Unit 3

Modules

Akshata S Bhayyar, Asst. Prof. Dept. of CSE, RIT

module

- A module is a file containing Python definitions and statements intended for use in other Python programs.

Random numbers

```
import random
rng = random.Random()
dice_throw = rng.randrange(1,7)
print(dice_throw)
```

The time module

- The time module has a function called clock.
- Whenever clock is called, it returns a floating point number representing how many seconds have elapsed since your program started running.
- So our function runs about 57% slower than the built-in one. Generating and summing up ten million elements

```
name=["Ram","Raj","Seema","riya"]
Computers=["Dell","HP","Apple","Acer"]

for i in range(0,len(name)):
    print("Computer used by",name[i],"is",Computers[i])
```

```
Computer used by Ram is Dell
Computer used by Raj is HP
Computer used by Seema is Apple
Computer used by riya is Acer
```

```
name=["Ram","Raj","Seema","riya"]
Computers=["Dell","HP","Apple","Acer"]

for i in range(0,len(name)):
    temp="Computer used by {} is {}"
    print(temp.format(name[i],Computers[i]))
```

```
Computer used by Ram is Dell
Computer used by Raj is HP
Computer used by Seema is Apple
Computer used by riya is Acer
```

```
import time

def do_my_sum(xs):
    sum = 0
    for v in xs:
        sum += v
    return sum

sz = 10000000 # Lets have 10 million elements in the list
testdata = range(sz)
t0 = time.clock()
my_result = do_my_sum(testdata)
t1 = time.clock()
print("my_result = {0} (time taken = {1:.4f} seconds)"
    .format(my_result, t1-t0))

t2 = time.clock()
their_result = sum(testdata)
t3 = time.clock()
print("their_result = {0} (time taken = {1:.4f} seconds)"
    .format(their_result, t3-t2))
```

```
my_result = 49999995000000 (time taken = 0.8088 seconds)
their_result = 49999995000000 (time taken = 0.4642 seconds)
```

The math module

- The math module contains the kinds of mathematical functions.
- Mathematical functions are "pure" functions

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.sqrt(2.0)
1.4142135623730951
>>> math.radians(90)
1.5707963267948966
>>> math.sin(math.radians(90))
1.0
```

Creating your own modules

- All we need to do to create our own modules is to save our script as a file with a .py extension.

```
def printh():
    print("Hello World")
location="Bangalore"
```

Save as Hello.py

```
import Hello

a=Hello.printh()
print(a)

place=Hello.location
print(place)
```

Namespaces

- A namespace is a collection of identifiers that belong to a module, or to a function,

```
# Module1.py

question = "What is the meaning of Life, the Universe, and Everything?"
answer = 42

# Module2.py

question = "What is your quest?"
answer = "To seek the holy grail."
```

Namespaces

```
1 import module1
2 import module2
3
4 print(module1.question)
5 print(module2.question)
6 print(module1.answer)
7 print(module2.answer)
```

will output the following:

```
What is the meaning of Life, the Universe, and Everything?
What is your quest?
42
To seek the holy grail.
```

Namespaces

Functions also have their own namespaces:

```
def f():
    n = 7
    print("printing n inside of f:", n)

def g():
    n = 42
    print("printing n inside of g:", n)

n = 11
print("printing n before calling f:", n)
f()
print("printing n after calling f:", n)
g()
print("printing n after calling g:", n)
```

```
printing n before calling f: 11
printing n inside of f: 7
printing n after calling f: 11
printing n inside of g: 42
printing n after calling g: 11
```

Scope and lookup rules

The scope of an identifier is the region of program code in which the identifier can be accessed, or used.

There are **three important scopes** in Python:

- **Local scope** refers to identifiers declared within a function.
- **Global scope** refers to all the identifiers declared within the current module, or file.
- **Built-in scope** refers to all the identifiers built into Python—those like range and min

Scope and lookup rules

```
1  n = 10
2  m = 3
3  def f(n):
4      m = 7
5      return 2*n+m
6
7  print(f(5), n, m)
```

Attributes and the dot operator

- Variables defined inside a module are called attributes of the module
- Attributes are accessed using the dot operator (.)
- Modules contain functions as well as attributes, and the dot operator is used to access them in the same way.

Files

Akshata S Bhayyar, Asst. Prof. Dept. of CSE, RIT

Files

- While a program is running, its data is stored in random access memory (RAM).
- RAM is fast and inexpensive, but it is also volatile, which means that when the program ends, or the computer shuts down, data in RAM disappears.
- To make data available the next time the computer is turned on and the program is started, it has to be written to a non-volatile storage medium, such a hard drive, usb drive, or CD-RW.
- Data on non-volatile storage media is stored in named locations on the media called **files**

Files

- Working with files is a lot like working with a notebook.
  - To use a notebook, it has to be **opened.**
  - When done, it has to be **closed.**
  - While the notebook is open, it can either be **read from or written to.**
  - In either case, the **notebook holder** knows where they are.

Writing our first file

```
myfile = open("test.txt", "w")
myfile.write("My first file written from Python\n")
myfile.write("---------------------------------\n")
myfile.write("Hello, world!\n")
myfile.close()
```

Writing our first file

- Opening a file creates what we call a file handle.
- In this example, the variable **myfile** refers to the new handle object.
- The open function takes two arguments.
  - The first is the name of the file, and the
  - second is the mode.

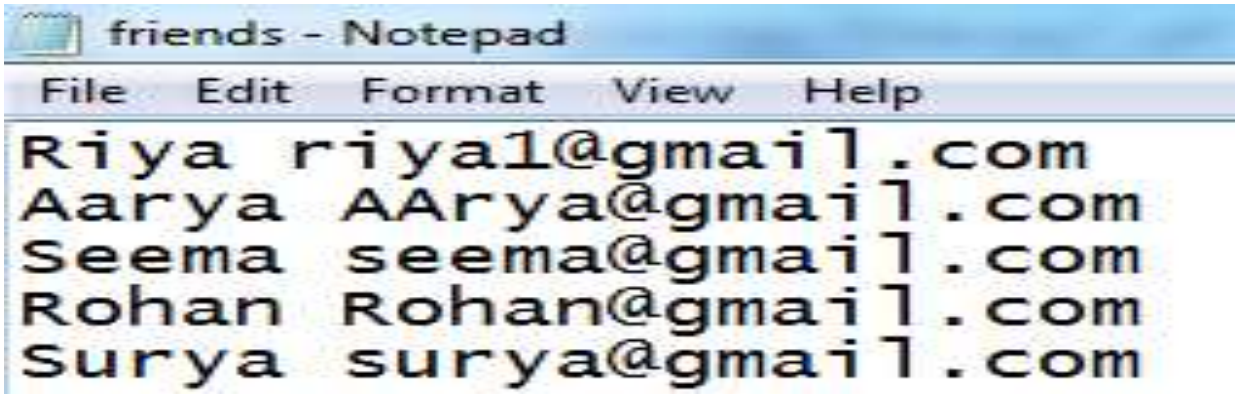
Reading a file line-at-a-time

```
mynewhandle = open("test.txt", "r")
while True: # Keep reading forever
    theline = mynewhandle.readline() # Try to read next line
    if len(theline) == 0: # If there are no more lines
        break # leave the loop

    # Now process the line we've just read
    print(theline, end="")
```

```
My first file written from Python
---------------------------------
Hello, world!
```

Turning a file into a list of lines

- It is often useful to fetch data from a disk file and turn it into a list of lines.
- Suppose we have a file containing our friends and their email addresses, one per line in the file.
- But we'd like the lines sorted into **alphabetical order.**
- A good plan is to read everything into a list of lines, then **sort** the list, and then write the sorted list back to another file:



Turning a file into a list of lines

MyF1.py - D:/Akshata/Python/Lab/MyF1.py (3.5.2)

File Edit Format Run Options Window Help

```
f = open("friends.txt", "r")
xs = f.readlines()
f.close()

xs.sort()

g = open("sortedfriends.txt", "w")
for v in xs:
    g.write(v)
g.close()
```

sortedfriends - Notepad

File Edit Format View Help

```
Aarya AArya@gmail.com
Riya riya1@gmail.com
Rohan Rohan@gmail.com
Seema seema@gmail.com
Surya surya@gmail.com
```

Reading the whole file at once

- Another way of working with text files is to read the complete contents of the file into a string, and then to use our string-processing skills to work with the contents.

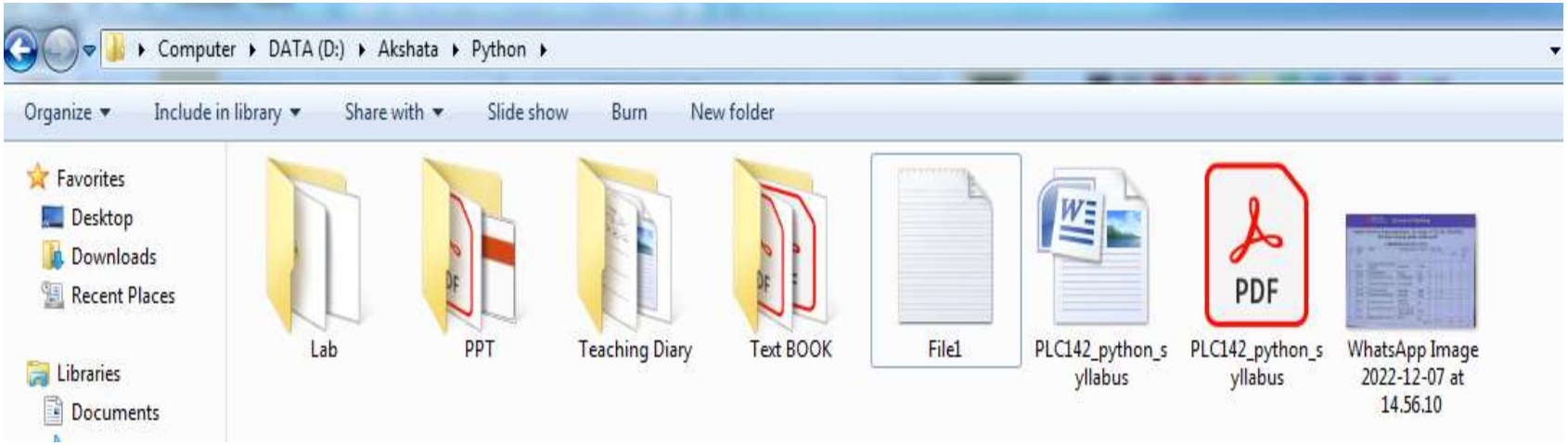
```
f = open("somefile.txt")
content = f.read()
f.close()

words = content.split()
print("There are {0} words in the file.".format(len(words)))

for i in range(len(words)):
    print(words[i])
```

Reading the whole file at once

```
There are 31 words in the file.
Another
way
of
working
with
text
files
is
to
read
the
complete
contents
of
the
file
into
a
string,
and
then
to
use
our
string-processing
skills
to
work
with
the
contents.
```

Your file paths may need to be explicitly named.

- In the above example, we're assuming that the file somefile.txt is in the same directory as your Python source code.
- If this is not the case, you may need to provide a full or a relative path to the file.



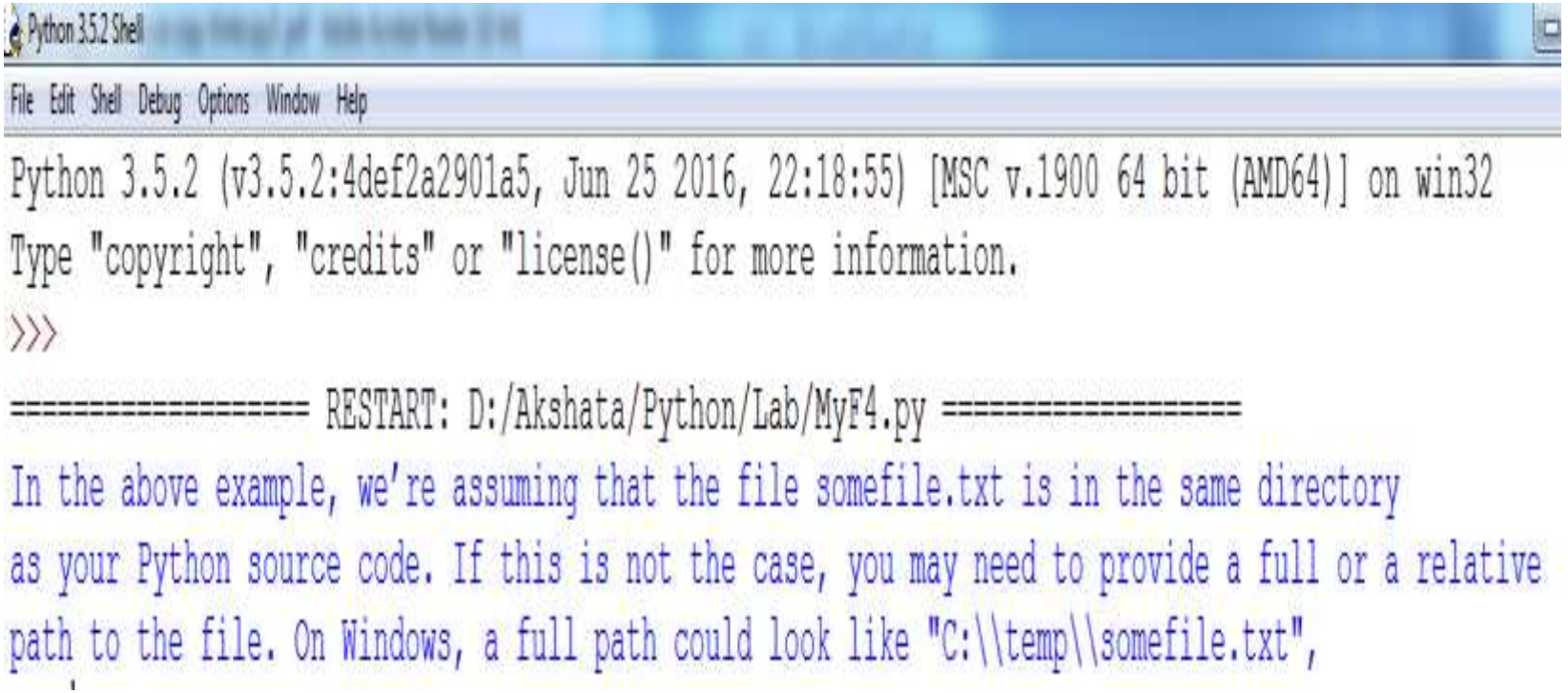
Your file paths may need to be explicitly named.

MyF4.py - D:/Akshata/Python/Lab/MyF4.py (3.5.2)

File Edit Format Run Options Window Help

```
mynewhandle = open("D:\Akshata\Python\File1.txt", "r")
while True: # Keep reading forever
    theline = mynewhandle.readline() # Try to read next line
    if len(theline) == 0: # If there are no more lines
        break # leave the loop

    # Now process the line we've just read
    print(theline, end="")
```

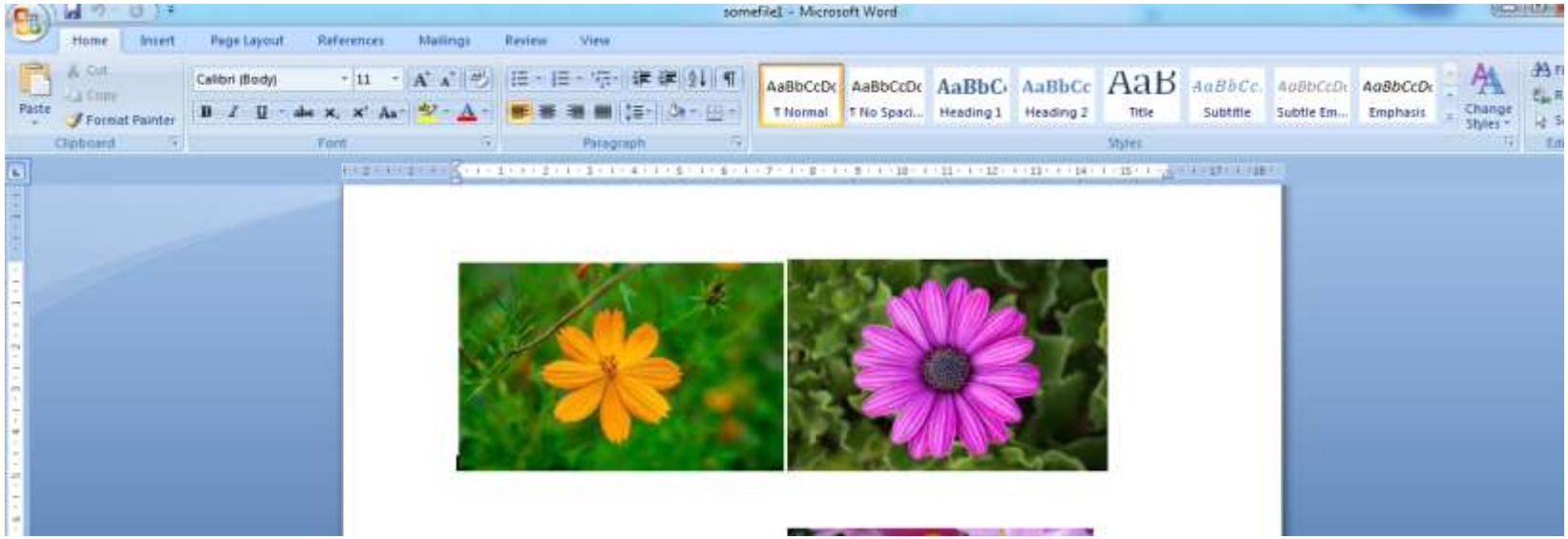
Your file paths may need to be explicitly named.



Working with binary files

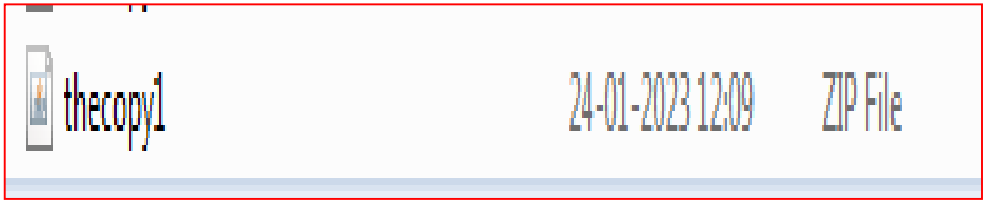
- Files that hold photographs, videos, zip files, executable programs, etc. are called **binary files**.
- They're not organized into lines, and cannot be opened with a normal text editor.
- Python works just as easily with binary files, but when we read from the file we're going to get bytes back rather than a string. Here we'll copy one binary file to another:

Working with binary files



Working with binary files

```
f = open("somefile.zip", "rb")
g = open("thecopy.zip", "wb")

while True:
      buf = f.read(1024)
      if len(buf) == 0:
            break
g.write(buf)

f.close()
g.close()
```



Directories

- Files on non-volatile storage media are organized by a set of rules known as a file system.
- File systems are made up of files and directories, which are containers for both files and other directories.
- When we create a new file by opening it and writing, the new file goes in the current directory
- Similarly, when we open a file for reading , Python looks for it in the current directory.
- If we want to open a file somewhere else, we have to specify the path to the file, which is the name of the directory (or folder) where the file is located:

Directories

```
>>> wordsfile = open("/usr/share/dict/words", "r")
>>> wordlist = wordsfile.readlines()
>>> print(wordlist[:6])
['\n', 'A\n', "A's\n", 'AOL\n', "AOL's\n", 'Aachen\n']
```

A Windows path might be

"c:/temp/words.txt" or "c:\\temp\\words.txt".

Because backslashes are used to escape things like newlines and tabs, we need to write two backslashes in a literal string to get one!

We cannot use / or \ as part of a filename; they are reserved as a delimiter between directory and filenames.

What about fetching something from the web?

```
import urllib.request
image_url = "https://media.geeksforgeeks.org/wp-content/cdn-uploads/20200623173636/Python-Tutorial1.png

urllib.request.urlretrieve(image_url,"image.png")
```

The urlretrieve function—just one call—could be used to download any kind of content from the Internet.


image - Windows Photo Viewer
File
Print
E-mail
Burn
Open
PYTHON
TUTORIAL
10:15
07-02-2023

Different example.

```
import urllib.request

def retrieve_page(url):
    """ Retrieve the contents of a web page.
        The contents is converted to a string before returning it.
    """
    my_socket = urllib.request.urlopen(url)
    dta = str(my_socket.readall())
    my_socket.close()
    return dta

the_text = retrieve_page("http://xml.resource.org/public/rfc/txt/rfc793.txt"
print(the_text)
```

Different example.

- Opening the remote url returns what we call a **socket.**
- This is a handle to our end of the connection between our program and the remote web server.
- We can call read, write, and close methods on the socket object in much the same way as we can work with a file handle.

Algorithms

- Linear search
- Binary search
- merging two sorted lists.

Binary Search

```
def binary_search(arr, low, high, x):

    # Check base case
    if high >= low:

        mid = (high + low) // 2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it can only
        # be present in left subarray
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)

        # Else the element can only be present in right subarray
        else:
            return binary_search(arr, mid + 1, high, x)

    else:
        # Element is not present in the array
        return -1
```

Binary Search Contd....

```
# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binary_search(arr, 0, len(arr)-1, x)

if result != -1:
    print("Element is present at index", str(result))
else:
    print("Element is not present in array")
```

```
Element is present at index 3
```

Merging two sorted lists.

```
l1=[5,8,2,0,-1]
l2=[66,9,6,33,-8]
l1.sort()
print(l1)
l2.sort()
print(l2)
size_1 = len(l1)
size_2 = len(l2)

res = []
i, j = 0, 0

while i < size_1 and j < size_2:
    if l1[i] < l2[j]:
        res.append(l1[i])
        i += 1

    else:
        res.append(l2[j])
        j += 1

res = res + l1[i:] + l2[j:]

# printing result
print("The combined sorted list is : " + str(res))
```

```
[-1, 0, 2, 5, 8]
[-8, 6, 9, 33, 66]
The combined sorted list is : [-8, -1, 0, 2, 5, 6, 8, 9, 33, 66]
```