Fundamentals of Programming

Unit 6
Functions & Basic I/O

CUSCS

Library

Make use of the wisdom from the others!

Library

Library

a reusable chunk of code that we may want to include in our programs/ projects

- ❑ Relationship with Modules and Packages
  - a module → a file with some Python code
    - a library is a collection of modules
  - a package → a directory for sub packages and modules
    - a package is a library that can be installed using a package manager

Python Standard Library

Python Standard Library

a collection of exact syntax, token, and semantics of Python

❑ Features

- come bundled with core Python distribution
- more than 200 core modules sit at the heart of the standard library
- access a growing collection of several thousand components from the Python Package Index (PyPI)

Common Libraries

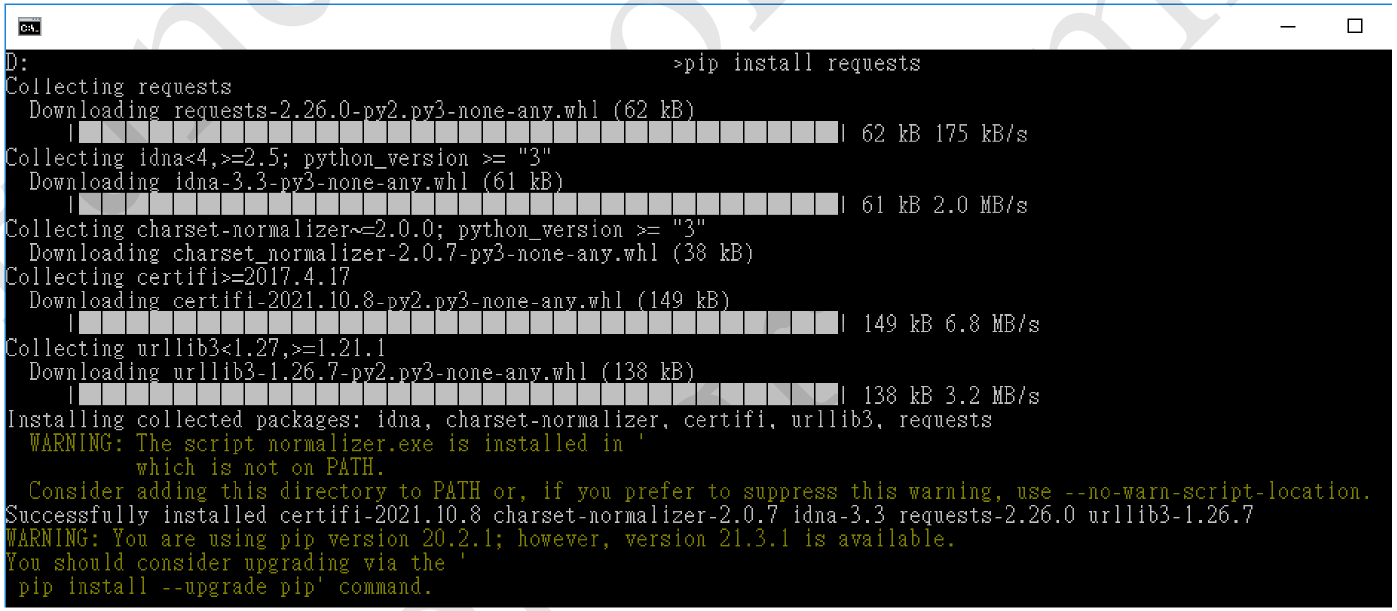
- ❑ Matplotlib
  - a numerical plotting library and working with data analysing
- ❑ Pandas
  - fast, expressive, and flexible data structures to easily work with structured and time-series data
- ❑ Requests
  - a Python Library that let us send HTTP/1.1 requests, add headers, form data, multipart files, and parameters

Common Libraries

❑ Example

- Install required libraries such as ‘requests’

Programming Code
pip install requests



```
D:                                                       >pip install requests
Collecting requests
  Downloading requests-2.26.0-py2.py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 175 kB/s
Collecting idna<4,>=2.5; python_version >= "3"
  Downloading idna-3.3-py3-none-any.whl (61 kB)
     |████████████████████████████████| 61 kB 2.0 MB/s
Collecting charset-normalizer~=2.0.0; python_version >= "3"
  Downloading charset_normalizer-2.0.7-py3-none-any.whl (38 kB)
Collecting certifi>=2017.4.17
  Downloading certifi-2021.10.8-py2.py3-none-any.whl (149 kB)
     |████████████████████████████████| 149 kB 6.8 MB/s
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.7-py2.py3-none-any.whl (138 kB)
     |████████████████████████████████| 138 kB 3.2 MB/s
Installing collected packages: idna, charset-normalizer, certifi, urllib3, requests
  WARNING: The script normalizer.exe is installed in '
           which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed certifi-2021.10.8 charset-normalizer-2.0.7 idna-3.3 requests-2.26.0 urllib3-1.26.7
WARNING: You are using pip version 20.2.1; however, version 21.3.1 is available.
You should consider upgrading via the '
 pip install --upgrade pip' command.
```

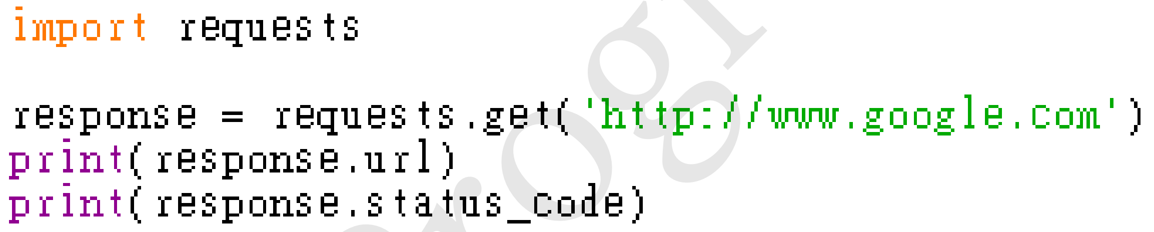
Common Libraries

- Example
  - Input the following code and see the results

Programming Code

```
import requests

response = requests.get('http://www.google.com')
print(response.url)
print(response.status_code)
```



```
import requests

response = requests.get('http://www.google.com')
print(response.url)
print(response.status_code)
```

Functions

Gather useful operations together!

Functions

Functions

a block of organized, reusable code that is used to perform a single, related action

❑ Features

- a function can be called multiple times to provide reusability and modularity to the Python program
- help to programmer to break the program into the smaller part
- organize the code very effectively and avoids the repetition of the code

Functions

❑ Rules to Define a Function

- begin with the keyword ***def*** followed by the function name and parentheses (( ))
- any input parameters or arguments should be placed within these parentheses
- the code block within every function starts with a colon (:) and is indented
- the statement ***return [expression]*** exits a function, optionally passing back an expression to the caller
- a ***return*** statement with no arguments is the same as return None

Functions

- ❑ Syntax

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

- ❑ Defining a function
  - give function a name, specifies the parameters that are to be included in the function and structures the blocks of code
- ❑ Calling a Function
  - execute the function by calling it from another function or directly from the Python prompt after the function is finalized

Arguments

Arguments

Information can be passed into functions as arguments

❑ Features

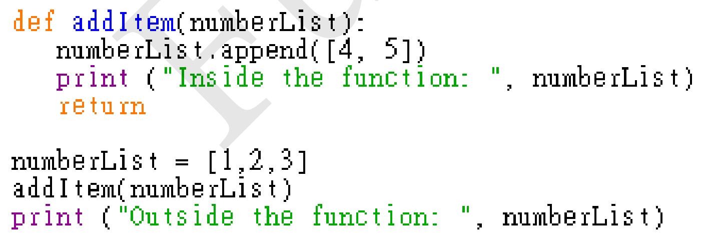
- specified after the function name, inside the parentheses
- can add as many arguments as we want, just separate them with a comma
- a parameter → the variable listed inside the parentheses in the function definition
- an argument → the value that is sent to the function when it is called

Pass By Reference

❑ Pass By Reference

- all parameters (arguments) in the Python language are passed by reference
- if we change what a parameter refers to within a function, the change also reflects back in the calling function

❑ Example



```
def addItem(numberList):
   numberList.append([4, 5])
   print ("Inside the function: ", numberList)
   return

numberList = [1,2,3]
addItem(numberList)
print ("Outside the function: ", numberList)
```

Programming Code

```
def addItem(numberList):
  numberList.append([4, 5])
  print ("Inside the function: ", numberList)
  return

numberList = [1,2,3]
addItem(numberList)
print ("Outside the function: ", numberList)
```

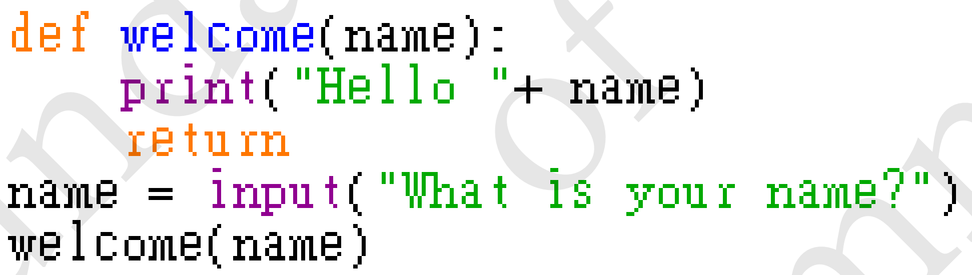
Arguments

- ❑ Types of Arguments
  1. Required arguments
  2. Keyword arguments
  3. Default arguments
  4. Variable-length arguments

- ❑ Required arguments
  - the arguments passed to a function in correct positional order
  - the number of arguments in the function call should match exactly with the function definition

Required arguments

- Example
  - Input the following code and see the results



```
def welcome(name):
    print("Hello "+ name)
    return
name = input("What is your name?")
welcome(name)
```

Programming Code

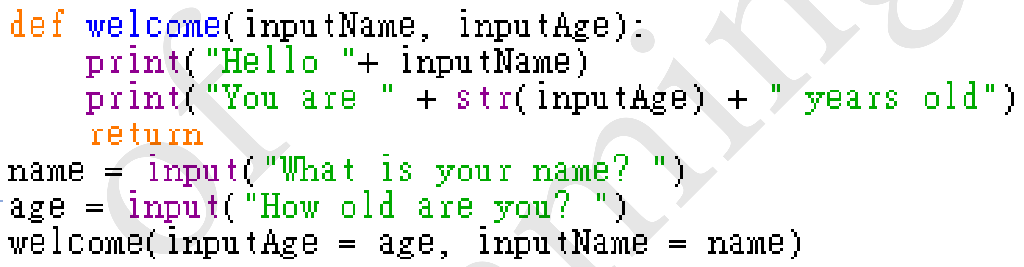
```
def welcome(name):
    print("Hello "+ name)
    return
name = input("What is your name?")
welcome(name)
```

Arguments

- ❑ Keyword Arguments
  - related to the function calls
  - when we use keyword arguments in a function call, the caller identifies the arguments by the parameter name
  - allows us to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters

Keyword Arguments

- Example
  - Input the following code and see the results



```
def welcome(inputName, inputAge):
    print("Hello "+ inputName)
    print("You are " + str(inputAge) + " years old")
    return
name = input("What is your name? ")
age = input("How old are you? ")
welcome(inputAge = age, inputName = name)
```

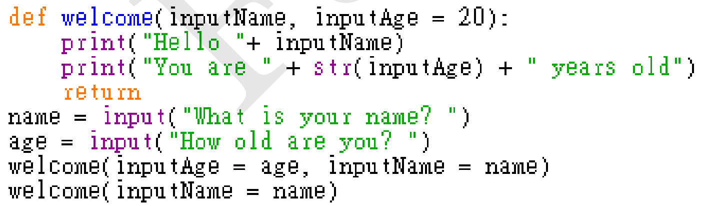
Programming Code

```
def welcome(inputName, inputAge):
    print("Hello "+ inputName)
    print("You are " + str(inputAge) + " years old")
    return
name = input("What is your name? ")
age = input("How old are you? ")
welcome(inputAge = age, inputName = name)
```

Arguments

- Default Arguments
  - an argument that assumes a default value if a value is not provided in the function call for that argument

- Example



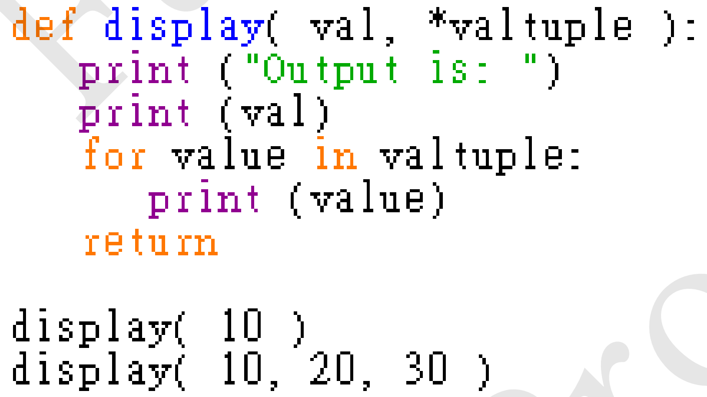
```
def welcome(inputName, inputAge = 20):
    print("Hello "+ inputName)
    print("You are " + str(inputAge) + " years old")
    return
name = input("What is your name? ")
age = input("How old are you? ")
welcome(inputAge = age, inputName = name)
welcome(inputName = name)
```

<u>**Programming Code**</u>

```
def welcome(inputName, inputAge = 20):
    print("Hello "+ inputName)
    print("You are " + str(inputAge) + " years old")
    return
name = input("What is your name? ")
age = input("How old are you? ")
welcome(inputAge = age, inputName = name)
welcome(inputName = name)
```

Arguments

- ❑ Variable-length Arguments
  - may need to process a function for more arguments than we specified while defining the function
  - these arguments are called variable-length arguments and not named in the function definition, unlike required and default arguments
- ❑ Example



```
def display( val, *valtuple ):
   print ("Output is: ")
   print (val)
   for value in valtuple:
      print (value)
   return

display( 10 )
display( 10, 20, 30 )
```

Programming Code

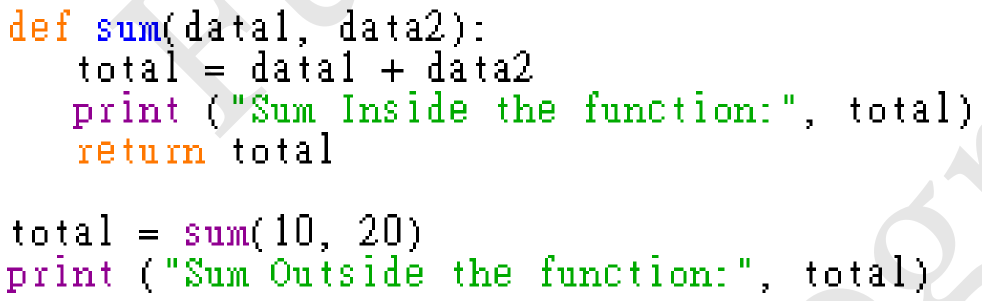
```
def display( val, *valtuple ):
  print ("Output is: ")
  print (val)
  for value in valtuple:
    print (value)
  return
display( 10 )
display( 10, 20, 30 )
```

return Statement

❑ *return* Statement

- exit a function, optionally passing back an expression to the caller
- a return statement with no arguments is the same as return None

❑ Example



```
def sum(data1, data2):
    total = data1 + data2
    print ("Sum Inside the function:", total)
    return total

total = sum(10, 20)
print ("Sum Outside the function:", total)
```

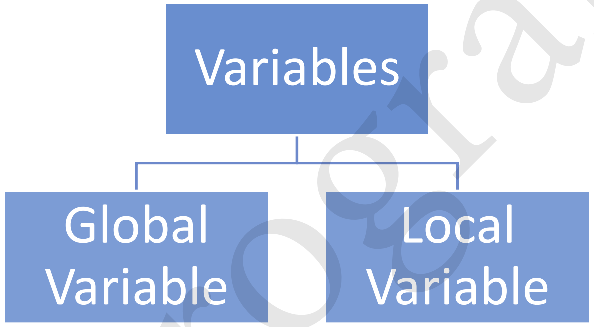
<u>**Programming Code**</u>

```
def sum(data1, data2):
  total = data1 + data2
  print ("Sum Inside the function:", total)
  return total

total = sum(10, 20)
print ("Sum Outside the function:", total)
```

Scope of Variables

- ❑ Scope of Variables
  - all variables in a program may not be accessible at all locations in that program
  - depends on where we have declared a variable
  - the scope of a variable determines the portion of the program where the programmer can access a particular identifier
  - 2 basic scopes of variables in Python:



Scope of Variables

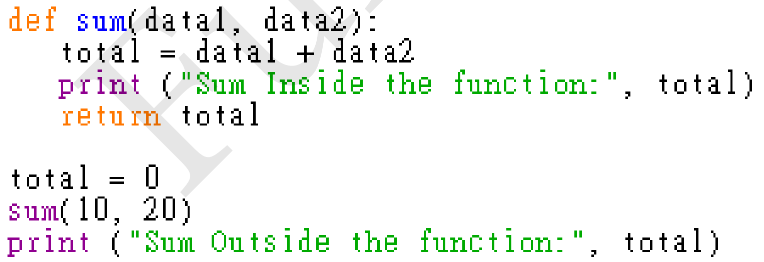
- ❑ General Concepts
  - ▪ variables that are defined inside a function body have a local scope, and those defined outside have a global scope

- ❑ Local Variables
  - ▪ can be accessed only inside the function in which they are declared

- ❑ Global Variables
  - ▪ can be accessed throughout the program body by all functions

Scope of Variables

- Example
  - Input the following code and see the results



```
def sum(data1, data2):
    total = data1 + data2
    print ("Sum Inside the function:", total)
    return total

total = 0
sum(10, 20)
print ("Sum Outside the function:", total)
```

Programming Code

```
def sum(data1, data2):
  total = data1 + data2
  print ("Sum Inside the function:", total)
  return total

total = 0
sum(10, 20)
print ("Sum Outside the function:", total)
```

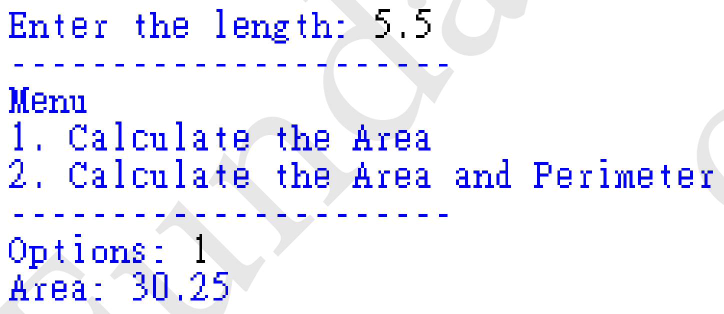
Functions

- Class Activity
  - Ask the user to input the length of a square.
  - Display a menu with two options which are calculating the area only and calculating both the area and perimeter respectively.
  - Calculations of the area and perimeters should be done with functions.
  - Ask the user to choose the appropriate options, otherwise, it will be treated as the invalid input.
  - Display the results as shown below.

Functions

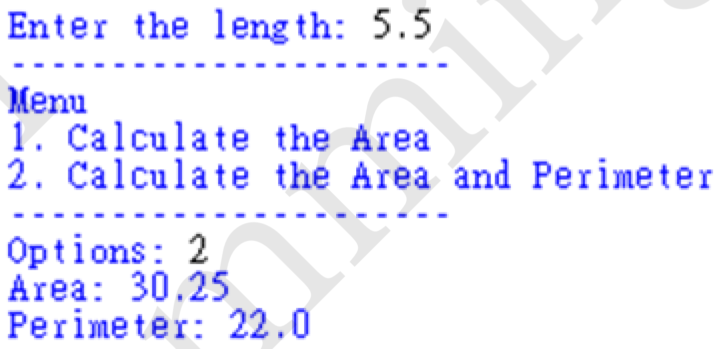
- Class Activity

Sample Output 1



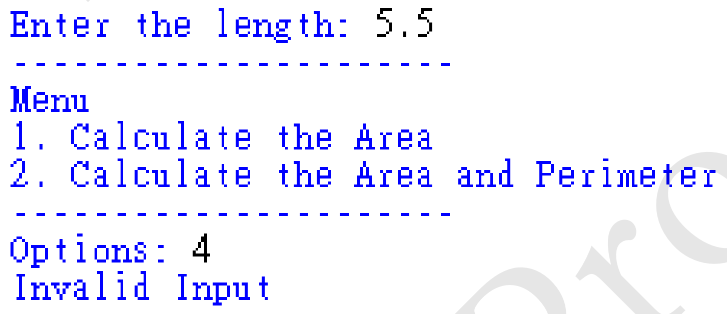
```
Enter the length: 5.5
-----------------------
Menu
1. Calculate the Area
2. Calculate the Area and Perimeter
-----------------------
Options: 1
Area: 30.25
```

Sample Output 2



```
Enter the length: 5.5
-----------------------
Menu
1. Calculate the Area
2. Calculate the Area and Perimeter
-----------------------
Options: 2
Area: 30.25
Perimeter: 22.0
```

Sample Output 3



```
Enter the length: 5.5
-----------------------
Menu
1. Calculate the Area
2. Calculate the Area and Perimeter
-----------------------
Options: 4
Invalid Input
```

Basic I/O Handling

Try to work with files in your computer!

File Handling

- File
  - a named location on disk to store related information
  - can access the stored information (non-volatile) after the program termination

- File Handling
  - the data needs to be stored permanently into the file

- Modes in Python
  - files are treated in two modes as text or binary
  - the file may be in the text or binary format, and each line of a file is ended with the special character

Opening a File

Opening a File

open() function that accepts two arguments, file name and access mode in which the file is accessed

❑ Features

- the function returns a file object which can be used to perform various operations like reading, writing, etc
- files can be accessed using various modes like read, write, or append

Opening a File

❑ Common Access Mode

Access Mode	Description
r	▪ open the file to read-only mode ▪ the file pointer exists at the beginning ▪ the file is by default open in this mode if no access mode is passed
r+	▪ open the file to read and write both ▪ the file pointer exists at the beginning of the file
w	▪ open the file to write only ▪ overwrite the file if previously exists or creates a new one if no file exists with the same name ▪ the file pointer exists at the beginning of the file

Opening a File

❑ Common Access Mode

Access Mode	Description
w+	▪ open the file to write and read both ▪ different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file ▪ create a new file if no file exists ▪ the file pointer exists at the beginning of the file
a	▪ open the file in the append mode ▪ the file pointer exists at the end of the previously written file if exists any ▪ create a new file if no file exists with the same name
a+	▪ open a file to append and read both ▪ the file pointer remains at the end of the file if a file exists ▪ create a new file if no file exists with the same name

Closing a File

Closing a File

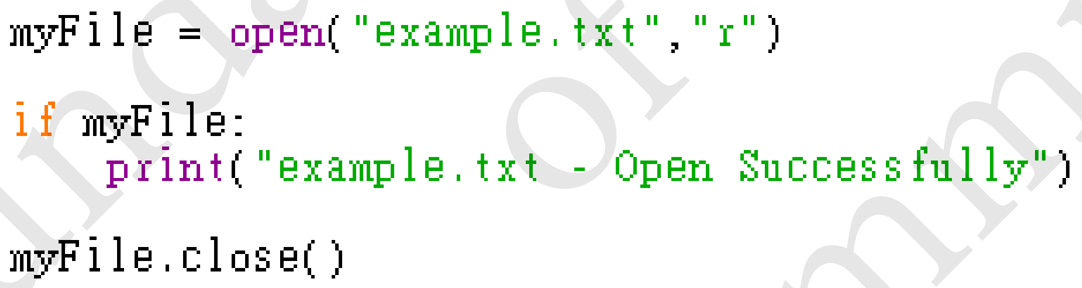
close the file using the ***close()*** method once all the operations are done on the file

❑ Features

- perform any operation on the file externally using the file system which is the currently opened in Python → it is good practice to close the file once all the operations are done
- any unwritten information gets destroyed once the ***close()*** method is called on a file object

Basic I/O

- Example
  - Input the following code and see the results



```
myFile = open("example.txt","r")

if myFile:
  print("example.txt - Open Successfully")

myFile.close()
```

<u>Programming Code</u>

```
myFile = open("example.txt","r")
if myFile:
    print("example.txt - Open Successfully")
myFile.close()
```

with statement

with statement

used in the scenario where a pair of statements is to be executed with a block of code in between

❑ Features

- introduced in python 2.5
- useful in the case of manipulating the files
- provide the guarantee to close the file regardless of how the nested block exits

Reading and Writing a File

Reading a File

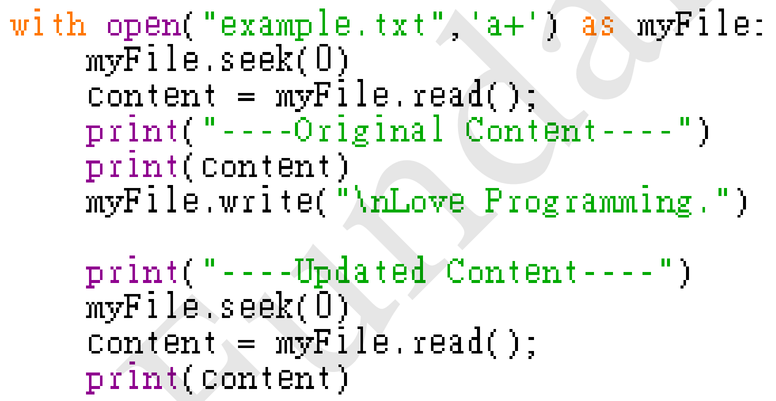
read() method reads a string from an open file
Python strings can have binary data apart from text data

Writing a File

write() method writes any string to an open file
write() method does not add a newline character ('\n') to the end of the string

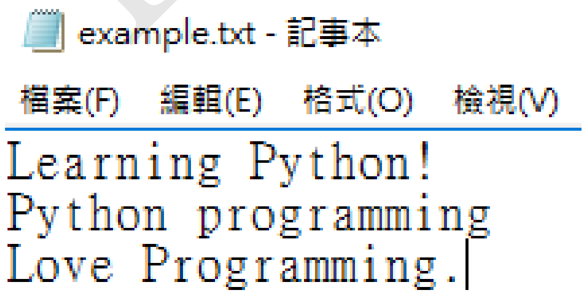
Basic I/O

- Example
  - Input the following code and see the results



```
with open("example.txt",'a+') as myFile:
    myFile.seek(0)
    content = myFile.read();
    print("----Original Content----")
    print(content)
    myFile.write("\nLove Programming.")

    print("----Updated Content----")
    myFile.seek(0)
    content = myFile.read();
    print(content)
```



```
Learning Python!
Python programming
Love Programming.
```

<u>Programming Code</u>

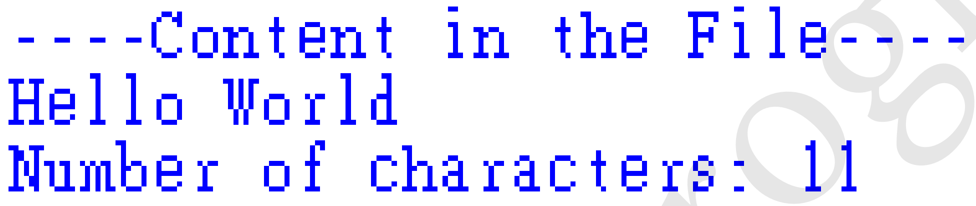
```
with open("example.txt",'a+') as myFile:
    myFile.seek(0)
    content = myFile.read();
    print("----Original Content----")
    print(content)
    myFile.write("\nLove Programming.")

    print("----Updated Content----")
    myFile.seek(0)
    content = myFile.read();
    print(content)
```

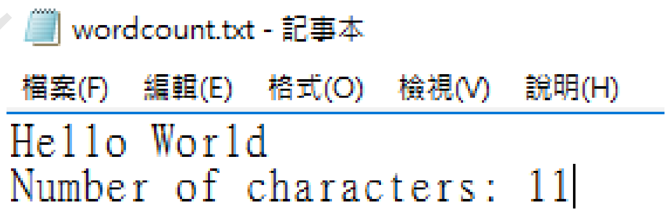
Basic I/O

❑ Class Activity

- Create a “wordcount.txt” text file in the location as same as that of your python scripts.
- Ask user to input a sentence and the program should write that sentence into the text file with calculating the number of characters in that sentence.
- The original content should be removed but only new content exists in the text file.
- Display the results as shown below.



```
----Content in the File----
Hello World
Number of characters: 11
```



```
Hello World
Number of characters: 11
```

Searching & Sorting

Find the key element or sort the data!

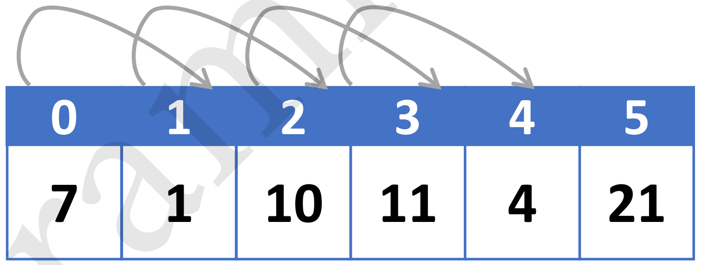
Linear Search

Linear Search

go across every element in the data structure and match it with the value we are searching for

❑ Features

- the simplest approach
- inefficient and rarely used



Linear Search

- ❑ Example
  - Input the following code and see the results

Programming Code

```
def linearSearch(values, search_for):
  search_at = 0
  search_res = False
  while search_at < len(values) and search_res == False:
    if values[search_at] == search_for:
      search_res = True
    else:
      search_at = search_at + 1
  return search_res
mylist = [27, 11, 50, 9]
print(linearSearch(mylist, 11))
print(linearSearch(mylist, 80))
```

Linear Search

- Example
  - Input the following code and see the results

```
def linearSearch(values, search_for):
    search_at = 0
    search_res = False
    while search_at < len(values) and search_res == False:
        if values[search_at] == search_for:
            search_res = True
        else:
            search_at = search_at + 1
    return search_res
mylist = [27, 11, 50, 9]
print(linearSearch(mylist, 11))
print(linearSearch(mylist, 80))
```

Insertion Sort

Insertion Sort

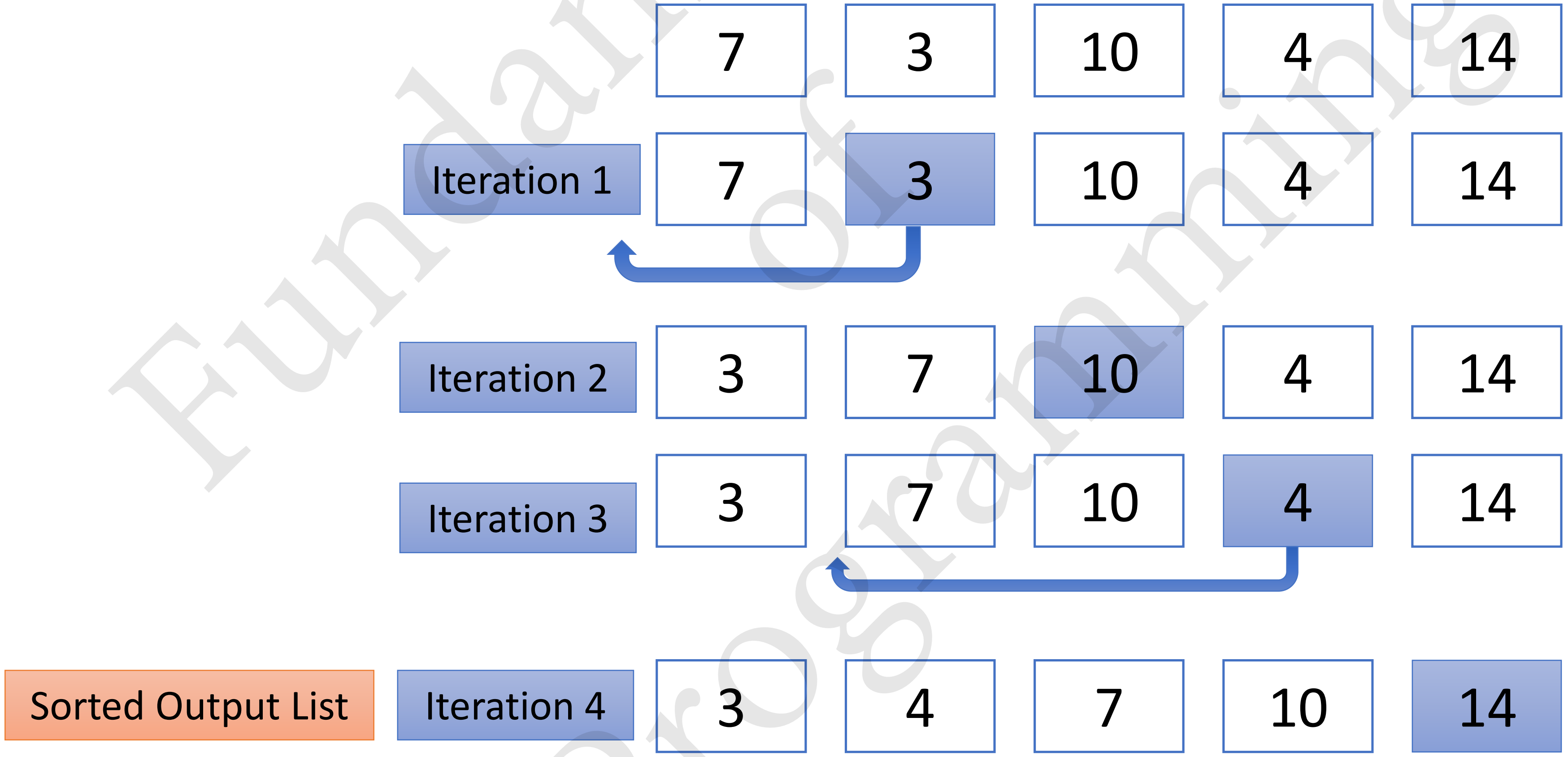
- virtually split into a sorted and an unsorted part
- values from the unsorted part are picked and placed at the correct position in the sorted part

❑ Algorithms (For Ascending Order)

1. iterate from the first element to last elements
2. compare the current element (key) to its predecessor
3. if the key element is smaller than its predecessor, compare it to the elements before → Move the greater elements one position up to make space for the swapped element

Insertion Sort

- Example



Insertion Sort

- Example
  - Input the following code and see the results

<u>Programming Code</u>

```
def insertionSort(InputList):
   for i in range(1, len(InputList)):
      j = i-1
      nxt_element = InputList[i]
   while (InputList[j] > nxt_element) and (j >= 0):
      InputList[j+1] = InputList[j]
      j=j-1
   InputList[j+1] = nxt_element
mylist = [27, 11, 50, 9]
insertionSort(mylist)
print(mylist)
```

Insertion Sort

- Example
  - Input the following code and see the results

```
def insertionSort(InputList):
    for i in range(1, len(InputList)):
        j = i-1
        nxt_element = InputList[i]

    while (InputList[j] > nxt_element) and (j >= 0):
        InputList[j+1] = InputList[j]
        j=j-1
    InputList[j+1] = nxt_element

mylist = [27, 11, 50, 9]
insertionSort(mylist)
print(mylist)
```

Hope you enjoy the class

Thank you