Python Supports JSON Natively!

Python comes with a built-in package called json for encoding and decoding JSON data.

```
import json
```

The process of encoding JSON is usually called serialization. This term refers to the transformation of data into a *series of bytes* (hence *serial*) to be stored or transmitted across a network.

Deserialization is the reciprocal process of decoding data that has been stored or delivered in the JSON standard.

Serializing JSON

json library exposes the dump() method for writing data to files. Simple Python objects are translated to JSON according to a fairly intuitive conversion.

Python	JSON
Dict	object
list, tuple	array
Str	string
int, long, float	number
True	true
False	false
None	null

```
data = {
   "president": {
      "name": "Murthy",
      "city": "Hyderabad"
   }
}
```

Using Python's context manager, you can create a file called data_file.json and open it in write mode. (JSON files conveniently end in a .json extension.)

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

Or, if you were so inclined as to continue using this serialized JSON data in your program, you could write it to a native Python str object.

```
json_string = json.dumps(data)
```

Deserializing JSON

In the json library, you'll find load() and loads() for turning JSON encoded data into Python objects.

Just like serialization, there is a simple conversion table for deserialization, though you can probably guess what it looks like already.

JSON	Python
Object	dict
Array	list
String	str
number (int)	int
number (real)	float
True	True
False	False
Null	None

A Simple Deserialization Example

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

A Real World Example (sort of)

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

```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}

# Map of userId to number of complete TODOs for that user
todos_by_user = {}

# Increment complete TODOs count for each user.
for todo in todos:
  if todo["completed"]:
    try:
      # Increment the existing user's count.
      todos_by_user[todo["userId"]] += 1
    except KeyError:
      # This user has not been seen. Set their count to 1.
      todos_by_user[todo["userId"]] = 1

# Create a sorted list of (userId, num_complete) pairs.
top_users = sorted(todos_by_user.items(),
                   key=lambda x: x[1], reverse=True)

# Get the maximum number of complete TODOs.
max_complete = top_users[0][1]

# Create a list of all users who have completed
# the maximum number of TODOs.
users = []
for user, num_complete in top_users:
```

```
    if num_complete < max_complete:
        break
    users.append(str(user))

max_users = " and ".join(users)
```

```
# Define a function to filter out completed TODOs
# of users with max completed TODOS.
def keep(todo):
    is_complete = todo["completed"]
    has_max_count = str(todo["userId"]) in users
    return is_complete and has_max_count

# Write filtered TODOs to file.
with open("filtered_data_file.json", "w") as data_file:
    filtered_todos = list(filter(keep, todos))
    json.dump(filtered_todos, data_file, indent=2)
```