Lec-06 in Python

Dr. Sushil Sharma

1. File Handling
2. Modules & PIP
3. Introduction to NumPy
4. Introduction to Matplotlib

https://tinyurl.com/tut-05

1. File Handling

What is File Handling & Why do we need it?

File handling refers to the process of performing operations on a file, such as **creating**, **opening**, **reading**, **writing** and **closing** it through a programming interface.

We need this to:

- 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.

open( ) - A key function for working with files in Python

The open() function takes two parameters; **filename**, and **mode**.

There are four different methods (modes) for opening a file:

- "r" - Read - Default value - Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition one can specify if the file should be handled as binary or text mode

- "t" - Text - Default value - Text mode
- "b" - Binary - Binary mode (e.g. images)

Syntax

Note: Make sure the file exists in cwd.

```
f = open("demofile.txt", "rt")
```

Let's create a `demofile.txt` & play with it

Create a `demofile.txt` in your colab, and fill in some text inside of it.

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

This is the text inside demofile.txt.

```
f = open("demofile.txt")
print(f.read())
```

The open() function returns a file object, which has read() method for reading the content of a file.

```
f = open("demofile.txt")
print(f.read(5))
f.close()
```

It is a good practice to close the file when you are done with it using close() method.

```
with open("demofile.txt") as f:
  print(f.read())
```

You can use with statement to *not worry about closing the file*.

read() vs readline() Method

The primary difference is that the read() method reads the entire content of a file into a single string, while the readline() method reads only one line at a time.

```
f = open("demofile.txt")
print(f.readline())
f.close()
```

```
with open("demofile.txt") as f:
  print(f.readline())
  print(f.readline())
```


```
with open("demofile.txt") as f:
  for x in f:
    print(x)
```

Looping through the file line by line

File Writing: “a” = Append & “w” = Write

```
with open("demofile.txt", "a") as f:
  f.write("Now the file has more content!")

#open and read the file after the appending:
with open("demofile.txt") as f:
  print(f.read())
```

```
with open("demofile.txt", "w") as f:
  f.write("Woops! I have deleted the content!")

#open and read the file after the overwriting:
with open("demofile.txt") as f:
  print(f.read())
```

File Creation (& Deletion): “x”=Create

```
f = open("myfile.txt", "x")
```

```
import os
os.remove("demofile.txt")
```

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Exception Handling

EH is used so that a ***program does not crash*** when something predictably can go wrong (missing file, wrong permission, etc)

- The **try** block lets you test a block of code for errors (the code you expect might fail).
- The **except** block lets you handle the error. (runs only if an error happened in **corresp. try**)
- The **else** block lets you execute code when there is no error. (runs if no error in **try block**)
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

```
try:
    f = open("demofile.txt")
except:
    print("Open failed")
else:
    print("Write OK")
finally:
    f.close()
```

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is
finished")
```

Raise an exception

To throw (or raise) an exception, use the `raise` keyword.

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Q1 - Log File Analyzer

You are given a file `log.txt` containing one message per line.

Some lines contain the word "ERROR".

Task: Write a program that:

1. Reads the file line by line
2. Counts how many error messages exist
3. Writes only the error messages into a new file errors.txt

Q2 - Student Record Cleaner

A file `students.txt` contains:

```
Alice,85
Bob,abc
Charlie,90
David,78
…
```

Some marks are invalid (not numbers).

Task: Create a new file `cleaned.txt` that contains only valid student records.

Expected Output:

```
Alice,85
Charlie,90
David,78
```

Q3 - File Mode Reasoning

What happens if a program uses "w" instead of "a" while saving user data?

Demonstrate this with code and explain the result.

2. Modules & PIP

Modules - A Code Library

A file containing a set of functions that we want to include in our program.

Let's create our own module, save the following code in a file named mymodule.py.

```
def greeting(name):
  print("Hello, " + name)
```

Use import statement to use the module we just created.

```
import mymodule
mymodule.greeting("Guido Von Rossum!")
```

```
person1 = {
  "name": "Ram",
  "age": 40,
  "country": "India"
}
```

```
import mymodule,importlib
importlib.reload(mymodule)
from mymodule import person1

a = person1["age"]
print(a)
```

Modules - A file containing a set of functions you want to include in your application.

- To create a module just save the code you want in a file with the file extension `.py`:

```
def greeting(name):
  print("Hello, " + name)
```

- Now we can use the module we just created, by using the `import` statement:

```
import mymodule
mymodule.greeting("Trump")
```

- Using alias:

```
import mymodule as mx
mx.greeting("Trump")
```

Note: When using a function from a module, use the syntax: module_name.function_name.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

```
person1 = {
  "name": "Konrad",
  "age": 24,
  "country": "Poland"
}
```

```
from mymodule import person1
```

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

There is a built-in function to list all the function names (or variable names) in a module. The `dir()` function:

There are several built-in modules in Python, which you can import whenever you like.

```
import platform

x = dir(platform)
print(x)
```

Example 1 - Time Module

Get current time and sleep for 1 second:

```
import time

start = time.time()
print(f'Start time: {start}')
time.sleep(1)
end = time.time()
print(f'Elapsed: {end - start:.2f} seconds')
```

Python time Module

Example 2 - Random Module

```
import random
a = [1, 2, 3, 4, 5, 6]
print(random.choice(a))
```

```
import random

a = [1, 2, 3, 4, 5, 6]
print(random.choice(a))

s = "geeks"
print(random.choice(s))

tup = (1, 2, 3, 4, 5)
print(random.choice(tup))
```

[Python Random Module - GeeksforGeeks]

```
import random
random.seed(5)
print(random.random())
print(random.random())
```

```
from random import random
print(random())
```

```
from random import sample

a = [1, 2, 3, 4, 5]
print(sample(a,3))

b = (4, 5, 6, 7, 8)
print(sample(b,3))

c = "45678"
print(sample(c,3))
```

```
import random
r1 = random.randint(5, 15)
print(r1)

r2 = random.randint(-10, -2)
print(r2)
```

```
import random
a = [1, 2, 3, 4, 5]

random.shuffle(a)
print("After shuffle : ")
print(a)

random.shuffle(a)
print("\nSecond shuffle : ")
print(a)
```

PIP is a package manager for Python packages, or modules if you like.

```
!pip install camelcase

!pip uninstall camelcase

!pip list
```

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

pip · PyPI

Q4 - Build Your Own Utility Module

Create a module `stats.py` with functions:

1. mean(numbers)
2. maximum(numbers)
3. minimum(numbers)

Then write a separate Python file that imports this module and analyzes this list:

```
[12, 45, 23, 67, 34]
```

Q5 - Aliasing & Namespace Conflict

Create two modules:

1. mathops.py with a function add
2. stringops.py with a function add (that concatenates strings)

Import both using aliases and demonstrate both add functions in the same program.

Q6 - External Package Investigation

Using PIP, install any package of your choice.

Write a short program that uses one useful feature from that package.

3. Introduction to NumPy

NumPy (Numerical Python)

Open source Python library that's widely used across all disciplines of science.

The NumPy library contains multidimensional array data structures, such as the homogeneous, N-dimensional ndarray, and a large library of functions that operate efficiently on these data structures.

Why use Numpy (& not lists)?

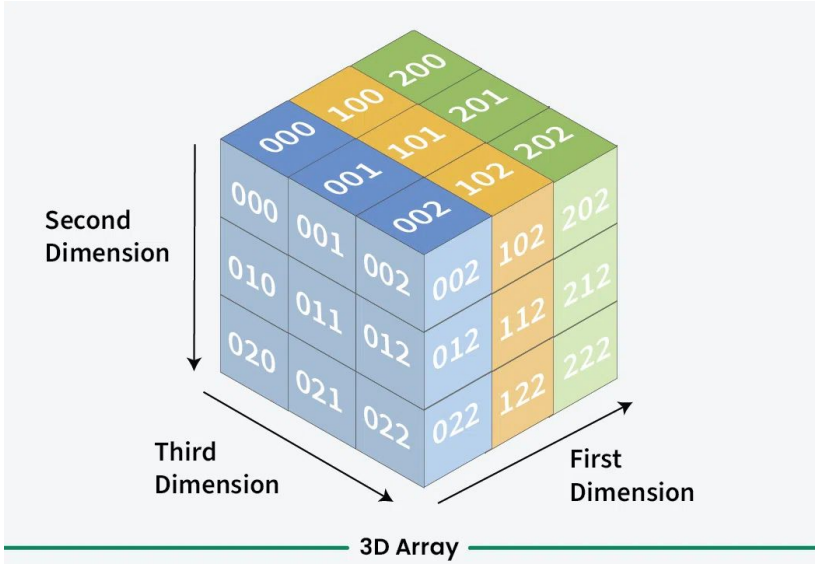
- improve speed,
- reduce memory consumption, and
- offer a high-level syntax for performing a variety of common processing tasks

1	5	2	0
8	3	6	1
1	7	2	9

What is an “array”?

1	5	2	0

In computer programming, an array is a structure for storing and retrieving data.



Restrictions on `ndarray`:

- All elements of the array must be of the same type of data.
- Once created, the total size of the array can’t change.
- The shape must be “rectangular”, not “jagged”; e.g., each row of a two-dimensional array must have the same number of columns.

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(np.__version__)

print(type(arr))
```

```python
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```python
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```python
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

```python
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Slicing *[start:end:step]*

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])
```

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

Q7 - Marks Matrix

You have marks of 3 students in 4 subjects:

```
[[80, 70, 90, 60],
 [75, 85, 95, 65],
 [60, 70, 80, 90]]
```

Using NumPy:

1. Store this in an array
2. Print the marks of the 2nd student
3. Print all marks of subject 3
4. Print the highest mark

Q8 - Slice Without Loop

From this array:

```
[10, 20, 30, 40, 50, 60, 70]
```

Extract only the even-positioned values using slicing.

Q9 - Copy vs View Trap

Write a program that shows what happens when:

- you change the original array after copying
- you change the original array after slicing

Explain the difference.

4. Introduction to Matplotlib

Matplotlib - A Visualization Tool

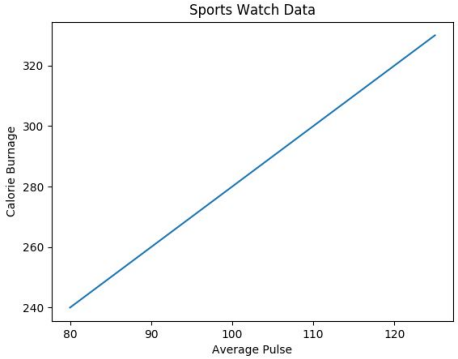
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

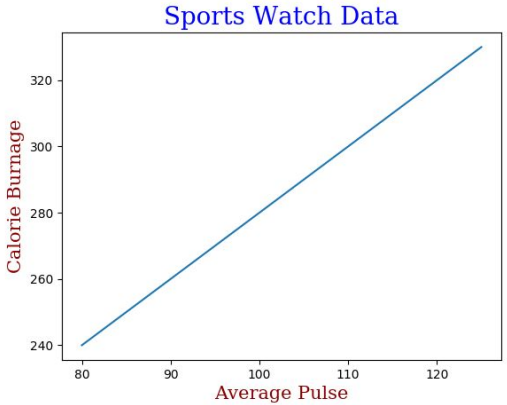
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints)
plt.show()
```

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120,
125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310,
320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```






```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115,
120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310,
320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 =
{'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```

Subplots in Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1,2,3,4])
temp = np.array([22, 24, 19, 23])
sales = np.array([200, 300, 250, 400])

plt.subplot(2, 1, 1)
plt.plot(x, temp)
plt.title("Temperature")

plt.subplot(2, 1, 2)
plt.plot(x, sales)
plt.title("Sales")

plt.show()
```

Q10 - Temperature Plot

You have daily temperatures for a week:

```
[22, 24, 19, 23, 25, 21, 20]
```

Plot:

- Day number on X-axis
- Temperature on Y-axis
- Title: "Weekly Temperature"

Q11 - Compare Two Students

Marks of two students:

```
A = [70, 75, 80, 85]

B = [60, 65, 78, 88]
```

Plot both on the same graph so you can visually compare performance.

Q12 - Debug This Graph

This code gives a wrong graph. Why?

```
x = np.array([1, 2, 3, 4])

y = np.array([10, 20, 30])

plt.plot(x, y)
```

Fix it and explain.

Q13 - Kinematics

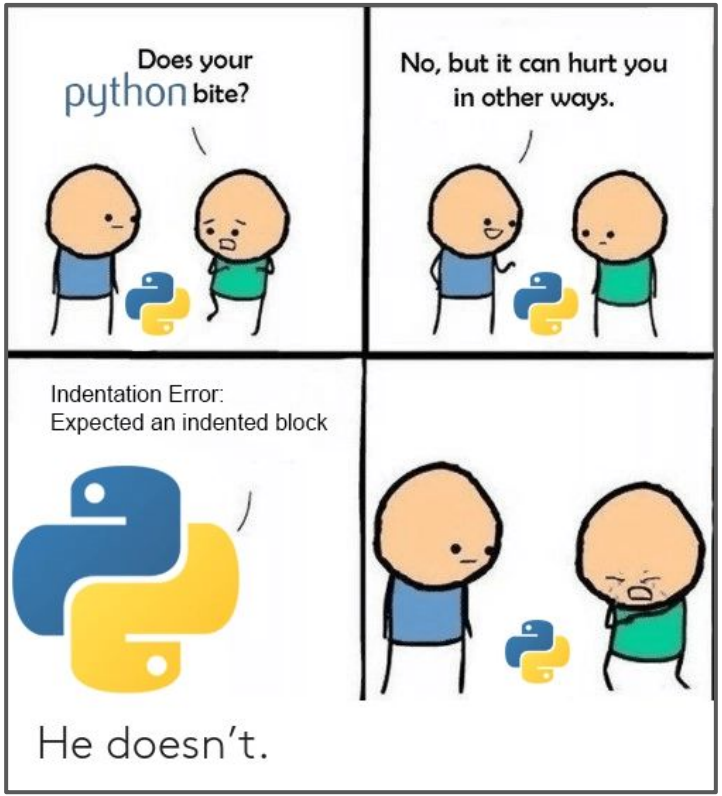
You are given:

```
speed = [10, 20, 30, 40]

time = [1, 2, 3, 4]

distance = [10, 40, 90, 160]
```

Plot:

- Speed vs Time in the first subplot
- Distance vs Time in the second subplot

Thank You - Do Widzenia! ^_^



Bonus Question - mixed.txt Analyzer

Separate:

- Errors → errors.txt
- Numbers → numbers.txt

```
ERROR Disk full
User login
75
ERROR Network down
hello
100
ERROR Access denied
42
```

Next Lecture: Using real-world dataset to showcase how plots are used for analysis.

Backup Slides

log.txt

System started successfully

User logged in

ERROR: Invalid password

Connection established

ERROR: Database not found

User logged out

ERROR: Timeout occurred

System shutdown

students.txt

Alice,85

Bob,abc

Charlie,90

David,78

Eva,9O

Frank,100

Grace,seven

Helen,88