Acceso a otros orígenes de datos (Teórico/Práctico)

tiiMiit by

Introducción

Existen distintos orígenes de datos que pueden ser accedidos. Los mismos pueden ser datos obtenidos de grandes almacenes de datos y exportados a datos específicos de distintos formatos. En este documento nos vamos a focalizar en orígenes de datos de tipo ASCII organizados en distintos tipos de formatos como ser TXT, CSV y JSON.

Al finalizar este módulo serás capaz de:

- Manipular estos tipos de archivos python, lo que te permitirá acceder a muchísima información para tus modelos.
- Podrás entender y comparar los distintos tipos de formatos de archivos y su uso.

Tema 1. Acceso a archivos en formato texto.

Objetivo

El objetivo de este tema es presentar los tipos de archivo de datos más conocidos y la manera de manipular cada tipo de dato con Python.

Información general

En el trabajo habitual de pruebas automatizadas, Python se usa a menudo para leer y escribir algunos archivos. Los formatos de archivo más utilizados son txt, log, json, csv, xml, zip, tar, gz, rar, excel, estos diez formatos de archivo.

Entre ellos, los cinco formatos de txt, log, json, csv y xml se pueden operar usando la biblioteca estándar de Python.

txt, lectura y escritura del archivo de registro

Los métodos de lectura y escritura de los archivos .txt y .log son los mismos. A continuación, solo se utilizan archivos .txt como ejemplo.

Escribir:

```
with open("test.txt","w") as f:
    f.write("test string")
```

Leer:

```
with open("test.txt","r") as f:
    print(f.read())
```

Asuntos que requieren atención:

- Modo de archivo general

  r: abrir en modo lectura
  w: abre escribiendo,
  a: Abrir en modo de adjuntar (comience desde EOF, cree un nuevo archivo si es necesario)

- MP3 multimedia, mp4 o modo que contenga caracteres especiales

rb: abrir en modo de lectura binaria
wb: Abrir en modo de escritura binaria
ab: abrir en modo de adición binaria

- otro:

  r+,w+,a+,rb+,wb+,ab+

Python lee archivos grandes (nivel de GB):

La asignación de memoria la maneja el intérprete de Python, que es muy eficiente en lectura y consume menos recursos.

```
with open("test.txt","r") as f:
    for line in f:
        print(line)
```

Archivo json leer y escribir

Escribir:

```
import json
test_dict = {'bigberg': [7600, {1: [['iPhone', 6300], ['Bike', 800], ['shirt', 300]]}]}
json_str = json.dumps(test_dict,indent=4,sort_keys=True)
with open('test.json',"w") as f:
    f.write(json_str)
```

Leer:

```
import json
with open('test.json',"r") as f:
    json_dic = json.load(f)
    print(json_dic["bigberg"])
```

4. lectura y escritura de archivos csv

- Utilice la clase de escritor en csv

  Escribir:

  ```
  import csv
  datas = [['name', 'age'],['Bob', 14], ['Tom', 23], ['Jerry', '18']]
  with open('example.csv', 'w', newline='') as f:
      writer = csv.writer(f)
      for row in datas:
          writer.writerow(row)
  ```

Leer:

```
import csv
with open("example.csv",'r') as f:
    reader = csv.reader(f)
    for row in reader:
        print(reader.line_num, row)
```

- Utilice la clase Dictwriter en csv

Escribir:

```
import csv
headers = ['name', 'age']
datas = [{'name':'Bob', 'age':23},
        {'name':'Jerry', 'age':44},
        {'name':'Tom', 'age':15}]
with open('example.csv', 'w', newline='') as f:
    writer = csv.DictWriter(f, headers)
    writer.writeheader()
    for row in datas:
        writer.writerow(row)
```

Leer:

```
import csv
filename = 'example.csv'
with open(filename) as f:
    reader = csv.DictReader(f)
    for row in reader:
        name = row['name']
        print(name)
```

Lectura y escritura del archivo xml

Escribir:

```
#coding=utf-8
import xml.dom.minidom
def GenerateXml():
    impl = xml.dom.minidom.getDOMImplementation()
    dom = impl.createDocument(None, 'CONFIG_LIST', None)
    root = dom.documentElement
    employee = dom.createElement('COMP')
    root.appendChild(employee)
    nameE = dom.createElement('path')
    nameE.setAttribute ("valor", "aaaaaaaaaaa") # Agregar atributo
```

```
    nameT = dom.createTextNode('linux')
    nameE.appendChild(nameT)
    employee.appendChild(nameE)
    f = open('config_new.xml', 'a')
    dom.writexml(f, addindent='  ', newl='\n')
    f.close()
GenerateXml()
```

Leer:

```
import xml.etree.cElementTree as ET
tree = ET.parse("config_new.xml")
root = tree.getroot()
COMP = root.findall ('COMP') [0]
print("Tag:", COMP.tag, "Attributes:", COMP.attrib, "Text:",
COMP.text.strip(), "Tail:", COMP.tail)

ruta = COMP.findall ("ruta") [0]
print("Tag:", path.tag, "Attributes:", path.attrib, "Text:", path.text.strip(),
"Tail:", path.tail)
```

La clase #Element y la clase ElementTree tienen el método iter () para atravesar recursivamente todos los elementos secundarios del elemento / árbol para el niño en tree.iter (etiqueta = 'ruta'): #tree es el objeto ElementTree

```
print("Tag:", child.tag, "Attributes:", child.attrib, "Text:", child.text.strip())
```

Tema 2. Acceso a API Rest

Objetivo

El objetivo de este tema es conocer qué es una API Rest y cuales son los comandos para accederla.

Qué es API REST

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones que tener en cuenta en la arquitectura software que usaremos para crear aplicaciones web respetando HTTP.

Según Fielding las restricciones que definen a un sistema RESTful serían:

Cliente-servidor: El servidor se encarga de controlar los datos mientras que el cliente se encarga de manejar las interacciones del usuario. Esta restricción mantiene al cliente y al servidor débilmente acoplados (el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente).

Sin estado: aquí decimos que cada petición que recibe el servidor debería ser independiente y contener todo lo necesario para ser procesada.

Cacheable: debe admitir un sistema de almacenamiento en caché. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.

Interfaz uniforme: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección o “URI”.

Sistema de capas: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

Hoy en día la mayoría de las empresas utilizan API REST para crear servicios web. Esto se debe a que es un estándar lógico y eficiente. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook o también la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, Google Maps, ...).

Métodos en una API REST

Las operaciones más importantes que nos permitirán manipular los recursos son:

GET es usado para recuperar un recurso.

POST se usa la mayoría de las veces para crear un nuevo recurso. También puede usarse para enviar datos a un recurso que ya existe para su procesamiento. En este segundo caso, no se crearía ningún recurso nuevo.

PUT es útil para crear o editar un recurso. En el cuerpo de la petición irá la representación completa del recurso. En caso de existir, se reemplaza, de lo contrario se crea el nuevo recurso.

PATCH realiza actualizaciones parciales. En el cuerpo de la petición se incluirán los cambios a realizar en el recurso. Puede ser más eficiente en el uso de la red que PUT ya que no envía el recurso completo.

DELETE se usa para eliminar un recurso.

Otras operaciones menos comunes pero también destacables son:

HEAD funciona igual que GET pero no recupera el recurso. Se usa sobre todo para testear si existe el recurso antes de hacer la petición GET para obtenerlo (un ejemplo de su utilidad sería comprobar si existe un fichero o recurso de gran tamaño y saber la respuesta que obtendremos de la API REST antes de proceder a la descarga del recurso).

OPTIONS permite al cliente conocer las opciones o requerimientos asociados a un recurso antes de iniciar cualquier petición sobre el mismo.

Dos términos a tener en cuenta son los de métodos seguros y métodos idempotentes. Se dice que los métodos seguros son aquellos que no modifican recursos (serían GET, HEAD y OPTIONS), mientras que los métodos idempotentes serían aquellos que se pueden llamar varias veces obteniendo el mismo resultado (GET, PUT, DELETE, HEAD y OPTIONS).

La idempotencia es de mucha importancia ya que permite que la API sea tolerante a fallos. Por ejemplo, en una API REST bien diseñada podremos realizar una operación PUT para editar un recurso. En el caso de que tuviésemos un fallo de Timeout (se ha superado el tiempo de espera de respuesta a la petición), no sabríamos si el recurso fue creado o actualizado. Como PUT es idempotente, no tenemos que preocuparnos de comprobar su estado ya que, en el caso de que se haya creado el recurso,

una nueva petición PUT no crearía otro más, algo que sí habría podido pasar con una operación como POST.

Algunas características de una API REST

El uso de hipermedios (procedimientos para crear contenidos que contengan texto, imagen, vídeo, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML (principio HATEOAS, Hypermedia as the engine of application state o hipermedia como el motor del estado de la aplicación).

Independencia de lenguajes. La separación en capas de la API permite que el cliente se despreocupe del lenguaje en que esté implementado el servidor. Basta con saber que las respuestas se recibirán en el lenguaje de intercambio usado (que será XML o JSON).

Los recursos en una API REST se identifican por medio de URI. Será esa misma URI la que permitirá acceder al recurso o realizar cualquier operación de modificación sobre el mismo.

Las APIs deben manejar cualquier error que se produzca, devolviendo la información de error adecuada al cliente. Por ejemplo, en el caso de que se haga una petición GET sobre un recurso inexistente, la API devolverá un código de error HTTP 404.

Algunos frameworks con los que podremos implementar nuestras APIs son: JAX-RS y Spring Boot para Java, Django REST framework para Python, Laravel para PHP, Rails para Ruby o Restify para Node.js.

Tema 4. Creación de una API REST

Objetivo

Tener los conocimientos básicos para crear un servicio de API REST.

Introducción

Para configurar un servidor Python, debe instalar Python, sugeriría cualquier versión superior a 3.7 a partir del año 2019.

Una vez instalado, abra su terminal o cmd para instalar el flask.

```
> pip install Flask
```

Una vez que se instala el flask, necesitaremos configurar un entorno virtual para ejecutar nuestra aplicación.

Configuración de un entorno virtual.

Comenzaremos creando una carpeta y agregando una venvcarpeta dentro.

```
> md sandbox
> cd sandbox
> py -m venv venv
```

Para activar el entorno, navegue hacia ./venv/Scripts/y en Linux . venv/bin/activate. En Windows, use cmd.

./code/sandbox/venv/Scripts/> activar

Navegue de regreso a sandbox cuál es la raíz y cree un archivo app.py.

Cree una aplicación Flask mínima.

Aquí, importamos la clase Flask y creamos una instancia de ella. Para crear una instancia, tendríamos que darle un nombre y usar (__name__) garantiza que pueda iniciarse como una aplicación o importarse como un módulo. Usamos el route()decorador para que nuestra aplicación de flask sepa qué URL debe activar el método correspondiente. Luego, la función simplemente devuelve un mensaje de cadena usando diferentes URL en el ejemplo.

Para ejecutar la aplicación, primero debemos completar algunas cosas. Primero, configure el entorno en desarrollo y dígale a su terminal la aplicación con la que trabajará exportando la variable FLASK_APP de entorno en Linux.

```
$ exportar FLASK_ENV=desarrollo
$ exportar FLASK_APP=app.py
```

y ventanas

```
> establecer FLASK_ENV=desarrollo
> establecer FLASK_APP=app.py
```

Ejecutar usando.

```
> py -m flask run
```

* Ejecutándose en http://127.0.0.1:5000

Por defecto, el puerto es 5000.

Has creado con éxito tu primer servidor de Python usando Flask. Es bastante básico y devuelve respuestas de cadena, vamos a animar un poco las cosas aprendiendo algunas cosas más que podemos hacer.

Enrutamiento.

Las rutas se consideran puntos finales, puede crear diferentes rutas para sus puntos finales que utilizan diferentes métodos.

Usamos el route()decorador para vincular una función a una URL. Aquí hay una serie de rutas con detalles en los comentarios.

De forma predeterminada, una ruta solo responde a las solicitudes GET. Tendrá que importar request desde el flask para identificar el tipo de método utilizado.

Plantillas de renderizado.

Cuando se usa express.js, Pug es el motor de plantilla predeterminado. Bueno, en Flask usamos Jinja2 . Flask configura Jinja2 automáticamente durante la instalación, y para generar plantillas, todo lo que necesita es importar render_template desde el flask y las variables que desea pasar al motor de plantillas como argumentos de palabras clave.

Acceso a los datos de la solicitud.

Es posible que desee pasar datos a través del método POST más seguro en lugar de exponerlos a través de la URL. Para acceder a los datos del formulario (transmitidos a través de POST o PUT métodos), puede utilizar el atributo de formulario.

Si las claves username o password no existen, entonces KeyError se genera un especial. Puede detectarlo como cualquier otro error, pero si no lo hace, se muestra una página de error HTTP 400 (Solicitud incorrecta). Para acceder a los parámetros enviados en la URL (?key=value), puede usar el atributo args .

palabra clave de búsqueda = solicitud.args.get('clave': '')

Se recomienda detectar KeyError cuando se utilizan parámetros de URL, ya que algunos usuarios pueden cambiar la URL, lo que puede generar una página de error de solicitud incorrecta.

Cargas de archivos.

Python es un lenguaje muy simple, se vuelve aún más simple usando Flask para cargar imágenes, archivos o videos. Flask le permite cargar archivos desde un objeto de formulario, sólo asegúrese de configurar enctype="multipart/form-data"el atributo en su formulario.

Si bien los archivos cargados se almacenan temporalmente en la memoria o en una ubicación temporal en el sistema de archivos, puede usar el método save() para almacenar el archivo en el sistema de archivos del servidor. Cuando crea un servidor, no se recomienda que almacene archivos en el servidor, debe almacenar archivos en un servicio como AWS Storage, Firebase (de Google), Azure (Microsoft), Dropbox y otros y solo mantener la URL de estos archivos almacenados en una base de datos separada como cadenas, tal vez incluso en el servidor. Sin embargo, así es como puede guardar archivos en el servidor en caso de que lo desee.

Puede acceder a su archivo utilizando el nombre de host de su servidor más el directorio de archivos, es decir, https://myapp.com/var/www/uploads/profilephoto.png después de guardarlo en el sistema de archivos.

Conclusión.

Es bastante fácil crear API con Flask. Puede responder con JSON serializando el valor en JSON y devolviéndole. Use el módulo json que viene con python para serializar sus datos en JSON. Puede conectarse a MongoDB y almacenar valores usando el popular pymongo, o conectarse a cualquier otra base de datos. SQLite3 viene por defecto en Python. No se

recomienda usar una base de datos SQLite; sin embargo, a medida que su aplicación se escala, no es lo suficientemente potente como para manejar una gran cantidad de datos.

Cierre

En este documento hemos presentado los fundamentos básicos del acceso a datos de tipo texto y las nociones para comprender API REST y establecer un servicio sencillo con esta tecnología web.

La combinación de estos conocimientos te permitirá acceder a mucha información de distintos orígenes de datos, manipularla y transformarla para tu propio interés.

Referencias

https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwiyloaEk-36AhWhQkgAHTwWDLsYABACGgJjZQ&ohost=www.google.com&cid=CAESbOD2SCmB_3RAAw6tYvDdoYzXOIEQHmEi-KHjGsZS5Rjk1JuBLqYAELbaIj9PyBNCUFa0i9liRhRJL3SsClrWcJTr7LdRseMK1aCUO-LKJyu7qC4Q6nJvH_PC646WpxH-NP9I8hOfzJFc4kCDbg&sig=AOD64_24QHh5urOKbssxYsgp115yPTG5PQ&q&adurl&ved=2ahUKEwjR-f-Dk-36AhWor5UCHXZ6DpkQ0Qx6BAgFEAE

https://www.redhat.com/en/topics/api/what-is-a-rest-api

https://www.bbvaapimarket.com/en/api-world/rest-api-what-it-and-what-are-its-advantages-project-development/

https://es.acervolima.com/guardar-texto-json-y-csv-en-un-archivo-en-python/

https://www.youtube.com/watch?v=jw8kUnS_1Uk

https://www.analyticslane.com/2018/07/16/archivos-json-con-python/

tiimiit by