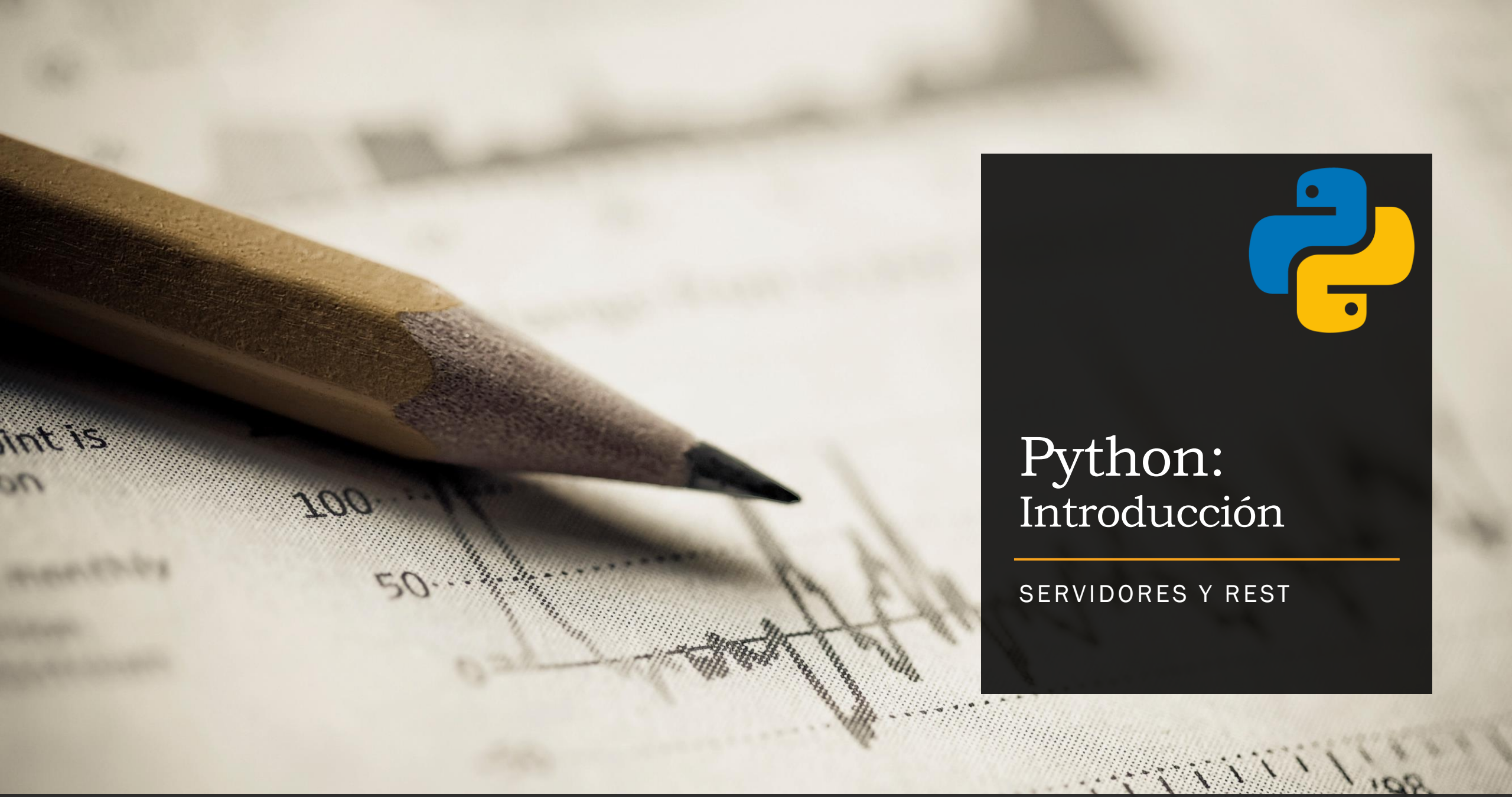


Python:
Introducción

SERVIDORES Y REST

Esta clase

- Servidores
- El protocolo HTTP
- APIs y REST
- Flask para servicios web y servidores

Recordando

Mediante Python podemos ofrecer una multitud de diferentes servicios en muchos contextos.

Entre ellos, la habilidad de utilizar servidores y proveer servicios web.

Concepto: Servidores

Con el pasar de los años, el mundo se ha convertido en una red de interconexiones entre múltiples entidades, personas y servicios.

Mediante el internet por ejemplo, podemos acceder a páginas web alojadas en múltiples lugares del mundo, o utilizar aplicaciones que internamente dependan de servicios externos para ofrecer distintas funcionalidades, como por ejemplo una aplicación de videollamadas.

Concepto: Servidores

Un servidor es simplemente un programa de computadora que provee un servicio a otro programa de computadora, conocido como el cliente.

Estos programas pueden encontrarse en el mismo contexto, o como suele ser más común, en distintos nodos o máquinas.

Una aplicación puede funcionar como un servidor para ciertas aplicaciones, y a su vez como un cliente para otras.

Servidores: Generalización

Un servidor de manera más general, es un proceso en un computador o máquina virtual, el cuál mediante protocolos de redes, se comunica con otros procesos.

Concepto: Dirección IP

Simplemente una cadena de caracteres que identifica un computador usando el protocolo de internet (IP) en una red.

Esto permite que cada unidad o equipo en una red, y en el mundo pueda ser encontrado e identificado por otras máquinas en la misma red.

Concepto: Puertos

Un puerto es un punto de acceso virtual en donde empiezan y terminan las conexiones de red dentro de un sistema operativo.

Por ejemplo, si nuestro computador expone la IP 1.2.3.4, pueden existir muchos procesos distintos que escuchan tráfico de red, por lo qué mediante puertos podemos diferenciarlos.

Un servidor ejecutándose en un computador suele escuchar uno o más puertos específicos.

Ejemplo: Puertos

- David tiene un computador con IP **1.2.3.4** asignada por la red
- Juan tiene un computador dentro de la misma red, cuya IP es **4.5.6.7**
- David creó un servidor en su lenguaje favorito con el fin de proveer su fecha de cumpleaños a cualquier persona que tenga acceso a la red
- David inició el servidor escuchando al puerto **5000**

Juan puede ahora solicitar mediante la red la información provista por la IP **1.2.3.4,** en el puerto **5000.** Esto quiere decir que su computador mediante protocolos de red obtendrá la fecha de cumpleaños de David.

Servidor web

Recordemos que mediante el internet uno puede hacer muchas cosas más aparte de navegar en páginas web, por ejemplo usar una aplicación de correo, jugar juegos en línea, etc.

Entonces es importante entender que un servidor puede tener muchas clases.

Un **servidor web** se enfoca enteramente en ofrecer acceso a páginas y servicios web, comúnmente mediante el protocolo **HTTP**.

Página web

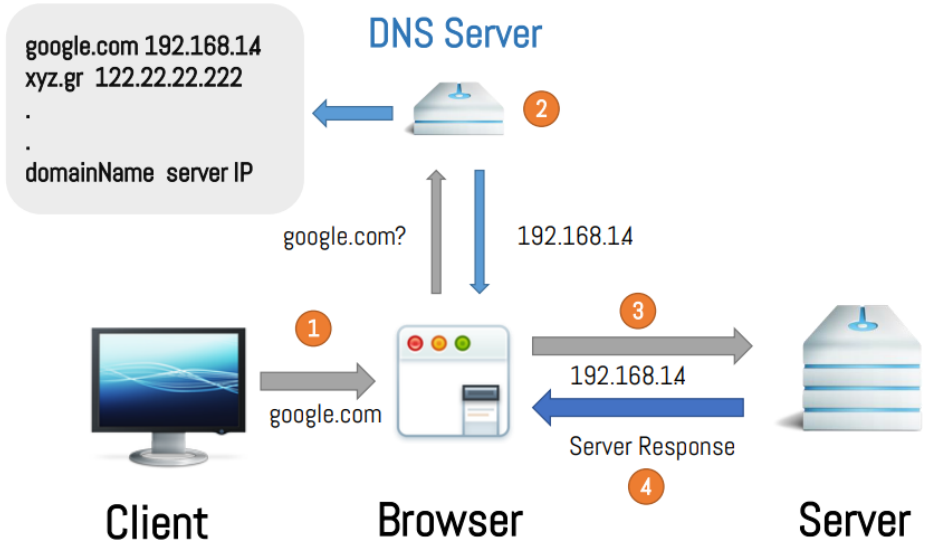
Recordemos que una página web no es nada más que una serie de archivos HTML, JavaScript y CSS que se encuentran ubicados en algún lugar del mundo.

La manera en que estos archivos llegan a nuestro explorador web es mediante un servidor web.

¿Cómo accedemos a una página?

Implícitamente nuestro explorador solicita el puerto 80 o 443 en la IP del servidor

El servidor DNS es una herramienta que traduce nombres de dominio a IPs.



[Entendiendo qué es un servidor DNS - Blog (godaddy.com)]

Servicio web

Otra manera en la que podemos usar servidores es para exponer servicios web.

Los servicios web esencialmente permiten acceder a funciones de software mediante la red, usando los mismos protocolos que utilizamos para resolver páginas web.

Servicio web

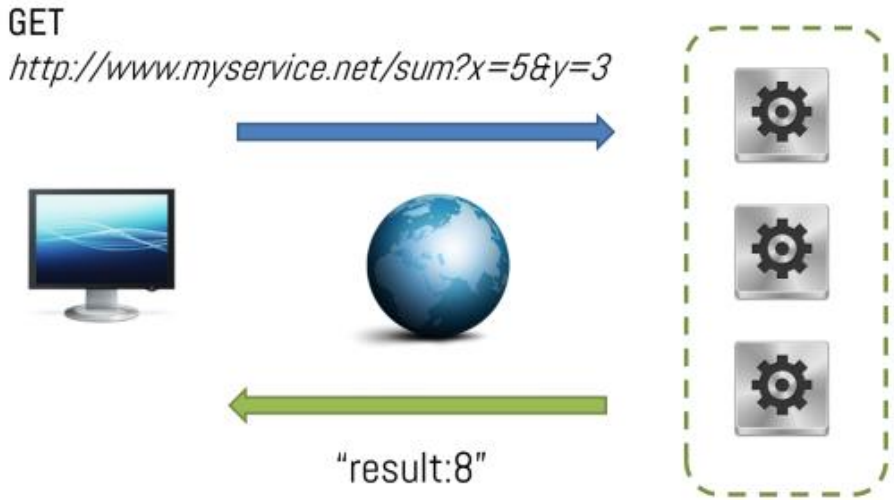
Cuando programamos, usamos funciones para operaciones repetitivas, o para obtener datos que permitan que nuestra aplicación cumpla su objetivo.

```
sumar(1, 2) > 3

obtener_cedula("Daniel") > 11000045678
```

Un servicio web es cualquier tipo de software disponible sobre la red y que utiliza un formato estandarizado de transmisión de mensajes (JSON, XML, etc.) con el fin de ejecutar funciones remotas.

Servicio web



- ✓ Un servicio web se accede mediante una red
- ✓ Tiene una interfaz bien definida de funciones disponibles
- ✓ No requieren un lenguaje de programación específico

Servicio web

Por lo tanto, es posible crear servicios web con nuestro lenguaje favorito, ya que casi todos proveen utilidades de red que permiten ofrecer las funcionalidades requeridas.

Interfaces de programación de aplicaciones (API)

Una **API** es simplemente la especificación que describe como los componentes de software deben interactuar.

Permite que el software intercambie mensajes con otros productos sin necesidad de conocer su implementación.

En un servicio web es necesario definir una **API** clara que permita acceder a sus funcionalidades.

https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Servicios REST

Un estilo arquitectónico que permite definir servicios web.

REpresentational

State

Transfer

- Aprovecha la tecnología y protocolos de comunicación web para proveer acceso a funcionalidades de software en lugar de páginas.
- Por lo tanto existe interoperabilidad en sistemas conectados por el internet.

Servicios REST

- Utiliza enteramente el protocolo HTTP como método de comunicación entre clientes y servidores.
- No retiene el estado de las operaciones, es decir cada operación es independiente y debe incluir toda la información requerida para la misma.
- Mediante el protocolo HTTP podemos transmitir información usando JSON, XML o cualquier estructura estandarizada, y recibir JSON, XML o cualquier estructura estandarizada.

¿Protocolos? ¿HTTP?

REST funciona enteramente aprovechando el protocolo HTTP como forma de comunicación entre procesos.

Un protocolo de red es simplemente una forma estandarizada mediante la cuál uno o más procesos pueden hablar entre sí.

HTTP fue ideado originalmente para acceder a recursos web

HTTP

Funciona mediante una serie de métodos enviados a través de la red, que son interpretados de acuerdo a su significado estandarizado.

GET:

- Instruye al servidor que transmita la información identificada por la URL pasada
- Normalmente es de solo lectura, es decir, que una operación GET no debería modificar ninguna clase de información

Por ejemplo, GET se usa para solicitar una página web, esto no debería en ningún caso cambiar alguna información.

HTTP

Funciona mediante una serie de métodos enviados a través de la red, que son interpretados de acuerdo a su significado estandarizado.

POST:

- Instruye al servidor que cree o actualice el recurso identificado por la URL pasada
- Las solicitudes adicionalmente incluyen un cuerpo de información que puede ser utilizado en el proceso de actualización o creación

- **Por ejemplo, POST se puede usar para enviar los datos de una ficha llenada por un usuario, lo que permitirá crear un registro en alguna base de datos.**

HTTP

Funciona mediante una serie de métodos enviados a través de la red, que son interpretados de acuerdo a su significado estandarizado.

PUT:

- Instruye al servidor que cree o actualice el recurso identificado por la URL pasada
- Similar a POST pero representa una acción idempotente.

HTTP

Funciona mediante una serie de métodos enviados a través de la red, que son interpretados de acuerdo a su significado estandarizado.

DELETE:

- Instruye al servidor que elimine el recurso identificado por la URL pasada

REST y HTTP

Entonces, REst como modelo arquitectónico de servicios web permite aprovechar HTTP para definir el significado de las operaciones provistas por la API del servicio.

Por ejemplo

- Se expone un punto de acceso GET para obtener una lista de nombres
- Se expone un punto de acceso POST para crear un estudiante en una base de datos
- Se expone un punto de acceso DELETE para eliminar una persona de un registro

HTTP: Códigos

Un servidor web responde con códigos de error de acuerdo a la respuesta del mismo, estos códigos han sido estandarizados en la documentación del protocolo HTTP.

Estos se pueden usar por el cliente para determinar la acción correcta a seguir.

- 200 OK
- 201 Recurso creado
- 401 Credenciales requeridas para acceder al servicio
- 500 Error en el servidor
- Entre otros...

Es la responsabilidad del creador de la API emitir los códigos más acertados

HTTP response status codes - HTTP | MDN (mozilla.org)

Alternativas: SOAP

SOAP es una alternativa bastante común para servicios web, no utiliza HTTP y provee un protocolo mucho más complejo.

Al no ser nativa en muchos contextos web no provee la misma interoperabilidad, por ejemplo, no es sencillo llamar a estos servicios dentro de un explorador de internet.

REST vs. SOAP (redhat.com)

REst

Aprendiendo un poco más

¿Qué es API RESTful? | Guía sobre API RESTful para principiantes | AWS (amazon.com)

Resumen

Entonces, un servidor es simplemente un proceso ejecutado en un computador, que se encuentra escuchando a un puerto correspondiente a la IP del equipo dentro de la red actual.

Resumen

Entonces, un servidor **web** es simplemente un proceso ejecutado en un computador, que se encuentra escuchando a un puerto correspondiente a la IP del equipo dentro de la red actual, y que se enfoca específicamente en proveer acceso a **páginas** o **servicios web REst.**

Resumen

Entonces, un servidor **web** es simplemente un proceso ejecutado en un computador, que se encuentra escuchando a un puerto correspondiente a la IP del equipo dentro de la red actual, y que se enfoca específicamente en proveer acceso a **páginas** o **servicios web REst.** Un servicio **REst** es un servicio que utiliza el protocolo **HTTP** a manera de transmitir información mediante una estructura estandarizada.

```
GET https://miservicio.com/datos/frutas

Respuesta:
200 OK
[
    {
    "nombre": "manzana",
    "color": "verde"
    },
    {
    "nombre": "banana",
    "color": "amarilla"
    },
]
```

```
POST https://miservicio.com/datos/crear_fruta
Cuerpo:
{
    "nombre": "mango",
    "color": "rojo"
}

Respuesta:
201 Creado
```

Conclusión

Preguntas?

Servicios en Python

Podemos recordar que Python posee un ecosistema muy extendido de librerías y utilidades.

Esto también aplica a herramientas para acceder a servicios web, o exponer servidores!

requests

A pesar de existir muchas liberías útiles, requests es una de las más sencillas y mejor documentadas.

Permite acceder a puntos HTTP, utilizar proxies, procesar códigos de respuesta, y mucho más.

```
py –m pip install requests
```

Usando requests

Podemos fácilmente utilizar los verbos HTTP mediante las funciones expuestas por la libería

```
import requests
respuesta = requests.get('https://xkcd.com/1906/')
codigo = respuesta.status_code
print(codigo)
contenido = respuesta.text
```

```
Out[17]: 200
```

Mediante el parámetro **text** podemos obtener el cuerpo de la respuesta, por ejemplo si esta es una página HTML, podemos posteriormente procesarla con ayuda de otras liberías.

Usando requests: descargas

Es muy fácil guardar el resultado de una solicitud, en caso de que por ejemplo sea un archivo.

```
import requests
resultado = requests.get("https://wwwnc.cdc.gov/travel/images/map-ecuador.png")
with open("alguna/ubicacion", 'wb') as f:
    f.write(resultado.content)
```

Recordemos que *open* nos permite trabajar con archivos de manera muy sencilla.

Usando requests: parámetros

Dentro del protocolo HTTP, las operaciones permiten proveer diferentes parámetros en la solicitud.

```
import requests
argumentos = {'argumento1':1,'argumento2':2}
r = requests.get('https://httpbin.org/get’, params=argumentos)
print(r.text)
```

En el caso de una solicitud GET, esto simplemente agregará los mismos a la URL

```
https://httpbin.org/get?argumento1=1&argumento2=2
```

Usando requests: POST

Como aprendimos anteriormente, existen diferentes verbos que podemos usar en HTTP, y cuando un servicio web soporta puntos de acceso POST, PUT o DELETE, los podemos acceder de manera similar.

```
import requests
argumentos = {'usuario':1,'clave':2}
r = requests.post('https://httpbin.org/post',params=argumentos)
```

A diferencia de GET, los parámetros serán enviados en el cuerpo de la solicitud, no en la URL.

Usando requests: JSON

Es posible que un servicio web responda directamente con información estructurada como objetos JSON y no texto plano. La siguiente función crea un diccionario a partir de ese JSON.

```
import requests
argumentos = {’username’: ’olivia’, ’password’: 123':}
respuesta = requests.post('https://httpbin.org/post',params=argumentos)

print(respuesta.json())
```

```
{'args': {}, 'data': '', 'files': {}, 'form': {'password': '123', 'username': 'olivia'}, 'headers': {'Accept': '*/*', 'Accept-E
ncoding': 'gzip, deflate', 'Content-Length': '28', 'Content-Type': 'application/x-www-form-urlencoded', 'Host': 'httpbin.org',
'User-Agent': 'python-requests/2.18.4'}, 'json': None, 'origin': '103.10.31.17, 103.10.31.17', 'url': 'https://httpbin.org/pos
t'}
```

Usando requests

Entonces, mediante **requests** podemos comunicarnos de manera sencilla con servidores HTTP, ya sea para interactuar con páginas web, o para vincularnos con servicios.

Conclusión

Preguntas?

Servidores en Python

De la misma manera que podemos interactuar con servidores externos mediante nuestro código de Python, podemos crear aplicaciones o servicios web de manera muy sencilla mediante un ecosistema de librerías y frameworks con este propósito.

[Welcome to Flask — Flask Documentation (2.1.x) (palletsprojects.com)]

Un mini-framework para servicios y aplicaciones web en Python!

Enfocado en la simpleza, muy fácil de utilizar y de entender. Permite por defecto ofrecer aplicaciones web muy rápidamente!

¿Qué es un framework web?

Una serie de utilidades y módulos que le permite a un desarrollador crear aplicaciones web sin preocuparse por gestionar código de nivel bajo como los protocolos, gestión de hilos, y muchas otras consideraciones.

Flask require simplemente que instalemos su dependencia base

```
py –m pip install flask
```

Un ejemplo básico!

```python
from flask import Flask

app = Flask(__name__)

# Usando la síntaxis de @ podemos definir las rutas y su comportamiento

@app.route("/")
def hola_mundo():
    return "<p>Hola a todos!</p>"

@app.route("/adios")
def adios():
    return "<p>Adios a todos!</p>"
```

Podemos definir la variable FLASK_APP con el nombre del módulo inicial

```
FLASK_APP=ejemplo.py python -m flask run
```

O alternativamente por defecto flask buscará el archivo **app.py**

Por defecto flask inicia el servidor en el puerto 5000

```
FLASK_APP=ejemplo.py python -m flask run –p OTROPUERTO
```

Flask entonces, opera bajo el concepto de rutas, cada función se puede asignar a una o más rutas dependiendo de su comportamiento. Por defecto, el método es GET.

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hola_mundo():
    return "<p>Hola a todos!</p>"
```

En el contexto de páginas web, no es eficiente escribir el código HTML en el código. Existe el concepto de plantillas o **templates.**

Por defecto, flask busca plantillas **jinja** en la carpeta **templates.**

[Template Designer Documentation — Jinja Documentation (3.1.x) (palletsprojects.com)]

```python
from flask import render_template, Flask

app = Flask(__name__)

@app.route('/hola')
def hola():
    return render_template('hola.html', nombre="Daniel")
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Mi pagina</title>
</head>
<body>
    Hola {{ nombre }}
</body>
</html>
```

Sin embargo, la función puede definirse para operar con otros métodos, por ejemplo POST.

```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/enviar_json', methods=['POST'])
def procesar_json():
    tipo = request.headers.get('Content-Type')
    if (tipo == 'application/json'):
        json = request.json
        # Hacer algo con el objeto JSON
        return json
    else:
        return 'Tipo de contenido en el objeto no soportado!'
```

Las rutas, no son recursos específicos, de hecho la cadena de texto es una “regla”.

```
@app.route('/ruta/')
```

La misma puede ser dinámica e incluir parámetros.

```
@app.route('/ruta/<variable>')
def procesar_ruta(variable):
    return f"Recibi el parametro {variable}"
```

127.0.0.1:8000/ruta/daniel
127.0.0.1:8000/ruta/juan

Así como es posible devolver texto plano, archivos o manejar plantillas. Es posible de igual manera devolver objetos formateados, por ejemplo JSON, algo muy útil en contextos de servicios web!

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/obtener_datos')
def procesar_informacion():
    informacion = {
        "nombre": "Daniel",
        "cedula": "123456"
    }
    #La función jsonify convierte un diccionario en una respuesta JSON!
    return jsonify(informacion)
```

Lo importante es que tenemos el poder entero de Python! Los datos que devolvamos pueden venir de una base de datos, un archivo, o inclusive otro servidor web (por ejemplo si usamos requests como intermediario!).

```python
from flask import Flask, jsonify

app = Flask(__name__)

def leer_archivo():
    archivo = None
    with open("archivo.csv") as a:
        archivo = a.readlines()
    return archivo

@app.route('/obtener_datos')
def procesar_informacion():
    contenido = leer_archivo()
    #Procesamiento de la línea del archivo!
    partes = contenido[0].split(',')
    #La función jsonify convierte un diccionario en una respuesta JSON!
    return jsonify({"nombre": partes[0], "cedula": partes[1]})
```

archivo.csv

```
daniel,12345
```

resultado

```
{
"cedula": "12345",
"nombre": "daniel"
}
```

Pero eso es solo la superficie! Mediante flask podemos gestionar sesiones, aplicar contextos separados, manejar integraciones con cookies, y muchas cosas más.

Revisar la documentación!

django

Una alternativa a flask.

Django es un proyecto más complejo y mucho más opinionado.

Pero al definir una estructura más establecida permite que las aplicaciones grandes sean más fáciles de gestionar en ciertos contextos.

Flask vs Django: What's the Difference Between Flask & Django? (guru99.com)

The web framework for perfectionists with deadlines | Django (djangoproject.com)

Siguientes clases

- Continuación Flask
- Lectura de archivos y bases de datos
- Introducción Github
- Ejercicio interactivo