

Python Summer Course

Course 3: Files, Data & Practice

Théophile Gentilhomme

July 11, 2025



Installing Python Packages

Most functionality in Python comes from external packages (libraries).

! Install with `pip` (Python's package installer)

```
1 !pip install numpy
```

Run this in:

- Jupyter notebooks (with !)
- Google Colab
- Terminal/command line (without !)

Importing a Package or Module

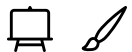
Once installed, use `import` to load the package in your code.

Python Code

↺ Start Over

▶ Run Code

```
1 import math
2
3 print(math.sqrt(16))
```



Import with Alias

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 import numpy as np
2
3 # We will see this next course!
4 print(np.array([1, 2, 3]) * 2)
```



Import Specific Functions

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 from math import pi, sin
2
3 print(sin(pi / 2)) # 1.0
```

Best Practice:

- Use standard aliases: `np` for numpy, `pd` for pandas, `plt` for matplotlib
- Only import what you need to keep code clean

Reading and Writing Files in Python

Python can read from and write to files using the built-in `open()` function.

Python Code

↺ Start Over

▶ Run Code

```
1 with open("example.txt", "w") as f:  
2     f.write("Hello, file!\n")
```



Reading from a Text File

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 ✓ with open("example.txt", "r") as f:  
2     content = f.read()  
3     print(content)
```

! Notes:

- `"w"` mode = write (overwrites file)
- `"r"` mode = read
- `"a"` mode = append
- Always use `with` so auto closes the file
- More functionalities (e.g. line by line) but we focus on specific formats here



JSON: Storing Structured Data

JSON (JavaScript Object Notation) is a text format for storing **structured data**, like Python dictionaries and lists.

✓ It's widely used in APIs, configs, and data exchange.

Python Code

↺ Start Over

▶ Run Code

```
1 import json
2
3 # this is our dict
4
5 data = {"name": "Alice", "age": 30, "skills": ["Python", "Data"]}
```


Save to JSON File

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 ✓ with open("data.json", "w") as f:  
2     # Write with some option for a more redable output  
3     json.dump(data, f, , indent=4, sort_keys=True)
```

Load JSON File

Python Code

[↺ Start Over](#)[▶ Run Code](#)

```
1 ✓ with open("data.json", "r") as f:  
2     loaded = json.load(f)  
3  
4 print(loaded["skills"])
```



JSON ↔ Dictionary

- `json.dump()` → save to file
- `json.load()` → read from file
- `json.dumps()` / `json.loads()` → convert to/from string

! JSON uses:

- `{ }` for objects (dicts)
- `[]` for arrays (lists)
- Keys must be strings

Study Case: Modeling Bacterial Colony Growth

We will work with simulated growth data of bacterial colonies under different conditions.

Objectives:

1. Load and inspect the dataset (JSON)
2. Create a `Colonie` class to represent each colony
3. Implement a method to **predict growth at a given time**
4. Create and test colony objects from data
5. Compare predicted vs actual growth data



Step 1: Load Dataset from JSON

- Open and parse the JSON file using `json.load()`
- Check structure: list of colonies, each with conditions + growth

Solution

```
1 import json
2
3 with open("colonies.json", "r") as f:
4     data = json.load(f)
5
6 print(data[0].keys())
```

Step 2: Define a Colonie

Attributes:

- `name, temperature, ph, sugar`

Method

- `predict_growth(time)`
- `math` package for exp, log functions

Using formula:

```
1 G(t) = S \cdot e^{\{-\frac{(T - 37)^2}{20}\}} \cdot \left(1 - 0.3 \cdot \left(1 - \frac{t}{T_{max}}\right)\right)
```

S: Sugar, T: temperature, pH: pH level, t: time (h)

Solution

```
1 import math
2
3 class Colonie:
4     def __init__(self, name, temperature, ph, sugar):
5         self.name = name
6         self.temperature = temperature
7         self.ph = ph
8         self.sugar = sugar
9
10    def predict_growth(self, t):
11        base = self.sugar * math.exp(-((self.temperature - 37) ** 2) / 20)
12        ph_factor = 1 - abs(self.ph - 7) * 0.3
13        return round(base * ph_factor * math.log(t + 1), 3)
```


Step 3: Create Objects and Compare Predictions

- Loop over data entries
- Create a `Colonie` object for each
- Predict growth at fixed times (e.g. 3, 5, 8)
- Compare with actual values in the dataset

Solution

```
1 fixed_times = [3, 5, 8]
2
3 for item in data:
4     cond = item["conditions"]
5     c = Colonie(item["name"], cond["temperature"], cond["ph"], cond["sugar"])
6     print(f"{c.name}:")
7
8     for t in fixed_times:
9         actual = float(item["growth"][str(t)])
10        predicted = c.predict_growth(t)
11        print(f"  t={t}h:  predicted={predicted}, actual={actual}")
```