# Computational Thinking

## Lecture 13: File I/O

University of Engineering and Technology

Vietnam National University

# Outline

- Motivation
- Open/Close
- Read/Write & With
- OS Functions
- Other file formats
- Hands-On Activity

# Motivation

# Problem

Before:

```
name = input("What's your name?" )
print(f"hello, {name}")
```

Or for multiple multiple names (input 3 times)

```
for _ in range(3):
    name = input("What's your name?" )
    print(name)
```

# Problem

Say we need to store 3 names:

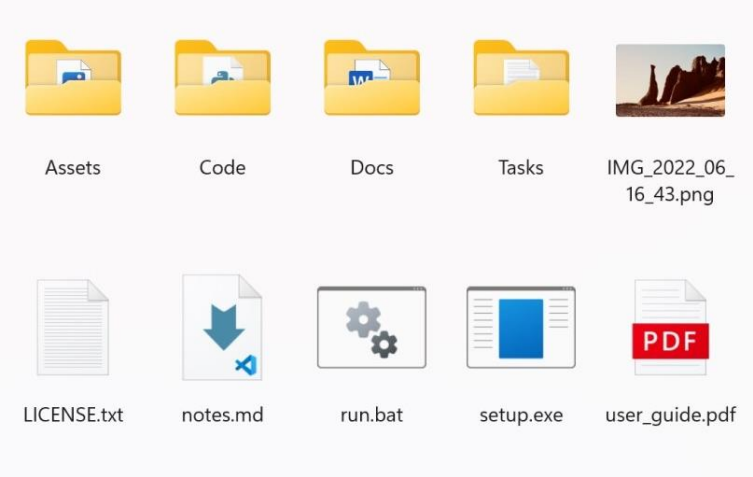
```
names = []

for _ in range(3):
   names.append(input("Give me your name:"))
```

Problem: All information of previous sessions is lost on program's execution

# Value of Files

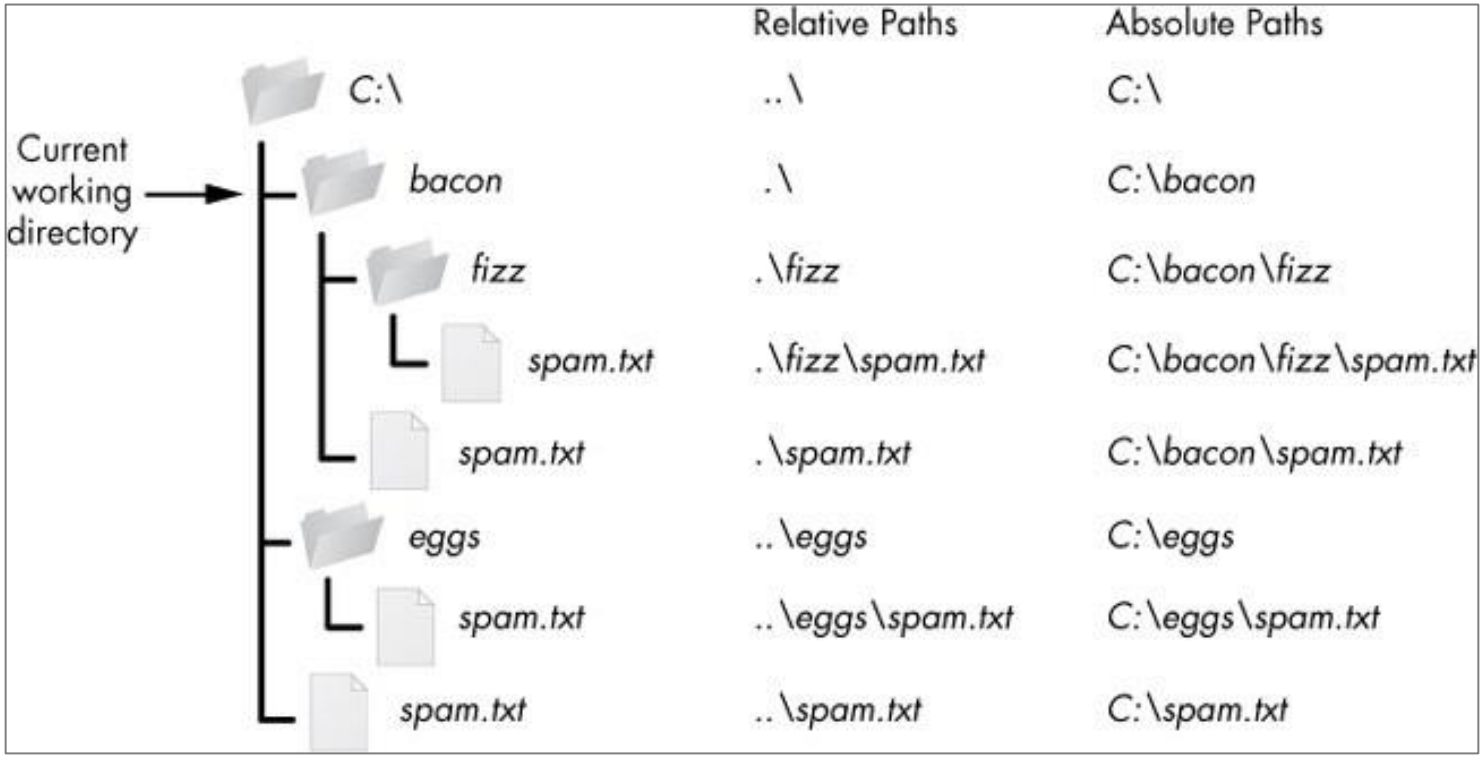
**Files** are persistent way to store programs, input data and output data




- Files are stored in the **memory** of your computer in an area allocated to the **file system**, which is typically arranged into a **hierarchy** of **directories**.
- The **path** to a particular file details **where** the file is **stored** within this **hierarchy**

# Relative Pathnames

A path to a file may be **absolute** or **relative**




Note: Backslash (\) is an **escape character** in Python, the solution is to use forward slash (/) or raw strings with r or R prefixes.

# File I/O

- To store data permanently, even after the program ends.
- To access external files like .txt, .csv, .json, etc.
- To process large files efficiently without using much memory.
- To automate tasks like reading configs or saving outputs.
- To handle input/output in real-world applications and tools.

# Manage Files in Python

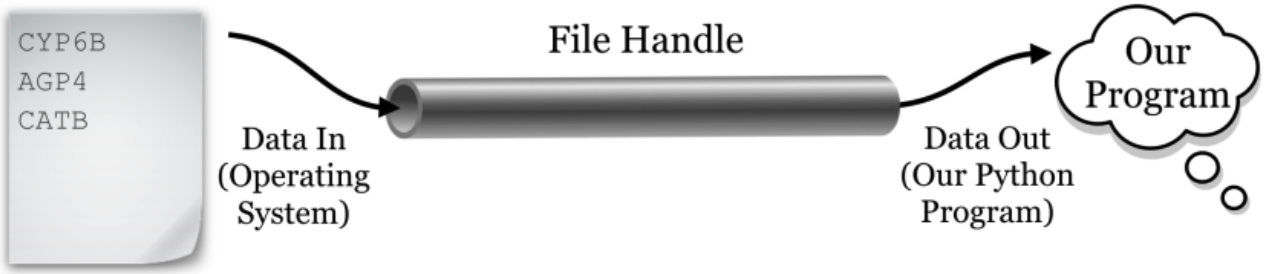
Python provides a simple, elegant interface to storing and retrieving data in files.

- open: establish a connection to the file and associate a local file handle with a physical file.
- close: terminate the connection to the file.
- read: input data from the file into your program
- write: output data from your program to a file.

# open/close

# Opening a File

Before your program can access the data in a file, it is necessary to *open* it. This returns a *file* object, also called a "*handle*," that you can use within your program to access the file.




It also informs the system how you intend for your program to interact with the file, the "*mode*."

# open()

**open**: open a file object to **read** from to **write** to it

```
file = open('filename.txt', 'mode')
```

- **filename.txt:** the file path to be opened.
- **mode:** mode in which you want to open the file (read – r, write - w, append - a, etc.). Default: read – r

Note: if full path is not provided, relative path to current working directory is used.

# open mode

**mode:**

- r (read): give reading access. Throw **FileNotFoundError exception** if given wrong directory or file does not exist
- w (write): give writing access. Create new file at given directory if file does not exist, and *overwrite* file's content if it exists.
- a (append): give writing access. Create new file at given directory if file does not exist, and *add* content at the end of the file otherwise

- Some other modes can be used for multiple access rights and other types of format (text/binary) – (More details)

# close()

**close**: closes the file and releases resources.

```
file = open('filename.txt', 'w')
file.close()
```

Why do we need to close any file open in Python?

- the file will be locked from access by any other program while you have it open;
- items you write to the file may be held in internal buffers rather than written to the physical file;
- if you have a file open for writing, you can't read it until you close it, and re-open for reading;
- it's just good programming practice.

# File properties

Once opened as an object, some of the file's properties can be checked

```
file = open('names.txt', 'r')
print(file.name)
print(file.mode)
print(file.closed)
```

file.name: the name of the file (names.txt)

file.mode: the mode the file was opened with (r)

file.closed: False if file is currently opened, otherwise True

# Read/Write a File

# Write a File

**write**: write any string to an opened file

```
file = open("filename.txt", 'w')
string = input('Enter your name:')
file.write(string)
file.close()
```

Note: write() does not add newline character ('\n') at the end of the string

# Raw Strings

There is a way in Python to treat a string as a **raw string**, meaning that escaped characters are treated just as any other characters.

```
>>> print("abc\ndef")
abc
def
>>> print(r"abc\ndef" )
abc\ndef
```

Prefix the string with an ”r”. You may or may not need to do the for Windows pathnames including "\"

# Read a File

**read**: read the content of an opened file

```
file = open("filename.txt", "r")
content = file.read()
print(content)
file.close()
```

Note:

- `file.read()` : return the entire content of the file as a string
- `file.read(k)` : return next k characters from the file as a string
- `file.readline()` : returns the next line as a string
- `file.readlines()` : return all remaining lines in the file as a list of strings.

# with

**with**: allows programs to automate the closing of a file

```
with open("filenames.txt", "r") as file:
     print(file.read())

name = input("What's your name? ")

with open("names.txt", "a") as file:
    file.write(f"{name}\n")
```

Note: Recommended when multiple files needs to be opened at once

# Example: Read Lines from File

```
import os.path

FILENAME = "Raven.txt "
def main ():
        ""Print lines from file, keeping a count."""
        if not os.path.isfile(FILENAME):
                print("File does not exist")
                return
        # Open file for input
        ravenFile = open(FILENAME, "r")
        line = ravenFile.readline()
        lineCount = 0
        while line:              # line is not empty string
                lineCount += 1
                print(format(lineCount , "3d" ) , ": " , \
                line.strip(), sep = "" )
                line = ravenFile.readline()
        print("\nFound ", lineCount, " lines.")
        ravenFile.close()
main ()
```

# OS Functions

# File Operations

**rename**: change current filename to a new one

```
import os
os.rename('names.txt', 'new_names.txt')
```

**remove**: delete chosen file

```
import os
os.remove('usernames.txt')
```

# Directory Operation

**mkdir**: create a new directory in the current directory with a given name

```
os.mkdir('newdir')
```

**rmdir**: remove a directory. Before removal, all contents of the directory should removed

```
os.rmdir('dirname')
```

**chdir:** change current working directory

```
os.chdir('dirname')
```

# Directory Operation

**getcwd**: display current working directory

```
os.getcwd() => Ex: D:/code
```

**listdir**: get a list of all files and directories in a specified directory: os.listdir('dirname')

```
import os

path = "D:/code"
contents = os.listdir(path)
print("Directory contents:", contents)
```

# Path Operation

**join**: joins one or more path components intelligently

```
import os

base_dir = "D:/data"
filename = "names.txt"

path = os.path.join(base_dir, filename)
print(path)

=> "D:/data/names.txt"
```

# Path Operation

**exists**: check if a path exists

```
import os

base_dir = "D:/data"
filename = "names.txt"

print(os.path.exists(os.path.join(base_dir,
filename)))

(check if there is anything at
"D:/data/names.txt" or not)
```

# Other File Formats

# Comma Separated Value (CSV)

CSV is a **simple representation** of tables.

Ex: "students.csv"

Hermione,Gryffindor
Harry,Gryffindor
Ron,Gryffindor
Draco,Slytherin

Remove trailing whitespace

Split rows into separate values

```
with open("students.csv") as file:
    for line in file:
        row = line.rstrip().split(",")
        print(f"{row[0]} is in {row[1]}")
```

# JavaScript Object Notation (JSON)

**JSON** is a lightweight data-interchange format, easy for human and machine to understand and create.

```
{
      "name": "Nam" ,
      "age": 18,
      "city": "Ha Noi"
}
```

=> Use json package to parse and operate on json files

# Load a JSON File

**json.load(filename)**: convert a JSON file into a Python dict object

```
import json
user = json.load("user.json")

print(user) => {"name": "Nam" , "age": 18,
"city": "Ha Noi")
print(user["name"]) => John
```

# Dump/Write a JSON File

**json.dumps(content, filename)**: convert a Python object into a JSON string

```
import json
user = {
      “name”: “Nam” ,
      “age”: 18,
      “city”: “Ha Noi”
}
json.dumps(user, “user.json”)
```

# Binary files: Image

A popular Python library for image files is **PIL**

```
from PIL import Image
path = <path_to_image>

#open the image file
im = Image.open(path)
#show the image file
im.show()
```

# Hands-On Activity

# Hands-On Activity

**Activity I: Lines of Code (LOC)**

Write a Python script to count the number of lines of code (LOC) in another script you have written.

**Activity II: "I was here"**

Write a Python script to add a line printing "I was here" to the end of a script you have written.

**Activity III: Correcting user information**

Using a Python script, create a JSON file "user.json" containing your name, age and city. Then, change the name to "UETer", age to "18" and city to "Hanoi".

# Summary

- Files provide **persistent storage** beyond program execution.
- Use open() (modes r/w/a) to **read/write**, and **close** files or use with.
- Work with file content via **read / write / append** and line-by-line processing.
- Use **os / os.path** for file & directory operations (paths, rename, remove, list, cwd).
- Handle other formats such as **CSV** and **JSON** for structured data exchange.