv\Scripts\activate
```

```
+ FullyQualifiedPath : UnauthorizedAccess  
PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass  
PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django> .\myVenv\Scripts\activate  
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django> █
```



## 7. DJANGO - Instalación y Creación Proyecto

### Paso 1: Instalación de Django

Abre tu terminal y ejecuta el siguiente comando para instalar Django utilizando pip:

```
pip install django
```

### Paso 2: Crear un nuevo proyecto Django

Una vez instalado Django, puedes crear un nuevo proyecto ejecutando el siguiente comando en tu terminal:

```
django-admin startproject nombre_del_proyecto
```

```
cd nombre del proyecto  
django-admin startapp nombre_de_app
```





## 7. DJANGO - Creación aplicación en el proyecto

### Paso 3: Crear una aplicación

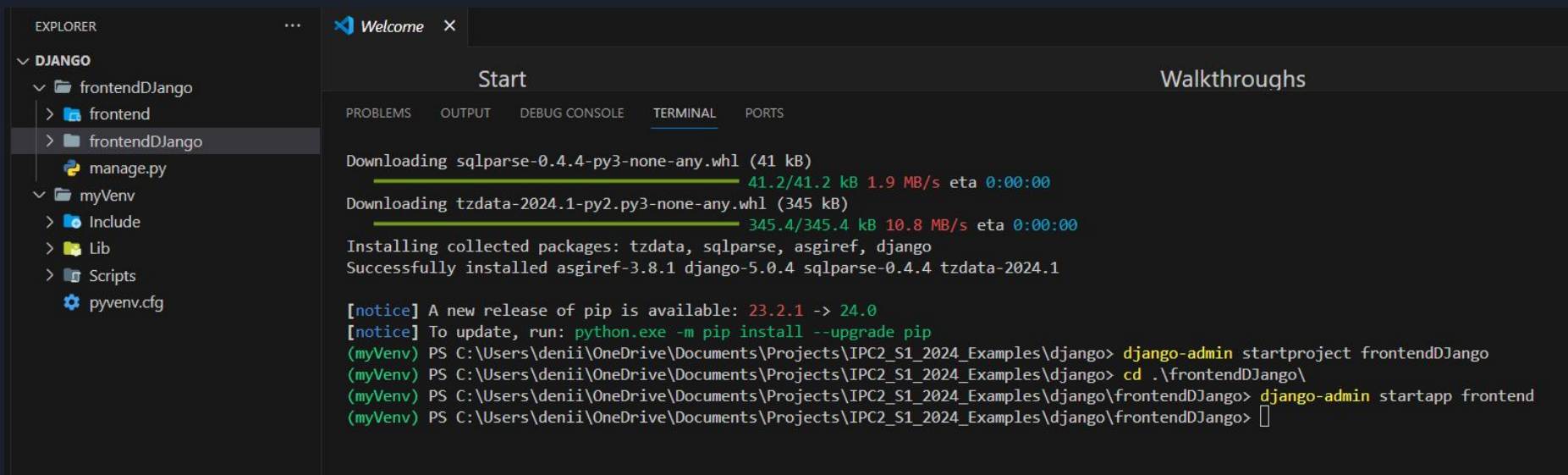
Una vez creado el proyecto Django.

Crea una aplicación Django dentro de tu proyecto ejecutando el siguiente comando en tu terminal (asegúrate de estar en la raíz de tu proyecto Django):

```
django-admin startapp nombre_de_app
```

```
python manage.py startapp nombre_de_app
```

## 7. DJANGO - Instalación y Creación Proyecto



The screenshot displays the Visual Studio Code interface with a project named 'DJANGO'. The Explorer sidebar on the left shows the file structure: 'frontendDjango' (containing 'frontend' and 'frontendDjango' subfolders), 'manage.py', 'myVenv', 'Include', 'Lib', 'Scripts', and 'pyvenv.cfg'. The main editor area is titled 'Start' and 'Walkthroughs'. The 'TERMINAL' tab is active, showing the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
41.2/41.2 kB 1.9 MB/s eta 0:00:00
Downloading tzdata-2024.1-py2.py3-none-any.whl (345 kB)
345.4/345.4 kB 10.8 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.0.4 sqlparse-0.4.4 tzdata-2024.1

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django> django-admin startproject frontendDjango
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django> cd .\frontendDjango\
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\frontendDjango> django-admin startapp frontend
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\frontendDjango> 
```



## 7. DJANGO - Migrate

El comando `python manage.py migrate` en Django se utiliza para aplicar las migraciones pendientes al esquema de la base de datos. Las migraciones son archivos de Python generados automáticamente por Django que representan los cambios en los modelos de datos. Cuando defines modelos en Django y realizas cambios en ellos, como agregar campos o modificar relaciones, necesitas crear migraciones y luego aplicarlas a la base de datos para reflejar esos cambios en el esquema de la base de datos.

```
python manage.py migrate
```



## 7. DJANGO - Migrate

El proceso generalmente involucra los siguientes pasos:

- Crear o modificar modelos en tu aplicación Django.
- Generar migraciones usando el comando `python manage.py makemigrations`. Esto crea archivos de migración en el directorio `migrations` de cada aplicación Django.
- Aplicar las migraciones pendientes a la base de datos usando el comando `python manage.py migrate`.

```
python manage.py migrate
```

## 7. DJANGO - Migrate

```
python manage.py migrate
```

```
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\fr
ontendDjango> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\fr
ontendDjango>
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\fr
ontendDjango>
```



## 7. DJANGO - Configurar usuario

Configurar el administrador (admin), no es obligatorio, pero es una buena práctica para administrar fácilmente los datos de tu aplicación durante el desarrollo. Configurar el administrador te permite crear, leer, actualizar y eliminar (CRUD) registros de tus modelos directamente desde una interfaz de administración web. Esto puede ser útil, especialmente durante el desarrollo y las pruebas de tu aplicación Django.

```
# Creamos usuario administrador (datos ejemplo, configurar prudentemente)
$ python manage.py createsuperuser
  > Email Address: admin@admin.com
  > Password: admin
  > Repeat Password Again: admin
```

## 7. DJANGO - Configurar usuario

Configurar el administrador (admin), no es obligatorio, pero es una buena práctica para administrar fácilmente los datos de tu aplicación durante el desarrollo. Configurar el administrador te permite crear, leer, actualizar y eliminar (CRUD) registros de tus modelos directamente desde una interfaz de administración web. Esto puede ser útil, especialmente durante el desarrollo y las pruebas de tu aplicación Django.

```
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\frontendDJango> python manage.py createsuperuser
Username (leave blank to use 'denii'): admin
Email address: admin@admin.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(myVenv) PS C:\Users\denii\OneDrive\Documents\Projects\IPC2_S1_2024_Examples\django\frontendDJango> █
```



## 7. DJANGO - Ejecutar

### Paso 4: Ejecutar el servidor de desarrollo

Una vez que hayas migrado tu proyecto Flask a Django, puedes ejecutar el servidor de desarrollo de Django para probar tu aplicación. Ejecuta el siguiente comando en tu terminal (asegúrate de estar en la raíz de tu proyecto Django):

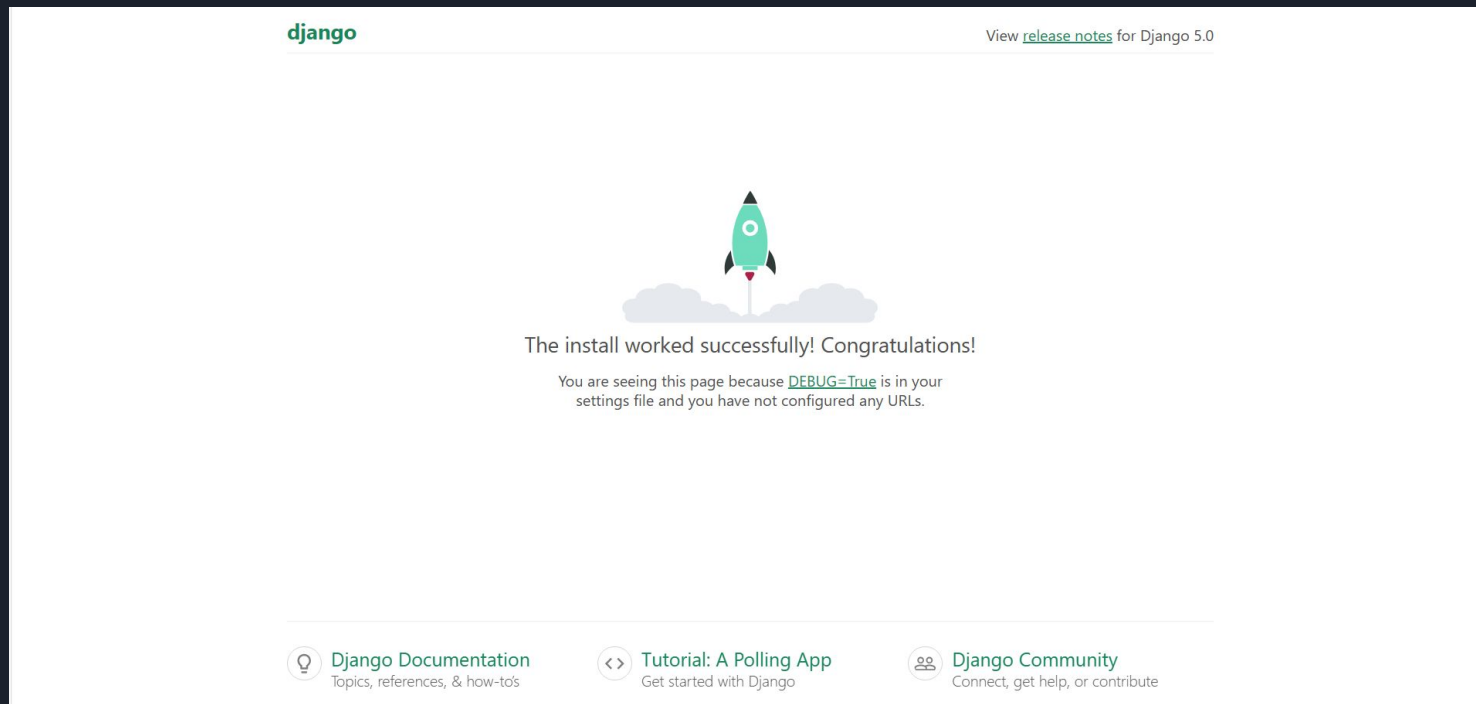
```
python manage.py runserver
```

Esto iniciará el servidor de desarrollo de Django, y podrás acceder a tu aplicación desde tu navegador web visitando la dirección `http://localhost:8000`.



## 7. DJANGO - Ejecutar

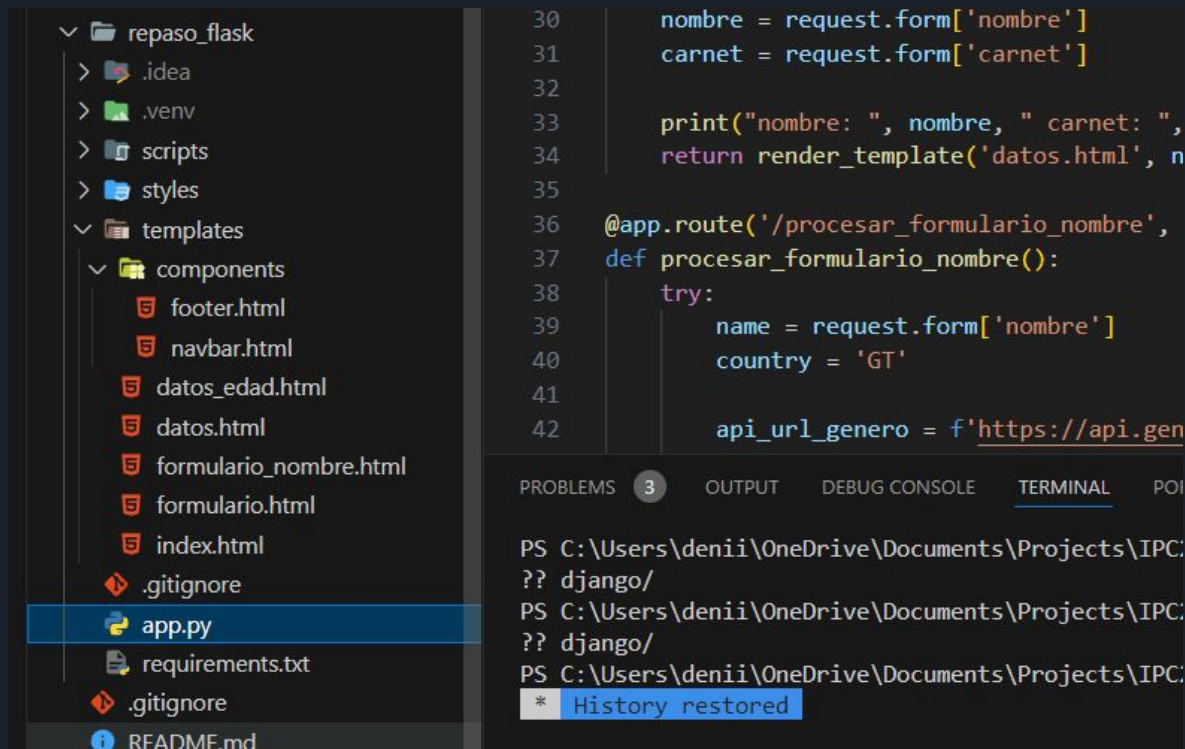
```
python manage.py runserver
```



# Migrar Ejemplo repaso de Flask (frontend) hacia Django



## 7. DJANGO - Flask a migrar



The screenshot displays a code editor interface. On the left, a file explorer shows the project structure for 'repaso\_flask'. The files listed are:

- repaso\_flask
  - .idea
  - .venv
  - scripts
  - styles
  - templates
    - components
      - footer.html
      - navbar.html
      - datos\_edad.html
      - datos.html
      - formulario\_nombre.html
      - formulario.html
      - index.html
    - .gitignore
    - app.py
    - requirements.txt
    - .gitignore
    - README.md

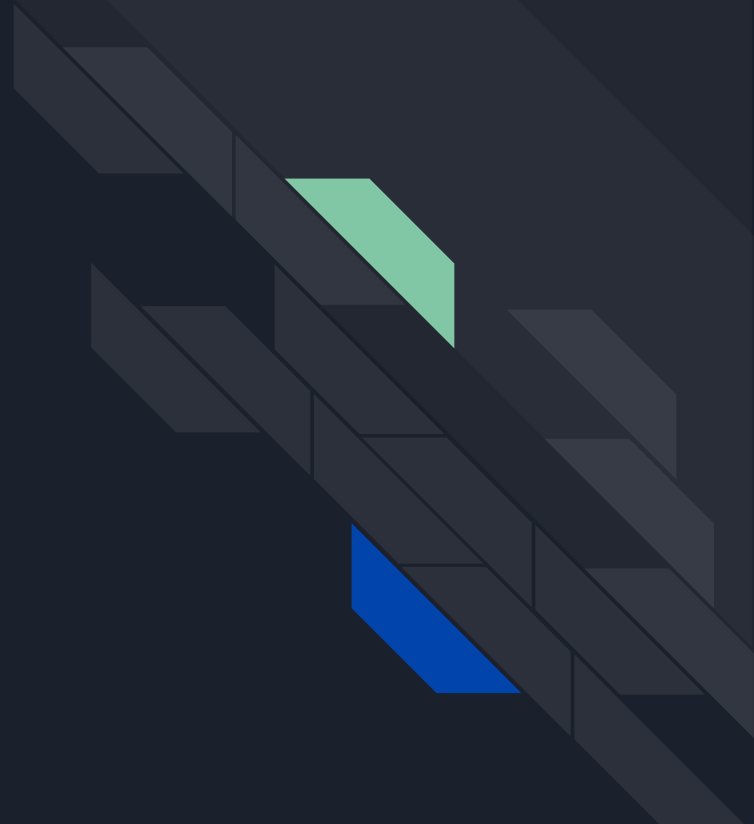
On the right, the code editor shows the following Python code:

```
30 nombre = request.form['nombre']
31 carnet = request.form['carnet']
32
33 print("nombre: ", nombre, " carnet: ",
34       return render_template('datos.html', n
35
36 @app.route('/procesar_formulario_nombre',
37 def procesar_formulario_nombre():
38     try:
39         name = request.form['nombre']
40         country = 'GT'
41
42         api_url_genero = f'https://api.gen
```

At the bottom, the terminal window shows the following commands and output:

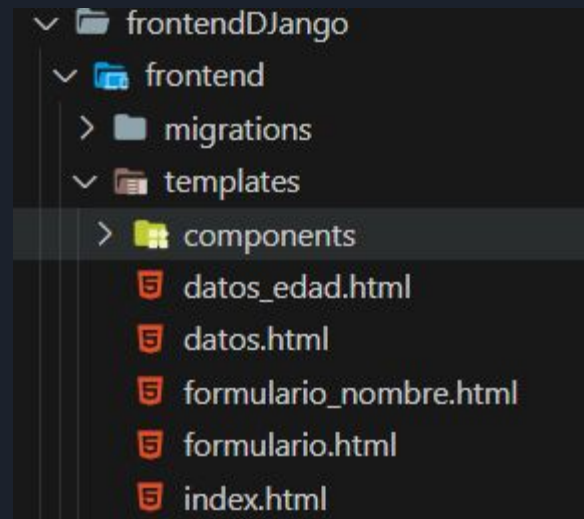
```
PS C:\Users\denii\OneDrive\Documents\Projects\IPC:
?? django/
PS C:\Users\denii\OneDrive\Documents\Projects\IPC:
?? django/
PS C:\Users\denii\OneDrive\Documents\Projects\IPC:
* History restored
```

# Configure Templates



## 7. DJANGO - Templates

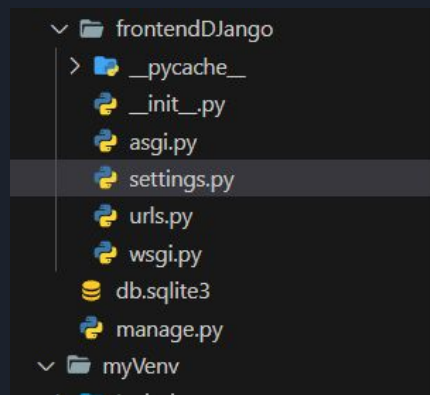
Creamos el directorio templates y  
agregamos los archivos html necesarios



## 7. DJANGO - Settings project

agregamos la app creada en el archivo settings.py de la carpeta project dentro de project

En este ejemplo sería “frontend”

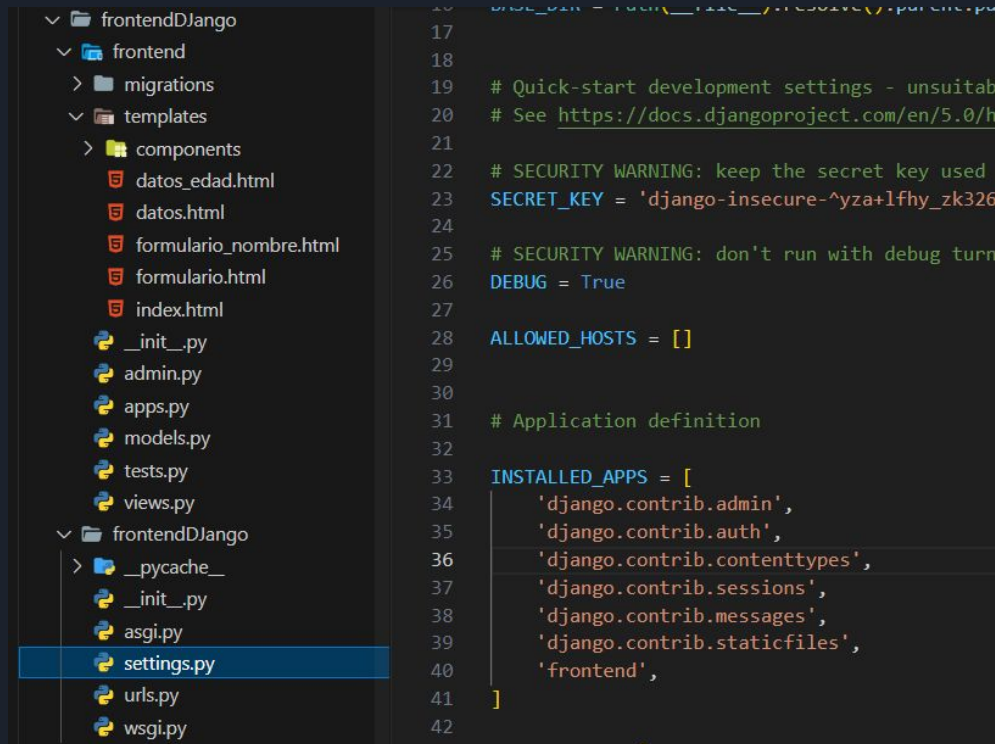


```
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'myApp'
41 ]
42
```

## 7. DJANGO - Settings project

agregamos la app creada en el archivo settings.py de la carpeta project dentro de project

En este ejemplo sería “frontend”



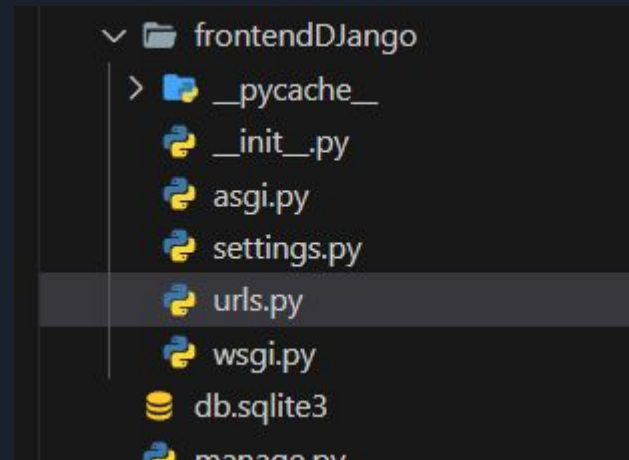
The screenshot shows a code editor with a dark theme. On the left is a file explorer showing a project structure. The 'frontendDJango' folder is expanded, showing subfolders 'frontend' and 'frontendDJango'. The 'frontend' folder contains 'migrations', 'templates', and 'components' (with HTML files). The 'frontendDJango' folder contains a 'pycache' folder and several Python files, with 'settings.py' highlighted. On the right, the content of 'settings.py' is visible, showing Django configuration settings like 'SECRET\_KEY', 'DEBUG', 'ALLOWED\_HOSTS', and 'INSTALLED\_APPS'.

```
16 BASE_DIR = Path(__file__).resolve().parent.parent
17
18
19 # Quick-start development settings - unsuitab
20 # See https://docs.djangoproject.com/en/5.0/h
21
22 # SECURITY WARNING: keep the secret key used
23 SECRET_KEY = 'django-insecure-^yza+lfhy_zk326
24
25 # SECURITY WARNING: don't run with debug turn
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'frontend',
41 ]
42
```

## 7. DJANGO - URLS project

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("myApp.urls"))
]
```



En nuestro ejemplo, myApp se llama “frontend”

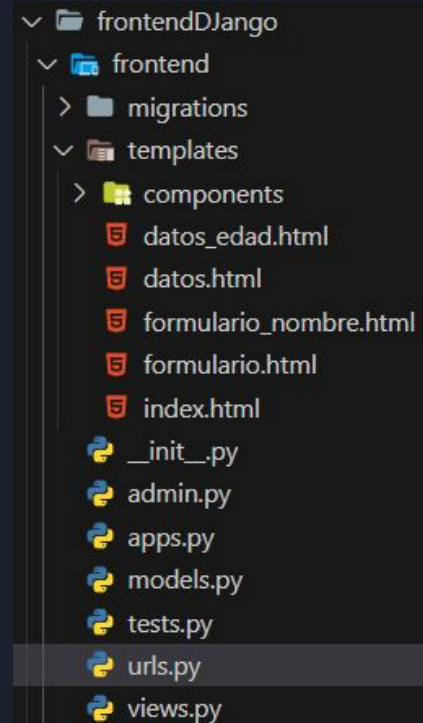


## 7. DJANGO - URLS App

```
from . import views
from django.urls import path

urlpatterns = [
    path("", views.viewName, name="home"),
]
```

Creamos el archivo "urls.py" que referenciamos anteriormente. Además, en este caso llamaremos a index.



## 7. DJANGO - VIEWS App

```
from django.shortcuts import render

def index(request):
    context = {}
    return render(request, "index.html", context)
```

La dirección a usar en “index.html” en este caso tiene que ir relativa al directorio /templates

