ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY Academy

NOME MODULO: *PYTHON*

UNITÀ DIDATTICA: *PYTHON 5*

ANDREA DIMITRI

Anno Accademico 2023-2024

NB. Le presentazioni devono pervenire alla Segreteria Didattica ITS (segreteria@its-ictacademy.com) possibilmente a inizio Modulo o al termine della singola Unità Didattica. I materiali didattici verranno condivisi con gli Allievi solo al termine del Modulo.

PYTHON – UNITA' 4

INDICE DEGLI ARGOMENTI

- La gestione della persistenza tramite file
- I file CSV
- I file xml
- **I file json**
- **Gestione di file json tramite python**

Tutti gli esempi di queste slide stanno in pythonMAC/pythonJson

Il formato json

Come per csv e per xml, json è un formato per comunicare dati, intendendo per dati informazioni relative ad una o più entità. Es. devo comunicare ad una terza parte informazione relative ad un gruppo di persone. Es. tutte le persone di un certo comune italiano.

Problema: ti mando un file con dentro tutte le informazioni dei cittadini del comune di Ladispoli.

Potrebbe sembrare semplice, ma se vogliamo essere precisi, evitare errori, mantenere sempre la coerenza, la soluzione non è semplice.

Es. ti mando prima il nome e poi il cognome.

Ma se c'è un cittadino che si chiama Giuseppe Marco Tulli, il nome è Giuseppe oppure è Giuseppe Marco?

Il formato json

Json nasce per risolvere questo tipo di problemi. Dobbiamo cioè poter rappresentare dati, insieme di entità, senza errori e ambiguità.

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

Il formato json

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

L'elemento di base è l'**Object** che in json inizia con la parentesi { e termina con }.

Un Object contiene coppie chiave/valore separate da una virgola.
I : separano la chiave dal valore.

Gli spazi bianche sono ignorati.

Il valore può essere di un tipo primitivo: string, number, boolean, null

Oppure può essere un oggetto, oppure può essere un array.

Un array è una lista di valori dello stesso tipo.

Il formato json: lo schema

Come per l'xml abbiamo associato l'xsd che ci dice tutti i vincoli associati ad un certo tag, analogamente per json abbiamo il json-schema. Si tratta di un file che ci dice le caratteristiche che devono avere gli oggetti json in un certo file json.

Es. consideriamo un catalogo di prodotti scritti in json:

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

Il formato json: lo schema

Es. consideriamo un catalogo di prodotti scritti in json:

```json
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

Esistono dei programmi che, dato un file json (es. prodotti.json) ed un file schema, ti dicono se il file prodotti.json è coerente con lo schema.

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "productId": {
      "description": "The unique identifier for a product",
      "type": "integer"
    }
  },
  "required": [ "productId" ]
}
```

Facciamo un esempio

Il Comune di Ladispoli e il Comune di Anzio si ritrovano è decidono che ciascuno pubblicherà nei confronti dell'altro un insieme di servizi per la consultazione dell'anagrafe. Il Comune di Anzio potrà richiedere al Comune di Ladispoli tutti i cittadini nati nel 2011, per esempio. Analogamente il Comune di Ladispoli potrà chiedere tutti i cittadini di Anzio che fanno di cognome 'Rossi'.

Entrambi i Comuni hanno le loro anagrafi su database. Ma le tabelle sono diverse. Es. il Comune di Anzio ha il campo indirizzo, non strutturato. Mentre Ladispoli ha TipoIndirizzo, Nome, NumeroCivico.

Per risolvere tutte le ambiguità i due Comuni decidono di scrivere uno schema json che condividono. Nello schema ci sono tutte le caratteristiche dell'Object Cittadino.

Facciamo un esempio

Entrambi i Comuni concordano che l'interrogazione può avvenire per Cognome, per Anno di Nascita e per Stato Civile. Il web service sarà allora del tipo

InterrogaAnagrafe(string sTipo, string sValore)

Un esempio di richiesta json è

```
{
"Tipo":"Cognome",
"Valore": "Rossi"
}
```

La risposta è un file json con tutti i cittadini che corrispondono alla richiesta effettuata.

Facciamo un esempio

Il web service del Comune ricevuta la richiesta, deve:

- tradurre la richiesta in una query SQL, considerando la struttura del suo DB.
- lanciare la query e ricevere la risposta
- scrivere la risposta in formato json, e validarla in base allo schema concordato tra i due Comuni
- inviare il file json al richiedente.

Questo fa si che i due comuni scambino dati senza che ciascuno si preoccupi della struttura interna dei dati dell'altro.

Proviamo ad implementare questo scenario in python!!!

Parser json in python

Python supporta json nativamente

```
import json
```

Serializzazione: è il processo di trasformazione di una struttura dati in un testo, es. json

Deserializzazione: è il processo inverso. L'input è una stringa contenente un testo in formato json. La deserializzazione trasforma tale stringa in una struttura dati.

Parser json in python: i dizionari

I dizionari sono insiemi di coppie <chiave,valore>. Mentre in una lista si accede ad un elemento con il suo progressivo o la sua posizione, in un dizionario ogni elemento ha una chiave univoc.

I dizionari di Python vengono definiti utilizzando una coppia di **parentesi graffe**

mioDict = {chiave:valore, ……,chiave:valore}

Esempio:

country_capitals = { "United States": "Washington", "Italy": "Rome", "England": "London"}

```
print(country_capitals["United States"])      # Washington
print(country_capitals["England"])            # London
```

Parser json in python: i dizionari

Nei python dictionary le chiavi devono essere immutabili, es. una stringa, oppure un numero. Non possono essere dei mutabili, es. una lista.

Posso avere, però, valori di tipo diverso:

thisdict = **{** "brand": "Ford",

"electric": False,

"year": 1964,

"colors": ["red", "white", "blue"]**}**

Ma anche chiavi di tipo diverso:

myDict = **{**1:"hello", "name":"John"**}**

Parser json in python: i dizionari

```
# add an item with "Germany" as key and "Berlin" as its value
country_capitals["Germany"] = "Berlin"
```

```
# delete it
del countr
```

Here are some of the commonly used dictionary methods.

Function	Description
pop()	Remove the item with the specified key.
update()	Add or change dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

Parser json in python

```python
# This is a sample Python script.
import json

def JsonSerialize(data):
    with open("data_file.json", "w") as write_file:
        json.dump(data, write_file)

def JsonDeserialize(sFile):
    with open(sFile, "r") as read_file:
        return json.load(read_file)

def print_dictionary(dData):
    for keys, values in dData.items():
        print(keys)
        print(values)


sFilePath = "/Users/andrea/Progetti/pythonMAC/pythonJson/prova.json"
data = JsonDeserialize(sFilePath)
```

L operazioni di serializzazione e deserializzazione in python fanno largo uso, anche se non esclusivo, dei dizionari.

Proviamo a vedere cosa succede se proviamo a serializzare strutture diverse da un dizionario.

Validare un json document con un json schema

In Python, possiamo usare la libreria jsonschema per validare un documento json:

$ pip install jsonschema

https://builtin.com/software-engineering-perspectives/python-json-schema#:~:text=from%20jsonschema%20import%20validate%20schema,error%2C%20the%20JSON%20is%20valid.

Web service REST

Cosa sono i web service? Siamo abituati al web come a dei server che ci mandano la pagina di un sito web. Nel caso dei web service invece il server web espone un servizio. Possiamo pensare ad una procedura o API, che si aspetta dei parametri in input e da un valore di ritorno.

Nei web service REST i dati di input ed i dati di ritorno sono comunicati usando il JSON.

Web service REST

Laddove i web service hanno come riferimento una certa risorsa, essi espongono delle API che servono per gestire tale risorsa.

HTTP method	Description
GET	Retrieve an existing resource.
POST	Create a new resource.
PUT	Update an existing resource.
PATCH	Partially update an existing resource.
DELETE	Delete a resource.

HTTP method	API endpoint	Description
GET	/customers	Get a list of customers.
GET	/customers/<customer_id>	Get a single customer.
POST	/customers	Create a new customer.
PUT	/customers/<customer_id>	Update a customer.
PATCH	/customers/<customer_id>	Partially update a customer.
DELETE	/customers/<customer_id>	Delete a customer.

A sinistra la descrizione generale dei metodi per gestire la risorsa.
A destra l'esempio di un web service associato ad un CRM, dove la risorsa sono i clienti di un'azienda.

Web service REST

Si parla di CRUD operations:

- **C**REATE
- **R**EA
- **U**PI
- **D**EL

HTTP method	Description
GET	Retrieve an existing resource.
POST	Create a new resource.
PUT	Update an existing reso
PATCH	Partially update an exis
DELETE	Delete a resource.

HTTP method	API endpoint	Description
GET	/customers	Get a list of customers.
GET	/customers/<customer_id>	Get a single customer.
POST	/customers	Create a new customer.
PUT	/customers/<customer_id>	Update a customer.
PATCH	/customers/<customer_id>	Partially update a customer.
DELETE	/customers/<customer_id>	Delete a customer.

Web service REST in python

In python i web service REST possono essere usati con la libreria **requests**.

```
$ python -m pip install requests
```

Il prossimo passo è quello di **consumare** un'API di un server esistente. Prendiamone uno già costruito per fare i test.

Web service REST in python

```
import requests

api_url = https://jsonplaceholder.typicode.com/todos/1
response = requests.get(api_url)
print(response.json())
print(response.status_code)
print(response.headers["Content-Type"])
```

Web service REST in python

Quello che abbiamo visto è un esempio essenziale di client REST, cioè consumer di un web service esistente.

Nel nostro esercizio abbiamo due Comuni italiani, ognuno dei quali richiede i servizi (le API) dell'altro, ma al tempo stesso pubblica le proprie API e quindi i propri service.

Dobbiamo allora imparare come si fa un server REST.

Richiamo di python: la variabile __name__

In python __name__ è una variabile speciale il cui contenuto è definito da python. La variabile __name__ vale **'__main__'** se siamo nel file passato come argomento a python. Altrimenti vale il nome del file dove si trova la funzione importata.

Attraverso la variabile main possiamo scrivere file python che contengono funzioni e un main. Questo main viene eseguito solo se l'utente lancia python con quel file come argomento.

(Vedi prova di questo in MAC/progetti/pythonMAc/pythonjson/main.py)

Web service REST in python: il server

Per implementare il nostro server REST usiamo la libreria Flask

$ pip install Flask

Flask definisce una classe. La creazione di una sua istanza prevede che venga passato il modulo corrente.

Qui accanto il codice di un semplice web service REST che implementa il metodo GET. Puoi provarlo anche con il browser.

```python
from flask import Flask, json
import prova_name as mn

companies = [{"id": 1, "name": "Company One"}, {"id": 2, "name": "Company Two"}]

api = Flask(__name__)

@api.route('/companies', methods=['GET'])
def get_companies():
  return json.dumps(companies)

if __name__ == '__main__':
    api.run(host="127.0.0.1", port=8080)
```

Web service REST in python: la post

Implementiamo anche il metodo POST.

Prima inviamo al server una nuova company che il server aggiungerà alla lista.

Dopo, usando il metodo GET, richiediamo le companies e verifichiamo che c'è anche quella appena aggiunta.

Programma client per la post

```
api_url = "http://127.0.0.1:8080/post_company"
data = {"id": 3, "name": "Company Three"}
response = requests.post(api_url,json=data)
#print(response.json())
print(response.status_code)
print(response.headers["Content-Type"])
```

Programma server per la post

```python
from flask import Flask, json, request
import prova_name as mn

companies = [{"id": 1, "name": "Company One"}, {"id": 2, "name": "Company Two"}]

api = Flask(__name__)

@api.route('/companies', methods=['GET'])
def get_companies():
  return json.dumps(companies)

@api.route('/post_company', methods=['POST'])
def process_json():
    print("Ricevuta chiamata")
    content_type = request.headers.get('Content-Type')
    print("Ricevuta chiamata " + content_type)
    if (content_type == 'application/json'):
        json = request.json
        print(json)
        companies.append(json)
        return json
    else:
        return 'Content-Type not supported!'

if __name__ == '__main__':
    api.run(host="127.0.0.1", port=8080)
```

PEP 8: E302 expected 2 blank lines, found 1

Reformat the file ⌥⇧⏎ More actions... ⌥⏎

Avviare il server da prompt di comandi. Poi lanciare il client che aggiunge la nuova company.

Con il browser digitare:
http://127.0.0.1/companies
E verificare che è stata aggiunta la nuova company.

Flask, messaggio di warning

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

Il messaggio ci avvisa che stiamo usando un ambiente server adatto per i test, quindi semplice da usare ma non efficiente. Da qui deriva la non adeguatezza in produzione.

In produzione bisogna usare un server differente al quale passeremo il codice Flask. Ma questo non è oggetto di questa lezione.

Ultimo mattone: interagire con un database

Un ultimo mattone manca per implementare la nostra applicazione: sapere interagire con un DBMS.

Nel nostro esempio useremo mySql all'interno del pacchetto XAMP.

Creeremo un DB per ognuno dei due Comuni.

Vedremo come, all'interno di un programma python possiamo fare delle query SQL per interagire con il DB.

Download e installazione di XAMP

Xamp è un programma che mette assieme un insieme di server tra i quali anche mysql server.

Ricordare che un server TCP è un programma che:

- si mette in ascolto su un determinato IP address e una porta TCP
- su tale network channel riceve le chiamate che arrivano dai client
- processa le chiamate e risponde ai client

Un server tipicamente è un'applicazione multithread, cioè ha più task che in parallelo processano le chiamate di più client generando le relative risposte.

Mysql è un server in ascolto su ip_local,3306. Riceve le chiamate in un apposito linguaggio che include l'SQL. Esegue le operazioni richieste su DB e genera la risposta per il client.

Creazione del database del Comune a partire da un dataset.

Per l'interazione con il database fare riferimento al progetto

db_client che sta in python_MAC

Per il dataset fare riferimento al file adult.data scaricato dal sito UCI MACHINE LEARNING.

Mettere tutto assieme

Ciascun Comune ha due script:

Comune_client.py

Comune_server.py

Il primo (Comune_client.py) sta in polling su una certa cartella, in base ad un file di configurazione. Se in quella cartella c'è un file allora lo usa per fare la richiesta e poi lo rinomina.

Il server sta in ascolto su un certo canale di rete e quando riceve una richiesta, interroga il DB, crea il json e manda la risposta.

ULTIMA SLIDE

RIFERIMENTI BIBLIOGRAFICI.

- .....
- .....
- .....

(FACOLTATIVO) Contatti email docente