# Python Summer Course

## Course 3: Files, Data & Practice

Théophile Gentilhomme

July 10, 2025

Downloading Pyodide

# Installing Python Packages

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

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

```
!pip install numpy
```

Run this in:

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

Downloading package: micropip

# Importing a Package or Module

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

Python Code  Start Over  Run Code

```
import math

print(math.sqrt(16))
```

# Import with Alias

Python Code Start Over Run Code

```python
import numpy as np

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

# Import Specific Functions

Python Code  Start Over  Run Code

```python
from math import pi, sin

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

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

# Reading from a Text File

Python Code | Start Over | Run Code

```python
with open("example.txt", "r") as f:
    content = f.read()
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 specifc 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

```python
import json

# this is our dict

data = {"name": "Alice", "age": 30, "skills": ["Python", "Data"]}
```

# Save to JSON File

Python Code Start Over Run Code

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

# Load JSON File

Python Code | Start Over | Run Code

```python
with open("data.json", "r") as f:
    loaded = json.load(f)

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

```python
import json

with open("colonies.json", "r") as f:
    data = json.load(f)

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:

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

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

## Solution

```python
import math

class Colonie:
    def __init__(self, name, temperature, ph, sugar):
        self.name = name
        self.temperature = temperature
        self.ph = ph
        self.sugar = sugar

    def predict_growth(self, t):
        base = self.sugar * math.exp(-((self.temperature - 37) ** 2) / 20)
        ph_factor = 1 - abs(self.ph - 7) * 0.3
        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

```python
fixed_times = [3, 5, 8]

for item in data:
    cond = item["conditions"]
    c = Colonie(item["name"], cond["temperature"], cond["ph"], cond["sugar"
    print(f"{c.name}:")

    for t in fixed_times:
        actual = float(item["growth"][str(t)])
        predicted = c.predict_growth(t)
        print(f"  t={t}h:  predicted={predicted}, actual={actual}")
```