Lecture 25: File I/O, JSON, and APIs

Comp 102

Forman Christian University

Your programs have

amnesia

Run it, enter data, close it — everything's **gone**.

What if you want to save...

- A high score?

What if you want to save...

- A high score?
- A to-do list?

What if you want to save...

- A high score?
- A to-do list?
- User settings?

What if you want to save...

- A high score?
- A to-do list?
- User settings?
- A diary entry?

What if you want to save...

- A high score?
- A to-do list?
- User settings?
- A diary entry?

Answer: **files** — the simplest form of persistent storage.

Today's Story

Files give your program **memory**

JSON gives it a **language**

APIs let it **talk to the world**

Part 1: Teaching Programs to Remember

The `open()` Function

- `open(filename, mode)` — returns a **file object**

The `open()` Function

- `open(filename, mode)` — returns a **file object**
- Modes:
  - `"r"` — **read** (default)
  - `"w"` — **write** (overwrites!)
  - `"a"` — **append** (adds to end)

Reading a Text File

Three ways to read:

```
# Entire file as one string
content = f.read()

# One line at a time
line = f.readline()

# List of all lines
lines = f.readlines()
```

The `with` Statement

Automatically closes the file — no forgetting!

```
with open("names.txt", "r") as f:
    content = f.read()
    print(content)
# File automatically closed here
```

The `with` Statement

Automatically closes the file — no forgetting!

```
1 with open("names.txt", "r") as f:
2     content = f.read()
3     print(content)
4 # File automatically closed here
```

Opens the file and calls it `f`

The `with` Statement

Automatically closes the file — no forgetting!

```
1 with open("names.txt", "r") as f:
2     content = f.read()
3     print(content)
4 # File automatically closed here
```

File closes automatically when block ends

The `with` Statement

Automatically closes the file — no forgetting!

```
with open("names.txt", "r") as f:
    content = f.read()
    print(content)
# File automatically closed here
```

Always use `with` — it's the safe way!

Gotcha: The Trailing \n

```
1 with open("names.txt", "r") as f:
2     for line in f:
3         print(repr(line))
```

Gotcha: The Trailing \n

```
1 with open("names.txt", "r") as f:
2     for line in f:
3         print(repr(line))
```

Output:

```
'Ayesha\n'
'Bilal\n'
'Chahat\n'
```

Gotcha: The Trailing \n

```
1 with open("names.txt", "r") as f:
2     for line in f:
3         print(repr(line))
```

Output:

```
'Ayesha\n'
'Bilal\n'
'Chahat\n'
```

Fix: use `line.strip()` to remove whitespace and \n

Writing Text Files

```
1 items = ["eggs", "milk", "bread"]
2
3 with open("shopping.txt", "w") as f:
4     for item in items:
5         f.write(item + "\n")
```

Writing Text Files

```
1 items = ["eggs", "milk", "bread"]
2
3 with open("shopping.txt", "w") as f:
4     for item in items:
5         f.write(item + "\n")
```

"w" mode **overwrites** the entire file!

Writing Text Files

```
1 items = ["eggs", "milk", "bread"]
2
3 with open("shopping.txt", "w") as f:
4     for item in items:
5         f.write(item + "\n")
```

- `f.write()` does **not** add `\n` automatically
- You must add it yourself!

Write vs Append

- `open("file.txt", "w")` — **write mode**
  - Creates new file or **overwrites** existing
  - Everything in the file is **deleted** first!

Write vs Append

- `open("file.txt", "w")` — **write mode**
  - Creates new file or **overwrites** existing
  - Everything in the file is **deleted** first!

- `open("file.txt", "a")` — **append mode**
  - Creates new file if it doesn't exist
  - **Adds to the end** of existing content

The Diary Program — It Remembers!

```
1  from datetime import date
2
3  entry = input("Today's diary entry: ")
4
5  with open("diary.txt", "a") as f:
6      f.write(f"{date.today()}: {entry}\n")
7
8  print("\n--- All Entries ---")
9  with open("diary.txt", "r") as f:
10     print(f.read())
```

The Diary Program — It Remembers!

```
1  from datetime import date
2
3  entry = input("Today's diary entry: ")
4
5  with open("diary.txt", "a") as f:
6      f.write(f"{date.today()}: {entry}\n")
7
8  print("\n--- All Entries ---")
9  with open("diary.txt", "r") as f:
10     print(f.read())
```

Append mode — adds to existing entries!

The Diary Program — It Remembers!

```
1  from datetime import date
2
3  entry = input("Today's diary entry: ")
4
5  with open("diary.txt", "a") as f:
6      f.write(f"{date.today()}: {entry}\n")
7
8  print("\n--- All Entries ---")
9  with open("diary.txt", "r") as f:
10     print(f.read())
```

Run it **twice** — your program now has **memory**!

Our program **remembers**. But what if we want to save something more complex?

A student with a name, GPA, and a list of courses?

In a plain text file, we'd have to **invent our own format** and write code to **parse it back**.

Our program **remembers**. But what if we want to save something more complex?

A student with a name, GPA, and a list of courses?

In a plain text file, we'd have to **invent our own format** and write code to **parse it back**.

There has to be a better way...

Part 2: A Universal Language for Data

The Problem with Plain Text

Try saving a student record to a `.txt` file:

```
Ayesha
3.8
COMP102,MATH101
```

The Problem with Plain Text

Try saving a student record to a `.txt` file:

```
Ayesha
3.8
COMP102,MATH101
```

- How do you read this back? You must remember what each line means.
- What if a name has a comma? What if you add a field?
- **It's fragile.**

Let's Try It — Saving

```
student = {
    "name": "Ayesha",
    "gpa": 3.8,
    "courses": ["COMP102", "MATH101"]
}

with open("student.txt", "w") as f:
    f.write(student["name"] + "\n")
    f.write(str(student["gpa"]) + "\n")
    f.write(",".join(student["courses"]) + "\n")
```

Let's Try It — Saving

```
student = {
    "name": "Ayesha",
    "gpa": 3.8,
    "courses": ["COMP102", "MATH101"]
}

with open("student.txt", "w") as f:
    f.write(student["name"] + "\n")
    f.write(str(student["gpa"]) + "\n")
    f.write(",".join(student["courses"]) + "\n")
```

OK, that wasn't *too* bad... but now **read it back**.

Let's Try It — Loading

```
with open("student.txt", "r") as f:
    name = f.readline().strip()
    gpa = float(f.readline().strip())
    courses = f.readline().strip().split(",")

student = {
    "name": name,
    "gpa": gpa,
    "courses": courses
}
```

Let's Try It — Loading

```
with open("student.txt", "r") as f:
    name = f.readline().strip()
    gpa = float(f.readline().strip())
    courses = f.readline().strip().split(",")

student = {
    "name": name,
    "gpa": gpa,
    "courses": courses
}
```

- You must remember: line 1 = name, line 2 = GPA, line 3 = courses
- You must manually convert types (`float()`, `split()`)
- What if you add a field later? **Everything breaks.**
- What about saving **multiple** students? **Nightmare.**

There's a format the **whole world** agreed on:

JSON

Java**S**cript **O**bject **N**otation

Python vs JSON — Spot the Difference

Python	JSON
`{"name": "Ali", "age": 20}`	`{"name": "Ali", "age": 20}`
`True` / `False` / `None`	`true` / `false` / `null`
Single or double quotes	**Must** use double quotes

Python vs JSON — Spot the Difference

Python	JSON
`{"name": "Ali", "age": 20}`	`{"name": "Ali", "age": 20}`
`True` / `False` / `None`	`true` / `false` / `null`
Single or double quotes	**Must** use double quotes

"If you know Python dicts, you already know **90% of JSON**."

The `json` Module — Four Functions

Direction	To File	To String
Python → JSON	`json.dump(data, f)`	`json.dumps(data)`
JSON → Python	`json.load(f)`	`json.loads(text)`

The `json` Module — Four Functions

Direction	To File	To String
Python → JSON	`json.dump(data, f)`	`json.dumps(data)`
JSON → Python	`json.load(f)`	`json.loads(text)`

Mnemonic: the "**s**" stands for "**s**tring"

Saving a Student Record with JSON

```
1  import json
2
3  student = {
4      "name": "Ayesha",
5      "gpa": 3.8,
6      "courses": ["COMP102", "MATH101"]
7  }
8
9  # Save to file
10 with open("student.json", "w") as f:
11     json.dump(student, f, indent=2)
```

Saving a Student Record with JSON

```
1  import json
2
3  student = {
4      "name": "Ayesha",
5      "gpa": 3.8,
6      "courses": ["COMP102", "MATH101"]
7  }
8
9  # Save to file
10 with open("student.json", "w") as f:
11     json.dump(student, f, indent=2)
```

Converts dict to JSON and writes to file

Loading It Back

```
1 import json
2
3 with open("student.json", "r") as f:
4     loaded = json.load(f)
5
6 print(loaded["name"])        # Ayesha
7 print(loaded["courses"][0]) # COMP102
8 print(type(loaded["gpa"]))  # <class 'float'>
```

Loading It Back

```
1 import json
2
3 with open("student.json", "r") as f:
4     loaded = json.load(f)
5
6 print(loaded["name"])          # Ayesha
7 print(loaded["courses"][0])    # COMP102
8 print(type(loaded["gpa"]))     # <class 'float'>
```

Reads JSON file and converts back to Python dict

Loading It Back

```
1 import json
2
3 with open("student.json", "r") as f:
4     loaded = json.load(f)
5
6 print(loaded["name"])        # Ayesha
7 print(loaded["courses"][0]) # COMP102
8 print(type(loaded["gpa"]))  # <class 'float'>
```

Types are preserved! Numbers stay numbers, lists stay lists.

Now you can save structured data and load it back **perfectly**.

But here's what's exciting:

You've just learned the **language of the internet**.

Every time you open a weather app, check prayer times, or convert currency — your phone is fetching **JSON** from a server.

Now you can save structured data and load it back **perfectly**.

But here's what's exciting:

You've just learned the **language of the internet**.

Every time you open a weather app, check prayer times, or convert currency — your phone is fetching **JSON** from a server.

What if your Python program could do the **same thing**?

Part 3: Connecting to the World

What is an API?

Application **P**rogramming **I**nterface

A way for programs to talk to each other over the internet.

What is an API?

Application **P**rogramming **I**nterface

A way for programs to talk to each other over the internet.

Restaurant analogy:

- You (your program) are the **customer**
- The kitchen (server) has the **data**
- The menu (API) lists what you can **request**
- The waiter (HTTP) **carries** your request and brings back the response

How Web APIs Work

How Web APIs Work

The prayer times app on your phone works **exactly** this way.

The `requests` Library

```
import requests
```

- `response = requests.get(url)` — make a GET request

The `requests` Library

```
import requests
```

- `response = requests.get(url)` — make a GET request
- Key attributes:
  - `response.status_code` — 200 means success
  - `response.text` — raw text of response
  - `response.json()` — parse JSON into Python dict

The `requests` Library

```
import requests
```

- `response = requests.get(url)` — make a GET request
- Key attributes:
  - `response.status_code` — 200 means success
  - `response.text` — raw text of response
  - `response.json()` — parse JSON into Python dict

Note: may need `pip install requests` in Thonny's package manager.

Your First API Call

```
1 import requests
2
3 url = "https://api.chucknorris.io/jokes/random"
4 response = requests.get(url)
5 data = response.json()
6 print(data["value"])
```

Your First API Call

```
1 import requests
2
3 url = "https://api.chucknorris.io/jokes/random"
4 response = requests.get(url)
5 data = response.json()
6 print(data["value"])
```

Three lines of code, and your program just talked to a server on the internet.

Demo: Prayer Times for Lahore

```
1  import requests
2
3  url = "https://api.aladhan.com/v1/timingsByCity"
4  params = {"city": "Lahore", "country": "Pakistan", "method": 1}
5  response = requests.get(url, params=params)
6  data = response.json()
7
8  timings = data["data"]["timings"]
9  print(f"Fajr:    {timings['Fajr']}")
10 print(f"Dhuhr:   {timings['Dhuhr']}")
11 print(f"Asr:     {timings['Asr']}")
12 print(f"Maghrib: {timings['Maghrib']}")
13 print(f"Isha:    {timings['Isha']}")
```

Demo: Prayer Times for Lahore

```
1  import requests
2
3  url = "https://api.aladhan.com/v1/timingsByCity"
4  params = {"city": "Lahore", "country": "Pakistan", "method": 1}
5  response = requests.get(url, params=params)
6  data = response.json()
7
8  timings = data["data"]["timings"]
9  print(f"Fajr:    {timings['Fajr']}")
10 print(f"Dhuhr:   {timings['Dhuhr']}")
11 print(f"Asr:     {timings['Asr']}")
12 print(f"Maghrib: {timings['Maghrib']}")
13 print(f"Isha:    {timings['Isha']}")
```

Parameters are added to the URL automatically

Demo: Prayer Times for Lahore

```python
import requests

url = "https://api.aladhan.com/v1/timingsByCity"
params = {"city": "Lahore", "country": "Pakistan", "method": 1}
response = requests.get(url, params=params)
data = response.json()

timings = data["data"]["timings"]
print(f"Fajr:    {timings['Fajr']}")
print(f"Dhuhr:   {timings['Dhuhr']}")
print(f"Asr:     {timings['Asr']}")
print(f"Maghrib: {timings['Maghrib']}")
print(f"Isha:    {timings['Isha']}")
```

This is the **exact same data source** your prayer-time apps use.

Demo: USD to PKR Exchange Rate

```python
import requests

url = ("https://cdn.jsdelivr.net/npm/"
       "@fawazahmed0/currency-api@latest"
       "/v1/currencies/usd.json")
response = requests.get(url)
data = response.json()

pkr_rate = data["usd"]["pkr"]
print(f"1 USD = {pkr_rate:.2f} PKR")

amount = float(input("Enter USD amount: "))
print(f"${amount} = Rs. {amount * pkr_rate:.2f}")
```

Demo: USD to PKR Exchange Rate

```python
import requests

url = ("https://cdn.jsdelivr.net/npm/"
       "@fawazahmed0/currency-api@latest"
       "/v1/currencies/usd.json")
response = requests.get(url)
data = response.json()

pkr_rate = data["usd"]["pkr"]
print(f"1 USD = {pkr_rate:.2f} PKR")

amount = float(input("Enter USD amount: "))
print(f"${amount} = Rs. {amount * pkr_rate:.2f}")
```

Navigate the JSON to find the rate

Demo: USD to PKR Exchange Rate

```python
import requests

url = ("https://cdn.jsdelivr.net/npm/"
       "@fawazahmed0/currency-api@latest"
       "/v1/currencies/usd.json")
response = requests.get(url)
data = response.json()

pkr_rate = data["usd"]["pkr"]
print(f"1 USD = {pkr_rate:.2f} PKR")

amount = float(input("Enter USD amount: "))
print(f"${amount} = Rs. {amount * pkr_rate:.2f}")
```

You just built a **currency converter**. The same way Google does it.

Demo: Dictionary / Word Lookup

```
1  import requests
2
3  word = input("Look up a word: ")
4  url = f"https://api.dictionaryapi.dev/api/v2/entries/en/{word}"
5  response = requests.get(url)
6
7  if response.status_code == 200:
8      data = response.json()
9      meanings = data[0]["meanings"]
10     for m in meanings:
11         print(f"\n{m['partOfSpeech']}:")
12         for d in m["definitions"][:2]:
13             print(f"  - {d['definition']}")
14 else:
15     print(f"Word '{word}' not found.")
```

Demo: Dictionary / Word Lookup

```
1  import requests
2
3  word = input("Look up a word: ")
4  url = f"https://api.dictionaryapi.dev/api/v2/entries/en/{word}"
5  response = requests.get(url)
6
7  if response.status_code == 200:
8      data = response.json()
9      meanings = data[0]["meanings"]
10     for m in meanings:
11         print(f"\n{m['partOfSpeech']}:")
12         for d in m["definitions"][:2]:
13             print(f"  - {d['definition']}")
14 else:
15     print(f"Word '{word}' not found.")
```

Always check status code before using data!

Demo: Current Weather in Lahore

```python
import requests

url = "https://api.open-meteo.com/v1/forecast"
params = {
    "latitude": 31.55,
    "longitude": 74.35,
    "current": "temperature_2m,"
               "relative_humidity_2m,"
               "wind_speed_10m"
}
response = requests.get(url, params=params)
data = response.json()

current = data["current"]
print(f"Temp:     {current['temperature_2m']}°C")
print(f"Humidity: {current['relative_humidity_2m']}%")
print(f"Wind:     {current['wind_speed_10m']} km/h")
```

Demo: Current Weather in Lahore

```python
import requests

url = "https://api.open-meteo.com/v1/forecast"
params = {
    "latitude": 31.55,
    "longitude": 74.35,
    "current": "temperature_2m,"
               "relative_humidity_2m,"
               "wind_speed_10m"
}
response = requests.get(url, params=params)
data = response.json()

current = data["current"]
print(f"Temp:     {current['temperature_2m']}°C")
print(f"Humidity: {current['relative_humidity_2m']}%")
print(f"Wind:     {current['wind_speed_10m']} km/h")
```

Every weather app works this way. You just built the backend of one.

You now know how to **read/write files**, work with **JSON**, and **fetch data** from APIs.

You now know how to **read/write files**, work with **JSON**, and **fetch data** from APIs.

Let's put **all three** together into one program that does something **actually useful**.

The Capstone — All Three Combined

Weather Tracker (Part 1: Fetch)

```python
import requests
import json
from datetime import date

# --- Part 3: Fetch from an API ---
url = "https://api.open-meteo.com/v1/forecast"
params = {"latitude": 31.55, "longitude": 74.35,
          "current": "temperature_2m,wind_speed_10m,relative_humidity_2m"}
response = requests.get(url, params=params)

if response.status_code == 200:
    # --- Parse JSON response ---
    data = response.json()
    current = data["current"]

    # Display nicely
    print(f"Lahore Weather for {date.today()}")
    print("-" * 35)
    for key, label in [("temperature_2m", "Temperature"),
                       ("wind_speed_10m", "Wind Speed"),
                       ("relative_humidity_2m", "Humidity")]:
        print(f"  {label:15s} {current[key]}")
```

Weather Tracker (Part 2: Save)

```
# --- Part 2: Structure data as JSON ---
record = {
    "date": str(date.today()),
    "temperature": current["temperature_2m"],
    "wind_speed": current["wind_speed_10m"],
    "humidity": current["relative_humidity_2m"]
}

# --- Part 1: Read/write files ---
try:
    with open("weather_history.json", "r") as f:
        history = json.load(f)
except FileNotFoundError:
    history = []

history.append(record)

with open("weather_history.json", "w") as f:
    json.dump(history, f, indent=2)

print(f"\nSaved! {len(history)} day(s) of weather data stored.")
```

Weather Tracker (Part 2: Save)

```
# --- Part 2: Structure data as JSON ---
record = {
    "date": str(date.today()),
    "temperature": current["temperature_2m"],
    "wind_speed": current["wind_speed_10m"],
    "humidity": current["relative_humidity_2m"]
}

# --- Part 1: Read/write files ---
try:
    with open("weather_history.json", "r") as f:
        history = json.load(f)
except FileNotFoundError:
    history = []

history.append(record)

with open("weather_history.json", "w") as f:
    json.dump(history, f, indent=2)

print(f"\nSaved! {len(history)} day(s) of weather data stored.")
```

Handle first run when file doesn't exist yet

One Program, Three Skills

1. **Part 3** — Fetch weather data from the API
2. **Part 2** — Structure the data as a JSON-ready dict
3. **Part 1** — Read/write the history file

Run it **twice** — watch the history grow!

Common Mistakes

Common Mistakes to Avoid

1. **Forgetting to close files** → always use `with`

Common Mistakes to Avoid

1. **Forgetting to close files** → always use `with`
2. **Using `"w"` when you meant `"a"`** — overwrites everything!

Common Mistakes to Avoid

1. **Forgetting to close files** → always use `with`
2. **Using `"w"` when you meant `"a"`** — overwrites everything!
3. **JSON requires double quotes** — single quotes cause errors

Common Mistakes to Avoid

1. **Forgetting to close files** → always use `with`
2. **Using `"w"` when you meant `"a"`** — overwrites everything!
3. **JSON requires double quotes** — single quotes cause errors
4. **Not checking `status_code`** before using API data

Common Mistakes to Avoid

1. **Forgetting to close files** → always use `with`
2. **Using `"w"` when you meant `"a"`** — overwrites everything!
3. **JSON requires double quotes** — single quotes cause errors
4. **Not checking `status_code`** before using API data
5. **Confusing `json.load()` vs `json.loads()`** — file vs string

Summary

The Story in One Sentence

Files give your program **memory**,

JSON gives it a **language**,

and **APIs** let it **talk to the world**.

Summary: Key Functions

Concept	Key Function
Read a file	`open("file.txt", "r")`, `f.read()`
Write a file	`open("file.txt", "w")`, `f.write()`
Safe file handling	`with open(...) as f:`
Python → JSON	`json.dump()` / `json.dumps()`
JSON → Python	`json.load()` / `json.loads()`
API call	`requests.get(url)`
Parse API response	`response.json()`

Free APIs — No Key Required

API	Endpoint	Returns
Prayer Times	`api.aladhan.com/v1/timingsByCity`	Prayer times for any city
Weather	`api.open-meteo.com/v1/forecast`	Current weather anywhere
Currency	`cdn.jsdelivr.net/.../usd.json`	Exchange rates (200+ currencies)
Dictionary	`api.dictionaryapi.dev/.../{word}`	Word definitions
Pokemon	`pokeapi.co/api/v2/pokemon/{name}`	Pokemon data (for fun!)
Trivia	`opentdb.com/api.php?amount=10`	Quiz questions
Chuck Norris	`api.chucknorris.io/jokes/random`	Random Chuck Norris joke

Practice Ideas

1. **Currency converter**: Fetch USD→PKR rate, convert amounts, save history to JSON
2. **Weekly prayer times**: Fetch for 7 days, save to JSON, display as table
3. **Personal dictionary**: Look up words via API, save favorites to JSON
4. **Weather logger**: Fetch weather each run, append to JSON, show trends

Questions?