

Programación de Computadores

Archivos JSON en Python

Jonatan Gómez Perdomo, Ph. D.

`jgomezpe@unal.edu.co`

Arles Rodríguez, Ph.D.

`aerodriguezp@unal.edu.co`

Camilo Cubides, Ph.D. (c)

`eccubidesg@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

Contenido

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



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. Está basado en un subconjunto del 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 llave es una cadena de caracteres delimitada por comillas dobles ("), y
- El 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, recursión!!!).



Ejemplo

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



Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



Diferencias

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 solo cadenas de caracteres delimitadas por comillas dobles.

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`. Es claro, que muchos objetos JSON se pueden escribir directamente en Python, pero no todos y viceversa.



Ejemplo

El objeto JSON:

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



Ejemplo

Se puede escribir como el diccionario Python:

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



Correspondencias

La siguiente tabla presenta las correspondencias entre un objeto JSON y un diccionario Python.

Python	JSON
dict	object
tuple, list	array
str	string
int, float, long	number
False	false
True	true
none	null

Tabla: Type correspondence between JSON and Python.



Python soporta JSON

Python viene con un paquete llamado json para procesar información en este formato:

```
import json
```



Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



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`:

```
data = {  
    "presidente": {  
        "nombre": "Ivan Duque",  
        "edad": 44  
    }  
}
```

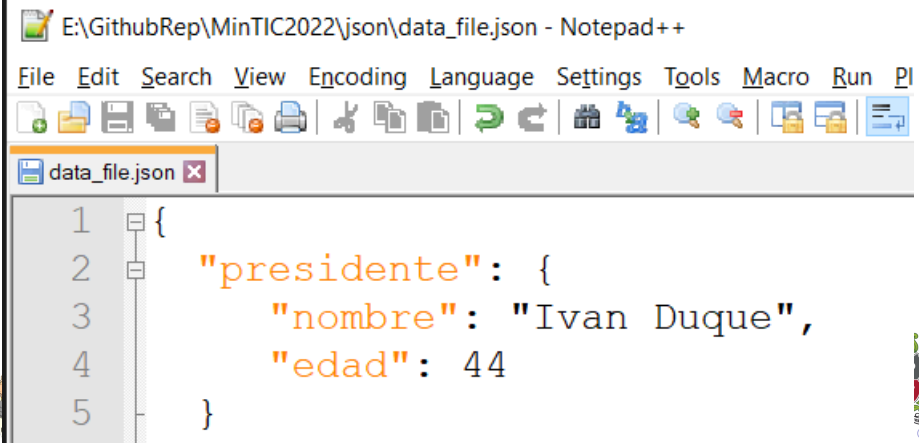


Serialización a archivo

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:



E:\GithubRep\MinTIC2022\json\data_file.json - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Pl

data_file.json

```
1 {  
2     "presidente": {  
3         "nombre": "Ivan Duque",  
4         "edad": 44  
5     }
```

Serialización a texto

Se puede asignar data a un string:

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

El programa genera como salida:

```
{"presidente": {"nombre": "Ivan Duque",  
"edad": 44}} <class 'str'>
```



Argumentos de la función dump

Es posible especificar el tamaño de la indentación para estructuras anidadas. El siguiente programa:

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

El programa imprime como salida:

```
{
    "presidente": {
        "nombre": "Ivan Duque",
        "edad": 44
    }
}
```



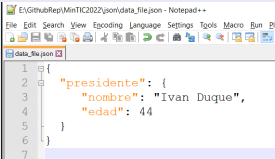
Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



Deserialización desde archivo

Es posible cargar un archivo de JSON desde un archivo.



```
1 {  
2   "presidente": {  
3     "nombre": "Ivan Duque",  
4     "edad": 44  
5   }  
6 }  
7 }
```

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

```
print(data["presidente"])
```

El programa imprime como salida:
'nombre': 'Ivan Duque', 'edad': 44



Deserialización desde texto

Es posible cargar un archivo de JSON desde un string.

```
json_string = '{ "presidente": { "nombre": "Ivan Duque", "edad": 44 } }'  
data = json.loads(json_string)  
print(data)
```

la salida del programa es:

```
'presidente': {'nombre': 'Ivan Duque', 'edad': 44}
```



Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



Conversión de archivos JSON a estructuras de python

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 comodamente en python:

```
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

Si se ejecutan las siguientes consultas a la variable data se tiene:

```
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["myList"]))
print(data["diccionario"], type(data["diccionario"]))

print(data["myList2"][1])
print(data["diccionario"]["papa"])
```

El código mostrará:

```
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 Diccionarios 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 archivo JSON



Usando request para leer archivos JSON desde internet

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:

```
import json
import requests
```



Usando request para leer archivos JSON desde internet

Para leer el archivo JSON de una url específica se utiliza:

`response = request.get(url)` y se carga con
`json.loads(response.text):`

```
import json
import requests
from pprint import pprint

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

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



Analizando la estructura de un JSON

Al procesar los dos últimos 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:

```
pendientes[:2] #Aquí se listan los últimos 2 registros
```

La salida observada es la siguiente:

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



Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON



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:

- Contar cuantas tareas han completado los usuarios
- Ordenar el conteo de tareas pendientes que se han completado
- 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]:
```



Escoger los usuarios que tienen el mismo número de tareas máximo

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

```
usuarios = []  
for usu, num_tareas_completas in items_ordenados:  
    if num_tareas_completas == maximas_tareas_completadas:  
        usuarios.append(str(usu))  
    else:  
        break  
  
usuarios_con_max = " y ".join(usuarios)  
print(f"los usuarios usuarios_con_max 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 a 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,  
  ...
```



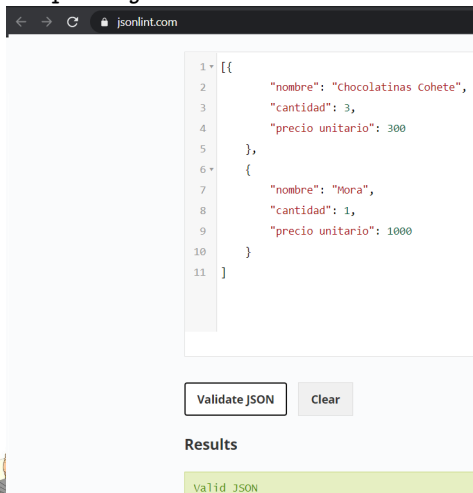
Agenda

- 1 JSON
- 2 JSON y Diccionarios 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 archivo JSON**



Sugerencia

Es posible validar un archivo de JSON utilizando la siguiente 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.

