[*Escuela de código*]

Apuntes de clase:

JSON y ETL

Tabla de contenido

JSON

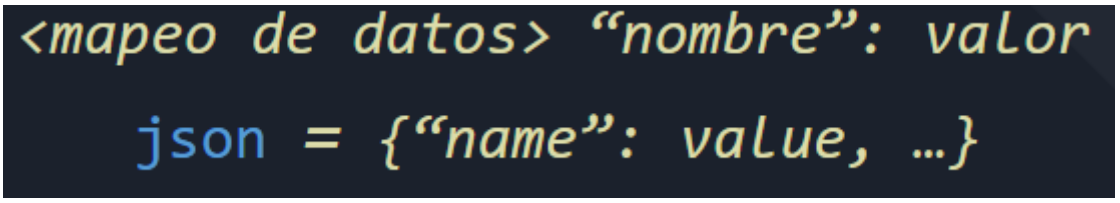
JSON (JavaScript Object Notation) es una sintaxis para el almacenamiento e intercambio de datos. Es un formato liviano utilizado para el intercambio de información que poco a poco fue desplazando a otros formatos como XML.

Hoy en día es imposible no cruzarnos con este tipo de formato de datos ya que se encuentra en todo el manejo e intercambio de información de páginas Webs, de APIs y de almacenamiento de información en bases NoSQL como aquellas del tipo "documento" (como MongoDB).

Estructura

JSON está compuesto por una colección de pares "name/value" separados por un limitador (como puede ser la coma). Dicho de otra manera, JSON se comporta exactamente igual que los diccionarios que venimos utilizando en Python en donde podemos tener datos tipo "key/value" separados por coma:

```
json_1 = {"Nombre": "Max", "Apellido": "Power"}
```



Las principales diferencias con los diccionarios son:

- Las claves "keys" de los dict en JSON se las llaman "name".
- Todos los "name" de JSON son del tipo string, no puede haber claves numéricas como en los diccionarios.
- Todos los string de JSON van con doble comillas **" "**, JSON no utiliza el concepto de simple comilla **' '** para los string.

En el uso JSON es muy común "anidar" listas o objetos generando una estructura más compleja de datos. Veamos un ejemplo:

```
json_2 = {
          "Nombre': "Max",
          "Apellido': "Power",
          "hijos": [
                {
                 "firstName": "Alice",
                 "age": 6
                },
                {
                 "firstName": "Bob",
                 "age": 8
                }
                ]
          }
```

JSON en Python

Python soporta el formato JSON de forma nativa, para poder realizar operaciones de JSON en Python debemos importar el siguiente módulo:

```
import json
```

Este módulo nos permitirá leer JSON desde archivos o desde una URL (API). La dualidad entre los nombres de los tipos de datos de JSON y Python son:

JSON	Python
object	dict
array	list
string	str
number(int)	int
number(float)	float
true	True
false	False
null	None

JSON Serialize

El método "dump" y "dumps" se utiliza para serializar un objeto JSON en una cadena de datos que pueden ser escritas "write" a un archivo o medio.

Comencemos utilizar inicialmente el método "dumps" para serializar el objeto JSON en un json_string:

```
emma_json = {
            "nombre": "Emma",
            "apellido": "Thompson",
            "hijos": []
            }
print('Imprimir json como un objeto')
print(emma_json)

print('Convertir JSON a json_string e imprimir en pantalla')
json_string = json.dumps(emma_json, indent=4)
print(json_string)
```

```
Imprimir json como un objeto
{'nombre': 'Emma', 'apellido': 'Thompson', 'hijos': []}

Convertir JSON a json_string e imprimir en pantalla
{
    "nombre": "Emma",
    "apellido": "Thompson",
    "hijos": []
}
```

Se puede ver que el hecho de utilizar "print" con el objeto cruzó "emma_json" en realidad se imprime como cualquier otro diccionario. Pero al utilizar el método "dumps" para convertir el objeto en un json_string ahora podemos visualizar correctamente los datos en el formato acorde, por ejemplo, los strings en el json_string tiene doble comillas **" "** y fue posible respetar el formato de indexado gracias a "indent=4".

```
json.dumps = (json_data, indent=4}
```

Ahora demostraremos el uso de "dump" con archivos, pero tambíen se puede utilizar para serializar datos para enviar a una página Web o servicio.

```
with open('mi_json.json', 'w') as jsonfile:
    json.dump(data, jsonfile, indent=4)
```

De la misma forma que trabajamos en el pasado con otros formatos de archivos, utilizamos la función "open" para abrir el archivo (en este caso con 'w' en modo de escritura) y bajamos los datos al archivo utilizando "**dump**".

```
json.dump = (json_data, file, indent=4}
```

JSON Deserialize

El método "load" se utiliza para deserializar un objeto JSON proveniente de un archivo, un request HTTP o API o WebApp en un dato objeto tipo JSON.

En este ejemplo lo veremos con archivos pero aplica a **request** HTTP o datos de telemetría enviados desde una API.

```
with open('mi_json.json', 'r') as jsonfile:
    json_data = json.load(jsonfile)
```

```
json_data = json.dump = (file)
```

JSON HTTP Request

La utilidad más importante del formato JSON es el envío de información mediante la nube con HTTP request que se utiliza para el manejo de información entres las páginas webs, entres las APIs y las WebApp.

Para poder hacer uso de "request" y poder consultar un servicio por los datos debemos incluir el siguiente módulo:

```
import requests
```

Para poner a prueba el módulo "requests" consultarlos la siguiente URL preparada para realizar pruebas con JSON:

"https://jsonplaceholder.typicode.com/todos/1"

```
response = requests.get("https://jsonplaceholder.typicode.com/todos/1")
# Se puede obtener el objeto JSON de dos formas distintas
data = json.loads(response.text)
data = response.json()
print('Imprimir los datos traídos de la nube')
print(json.dumps(data, indent=4))
```

Respuesta:

```
Imprimir los datos traídos de la nube
{
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
}
```

Otro ejemplo más complejo de datos que se reciben de un request:

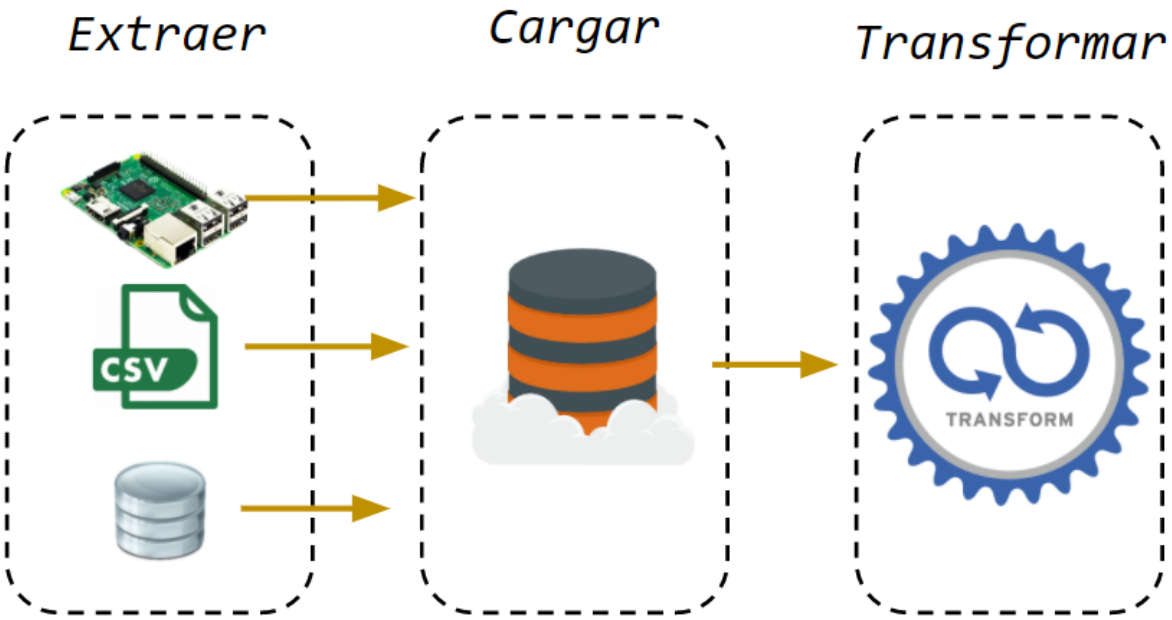
"https://jsonmock.hackerrank.com/api/transactions/search?txnType=debit&page=1"


```
{
 "page":"1",
 "per_page":10,
 "total":155,
 "total_pages":16,
 "data":[{
          "id":1,
          "userId":1,
          "userName":"John Oliver",
          "timestamp":1549536882071,
          "txnType":"debit",
          "amount":"$1,670.57",
          "location":{
                      "id":7,
                      "address":"770, Deepends, Stockton Street","city":"Ripley",
                      "zipCode":44139
                      },
          "ip":"212.215.115.165"
         },
```


En este ejemplo los datos se encuentran dentro de una lista de nombre "data", y hay un diccionario por cada usuario y sus datos.

Procesos ETL

El proceso ETL es un proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos.



Es un método potente que puede trabajar junto a herramientas de gestión e integración de datos. Ejemplos de uso:

- Recolección y fusión de datos desde proveedores o partners externos.
  - Dispositivos conectados a internet (IoT)
  - Archivos
  - Bases de datos
  - APIs
- Integración de nuevas fuentes de datos como social media, videos, dispositivos conectados a internet de las cosas, entre otras.
- Migración de datos desde sistemas legacy con formatos de datos distintos.
- Consolidación de datos como consecuencia de una fusión empresarial.

Extraer, Transformar y Cargar

- Extracción (**Extract**): La información es extraída desde todas nuestras fuentes de datos, sean estas bases de datos relacionales, XML, o ficheros no estructurados. El volumen de datos extraídos, así como el intervalo de tiempo entre extracciones, depende de las necesidades y requisitos del negocio.
- Transformación (**Transform**): Se analizan los datos extraídos para luego transformarlos en el formato deseado manteniendo su integridad, y llevando a cabo operaciones como validación, cálculos, codificación, filtrado, remoción de duplicados.
- Carga (**Load**): Se cargan los datos en un formato consistente y homogéneo en el almacén de destino, generalmente el Data Warehouse. Los datos están listos para ser explotados y convertidos en conocimiento.

Ingeniero de Datos

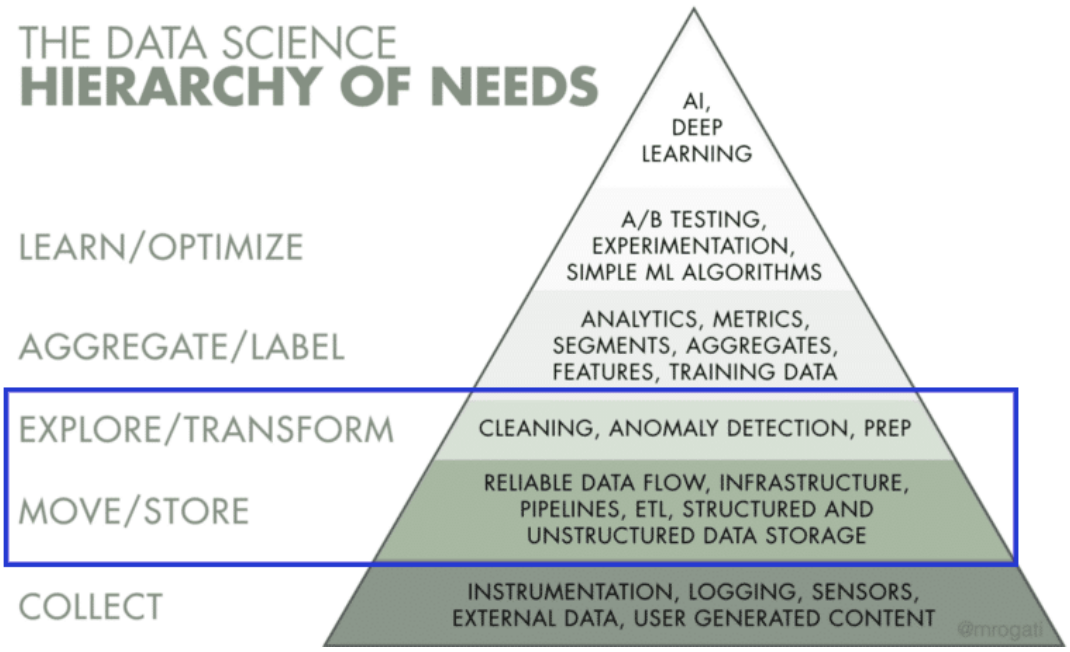
El Ingeniero de Datos (Data Engineer) es el encargado de crear la interfaz y mecanismo para el permitir el acceso y el flujo de la información. Se dedica principalmente al manejo de bases de datos, procesos de ETL, mantenimiento de los sistemas y desarrollo de APIs y microservicios que garanticen el flujo de información.

El ingeniero de datos es aquel que prepara la infraestructura para el consumo de la información, transformación y carga a fin de dejar preparada la información para que un Data Science la intérprete o un sistema de inteligencia artificial la consuma.

Inteligencia de Negocios

El Data Engineer forma parte del equipo de inteligencia de negocio (Business Intelligence - BI) junto con el Data Science y el Machine Learning Engineer.

En la pirámide del manejo de información, el Data Engineer es parte de los cimientos ya que es el encargado de garantizar el flujo de datos:



Data Pipeline

Un "Data Pipeline" es una suma de herramientas y procesos para la integración de información. Captura datos de múltiples fuentes y los une o inserta en una base de datos, aplicación o herramienta que luego pueda ayudar al equipo de inteligencia de negocios (BI).

Construir "Data Pipeline" es la responsabilidad principal de un Data Engineer, requiere manejo de muchas herramientas distintas y diseñar programas para que crezcan y escalen por su cuenta y automaticen tareas.

Un tipo de "Data Pipeline" son los procesos ETL, son el tipo de pipeline genérico por excelencia ya que siempre habrá involucrado un proceso de extracción y transformación. A este pipeline se puede sumarle procesos de inteligencia artificial, consultas, etc.

Datetime

Datetime es un módulo nativo de Python para poder realizar operaciones aritméticas con fechas y tiempos sin necesidad de tener que realizar complicados procedimientos para calcular la diferencia entre dos horas diferentes o fechas.

Datetime permite realizar cualquier tipo de operación lógica y además convertir la información o el resultado en la base de tiempo que necesitemos (días, horas, segundos, etc). Lo primero es importar el módulo datetime en nuestro programa para poder utilizarlo:

```
from datetime import datetime
```

La forma más usual de construir un objeto datetime son las siguientes:
Utilizando el constructor:

```
date1 = datetime(year=2019, month=6, day=15, hour=20, minute=45, second=25)
```

Convirtiendo una fecha en formato string a objeto datetime:

```
date2_str = '2019-06-16 20:45:25'
date2 = datetime.strptime(date2_str, '%Y-%m-%d %H:%M:%S')
```

Código	Valor	Descripción
%Y	2019	Años, representado en 4 dígitos
%m	06	Meses, 2 dígitos
%d	16	Días, 2 dígitos
%H	20	Horas, 2 dígitos en formato 24hs
%M	45	Minutos, 2 dígitos
%S	25	Segundos, 2 dígitos

Operaciones

Supongamos que tenemos dos objetos datetime creados cada uno con una fecha distinta:
Podemos realizar cualquier operación lógica (suma, resta) pero lo más usual es medir el tiempo entre dos fechas

```
time_diff = date2 - date1
```

Luego de esa diferencia se puede obtener cuantos segundos en total representa ese tiempo "timedelta":

```
print('Diferencia de segundos:', time_diff.total_seconds())
print('Diferencia de minutos:', time_diff.total_seconds() / 60)
```

Si quisiéramos podríamos convertir la fecha date1, sus horas minutos y segundos a segundos simplemente de la siguiente forma:

```
date1_seconds = date1.hour*3600 + date1.minute*60 + date1.second
```

Si deseara podría re convertir el objeto date1 en formato string en el formato que me sea más conveniente, por ejemplo:

```
date1_str = date1.strftime('%d-%m-%Y | %H:%M')
```

Time since epoch

¿Cómo funciona el sistema de tiempos?
El objeto datetime en el fondo lo que está haciendo es contar cuantos segundos pasaron desde el "epoch", en **Unix time** el epoch se considera el 1-Enero-1970 y con ello se construye cuando segundos pasaron desde ese día y se construye lo que se llama el UTC time o el timestamp.

Datetime posee un método "timestamp" que calcula y devuelve los segundos pasados desde el epoch time hasta la fecha indicada en el objeto datetime.

Descargar un repositorio de GitHub

Para poder realizar las actividades de aquí en adelante debemos tener instalado y configurado nuestro GitHub. Todos los ejemplos prácticos estarán subidos al repositorio GitHub de **InoveAlumnos**, para aprender como descargar estos ejemplos desde el repositorio referirse al "Instructivo de GitHub: Descargar un repositorio" disponible entre los archivos del campus. De no encontrarse allí, por favor, tenga a bien comunicarse con alumnos@inove.com.ar para su solicitud.

Debemos descargar el repositorio que contiene los ejemplos de clase de ésta unidad:
https://github.com/InoveAlumnos/json_etl_python

Hasta la próxima!

Con esto finaliza el tema "JSON y XML", a partir de ahora tienen las herramientas para poder comenzar a consumir APIs!.

Si desean conocer más detalles sobre el contenido pueden iniciar un tema de discusión en el foro del campus, o visitar los "Links de interés" que se encuentran al final de este apunte.

Links de interés

- JSON página oficial
- Python JSON Parser
- Ejemplos JSON y Python
- Página ensayar JSON request
- Python XML Parser
- Datetime documentación oficial
- Datetime ejemplos Python