Calendario

- Arquitectura web
- Requests
- Json
- Servicios web

Arquitectura web

1) Conceptos claves: Protocolo, ISP, IP, DNS, Dominios: https://www.youtube.com/watch?v=MwxMsaFFycg

2) Conceptos claves: TCP, Paquetes, Router: https://www.youtube.com/watch?v=aD_yi5VjF78

3) Conceptos claves: URL, HTTP, GET, POST, HTML, SSL: https://www.youtube.com/watch?v=1K64fWX5z4U

Requests: HTTP for Humans - GET

- Doc: http://docs.python-requests.org/en/master/. Una vez instalado e importado, *requests* permite que desde nuestro código podamos realizar fácilmente peticiones(requests), usando cada uno de los verbos HTTP (o sea, podemos consumir desde el código PY un servicio web, también podemos leer una página web, o enviar datos a determinado sitio).
- Antes de empezar a probarlo, tenemos que instalarlo. Usemos **pip**: *pip install requests* (quizá deban pasar las credenciales de un proxy)
- Chequeen que haya quedado (*pip list*) y luego crearemos el módulo:

```
import requests
r = requests.get('https://www.antel.com.uy')
print(r.status_code)
print(r.text)
```

- Hicimos un GET a la URL de Antel. SI todo fue bien, r será un objeto que quedará cargado con información relativa al request realizado.

```
print(r.status_code) >> 200 (OK)

print(r.text) >> El HTML de la página de Antel
```

Requests: HTTP for Humans - GET (2)

1) GET (un GET se puede probar siempre desde un browser, si devuelve HTML, el browser dibujará el sitio):

- Ya hicimos un GET a la página de Antel
- El protocolo HTTP permite que en los requests GET puedan enviarse datos. Podemos probarlo usando *httpbin.org* (un sitio que nos sirve para testear)
- Requests nos permite enviar los datos como un diccionario PY

  payload = {'key1': 'value1', 'key2': 'value2'}

  r = requests.get('https://httpbin.org/get',params=payload)

  Usando la propiedad **url** del objeto **r** (devuelto por *requests.get*) podemos ver la URL construida:

  print(r.url) #https://httpbin.org/get?key2=value2&key1=value1

Requests: HTTP for Humans - GET (3)

- Como vimos, para leer el body (o sea, el HTML) del HTTP respuesta

  print(r.text)

- Otras propiedades del objeto de cargado como respuesta (r) que pueden investigar:
- *encoding*
- *content*
- Tratamiento de imágenes: Ver biblioteca **Pillow**
- *https://pillow.readthedocs.io/en/5.3.x/handbook/tutorial.html*
- *Headers*. Son los cabezales del mensaje HTTP respuesta, se pueden obtener como un diccionario. Los cabezales vienen cargados de adicionales al body (que es el contenido HTML de las páginas)

Requests: HTTP for Humans - GET (4)

Ejemplo: *print(r.headers)*

Salida:

```
{
 'content-encoding': 'gzip',
 'transfer-encoding': 'chunked',
 'connection': 'close',
 'server': 'nginx/1.0.4',
 'x-runtime': '148ms',
 'etag':
'"e1ca502697e5c9317743dc078f67693f"',
 'content-type': 'application/json'
}
```

Requests: HTTP for Humans - GET (5)

- **Ejercicio:** Consumir un **recurso** expuesto por la plataforma abierta de **web services** de la IM (revisar la API en *https://github.com/dti-montevideo/servicios-abiertos/blob/master/ubicaciones.md*), con el fin de que, dado un string, se busquen todas las calles que en su nombre lo contengan.
- Ejemplo: Supongamos que nuestro programa reciba “agra”, entonces deberá responder una lista de calles como agraciada y josé luis villagran (entre otras), ya que contienen agra
- ¿¿No se entendió nada??!? ¿Qué es un recurso, qué es un web service, qué es una API?

Web Service

- Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
- Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes como Internet.
- La interoperabilidad se consigue mediante la adopción de estándares abiertos

API REST

- REST (Representational State Transfer) es uno de los estándares utilizados para implementar web services.
- Hace uso del protocolo HTTP
- Propone utilizar cada uno de sus verbos (GET, POST, PUT, DELETE, PATCH) para poder realizar diferentes operaciones entre el servicio web y el cliente (el consumidor).
- Una API sólo describe qué recursos (operaciones) provee un web service, y cómo deben utilizarse,

JSON

JSON: JavaScript Object Notation: Formato de **texto ("un Json" ES UN STRING, pero con un formato específico!!! NO olvidar)** para intercambio de datos. Es muy similar al tipo diccionario en Python. Veremos cómo usar Json en Python

```
import json

# pasar de un string (con formato json) a una variable Python

ejemplo_json = json.loads('{"ejemplo":["cadena", 1.0, 2]}')

print(type(ejemplo_json))

# pasar de una variable de PY (debe ser diccionario o lista

de diccionarios) a un string con formato json

data = {'nombre': 'Gustavo',

 'ci': '12345678'}

string_jsoneado= json.dumps(data)

# para manejar los encodings correctamente

string_jsoneado_utf8=
json.dumps(data,ensure_ascii=False).encode('utf8')
```

Requests: HTTP for Humans - POST

- *http://docs.python-requests.org/en/master/user/quickstart/#more complicated-post-requests*
- Ya vimos cómo hacer GETs, es hora de POST

2) POST. En estos ejemplos se mandan datos (clave-valor) a la URL indicadas.

a) Los datos se envían como diccionario

```
payload = {'key1': 'value1', 'key2': 'value2'}

r = requests.post("https://httpbin.org/post", data=payload)

print(r.text)
```

b) En este segundo ejemplo los datos se pasan directamente en Json (o sea, en un string)

```
import json

url = 'https://api.github.com/some/endpoint'

payload = {'some': 'data'}

r = requests.post(url, json=payload)
```

Ejecicio (consumiendo un web service REST)

- La idea es consumir con Request un recurso expuesto en la plataforma abierta de web services de la IM. Debemos escribir un programa que pida un string y devuelva todas las calles que en su nombre contengan dicho string
- Para implementarlo, el programa debe consumir el recurso http://www.montevideo.gub.uy/ubicacionesRest/calles/?nombre=<substring_a_buscar> y procesar los datos devueltos (devuelve Json, pero solo nos interesa que se devuelva el nombre de la calle). En caso de no encontrarse, dar un mensaje acorde
- Ejemplo: Supongamos que nuestro programa reciba “agra”, entonces deberá responder una lista de calles como agraciada y josé Luis villagran (entre otras), ya que contienen "agra"
- En este caso, éste es el recurso a consumir con requests: http://www.montevideo.gub.uy/ubicacionesRest/calles/?nombre=agra (Prueben esa URL desde un browser)

  > Ingrese (sub)string a buscar: agra

  > Salida:

  Av. Agraciada

  José Luis Villagrán

  ...
- La API completa de la IM puede ser chequeada en:https://github.com/dti-montevideo/servicios-abiertos/blob/master/ubicaciones.md