

# Estructuras de Datos VI

## JSON en Python

Jonatan Gómez Perdomo, Ph. D.

[jgomezpe@unal.edu.co](mailto:jgomezpe@unal.edu.co)

Arles Rodríguez, Ph.D.

[aerodriguezp@unal.edu.co](mailto:aerodriguezp@unal.edu.co)

Camilo Cubides, Ph.D. (c)

[eccubidesg@unal.edu.co](mailto:eccubidesg@unal.edu.co)

Carlos Andrés Sierra, M.Sc.

[casierrav@unal.edu.co](mailto:casierrav@unal.edu.co)

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)

Computer and System Department

Engineering School

Universidad Nacional de Colombia

# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Definición

**JSON (JavaScript Object Notation):** Es un estándar utilizado para el intercambio de información entre aplicaciones. Es un formato de archivo de texto que es fácil de leer tanto por humanos como por máquinas.

JSON está basado estructuralmente en el lenguaje de programación JavaScript y por su definición simple, se puede usar en diferentes lenguajes de programación.



# Estructura

Un objeto JSON es un diccionario, donde para cada ítem:

- La clave debe ser una cadena de caracteres delimitada por comillas dobles ("), y
- El campo del valor puede ser un número (no se distingue entre enteros, reales u otros), un valor de verdad (en JavaScript se usan las palabras reservadas `true` y `false`), una cadena de caracteres (usando comillas dobles como delimitador), un elemento nulo (`null`), o una lista de estos mismos elementos o nuevamente un objeto JSON (un diccionario con estas propiedades —definido recursivamente—).



# Ejemplo de un objeto JSON

## Ejemplo

```
{  
  "Nombre": "Douglas",  
  "Apellido": "Crockford",  
  "pasatiempos": ["trotar", "bucear", "cantar"],  
  "edad": 64,  
  "empleado": false,  
  "jefe": null,  
  "hijos": [  
    {"Nombre": "Alice", "edad": 16},  
    {"Nombre": "Bob", "edad": 8}  
  ]  
}
```



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Diferencias

- 1 No todo diccionario en Python es un objeto JSON: Los diccionarios en Python pueden tener como claves números, cadenas delimitadas por el apóstrofe o tuplas. Los objetos JSON sólo permite cadenas de caracteres delimitadas únicamente por comillas dobles.
- 2 Un objeto JSON es “casi” un diccionario en Python, diccionario que usa las palabras `false`, `true`, `null` y no las de Python `True`, `False`, `None`.
- 3 Es claro, que muchos objetos JSON se pueden escribir directamente en Python, pero no todos y viceversa.



# Ejemplo de un objeto JSON en Python I

## Ejemplo

El objeto JSON:

```
{
  "Nombre": "Douglas",
  "Apellido": "Crockford",
  "pasatiempos": ["trotar", "bucear", "cantar"],
  "edad": 64,
  "empleado": false,
  "jefe": null,
  "hijos": [
    {"Nombre": "Alice", "edad": 16},
    {"Nombre": "Bob", "edad": 8}
  ]
}
```



# Ejemplo de un objeto JSON en Python II

## Ejemplo (continuación)

Se puede escribir como el diccionario de Python así:

```
{
    "Nombre": "Douglas",
    "Apellido": "Crockford",
    "pasatiempos": ["trotar", "bucear", "cantar"],
    "edad": 64,
    "empleado": False,
    "jefe": None,
    "hijos": [
        {"Nombre": "Alice", "edad": 16},
        {"Nombre": "Bob", "edad": 8}
    ]
}
```

# Correspondencias

La siguiente tabla presenta las correspondencias entre los tipos de un objeto JSON y los tipos dentro de un diccionario Python.

Tipo en Python	Tipo en JSON
dict	object
tuple, list	array
str	string
int, float	number
False	false
True	true
None	null

Table: Correspondencia de tipos entre Python y JSON.



# Python soporta JSON

Python viene con un módulo llamado `json` que permite automatizar la tarea de procesar información en este formato, éste se invoca mediante la instrucción

```
import json
```



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 **Serialización**
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Serialización

El proceso de transformar datos en series de bytes para ser enviados por una red o ser guardados como archivo se conoce como serialización. Los archivos JSON se pueden serializar. La librería `json` expone el método `dump()` para escribir datos en un archivo. Si se tiene el siguiente diccionario

```
data = {  
    "cientifico": {  
        "nombre": "Alan Mathison Turing",  
        "edad": "41"  
    }  
}
```



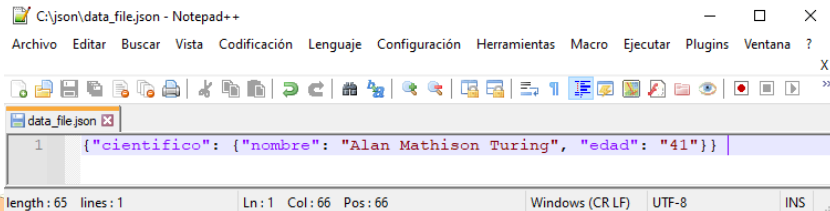
# Serialización en archivo

El objeto data se puede serializar en archivo así

```
with open("json/data_file.json", "w") as write_file:  
    json.dump(data, write_file)
```



Se creará el siguiente archivo



# Serialización a texto

El objeto data se puede asignar a un string de la siguiente manera

```
json_string = json.dumps(data)  
print(json_string, type(json_string))
```



El programa genera como salida:

```
{"cientifico": {"nombre": "Alan Mathison Turing", "edad": "41"}}  
<class 'str'>
```



# Argumentos de la función dump

Es posible especificar el tamaño de la indentación para estructuras anidadas. El siguiente programa establece que la indentación se realiza a 4 espacios y se imprime en forma de árbol expandido hacia la derecha.

```
json_string = json.dumps(data, indent=4)
print(json_string)
```



El programa imprime como salida:

```
{
    "cientifico": {
        "nombre": "Alan Mathison Turing",
        "edad": "41"
    }
}
```





# Agenda

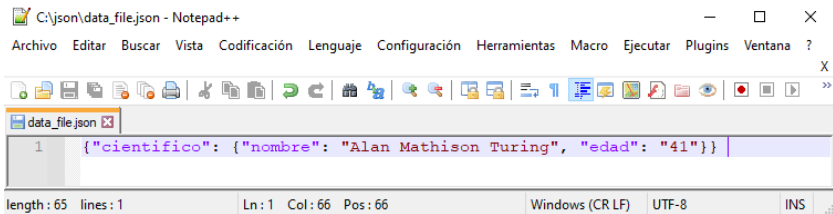
- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización**
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Deserialización desde archivo I

Es posible cargar un archivo de JSON desde un archivo de la siguiente manera

Para el archivo `data_file.json` ubicado en la carpeta `json`



The screenshot shows a Notepad++ window titled "C:\json\data\_file.json - Notepad++". The menu bar includes Archivo, Editar, Buscar, Vista, Codificación, Lenguaje, Configuración, Herramientas, Macro, Ejecutar, Plugins, and Ventana. The toolbar contains various icons for file operations and editing. The text area shows a single line of JSON code: `{"cientifico": {"nombre": "Alan Mathison Turing", "edad": "41"}}`. The status bar at the bottom indicates "length: 65", "lines: 1", "Ln: 1", "Col: 66", "Pos: 66", "Windows (CR LF)", "UTF-8", and "INS".

```
1 {"cientifico": {"nombre": "Alan Mathison Turing", "edad": "41"}}
```

length: 65 lines: 1 Ln: 1 Col: 66 Pos: 66 Windows (CR LF) UTF-8 INS



# Deserialización desde archivo II

El siguiente código carga el objeto JSON a la variable data y luego imprime uno de los registros del objeto JSON data, así

```
with open("json/data_file.json", "r") as read_file:  
    data = json.load(read_file)  
  
print(data["cientifico"])
```



La salida obtenida es

```
{'nombre': 'Alan Mathison Turing', 'edad': '41'}
```



# Deserialización desde texto

Es posible cargar un archivo de JSON desde un string. En el siguiente código se crea un string (en dos líneas por espacio, para esto se encierra entre tres comillas simples ''' ), luego se carga el objeto JSON a la variable data y por último ésta se imprime

```
json_string = '''{"cientifico":  
{"nombre": "Alan Mathison Turing", "edad": "41"}}'''  
data = json.loads(json_string)  
print(data)
```



La salida del programa es



```
{'cientifico': {'nombre': 'Alan Mathison Turing', 'edad': '41'}}
```



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Conversión de archivos JSON a estructuras de Python I

Como se observaba en la tabla de correspondencias, JSON maneja `null` en vez de `None`, `false` en vez de `False` y `true` en vez de `True`. Al importar el código con la librería `json` se realizan las conversiones internas para poder trabajar cómodamente en Python. Así como se puede apreciar en el código presentado en la siguiente diapositiva.

Para imprimir se utiliza la función `pprint` del módulo `pprint`, con el fin de que el resultado aparezca impreso de forma bonita (*Pretty Print*).



# Conversión de archivos JSON a estructuras de Python II

## Ejemplo

```
import json
from pprint import pprint
strjson = '''{
"boolean1": null,
"diccionario": {"papa": 2000, "arroz": 5000},
"intValue": 0, "myList": [],
"myList2":["info1", "info2"],
"littleboolean": false, "myEmptyList": null,
"text1": null, "text2": "hello", "value1": null,
"value2": null}'''
data = json.loads(strjson)
pprint(data)
```



# Conversión de archivos JSON a estructuras de Python III

## Ejemplo (continuación)

La salida del programa es

```
{'boolean1': None,  
  'diccionario': {'arroz': 5000, 'papa': 2000},  
  'intValue': 0,  
  'littleboolean': False,  
  'myEmptyList': None,  
  'myList': [],  
  'myList2': ['info1', 'info2'],  
  'text1': None,  
  'text2': 'hello',  
  'value1': None,  
  'value2': None}
```



# Conversión de archivos JSON a estructuras de Python IV

## Ejemplo

Si se ejecutan las siguientes consultas a la variable data

```
print(data["text2"], type(data["text2"]))
print(data["text1"], type(data["text1"]))
print(data["intValue"], type(data["intValue"]))
print(data["myList"], type(data["myList"]))
print(data["myList2"], type(data["myList2"]))
print(data["diccionario"], type(data["diccionario"]))
print(data["myList2"][1])
print(data["diccionario"]["papa"])
```



# Conversión de archivos JSON a estructuras de Python V

## Ejemplo (continuación)

El resultado obtenido es:

```
hello <class 'str'>
None <class 'NoneType'>
0 <class 'int'>
[] <class 'list'>
['info1', 'info2'] <class 'list'>
{'papa': 2000, 'arroz': 5000} <class 'dict'>
info2
2000
```



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet**
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Usando request para leer archivos JSON desde internet I

Existe un recurso en línea gratuito llamado [JSONPlaceholder](#) para practicar peticiones que se pueden realizar en formato JSON. Para realizar una petición se utiliza la librería request así

```
import json
import requests
```

Para leer un archivo JSON de una url específica, se utiliza la instrucción

```
response = request.get(url)
```

y se carga con

```
json.loads(response.text)
```



# Usando request para leer archivos JSON desde internet II

## Ejemplo

En el siguiente código se carga un JSON de la url

<https://jsonplaceholder.typicode.com/todos>

```
import json
import requests
from pprint import pprint

response = requests.get(
    "https://jsonplaceholder.typicode.com/todos")
pendientes = json.loads(response.text)

#imprime el json cargado
pprint(pendientes)
```



# Analizando la estructura de un JSON I

Al procesar los registros del JSON de pendientes cargados se observa que cada registro posee un usuario `userId`, un identificador de pendiente `id`, un título de tarea específico `title` y un estado `completed` que indica si la tarea ha sido realizada o no.

Con la siguiente instrucción se obtienen los dos registros iniciales del JSON:

```
pendientes[:2]
```



# Analizando la estructura de un JSON II

La salida observada de la instrucción `pendientes[:2]` es:

```
[{'completed': False,  
  'id': 1,  
  'title': 'delectus aut autem',  
  'userId': 1},  
 {'completed': False,  
  'id': 2,  
  'title': 'quis ut nam facilis et officia qui',  
  'userId': 1}]
```

¿Qué se obtiene si se ejecuta la siguiente instrucción?

```
pendientes[-2:]
```



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON**
- 8 Validación de archivos con formato JSON
- 9 Ejercicios





# Saber cuantas tareas ha completado un usuario

Es posible observar que hay varios usuarios cada uno con un identificador único y cada tarea tiene un estado que indica si la tarea se completó o no. Procesaremos este archivo para obtener cuantas tareas ha completado cada usuario y establecer los usuarios que más tareas han completado.

Esto se puede hacer en tres fases:

- 1 Contar cuántas tareas han completado los usuarios.
- 2 Ordenar el conteo de tareas pendientes que se han completado.
- 3 Escoger los usuarios que tienen el mismo número de tareas máximo.



# Contar tareas completadas por usuario

La primera parte lleva un conteo de tareas completadas por usuario:

```
pendientes_por_usuario= {}  
  
# Lleva un conteo de los pendientes  
# que ha completado cada usuario  
for pendiente in pendientes:  
    if pendiente["completed"]:  
        if pendiente["userId"] in pendientes_por_usuario:  
            pendientes_por_usuario[pendiente["userId"]] += 1  
        else:  
            pendientes_por_usuario[pendiente["userId"]] = 1
```



# Ordenar por número de pendientes completos

La segunda parte ordena el diccionario de pendientes por usuario por su conteo de tareas\*:

```
# ordena por id los usuarios
items_ordenados = sorted(pendientes_por_usuario.items(),
key=lambda x: x[1], reverse=True)

maximas_tareas_completadas = items_ordenados[0][1]
```



\*En este ejemplo se usa una de las características más importantes de la programación funcional en Python: **Notación lambda**. Este tema esta fuera del alcance de este curso.



# Escoger los usuarios que tienen el número de tareas máximo

La tercera parte obtiene las personas con el número de tareas máximas

```
usuarios = []  
for usuario, num_tareas_completas in items_ordenados:  
    if num_tareas_completas == maximas_tareas_completadas:  
        usuarios.append(str(usuario))  
    else:  
        break  
  
usuarios_con_max = " y ".join(usuarios)  
print("los usuarios", usuarios_con_max)  
print("han completado ", maximas_tareas_completadas, "tareas")
```



La salida de este código completo es la siguiente

```
los usuarios 5 y 10  
han completado 12 tareas
```



# Filtrar usuarios con el máximo número de pendientes

Es posible filtrar las tareas de los usuarios que han hecho la mayor cantidad de pendientes completados y escribirlas en un archivo json. Para esto nos apoyaremos de la función `filter` que determina a través de una función que retorna un booleano si incluir al elemento en la lista de salida o no

```
def filtro(pendiente):  
    esta_completa = pendiente["completed"]  
    esta_en_el_maximo_conteo = str(pendiente["userId"]) in usuarios  
    return esta_completa and esta_en_el_maximo_conteo  
  
with open("json/tareas_filtradas.json", "w") as archivo_salida:  
    tareas_filtradas = list(filter(filtro, pendientes))  
    json.dump(tareas_filtradas, archivo_salida, indent=2)
```



# Filtrar usuarios con el máximo número de pendientes

Una parte de la salida anterior nos muestra tareas con `completed=True`

```
[{'completed': True, 'id': 81, 'title': 'suscipit qui totam', 'userId': 5},  
 {'completed': True,  
  'id': 83,  
  'title': 'quidem at rerum quis ex aut sit quam',  
  'userId': 5},  
 {'completed': True, 'id': 85, 'title': 'et quia ad iste a', 'userId': 5},  
 {'completed': True, 'id': 86, 'title': 'incidunt ut saepe autem',  
  'userId': 5},  
 {'completed': True,  
  'id': 87,  
  'title': 'laudantium quae eligendi consequatur quia et vero autem',  
  'userId': 5},  
 {'completed': True, 'id': 89, 'title': 'sequi ut omnis et', 'userId': 5},  
 ...
```



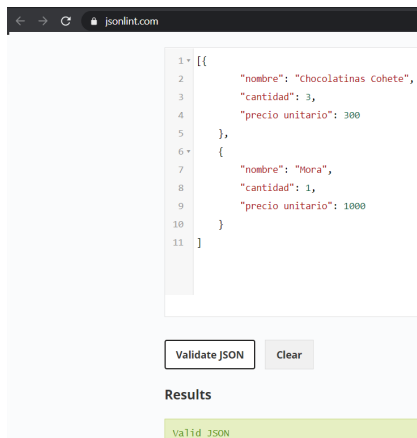
# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Validación de archivos con formato JSON

Es posible validar un archivo de JSON utilizando la página web <https://jsonlint.com/>





# Sugerencia

Se sugiere consultar un manual de Python o de sus librerías para determinar si ya existe un método para lo que se quiera realizar con JSON.



# Agenda

- 1 JSON
- 2 JSON y los diccionarios de Python
- 3 Serialización
- 4 Deserialización
- 5 Conversión de archivos JSON a estructuras de Python
- 6 Leer JSON desde Internet
- 7 Filtrar información de un archivo JSON
- 8 Validación de archivos con formato JSON
- 9 Ejercicios



# Ejercicios I

## Problemas

Para el JSON con la estructura mostrada

```
{
  "jadiazcoronado":{
    "nombres": "Juan Antonio",
    "apellidos": "Díaz Coronado",
    "edad":19,
    "colombiano":true,
    "deportes":["Fútbol","Ajedrez","Gimnasia"]
  },
  ...
  "dmlunasol":{
    "nombres": "Dorotea Maritza",
    "apellidos": "Luna Sol",
    "edad":25,
    "colombiano":false,
    "deportes":["Baloncesto","Ajedrez","Gimnasia"]
  }
}
```

# Ejercicios II

## Problemas

Cree un programa que lea de un archivo con dicho JSON y

- 1 Imprima los nombres completos (nombre y apellidos) de las personas que practican el deporte ingresado por el usuario.
- 2 Imprima los nombres completos (nombre y apellidos) de las personas que estén en un rango de edades dado por el usuario.
- 3 Cree un JSON de deportes como sigue:

```
{  
  "Ajedrez": ["jadiazcoronado", ..., "dmlunasol"],  
  "Futbol": ["jadiazcoronado", ...],  
  "Gimnasia": ["jadiazcoronado", ..., "dmlunasol"],  
  ...  
  "Baloncesto": [..., "dmlunasol"]  
}
```



# Ejercicios III

## Problemas

- 4 Desarrolle un programa que lea dos archivos JSON, y encuentre los componentes clave:valor que son iguales en ambos. Genere un nuevo archivo JSON con las coincidencias exactas entre los dos archivos.
- 5 Desarrolle un programa que lea un archivo JSON, en el cual se encuentran las notas de los estudiantes del curso. Cada llave corresponde al código de cada estudiante, y su valor es una lista con las notas obtenidas en las actividades del curso. Se debe generar un nuevo archivo JSON que para uno de los estudiantes solo guarde el promedio de las notas obtenidas.



# Ejercicios IV

## Problemas

- 6 Desarrollar un programa que lea un archivo JSON que contiene una serie de cadenas de caracteres en minúscula, cada una con su propia llave. Estas llaves tienen una codificación, a forma de encriptación, en donde las vocales están descritas como otros símbolos: \$ en vez de a, # en vez de e, \* en vez de i, ¬ en vez de o, y + en vez de u. Una vez leído el archivo, realice una desencriptación de todas las cadenas, es decir, convierta los símbolos a sus vocales correspondientes (si la cadena de entrada es "h¬l\$", la cadena resultante sería "hola"), y guarde el resultado en un nuevo archivo JSON.

