Sriphagaarucht Srinhichaarnun
Rachanee Kalayavinai

A-Z IN PYTHON

FROM BEGINNER TO PROGRAMMER

VARIITSRIS

CHAPTER 1 INTRODUCTION

Python is a popular programming language created by Guido van Rossum and launched in 1991. It can be used to build web applications, servers, in combination with other software to create workflows, connect to a relational database system that uses Structured Query Languages (SQLs) such as Oracle, DB2 or MySQL, as well as connect to non-relational database systems (NoSQL) such as MongoDB. This is explained in detail in the Advanced Python Programming Textbook.

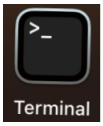
Advantages of the Python language include:

1. Being able to handle a lot of data and perform complex mathematical calculations.
2. Rapid prototyping or development of production-ready software
3. Works on different OSs (Windows, Mac, Linux, Raspberry Pi, etc.)
4. Simple grammar similar to English, making it easy to program.
5. It contains syntax that allows programmers to write programs with a smaller number of lines compared to other programming languages.
6. Works on interpreter systems, so codes can be executed as soon as it is written, which means prototyping can be done quickly.
7. Able to act in a procedural, object-oriented or function-oriented manner.
8. Python uses a line break to complete a command, as opposed to other programming languages that often use semicolons or

brackets. Relying on indents using spaces to define boundaries, such as loop boundaries, functions, and classes, other programming languages often use brackets for this purpose.

Installing Python

Many PCs and Macs have Python pre-installed. To check, launch the Terminal App on Mac, or Command Line on the PC and enter the command below.

For Mac or Linux OSs, go to the terminal, the 

console will appear to enter the command, with the host name appearing differently depending on the name defined. The author uses the name “VariitSris”, so the following line of command will be displayed.

```
VaritSris>
```

To find out if the machine has Python installed and which version, use the following command:

```
VaritSris> python3 --version
```

```
Python 3.12.3
```

Note: For Windows operating systems, go to the Command Line and use the py --version command.

If not installed, go to:

https://www.python.org/downloads

To download the latest version of python according to your operating system, as shown in Figure 1-1: download window for the latest version of

python.

Figure 1-1: download window for the latest version of

python

Download the appropriate version for your operating system (in this example we will install on macOS) using the following steps:

1. Select macOS to download. You should receive file python-3.12.3-macos11.pkg. Double click the file. The result is shown in Figure 1-2.

Figure 1-2: 1st python installation window for macOS

2. Click Continue , result is shown as Figure 1-3.

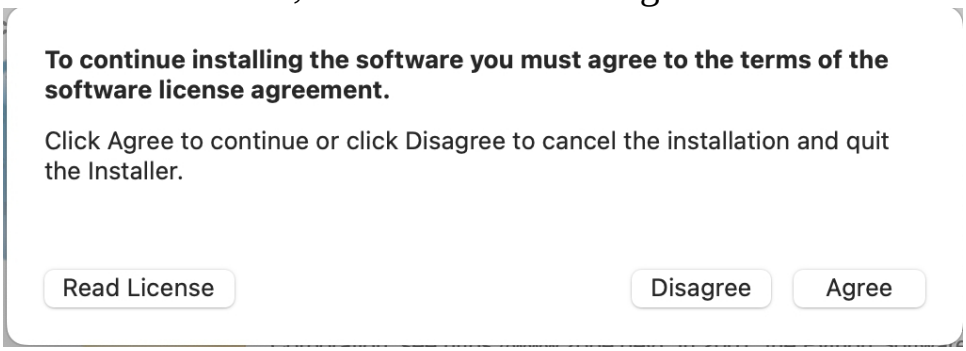


Figure 1-3: 2nd Python installation window for macOS

3. Click Agree , result is shown as Figure 1-4.

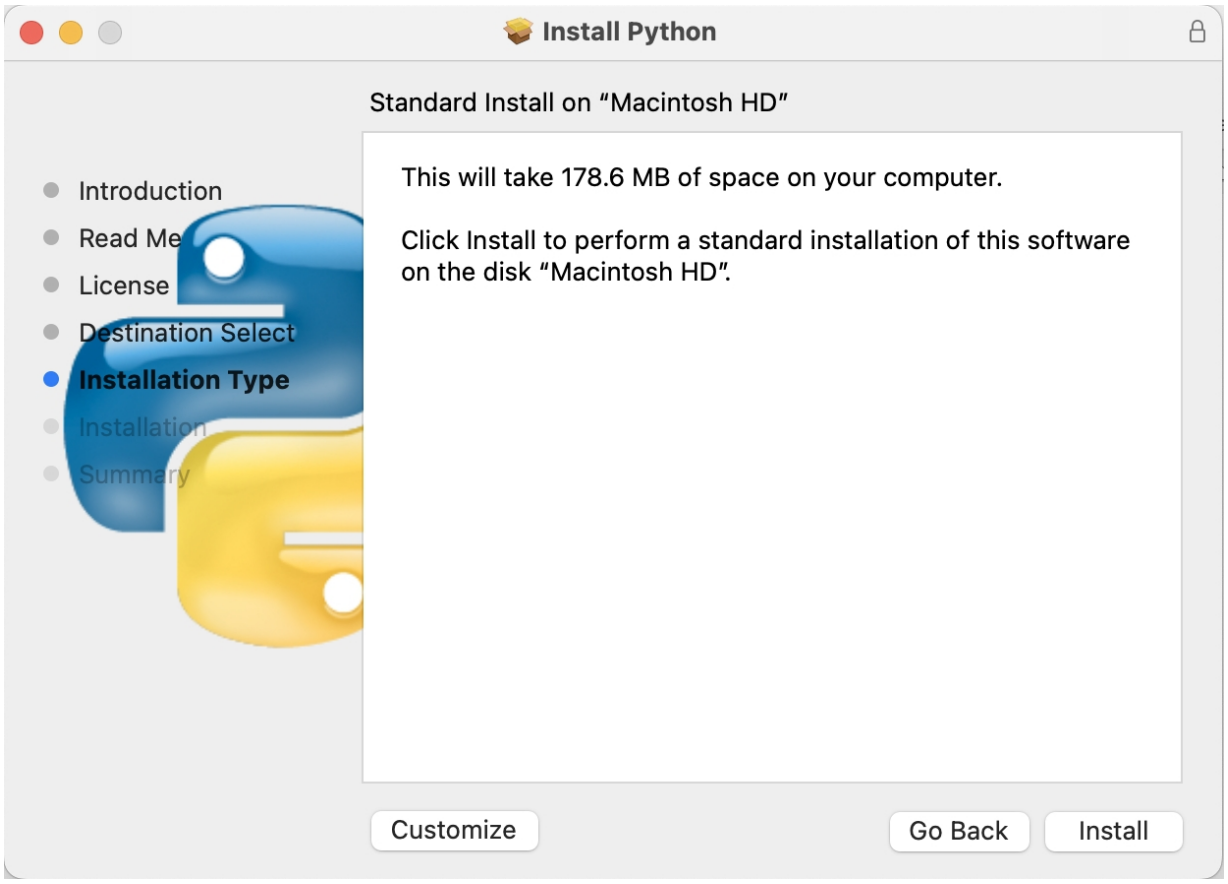


Figure 1-4: 3rd python installation window for macOS

4. Click Install
5. You will find various files for programming with python. The result is shown as Figure 1-5.

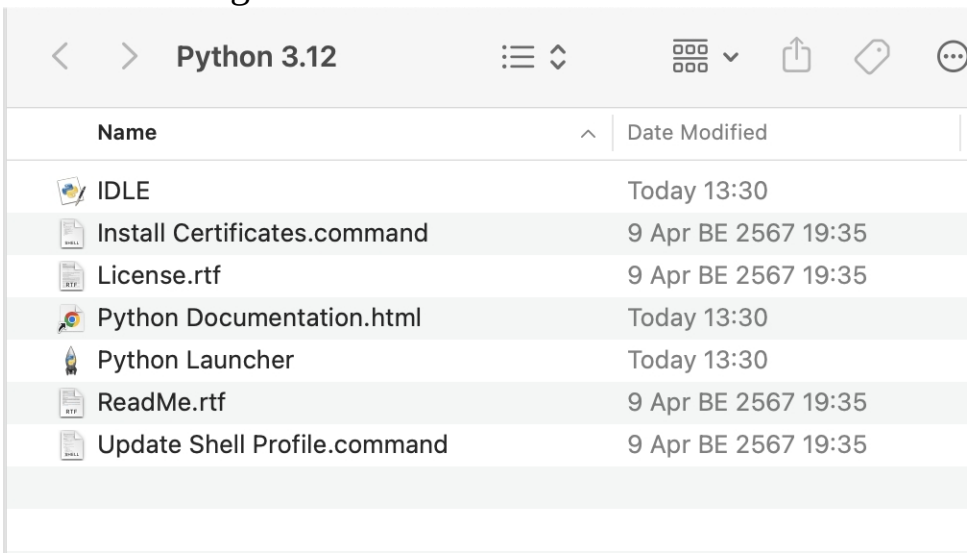


Figure 1-5: folder showing installed python files

Now you are not ready to program with python!

Installing Packages or Modules

To install Packages or Modules for Python, you need an installer program or “pip.” Therefore, it is necessary to check if pip is already installed. For machines running macOS, Linux or Windows operating systems, use command:

```
VaritSris>pip3 –version
```

Results

```
pip 24.0 from
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pip (python 3.12)
```

If not, for Mac or Linux operating systems, install the following:

```
VaritSris> python -m ensurepip –upgrade
```

```
VaritSris>python get-pip.py
```

Note: For Windows operating systems, use py -m ensurepip –upgrade, then py get-pip.py

Installing a Package

After installing pip, try installing the “numpy” package

```
VaritSris>pip3 install numpy
```

Results

```
Collecting numpy
  Downloading numpy-1.26.4-cp312-cp312-macosx_10_9_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 1.2 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp312-cp312-macosx_10_9_x86_64.whl (20.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.3/20.3 MB 6.0 MB/s eta 0:00:00
Installing collected packages: numpy
        Successfully installed numpy-1.26.4
        To check which packages or modules are installed, use the following command:
```

```
VaritSris>pip3 list
```

Results

```
Package       Version
-------       -------
        numpy         1.26.4
   pip       24.0
```

To use Terminal to program Python, follow these steps:

```
VaritSris>python3
```

Results

```
Python 3.12.3 (v3.12.3:f6650f9ad7, Apr  9 2024, 08:18:48) [Clang 13.0.0
(clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

When using the command print('Hi')

```
>>> print('Hi')
```

Results

```
Hi
```

Program Python Using Visual Studio Code

To use visual studio code in programming python, go to the following link to download the appropriate installer, according to your operating system:

https://code.visualstudio.com/download

to download the appropriate installer, according to your operating system. Visual Studio Code is supported on Windows, Linux and Mac, as shown in Figure 1-6.

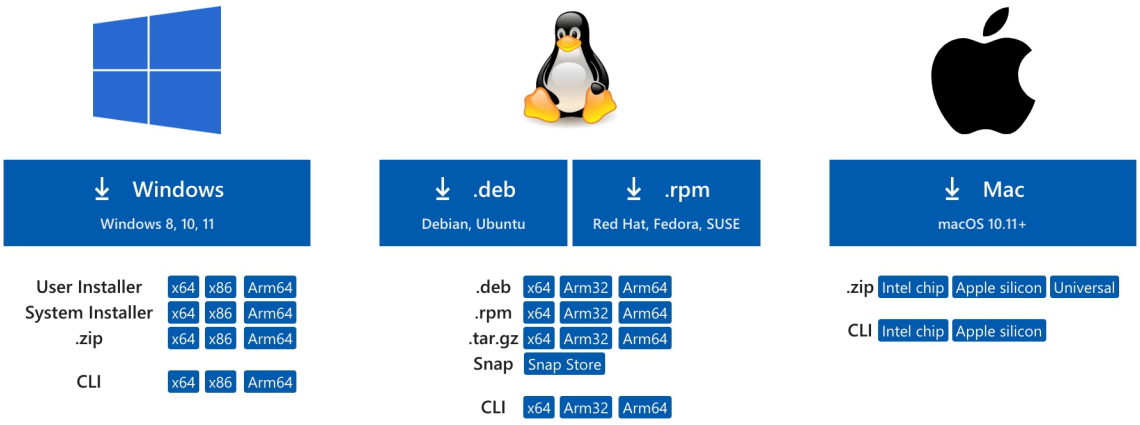
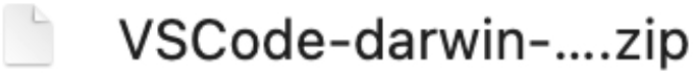


Figure 1-6: Visual Studio Code Download Window

Click to download the appropriate installer and the installation file will be shown.



Double click the downloaded file to show:

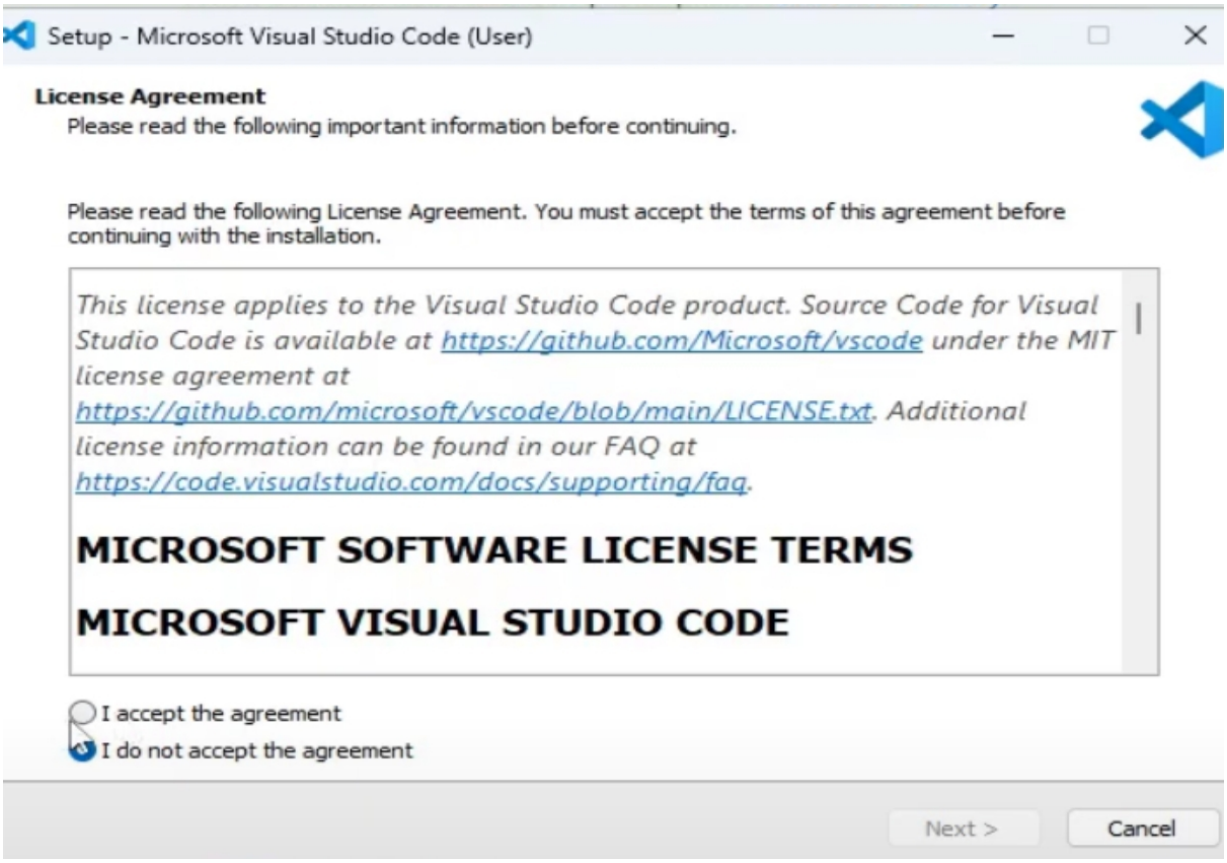


Figure 1-7: 1st installation window of Visual Studio Code

Select I accept the agreement to allow the installation to begin, the next button should become visible. Click Next > To set the desired installation path for Visual Studio Code. In this case, you can use Microsoft's the default path: C:\Users\appzs\AppData\Local\Programs\Microsoft VS Code click Next > , The result is shown as Figure 1-8.

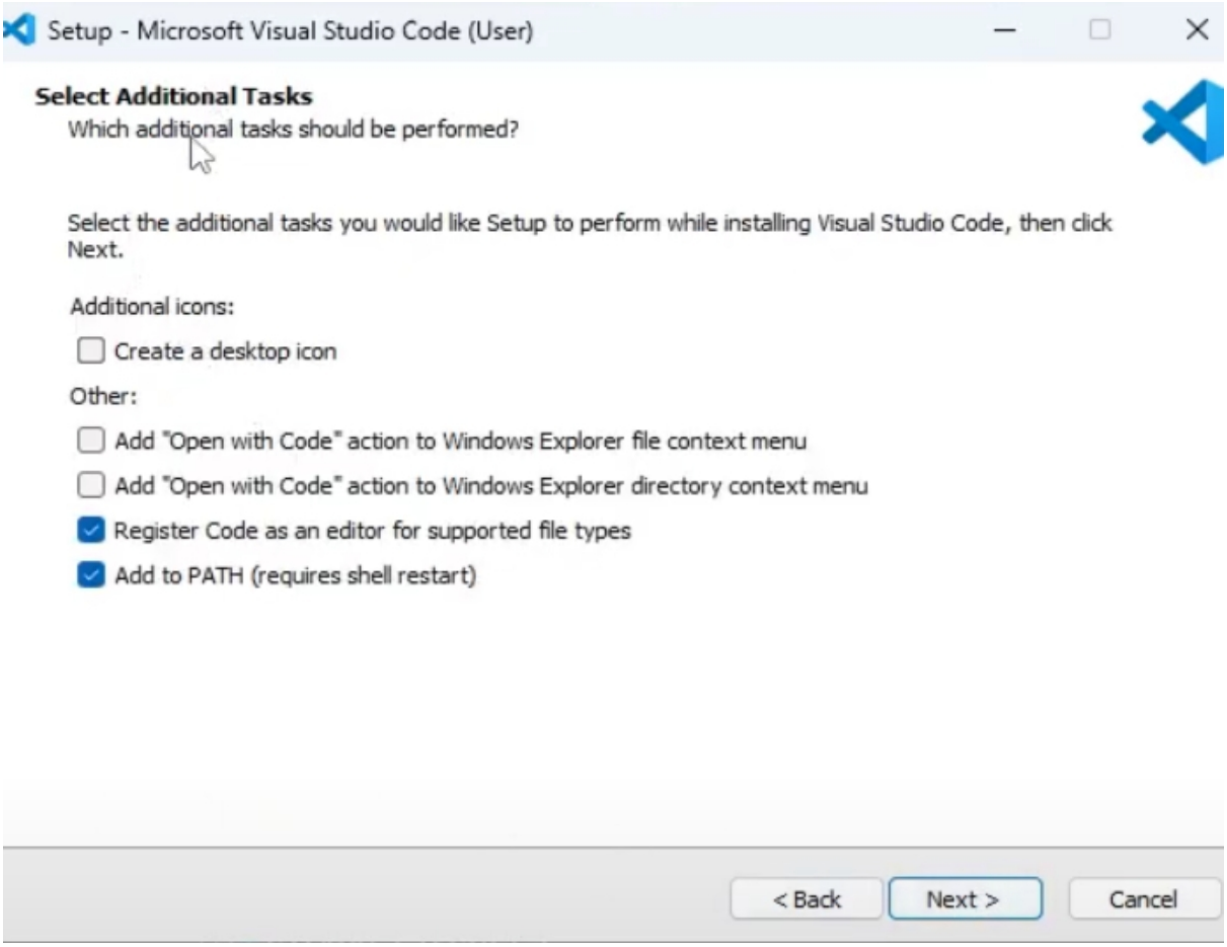
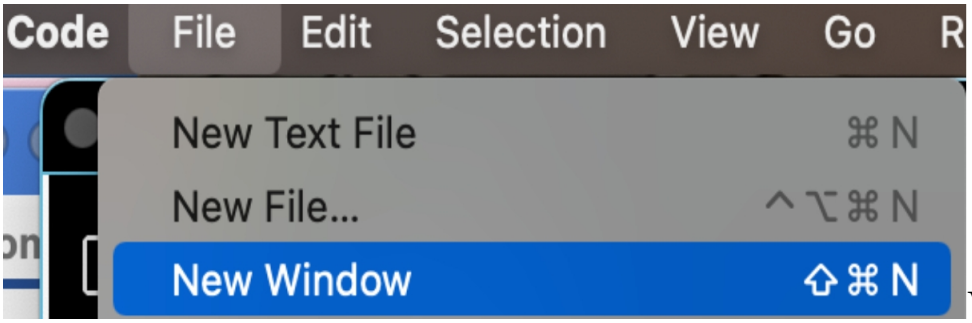


Figure 1-8: 2nd installation window of Visual Studio Code

Advantages of VS Code

1. The “IntelliSense” feature helps programmers by predicting what program code will be written in Python language. This is like having an assistant, making it easier for programmers to perform tasks such as entering code, show parameter information, quick information, member lists, variable names, and code hints, as well as clearly separating by colors that the programmer can adjust as desired (Color Themes).
2. Built-in Python debugging support makes editing programs easy.
3. It is open source and can be used free of charge.
4. Its Secure Shell (SSH) is designed for connecting to other computers within a network and is highly secure. Working with

multiple machines and networks, will be discussed in detail in the Advanced Python Programming Textbook.



VSCODE Has its own terminal that runs python, so we need to create a work folder to keep all programming files. For this textbook, we will use the folder "ExamplePythonCode." Click VSCODE, then File->New Window as show in Figure 1-9.

Figure 1-9: Starting Visual Studio Code

And the following new window will appear in Figure 1-10.

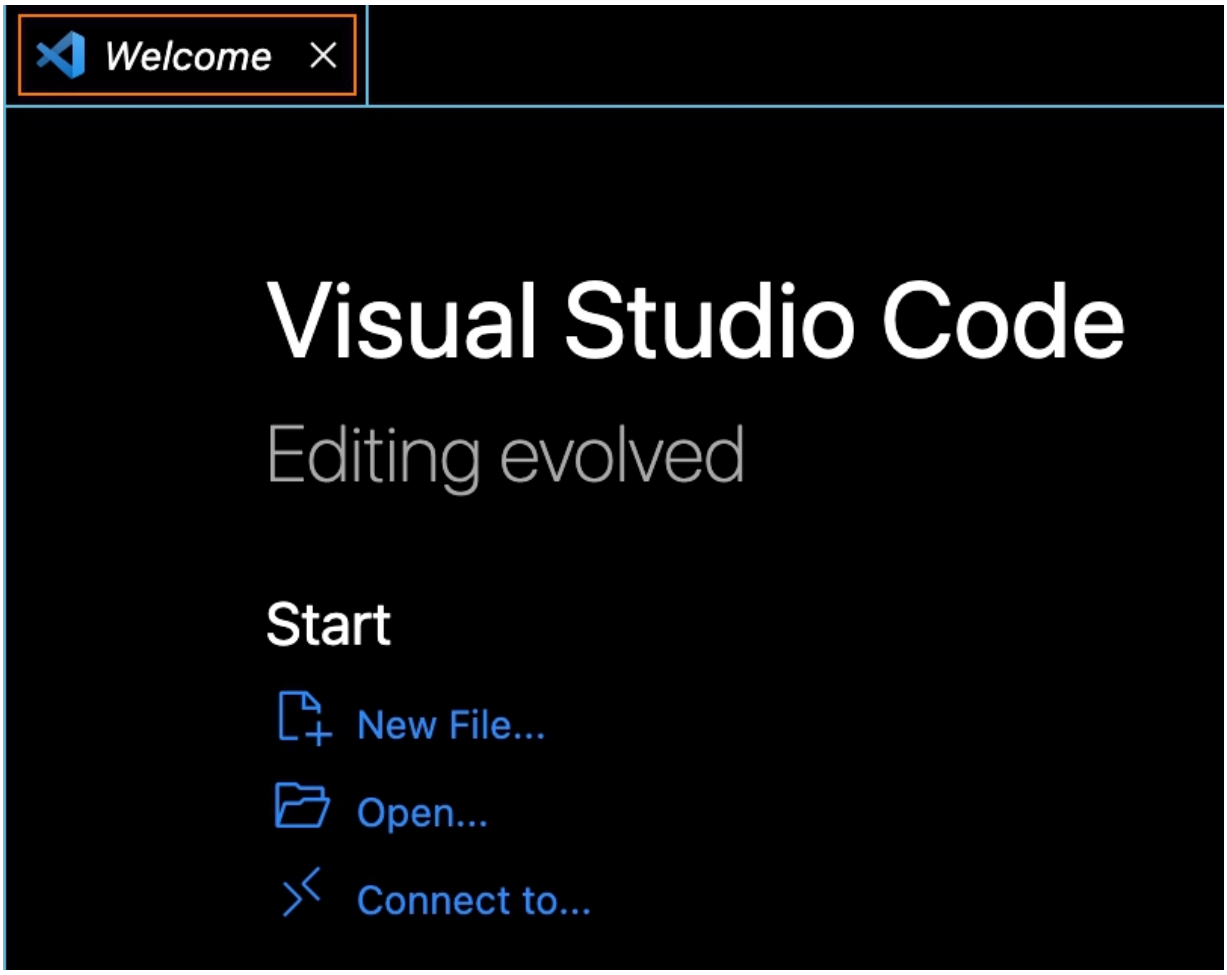


Figure 1-10: Visual Studio Code Welcome Screen

Click open, then open the newly created folder used to keep our python files as in Figure 1-11.

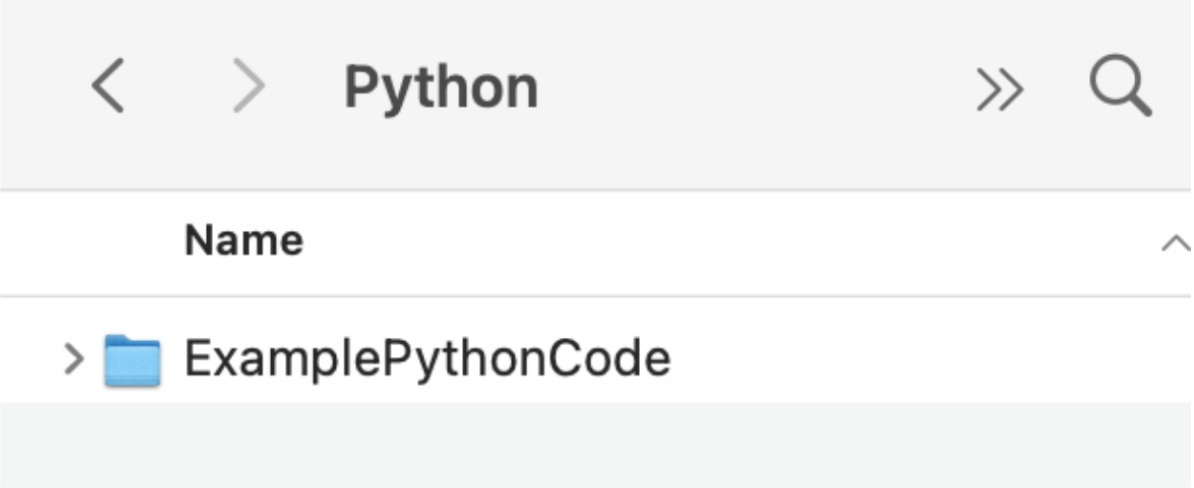


Figure 1-11: folder created to keep files for Python software development

Select and open the ExamplePythonCode folder and the following window will appear in Figure 1-12.

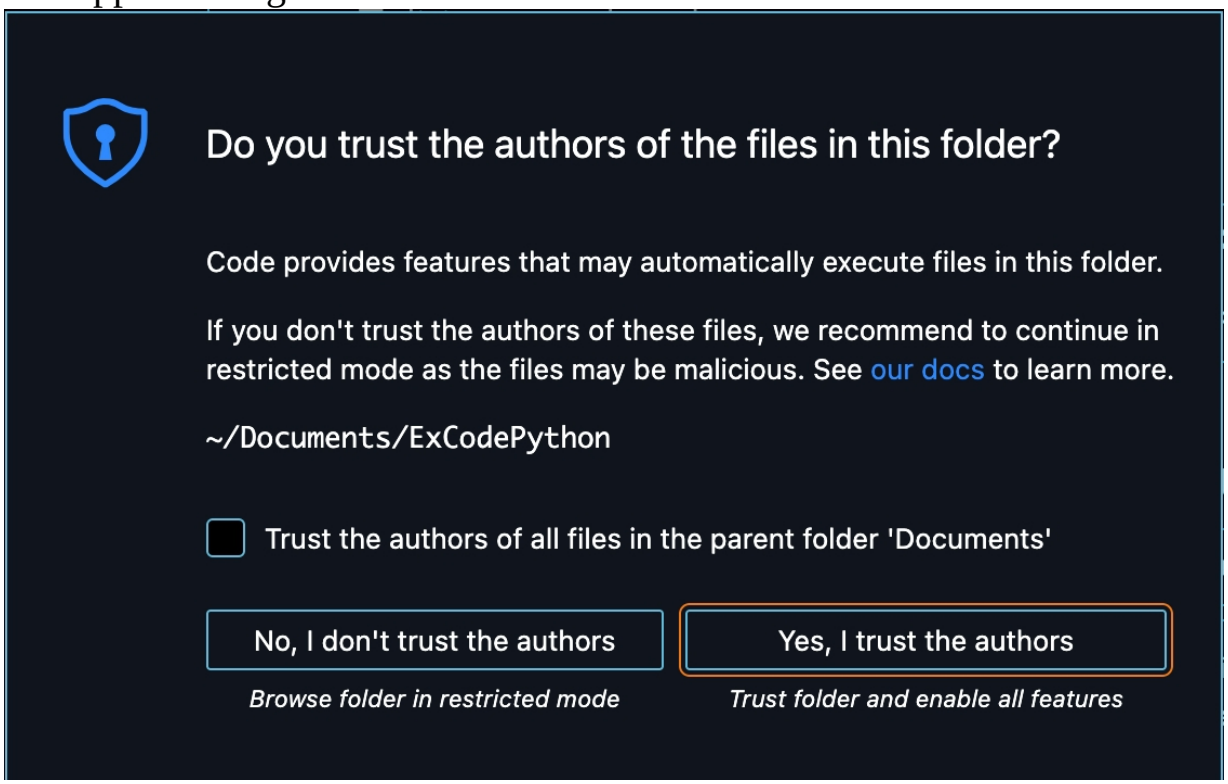


Figure 1-12: Visual Studio Code asking for permission to use the folder

Choose 

to confirm that you trust the authors, since you created this folder yourself.
When selected this screen will be shown in Figure 1-13.

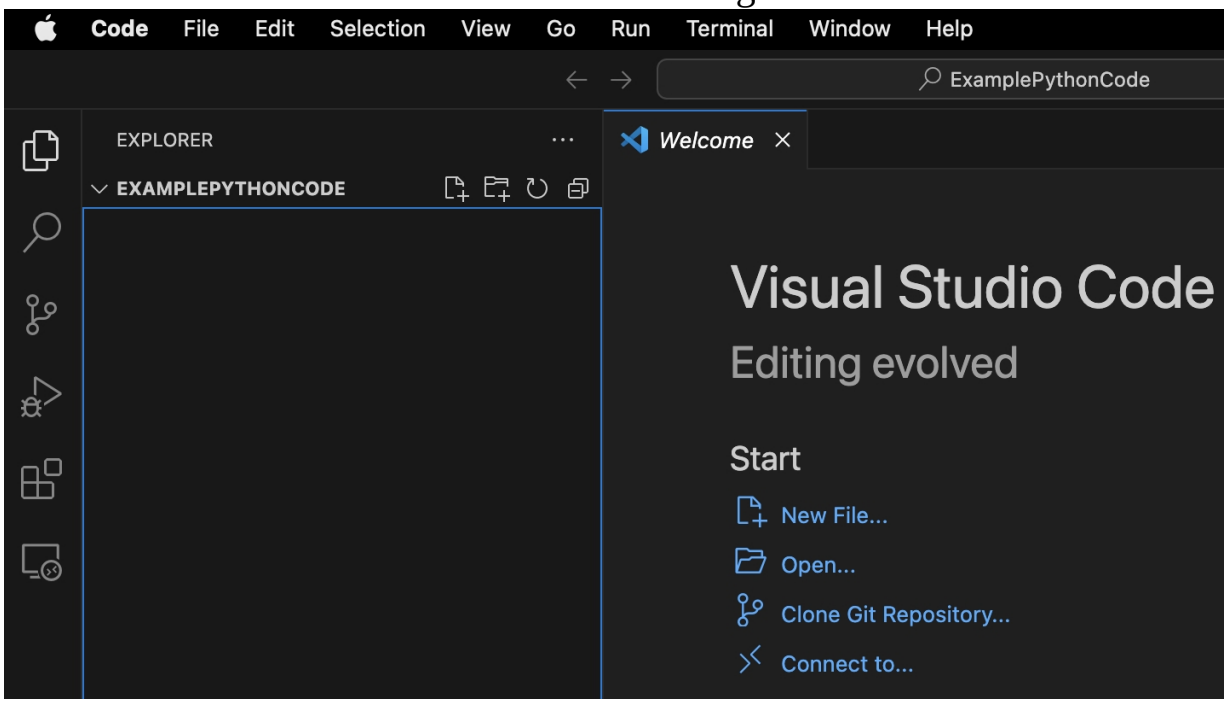
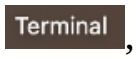
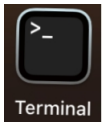


Figure 1-13: Visual Studio Code starting window

A new menu, 

, will appear, and can be used instead of 

in Figure 1-14.

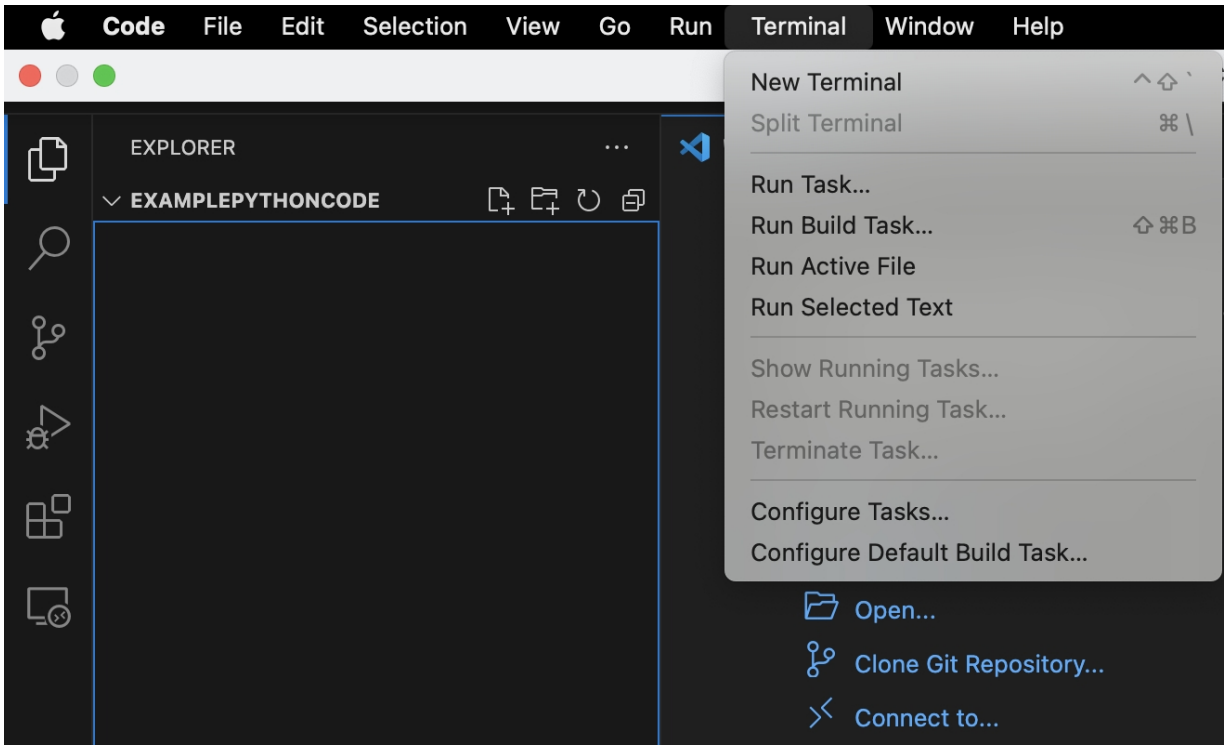


Figure 1-14: Visual Studio Code menu

Then select “New terminal.” A new terminal window will appear as shown in Figure 1-15.

Figure 1-15: Visual Studio Code terminal window

To write a program, click the space below EXAMPLEPYTHONCODE to reveal these icons in Figure 1-16.

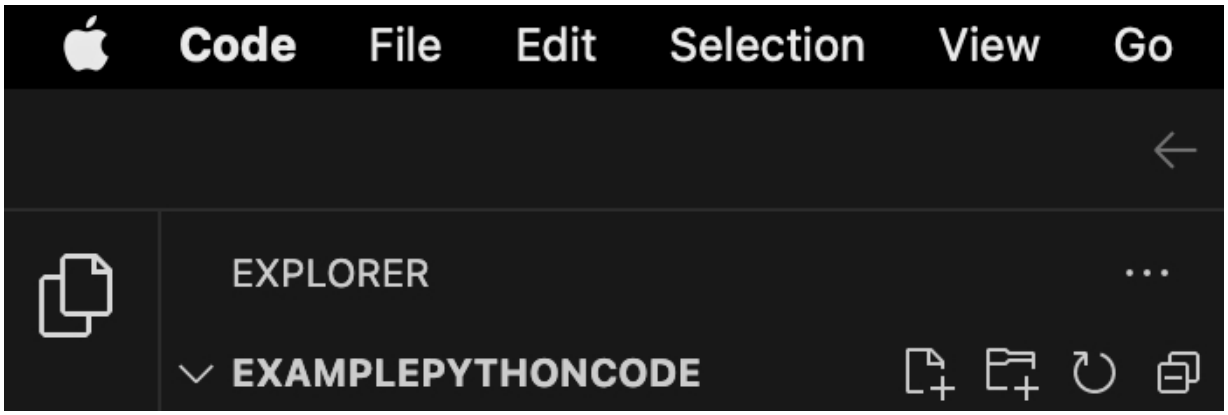


Figure 1-16: Visual Studio Code menu bar

 means creating a new file, while  means creating a new folder under EXAMPLEPYTHONCODE. Start by creating a new file named “Hi.py" and enter the following instructions:

```
print("Hi "+input("What is your name?"))
```

Select File->Autosave in Figure 1-17, so that any changes are saved automatically.

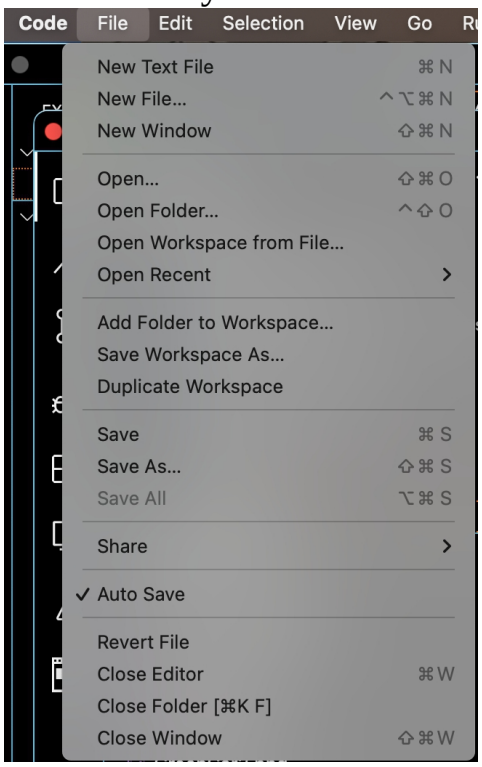


Figure 1-17: Visual Studio Code file submenu

After that, select the terminal window and enter the following instructions:

```
VaritSris>python Hi.py
```

Results

```
What is your name?
```

Enter VariitSris

```
What is your name?VariitSris
```

Results

```
Hi VariitSris
```

The Code input('What is your name?') is using the text “What is your name?” To query its user to enter information.
Changing the program text to

```
print("Welcome "+input("What is your name?")+ " to Thailand")
```

Then running Hi.py

```
VaritSris>python3 Hi.py
```

Results

```
What is your name? VariitSris
```

Results

```
Welcome VariitSris to Thailand
```

Example

```
print('No.of Character '+

str(len(input('What is your name? '))))
```

Results

```
What is your name? VariitSris

No.of Character 10
```

Example

```
name = "Jack"
```

```
print(name)

name = "Angela"

print(name)

name = input("What is your name?")

length = len(name)

print(length)
```

Results

```
Jack
Angela
What is your name?VariitSris
10
```

virtual environment

In case you want to use multiple versions of Python and mutiple projects on single machine

The methods and steps for using a virtual environment vary according to different operating systems. For this textbook, let's use macOS as an example.

1. Install brew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Update brew

```
brew update
```

3. Install pyenv

```
brew install pyenv
```

4. Install python. Required version for use in Part II, use python 3.9.16

```
pyenv install 3.9.16
```

Note :

In case you want to develop other software projects that use packages or modules. If appropriate for your project, you must check which version of Python is appropriate and install to match that version.

5. Install pyenv-virtualenv

```
brew install pyenv-virtualenv
```

6. Give the environment an appropriate name for your project so it is make easy to remember. For Part II, the environment names are set as follows.

Game

```
pyenv virtualenv 3.9.16 PyGame
```

Web Application

```
pyenv virtualenv 3.9.16 PyWebApp
```

Data Analysis

```
pyenv virtualenv 3.9.16 PyDataAnalysis
```

AI

```
pyenv virtualenv 3.9.16 PyAI
```

The command used to create a virtual environment with python version is shown as follows.

```
pyenv virtualenv <python_version> <environment_name>
```

7. To access that environment, use the command.

```
pyenv activate <environment_name>
```

If there is an error, it will show as follows.

```
Failed to activate virtualenv.
```

Perhaps pyenv-virtualenv has not been loaded into your shell properly.
Please restart current shell and try again.

Use the following command to resolve the error:

```
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

8. Check python version.

Inside the environment.

```
python –version
```

Result

Python 3.9.16

Outside the environment.

Result

```
The `python' command exists in these Python versions:
  3.9.16
  3.9.16/envs/PyAI
  3.9.16/envs/PyDataAnalysis
  3.9.16/envs/PyGame
  3.9.16/envs/PyWebApp
  PyAI
  PyDataAnalysis
  PyGame
  PyWebApp
```

9. Exit environment.

```
source deactivate
```

10. uninstall environment.

```
pyenv uninstall <environment_name or python version>
```

Exercises

1. Show the results of running print("Hi,"+input("My name is "))
2. Write a program that queries the user's name and address, and displays the name and address of the user.

CHAPTER 2 DATA TYPES

Python has the following Data Types:

- Integers
  Data that is any whole number, such as 123247289343214
- Floats
  Data that is any numbers with decimal places, such as 23.5 or 8.63
- Strings
  Characters enclosing quotes, such as “Hi”
- Boolean
  Can only be True or False
- Lists/Dictionaries/Sets/Tuples
  (Examples and programming functions in Chapter 3)

To determine the correct data class that will function as such, you must use variables to configure the data class you need.

Integers

If you want to determine the number of tourists (NoOfTourist) as a whole number (integer) of 115, then program:

```
NoOfTourist = 115
```

Floats

If you want to determine the Tourist’s Vote point average (VoteTourist) as a decimal (float) of 3.50, then Program:

```
VoteTourist = 3.50
```

In case of decimals within the data, it will not be added to the sum as integer data, so all data needs to be configured as float data.

For example:

```
print(float(2.5)+float(3))
```

This will convert all data to Float first before running.

Results

```
5.5
```

For example:

```
print(float(2.5)+int(3.2))
```

This will combine the decimals of 2.5 with just the integer of 3.

Results

```
5.5
```

```
print(float(2.5)+float(3.2))
```

This will add all integer and decimal data between 1.2 and 3.6

Results

```
5.7
```

For examples of functions, refer to Table 2-1.

Table 2-1: Functions Used In Data Type Conversion For Python

Functions Used in Data Type Conversion for Python		
Function	**Description**	**Example**
int()	Make the value in brackets an integer, perform calculations with integers. Results in an integer value.	x=int(1.7) print(x) print(type(x)) Results 3 <class 'int'> Where values in brackets contain decimals.

Functions Used in Data Type Conversion for Python		
Function	**Description**	**Example**
float()	Make the value in brackets a decimal, perform calculations with decimals. Results in a decimal value.	x=float(3.56) print(x) print(type(x)) Results 3.56 <class 'float'> Where values in brackets is a whole number.
str()	Make the value in brackets a String. Display results as a character.	x=str(5242) print(x) print(type(x)) Results 5242 <class 'str'>

Functions Used in Data Type Conversion for Python		
Function	**Description**	**Example**
bool()	Make the value in brackets Boolean	x= bool("Hi Python") print(x) print(type(x)) Results True <class 'bool'> num= bool(3242) print(num) print(type(num)) Results True <class 'bool'> list= bool(["Peter", "Top", "Lisa"]) print(list) print(type(list)) Results True <class 'bool'> False values are bool(False) bool(None) bool(0) bool("") bool(()) bool([]) bool({})

Strings

In case of String Class Data

Determine the Tourist's name (TouristName) as Character strings by using quotation marks. either "" or '' will work.

TouristName =’Peter’ or TouristName =”Peter”

String Functions

String values have the following functions:

```
print(“Top"[0]) Results: T
print("Top"[2]) Results: p
print("56"+"211") Results 56211 (as a string)
```

We can also find the number of characters in the string:

```
num_char = len(input("What is your name? "))

print("Your name has "+str(num_char)+" characters.")
```

Results

```
What is your name? Top
Your name has 3 characters.
```

We can combine string values, but if the value is not a string value but an integer, you convert its class into string like this:

```
num_char = len(input("What is your name? "))

print(type(num_char))

print("Your name has "+str(num_char)+" characters.")
```

Results

```
What is your name? Top
<class 'int'>
Your name has 3 characters.
```

```
print(str(4.8)+str(9)) to convert values to string
```

Results

```
4.89
```

Boolean

Boolean has a value of True or False only, as in the example.

Finding a value in a string whether it exists or not. If there is a result, it will be True, if it is not there, it will be False.

```
sentence = "Lisa is beautiful girl"

print("Lisa" in sentence)
```

Result

```
True
```

```
sentence = "Lisa is beautiful girl"

print("Peter" not in sentence)
```

Result

```
True
```

For examples of functions, refer to Table 2-2.

Table 2-2: Frequently Functions Used in String Type Data for Python

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
capitalize()	Capitalizes first character	sentence = "lisa is a beautiful girl" print(sentence.capitalize()) Results Lisa is a beautiful girl
casefold()	Make all characters lowercase	sentence = "lISa Is a beaUtiful Girl" print(sentence.casefold()) Results lisa is a beautiful girl
center()	Centers all characters according to the number of characters in brackets.	sentence = "Lisa is beautiful " print(sentence.center(40)) Results Lisa is beautiful

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
count()	Find the number of words or letters in brackets of a string	sentence = "Lisa is a beautiful girl. " print(sentence.count("a")) Results 3
		sentence = "Lisa is a beautiful girl. She has a child. She has two dogs." print(sentence.count("has")) Results 2 sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden" print(sentence.count("i")) Results 9 sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden" print(sentence.count("She")) Results 2

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
endswith()	Find out if the end of a string includes words enclosed in brackets at the end of the string. If there is, the result is True. If not, False is displayed.	Find if the value “sentence” ends with “.” sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden" print(sentence.endswith(".")) Results False sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden" print(sentence.endswith("her dogs")) Results False
expandtabs()	Set the space as defined in the brackets of the string.	Set the space as defined in the brackets of the string, using \t between spaces (tab) sentence = "Lisa\tis\ta\tbeautiful\tgirl." print(sentence.expandtabs(2)) Results Lisa is a beautiful girl.

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
find()	Searches for value enclosed in brackets, if found, returns a return value. The first position in the string is 0 until all the letters in the string are checked. Works like index(), except that if not found, -1 is returned.	Finds a value enclosed in brackets, The first position in the string is 0. sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden." findPosition = sentence.find("a") print(findPosition) Results 3 findPosition = sentence.find("A") print(findPosition) Results -1 -1 means not found

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
		sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." findPosition = sentence.find("She") print(findPosition) <u>Results</u> 26 First found in position 26. sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden." findPosition = sentence.find("xxx") print(findPosition) <u>Results</u> -1
index()	Searches for value enclosed in brackets, if found, returns a return value. The first position in the string is 0 until all the letters	Finds a value enclosed in brackets, The first position in the string is 0. sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence.index("A") print(indexPosition) <u>Results</u> 0

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
	in the string are checked.	
	Works like find(), except that if not found, error is returned.	sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence.index("She") print(indexPosition) <u>Results</u> 26 First found in position 26. sentence = "Lisa is a beautiful girl. She likes to eat bananas, and likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence. index("xxx") print(indexPosition) Returns <u>Results</u> ValueError: substring not found

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
format()	Format decimal places. e.g.: .2f, displays 2 decimal places, rounded up. .0f displays an integer.	To display the amount decimal value, sentence = "Yesterday, Lisa bought bananas for {amount:.2f} baht.” print(sentence.format(amount=50)) <u>Results</u> Yesterday, Lisa bought bananas for 50.00 baht. If the next decimal is 5 or more, it will round up. 50.558 rounds to 50.56 sentence = "Yesterday, Lisa bought bananas for {price:.2f} baht." print(sentence.format(price=50.558)) <u>Results</u> Yesterday, Lisa bought bananas for 50.56 baht. to add comma separators, use ,.2f sentence = "Yesterday, Lisa bought bananas for {price:,.2f} baht." print(sentence.format(price=12250.558))

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
		Results Yesterday, Lisa bought bananas for 12,250.56 baht. You can also add currency signs in front sentence = "Yesterday, Lisa bought bananas for ${price:,.2f} baht." print(sentence.format(price=12250.558)) Results Yesterday, Lisa bought bananas for $12,250.56 baht.
isalnum()	If all the characters in a string are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9), returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isalnum()) Results False Since it includes space, “,” and “.” sentence = "YesterdayAnnaboughtbananasfor50baht" print(sentence.isalnum()) Results True

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
isalpha()	If all the characters in a string are alphabet letters (a-z) (A-Z) only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isalpha()) Results False Since it includes space, “,” and “.” sentence = "YesterdayAnnaboughtbananasfor50baht" print(sentence.isalpha()) Results False Since it includes the number 50 sentence = "YesterdayAnnaboughtbananasforbaht" print(sentence.isalpha()) Results True
isdecimal()	If all the characters in a string are decimals only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isdecimal()) Results False not decimal sentence = "3341341" print(sentence.isdecimal()) Results True

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
isdigit()	If all the characters in a string are digits only, returns True. Note: Digits are values with numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in a string. e.g. 25, 45672 etc.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isdigit()) Results False Not a Digit sentence = "3341341" print(sentence.isdigit()) Results True sentence = "33,413.41" print(sentence.isdigit()) Results False Not all digits. contains “,” and “.”

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
islower()	If all the characters in a string are lowercase only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.islower()) Results False Contains Uppercase Y and A sentence = "yesterday, anna bought bananas for 50 baht." print(sentence.islower()) Results True sentence = "33,413.41" print(sentence.islower()) Results False Numbers cannot be lowercase
isupper()	If all the characters in a string are uppercase only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isupper()) Results False Contains Uppercase Y and A sentence = "yesterday, anna bought bananas for 50 baht." print(sentence.isupper())

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
		Results False sentence = "YESTERDAY, ANNA" print(sentence.isupper()) Results True
upper()	Change all character in string to uppercase	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.upper()) Results YESTERDAY, ANNA BOUGHT BANANAS FOR 50 BAHT.

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
isnumeric()	If all the characters in a string are numeric only, returns True. Note: Numeric are values with numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 in a string. e.g. 25, 45672 etc.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isnumeric()) Results False sentence = "3451" print(sentence.isnumeric()) Results True sentence = "-33,413.41" print(sentence.isnumeric()) Results False Contains “-”, “,” and “.”

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
isprintable()	If all the characters in a string are printable only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isprintable()) Results True sentence = "Yesterday, \n Lisa bought bananas for 50 baht." print(sentence.isprintable()) Results False Contains \n, a character for creating a new line. sentence = "Yesterday, \n Lisa bought bananas for 50 baht." print(sentence) Results Yesterday, Lisa bought bananas for 50 baht.
isspace()	If all the characters in a string are spaces only, returns True.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.isspace()) Results False sentence = “ ” print(sentence.isspace()) Results True

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
istitle()	If all words in a text start with an uppercase letter, AND the rest of the word are lower case letters, returns True.	sentence = "Yesterday, Lisa Bought Bananas For 50 Baht." print(sentence.istitle()) Results True sentence = “YESTERDAY, Lisa bought bananas for 50 baht.” print(sentence.istitle()) Results False
title()	Capitalize all words in the string.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.title()) Results Yesterday, Lisa Bought Bananas For 50 Baht.
lower()	Change all characters in string to lowercase.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.lower()) Results yesterday, anna bought bananas for 50 baht.

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
replace()	Find and replace instances of a word in string.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.replace("bananas", "apples")) <u>Results</u> Yesterday, Lisa bought apples for 50 baht. sentence = "Lisa is beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." print(sentence.replace("bananas", "apples",1)) <u>Results</u> Lisa is a beautiful girl. She likes to eat apples, and likes to give bananas to her friends. She grows bananas in her garden. Replace only the first instance by adding ,1

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
rfind()	From the rightmost, finds the last occurrence of the specified value in a string. Returns -1 if the value is not found.	sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." print(sentence.rfind("bananas")) Results 104 sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." print(sentence.rfind("apples")) Results -1 Returns -1 if the value is not found sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." print(sentence.rfind("bananas",50,100)) to search between 50 and 100 Results 70
rindex()	From the rightmost, finds the	Finds the last occurrence of the specified value.

Frequently Used Functions Used in String Type Data for Python		
Function	**Description**	**Example**
	last occurrence of the specified value in a string. Functionally identical to rfind(), but returns “ValueError: substring not found” instead if the value is not found.	sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence.rindex("A") print(indexPosition) <u>Results</u> 52 sentence = "Lisa is beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence.rindex("She") print(indexPosition) <u>Results</u> 94 Found in last position 94. sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." indexPosition = sentence.rindex("xxx") print(indexPosition) Returns <u>Results</u> ValueError: substring not found

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
split()	Split words in a string.	sentence = "Lisa is a beautiful girl. She likes to eat bananas. And likes to give bananas to her friends. She grows bananas in her garden." splitSentence = sentence.split() print(splitSentence) <u>Results</u> [‘Lisa’, ‘is’, ‘a’, ‘beautiful’, ‘girl.’, ‘She’, ‘likes’, ‘to’, ‘eat’, ‘bananas,’, ‘and’, ‘likes’, ‘to’, ‘give’, ‘bananas’, ‘to’, ‘her’, ‘friends.’, ‘She’, ‘grows’, ‘bananas’, ‘in’, ‘her’, ‘garden.’]
splitlines()	Split lines in a string. requires /n to create line breaks.	sentence = "Lisa is a beautiful girl. \nShe likes to eat bananas. And likes to give bananas to her friends. \nShe grows bananas in her garden." splitLinesSentence = sentence.splitlines() print(splitLinesSentence) <u>Results</u> [‘Lisa is a beautiful girl.’, ‘She likes to eat bananas, and likes to give bananas to her friends.’, ‘She grows bananas in her garden.’]
startswith()	Check if The value in brackets is the first	Check if “Hi” is the first value of the string. sentence = "Lisa is a beautiful girl. \nShe likes to eat bananas. And likes to

Frequently Used in String Type Data for Python		
Function	**Description**	**Example**
	value of the string.	give bananas to her friends. \nShe grows bananas in her garden." startswithSentence = sentence.startswith("Hi") print(startswithSentence) Results False sentence = "Lisa is a beautiful girl. \nShe likes to eat bananas. And likes to give bananas to her friends. \nShe grows bananas in her garden." startswithSentence = sentence.startswith("Lisa") print(startswithSentence) Results True sentence = "Lisa is a beautiful girl. \nShe likes to eat bananas. And likes to give bananas to her friends. \nShe grows bananas in her garden." startswithSentence = sentence.startswith("Lisa is a beautiful") print(startswithSentence) Results True

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
strip()	Remove spaces at the beginning and at the end of the string before combining with other string values.	Remove spaces at the beginning and at the end of the string. fruit = " bananas " print(fruit.strip()) sentence="She likes to eat "+fruit.strip()+". And likes to give "+fruit.strip()+" to her friends." print(sentence) <u>Results</u> bananas She likes to eat bananas. And likes to give bananas to her friends.
rstrip()	Remove any spaces at the end of the stringbefore combining with other string values.	fruit = " bananas " print(fruit.rstrip()) sentence="She likes to eat "+fruit.rstrip()+". And likes to give "+fruit.rstrip()+" to her friends." print(sentence) <u>Results</u> ….bananas She likes to eat bananas. And likes to give bananas to her friends.

Frequently Functions Used in String Type Data for Python		
Function	**Description**	**Example**
swapcase()	Swap uppercase characters with lowercase, and vice versa.	sentence = "Yesterday, Lisa bought bananas for 50 baht." print(sentence.swapcase()) Results yESTERDAY, aNNA BOUGHT BANANAS FOR 50 BAHT.
zfill()	adds 0 at the beginning of the number, until it reaches the length Specified in brackets.	To add 0 In front of the number 20 to reach the length specified, which is 5 digits sentence = "20" print(sentence.zfill(5)) Results 00020

Exercises

1. 23.42 is what python data type?
2. “I love Thailand.” is what python data type?
3. What python data type has the value “true”?
4. What value is returned with: print(int(4.3)+int(3.5))? What data type is it?
5. Write a program to display ‘a’ from the string “How are U?”

information = “Where is the best place in
Thailand?”

6. Write a program to display the “information” string but all uppercase.
7. Write a program to count all the ‘e’s in the “information” string.
8. Write a program to find ‘place’ in the “information” string.

BKKPeople = “In 2016, Bangkok had an estimated population of 9 million according to data from the 2010 census.”

9. write a program to display the estimated population as 8.28 million
10. Write a program to split all the words in the “BKKPeople” string.

CHAPTER 3 LISTS, DICTIONARIES, SETS, TUPLES

Lists

A **List** is what python uses instead of arrays to store multiple values, sorted alphabetically in ascending order, in a single variable that can change values and store repeating values.

One syntax for creating a list is Syntax 3-1.

Syntax 3-1: Create Lists 1

To store multiple fruits (fruit) including “Orange, Apple, Mango, Mangosteen, Rambutan, Durian, Pineapple, Papaya” in a single variable as a list, proceed as follows:

```
fruit = ["orange","mango","banana","rambutan", "durian", "pineapple","papaya"]
print(fruit)
```

Results

[‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’]

Using the data kept in the list requires an index number, starting with the number 0, followed by 1, 2, 3, up until the last index. The syntax for index number of element in List is Syntax 0-1.

Syntax 3-2: Index number of element in List

Example

```
print(fruit[0])
```

Results

```
orange
```

```
print(fruit[1])
```

Results

```
mango
```

If you want the last index value in the list, we need to know the total number of items in a list with this function:
len(List)
Example

```
print(len(fruit))
```

Results

```
7
```

Therefore, if you want to return the last value in the list, then we need to use **fruit[6]** as the last value is always len(List)-1 with the first index being 0.
In case you don't want to search with len(List), alternatively we could use negative indexes, with -1 as the last index as shown in Syntax 3-3.

Syntax 3-3: Negative index of element in List

```
listname=[listname[-len(listname)],...,listname[-2],listname[-1]]
```

```
print(fruit[-1])
```

Results

```
papaya
```

If the index is -3, it will return the 3rd item from the end.

```
print(fruit[-3])
```

Results

```
durian
```

For functions, refer to Table 3-1.

Table 3-1: Functions Used in Lists for Python

Functions Used in Lists for Python		
Function	**Description**	**Example**

Functions Used in Lists for Python		
Function	**Description**	**Example**
append()	Append the new value at the end of the list. Duplicates can only be added one value at a time.	fruit = ["orange","mango","banana", "rambutan", "durian","pineapple", "papaya"] fruit.append("strawberry") print(fruit) Results [‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’, ‘strawberry’]
clear()	Clear the list. No values kept.	Clear the “fruit” list of all values. fruit = ["orange","mango","banana","rambutan", "durian","pineapple","papaya"] fruit.clear() print(fruit) Results []
copy()	Copy the list and submit the value.	Copy “fruit” with “copy_fruit” fruit = ["orange","mango","banana","rambutan", "durian","pineapple","papaya"] copy_fruit=fruit.copy() print(copy_fruit) Results [‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’, ‘strawberry’]
count()	Count the number of times a value is in the list and submit the result.	Count the number of "cherry" in the “fruit” list fruit = ["orange","mango","banana","rambutan", "durian","pineapple","papaya"] NoOfCherry = fruit.count("cherry") print(NoOfCherry) Results 0

Functions Used in Lists for Python		
Function	**Description**	**Example**
extend()	Add the values from one list to another.	Extend the values in the “fruit” list With the values in “chineseFruit” fruit = ["orange","mango","banana", "rambutan","durian","pineapple", "papaya"] chineseFruit = ["loquat","lychee"] fruit.extend(chineseFruit) print(fruit)
extend()		Results [‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’, ‘strawberry’, ‘loquat’, ‘lychee’]
index()	Returns Index number of a value.	Find the index of value "mango" fruit = ["orange","mango","banana", "rambutan","durian","pineapple", "papaya"] indexOfMango=fruit.index("mango") print(indexOfMango) Results 1
insert()	Insert a value into a list by specifying its index number.	Add "strawberry" as the first value in the list. fruit = ["orange","mango","banana", "rambutan","durian","pineapple", "papaya"] fruit.insert(0, "strawberry") print(fruit) Results [‘strawberry’, ‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’]
pop()	Removes a value from the list using its index.	To remove the value indexed as 0, "orange" from the “fruit” list. fruit = ["orange","mango","banana", "rambutan","durian","pineapple", "papaya"] fruit.pop(0) print(fruit) Results [’mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’]

Functions Used in Lists for Python		
Function	**Description**	**Example**
remove()	Removes the specified value from the list.	To remove the value "orange" from the list fruit = ["orange","mango","banana", "rambutan","durian","pineapple","papaya"] fruit.remove("orange") print(fruit) Results [‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’]
sort()	Sort the values alphabetically or numerically in ascending order.	Tube the values alphabetically in ascending order of fruit list: fruit = ["orange","mango","banana", "rambutan","durian", "pineapple","papaya"] fruit.sort() print(fruit)
		Results [‘durian’,‘mango’, ‘banana’,’orange’,’papaya’, ‘pineapple’, ‘rambutan’] If the values within the list have multiple languages, upper and lowercase, as well as numbers, the values will be sorted by numbers value, then all the uppercase values, then all lowercase values, and finally any other languages. listSort = ["orange","mango", "banana", "Apple","Tomato", "Peach","5","4","2", "6"] listSort.sort() print(listSort) Results ['2', '4', '5', '6', 'Apple', 'Peach', 'Tomato', 'mango', 'banana', 'orange']

Functions Used in Lists for Python		
Function	**Description**	**Example**
join()	Join the values in the list and display.	to display values in the fruit list separated by / fruit = ["orange","mango","banana", "rambutan","durian","pineapple","papaya"] print("/".join(fruit)) Results orange/mango/banana/rambutan/durian/pineapple/papaya

To combine two lists together, use Syntax 3-4.

Syntax 3-4: Combining Two Lists

$ListName_1 = [ListName_1[0], ListName_1[1], ..., ListName_1[len_1-1]]$

$ListName_2 = [ListName_2[0], ListName_2[1], ..., ListName_2[len_2-1]]$

$ListName_3 = [ListName_1, ListName_2]$

Results

$ListName_3 = [[ListName_1[0], ListName_1[1], ..., ListName_1[len_1-1]], [ListName_2[0], ListName_2[1], ..., ListName_2[len_2-1]]]$

Note

len1 is number of value in ListName1
len2 is number of value in ListName2
Number of value in ListName3 is len1 + len2

Example:

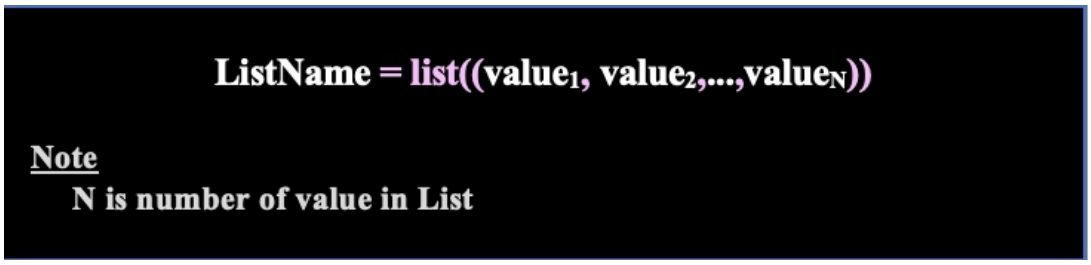
```
fruit = ["orange","mango","banana","rambutan", "durian","pineapple", "papaya"]
vegetable = ["tomato","avocado","bean","pepper", "pumpkin"]
plant = [fruit,vegetable]
print(plant)
```

Results

```
[[‘orange’, ‘mango’, ‘banana’, ‘rambutan’, ‘durian’, ‘pineapple’, ‘papaya’], [‘tomato’, ‘avocado’, ‘bean’, ‘pepper’, ‘pumpkin’]]
```

Another way to create lists is with Syntax 3-5.

Syntax 3-5: Create Lists 2



Example:

```
fruit = list(("orange","mango","banana","rambutan","durian","pineapple", "papaya"))
print(fruit)
```

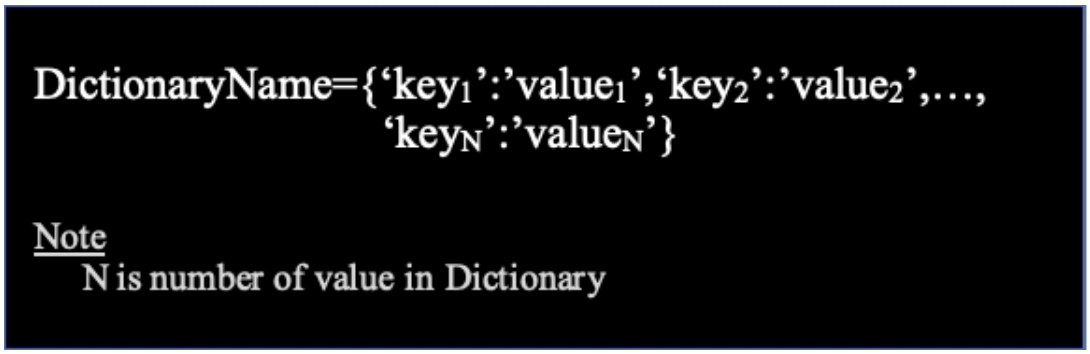
Results

['orange', 'mango', 'banana', 'rambutan', 'durian', 'pineapple', 'papaya']

Dictionaries

Dictionaries in Python are data kept in pairs, using the syntax key:value, as shown in Syntax 3-6.

Syntax 3-6: Create Dictionaries



Dictionaries are useful in collecting related data, such as
Tourist name and gender, and using the same indexes to call the related data in a database. They're written using curly braces {} separated by colons.

```
Tourist_gender = {'Ammy':'Female','Jame':'Male','Mike':'Male','Minnie':'Female'}
print(Tourist_gender)
print(Tourist_gender['Jame'])
```

Results

```
{'Ammy': 'Female', 'Jame': 'Male', 'Mike': 'Male', 'Minnie': 'Female'}
Male
```

In case of Tourist age:

```
Tourist_age = {'Ammy':'16','Jame':'12','Mike':'17','Minnie':'14'}
print(Tourist_age)
print(Tourist_age['Jame'])
```

Note that key values cannot repeat. If it does, then only the last key and value will be referenced.
For example:

```
Tourist_age = {'Ammy':'16','Jame':'12','Mike':'17','Minnie':'14',
'Minnie':'18','Minnie':'19'}
print(Tourist_age)
print(Tourist_age['Minnie'])
```

Results

```
{'Ammy': '16', 'Jame': '12', 'Mike': '17', 'Minnie': '19'}
19
```

And since duplicate keys do not count, the len command will result as:

```
print(len(Tourist_age))
```

Results

```
4
```

We can, however, store a list of values within the same key, for example, to store a Tourist's name, age, then a list grades in three years:

```
Tourist = {'Name':'Jame','Age':18,'Vote':['A','C','F']}
print(Tourist)
print(Tourist['Vote'])
```

Results

```
{'Name': 'Jame', 'Age': 18, 'Vote': ['A', 'C', 'F']}
['A', 'C', 'F']
```

To show their first-year grade:

```
print(Tourist['Vote'][0])
```

Results

```
A
```

For functions, refer to Table 3-2.

Table 3-2: Functions Used in Dictionaries for Python

Functions Used in Dictionaries for Python		
Function	**Description**	**Example**
clear()	Clears the dictionary. No values kept.	Tourist=dict(Name = "Jame", Age = 18, Vote = ['A','C','F']) print(Tourist) Tourist.clear() print(Tourist) Results {'Name': 'Jame', 'Age': 18, 'Vote': ['A', 'C', 'F']} {}

Sets

Sets in Python are multiple items stored under a single variable. Items cannot be repeated, are randomly ordered and cannot be changed. To create a set, use Syntax 3-7.

Syntax 3-7: Create Sets

$$SetName = \{value_1, value_2, value_3, \ldots, value_N\}$$

Note
N is number of value in Set

```
Tourist={'Ammie','Ann','Minnie','Jame','Mike','Ann'}
print(Tourist)
```

Results

```
{'Minnie', 'Ann', 'Jame', 'Ammie', 'Mike'}
```

For functions, refer to Table 3-3.

Table 3-3: Functions Used in Sets for Python

Functions Used in Sets for Python		
Function	**Description**	**Example**
add()	Add values into the set must be a value not already contained in the set, otherwise it will not be added.	fruit = {"apple","banana", "cherry","orange", "mango","banana","rambutan","durian", "pineapple", "papaya"} fruit.add("papaya") print(fruit) Results {‘cherry’, ‘mango’, ‘rambutan’, ‘papaya’, ‘banana’, ‘apple’, ‘banana’, ‘durian’, ‘pineapple’, ‘orange’} fruit = {"apple","banana","cherry","orange", "mango", "banana", "rambutan", "durian", "pineapple", "papaya"} fruit.add(“strawberry”) print(fruit) Results {‘orange’, ‘durian’, ‘mango’, ‘banana’, ‘strawberry’, ‘pineapple’, ‘cherry’, ‘papaya’, ‘apple’, ‘rambutan’, ‘banana’}

Functions Used in Sets for Python		
Function	**Description**	**Example**
clear()	Clear the set of all values.	fruit = {"apple","banana","cherry","orange", "mango","banana","rambutan","durian", "pineapple","papaya","papaya"} fruit.clear() print(fruit) Results set()
copy()	Copy the set.	fruit = {"apple","banana","cherry","orange", "mango","banana","rambutan","durian", "pineapple","papaya","papaya"} Copy_fruit=fruit.copy() print(Copy_fruit) Results {‘mango’, ‘papaya’, ‘durian’, ‘cherry’, ‘banana’, ‘pineapple’, ‘rambutan’, ‘orange’, ‘apple’, ‘banana’}

Functions Used in Sets for Python		
Function	**Description**	**Example**
difference()	Display the differences between two sets.	interfruit = {"apple","banana","cherry","orange", "mango", "banana", "rambutan", "durian", "pineapple", "papaya","papaya"} Thaifruit = {"banana","orange","mango", "banana", "rambutan", "durian", "pineapple", "papaya"} print(interfruit.difference(Thaifruit)) Results {'cherry', 'apple'} Since the interfruit set has apple, cherry, Which does not appear in Thaifruit, whereas print(Thaifruit.difference(interfruit)) Results set() Since all fruits in Thaifruit are in interfruit.
difference_ update()	Show the different values between two sets. Differences will be cleared.	interfruit = {"apple","banana", "cherry","orange", "mango", "banana", "rambutan","durian", "pineapple", "papaya","papaya"} Thaifruit = {"banana","orange","mango", "banana", "rambutan","durian","pineapple", "papaya"} interfruit.difference_update(Thaifruit) print(interfruit) Results {‘apple’,’cherry’}

Functions Used in Sets for Python		
Function	**Description**	**Example**
discard()	Clears the values in brackets from the set	fruit = {"apple","banana", "cherry","orange", "mango", "banana", "rambutan","durian", "pineapple","papaya","papaya"} fruit.discard("apple") print(fruit) Results {‘pineapple’, ‘orange’, ‘banana’, ‘rambutan’, ‘cherry’, ‘durian’, ‘banana’, ‘papaya’, ‘mango’}
intersection()	Display like values between 2 or more sets.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple", "papaya"} fruit2 = {"apple","mango","banana", "rambutan","durian","pineapple", "papaya"} print(fruit1.intersection(fruit2)) Results {‘papaya’, ‘pineapple’, ‘durian’, ‘apple’, ‘rambutan’, ‘banana’} fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"banana","rambutan","durian", "pineapple","banana"} fruit3 = {"rambutan","banana"} print(fruit1.intersection(fruit2,fruit3)) print(fruit2.intersection(fruit1,fruit3)) print(fruit3.intersection(fruit1,fruit2)) Result of each computation will be the same, as follows: Results {'banana', 'rambutan'} {'banana', 'rambutan'} {'banana', 'rambutan'}

Functions Used in Sets for Python		
Function	**Description**	**Example**
intersection_update()	Display like values between 2 or more sets, then remove all other values from the set preceding .intersection_update	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"apple","mango","banana","rambutan", "durian","pineapple","papaya"} fruit1.intersection_update(fruit2) print(fruit1) Results {‘banana’, ‘rambutan’, ‘apple’, ‘pineapple’, ‘durian’, ‘papaya’} fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"banana","rambutan", "durian","pineapple","banana"} fruit3 = {"rambutan","banana"} fruit1.intersection_update(fruit2,fruit3) print(fruit1) Result of each computation will be the same, as follows: {'rambutan', 'banana'}

Functions Used in Sets for Python		
Function	**Description**	**Example**
isdisjoint()	Find the same value between two sets. If there is none, return true, if not, false.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"apple","mango","banana", "rambutan","durian","pineapple","papaya" } print(fruit1.isdisjoint(fruit2))] Results False Contains like values. fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"mango"} print(fruit1.isdisjoint(fruit2)) Results True
issubset()	check whether the first set is a subset of the second. If yes, return true, if not then false.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"banana","banana","rambutan", "durian","pineapple"} print(fruit1.issubset(fruit2)) Results False print(fruit2.issubset(fruit1)) Results True
issuperset()	check whether the first set is a superset of the second. If yes, return true, if not then false.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit2 = {"banana","banana","rambutan", "durian","pineapple"} print(fruit1.issuperset(fruit2))

Functions Used in Sets for Python		
Function	**Description**	**Example**
		Results True print(fruit2.issuperset(fruit1)) Results False
pop()	Removes a value from the set. (Random)	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya" } fruit1.pop() print(fruit1) Results {‘pineapple’, ‘banana’, ‘banana’, ‘rambutan’, ‘apple’, ‘papaya’} Removed value is “durian”
remove()	Removes value specified in brackets from the set.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papa ya"} fruit1.remove("papaya") print(fruit1) Results {‘banana’, ‘rambutan’, ‘apple’, ‘pineapple’, ‘durian’, ‘papaya’}
symmetric_ difference ()	Display the differences between both sets.	fruit1 = {"apple","banana","banana", "rambutan","durian", "pineapple","papaya", "mango"} fruit2 = {"banana","banana", "rambutan","durian", "pineapple","strawberry"} print(fruit1.symmetric_difference(fruit2)) Results {‘strawberry’, ‘apple’, ‘papaya’, ‘mango’}

Functions Used in Sets for Python		
Function	**Description**	**Example**
union()	Combine the values of two or more sets, Removing all duplicates.	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya", "mango"} fruit2 = {"banana","banana", "rambutan","durian","pineapple","strawberry"} print(fruit1.union(fruit2)) Results {‘papaya’, ‘durian’, ‘banana’, ‘pineapple’, strawberry’, ‘banana’, ‘rambutan’, ‘apple’, ‘mango’}
union()		fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya"} fruit2 = {"banana","rambutan","durian", "pineapple","banana"} fruit3 = {"rambutan","banana"} print(fruit1.union(fruit2,fruit3)) Results {‘banana’, ‘banana’, ‘papaya’, ‘durian’, ‘apple’, ‘rambutan’, ‘pineapple’}
update()	Updates the set by adding values from another set without creating duplicates	fruit1 = {"apple","banana","banana", "rambutan","durian","pineapple","papaya", "mango"} fruit2 = {"banana","banana","rambutan", "durian","pineapple","strawberry"} fruit1.update(fruit2) print(fruit1) Results {‘pineapple’, ‘papaya’,‘banana’, ‘banana’, ‘durian’, ‘apple’, ‘mango’, ‘rambutan’, ‘strawberry’}

Tuples

Tuples in python are used to store multiple items ordered under a single variable. It is unchangeable, but does allow duplicates. To write tuples, use Syntax 3-8.

Syntax 3-8: Create Tuples

$$TupleName = (value_1, value_2, \ldots, value_N)$$

Note

N is number of value in Tuple

Example

```
fruit = ("apple","banana","cherry","orange","mango","banana",
"rambutan","durian","pineapple", "papaya")
print(fruit)
```

Results

('apple', 'banana', 'cherry', 'orange', 'mango', 'banana', 'rambutan', 'durian', 'pineapple', 'papaya')

With duplicates:

```
fruit = ("apple", "banana","cherry","orange","mango","banana",
"rambutan","durian","pineapple","papaya", "papaya")
print(fruit)
```

Results

('apple', 'banana', 'cherry', 'orange', 'mango', 'banana', 'rambutan', 'durian', 'pineapple', 'papaya')

Therefore, when using len() to count the number of items, all items will be counted.

```
print(len(fruit))
```

Results

```
11
```

There are two different data types to keep a variable as a single item. These are:

Case 1

As a tuple

```
Tourist = ("Jame",)
print(type(Tourist))
```

Results

```
<class 'tuple'>
```

Case 2

As a String

```
Tourist = ("Jame")
print(type(Tourist))
```

For functions, refer to Table 3-4.

Table 3-4: Functions Used in Tuples for Python

Functions Used in Tuples for Python		

Function	Description	Example
count()	Count the instances of the bracketed value in the tuple.	fruit = ("apple","banana", "cherry","orange", "mango","banana", "rambutan","durian", "pineapple","papaya","papaya") print(fruit.count("papaya")) Results 2
index()	Find the first indexed location of the bracketed value in the tuple, Starting from 0.	fruit = ("apple","banana","cherry","orange", "mango","banana","rambutan","durian", "pineapple","papaya","papaya") print(fruit.index("papaya")) Results 9 fruit = ("apple","banana", "cherry","orange", "mango", "banana", "rambutan","durian", "pineapple","papaya","papaya") print(fruit.index("apple")) Results 0

Table 3-5: Differences between lists, dictionaries, sets and tuples

Item	List	Dictionary	Set	Tuple
Changeable	✓	✓	X	X
Orderable	✓	✓	X	✓
Duplicates	✓	X	X	X
Indexable	✓	✓	X	✓

Note: ✓ Yes X No

Exercises

1. Create a list to store the names and telephone numbers of at least 5 members and display them.
2. Display the name of the 3rd member.
3. Add at least 2 members.
4. Make a copy of the members list named copyMember.
5. Display the number of members.
6. Insert a new value for the 3rd member.
7. Display results of copyMember.pop(2)
8. Display the phone numbers of all members.
9. Write a program that uses issubset().
10. Create a tuple of the members.

CHAPTER 4 OPERATORS

Python has the following operators:

- Arithmetic operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators
- Assignment operators

Arithmetic Operators

Used in basic calculations, Table 4-1 contains all the arithmetic operators used by Python.

Table 4-1: Arithmetic Operators

Arithmetic Operators		
Arithmetic Operator	**Meaning**	**Example**
+	Add	x = 8 y = 3 print(x + y) Results 11

Arithmetic Operators		
Arithmetic Operator	**Meaning**	**Example**
-	Subtract	x = 8 y = 3 print(x -y) Results 5
*	Multiply	x = 8 y = 3 print(x * y) 24
/	Divide	x = 8 y = 3 print(x / y) Results 2.6666666666666665

//	Floor Division (Integer only, discarding remainders and decimals.)	x = 8
		y = 3
		print(x // y)
		2

**	Exponentiation (power of)	x = 8 y = 3 print(x ** y) Results 512
%	Modulus (finds remainder)	x = 8 y = 3 print(x%y) Results 2

Arithmetic statements follow Table 4-2, from left to right, bracketed values first.

Table 4-2: Order of Operations

Order	Operation
First	() Bracketed values
Second	** Exponents
Third	% Modulo, * Multiply, / Divide, // Floor Divide
Fourth	+ Add, -Subtract

Example 1

```
print(5+2**2+8/4*10%3%2)
```

The result of the order of operations are as follows: Starting from the second order, which is exponents, 2 To the power of 2 equals 4. Next or third order operations from left to right: 8 divided by 4 equals 2, 2 multiplied by 10 equals 20, then modulo operations, 20 divided by 3 leaves remainder 2, 2 divided by 2 leaves remainder 0. Finally, do additions from the fourth order: 5 plus 4 plus 0 equals 9.0

Results

```
9.0
```

Example 2

```
print(5+2/2*5+8)
```

Start with the third order: 2 divided by 2 equals 1, 1 multiplied by 5 equals 5. Then the addition: 5 plus 5 plus 8 equals 18.0

Results

```
18.0
```

```
print(5+2**2/2*5+8)
```

Start with exponents: 2 to the power of 2 equals 4. 4 divided by 2 equals 2. 2 multiplied by 5 equals 10. 5 plus 10 plus 8 equals 23.

Results

```
23.0
```

Adding brackets will change the order of operations. For example:

```
print((5+2)**2/(2*5)+8)
```

Starting with bracketed values: 5 plus 2 equals 7, 2 times 5 equals 10. 7 to the power of 2 equals 49. 49 divided by 10 equals 4.9. 4.9 plus 8 equals 12.9.

Results

```
12.9
```

Example 3

Let's say we want to use python to calculate body mass index (BMI) to display whether the person is underweight, normal or overweight, using the formula weight(kg) divided by height(m) to the power of 2, we can determine the results using Table 4-3:

Table 4-3: BMI results meaning

BMI	Category
below 18.5	Underweight
between 18.5 and 22.9	Normal
above 22.9	Overweight

```
Weight_kg = float(input('Your Weight(Kg) is '))
Height_m = float(input('Your Height(M) is '))
print('Your BMI is '+str(Weight_kg/Height_m**2))
```

Results

```
Your Weight(Kg) is 80
Your Height(M) is 1.8
Your BMI is 24.691358024691358
```

and if you want the text to display only 2 decimal places, add this code:

```
Weight_kg = float(input('Your Weight(Kg) is '))
Height_m = float(input('Your Height(M) is '))
BMI=round(float(Weight_kg/Height_m**2),2)
print('Your BMI is '+str(BMI))
```

Results

```
Your Weight(Kg) is 50
Your Height(M)is 1.50
Your BMI is 22.22
```

Comparison Operators

Used in comparing statements, Table 4-4 contains all the comparison operators used by Python.

Table 4-4:Comparison Operators

Comparison Operators		
Comparison Operator	**Meaning**	**Example**
==	equal	x = 8 y = 3 print(x == y) Results False
!=	not equal	x = 8 y = 3 print(x != y) Results True
>	more than	x = 8 y = 3 print(x > y) Results True
<	less than	x = 8 y = 3 print(x < y) Results False

Comparison Operators		
Comparison Operator	**Meaning**	**Example**
>=	greater or equal to	x = 8 y = 3 print(x >= y) Results True x = 8 y = 8 print(x >= y) Results True
<=	lesser or equal to	x = 8 y = 3 print(x <= y) Results False x = 8 y = 8 print(x <= y) Results True

Logical Operators

Logical operators combine two conditional propositions. They return true or false under the following tables:

For AND operators, use Table 4-5.

Table 4-5: AND Truth Table

AND Truth Table		
p	**q**	**p and q**
True	True	True

True	False	False
False	True	False
False	False	False

For OR operators, use Table 4-6.

Table 4-6: OR Truth Table

OR Truth Table		
p	**q**	**p or q**
True	True	True
True	False	True
False	True	True
False	False	False

For XOR operators, use Table 4-7.

Table 4-7: XOR Truth Table

XOR Truth Table		
p	**q**	**p ^ q**
True	True	False
True	False	True
False	True	True
False	False	False

For NOT operators, use Table 4-8.

Table 4-8: NOT Truth Table

NOT Truth Table	
p	**Not p**
True	False
False	True

Note: Statements are either true or false, where symbol p Represents the first proposition or sq represents the second.

For example:
a = 5
b = 6

p proposes a<b, which is True
q proposes a=b, which is False
Considering the table,
p and q, True and False, Result is False
p or q , True and False, Result is True
not p, not True, result is False

Examples in Table 4-9.

Table 4-9: Logical Operators

Logical operators	
Logical Operator	**Example**
and	x = 8 y = 3 print(x == y and x>y) Results False x==y is false x>y is True When checking the AND Truth Table, Result is False.
or	x = 8 y = 3 print(x == y or x>y) Results True x==y is false x>y is True When checking the OR Truth Table, Result is True.

Logical operators	
Logical Operator	**Example**
not	x = 8 y = 3 print(not(x == y or x>y)) Results False x==y is false x>y is True When checking the OR Table, Result is True, adding not, result is False. print(not(x == y and x>y)) Results True x==y is false x>y is True When checking the And Table, Result is False adding not, result is True.

Identity operators

Identity operators as shown in Table 4-10.

Table 4-10: Identity Operators

Identity Operators	
Identity operator	**Example**

Identity Operators	
Identity operator	**Example**
is	x = 8 y = 3 print(x is y) Results False x = 8 y = 8 print(x is y) Results True x = 8 y = "8" print(x is y) Results False

Identity Operators	
Identity operator	**Example**
is not	x = 8 y = 3 print(x is not y) Results True x = 8 y = 8 print(x is not y) Results False x = 8 y = "8" print(x is not y) Results True

Membership operators

Membership operators as shown in Table 4-11.

Table 4-11: Membership Operators

Membership Operators		
Membership operator	**Meaning**	**Example**

Membership Operators		
Membership operator	**Meaning**	**Example**
in	is a member of	fruit = ["strawberry", "mango", "banana"] print("banana" in fruit) Results True
		print("papaya" in fruit) Results False
not in	is not a member of	fruit = ["strawberry", "mango", "banana"] print("banana" not in fruit) Results False print("papaya" not in fruit) Results True

Bitwise operators

Bitwise operators as shown in Table 4-12.

Table 4-12: Bitwise Operators

Bitwise Operators		
Bitwise operator	**Meaning**	**Example**
&	And (for bits), that is, Converting both values in the logical operator into bits (binary) before performing said logical operator for each digit.	x = 8 y = 3 print(x & y) Results 0 This will compare each of the 16 digits, one by one, using the AND truth table, where 0 is false and 1 is true. 8 in binary is 0000000000001000 3 in binary is 0000000000000011 Using the AND logic operator for each bit: 0000000000001000 0000000000000011 --- 0000000000000000 Therefore, result is Results 0

Bitwise Operators		
Bitwise operator	**Meaning**	**Example**
|	Or (for bits), that is, Converting both values in the logic operator into bits (binary) before performing said logical operator for each digit.	x = 8 y = 3 print(x | y) Results 11 This will compare each of the 16 digits, one by one, using the OR truth table, where 0 is false and 1 is true. 8 in binary is 0000000000001000 3 in binary is 0000000000000011 Using the OR logic operator for each bit: 0000000000001000 0000000000000011 --- 0000000000001011 Therefore, result is Results 11
^	Xor (for bits), that is, Converting both values in the logic operator into bits (binary) before performing said	x = 5 y = 6 print(x^y) Results 3

	Bitwise Operators	
Bitwise operator	**Meaning**	**Example**
	logical operator for each bit	This will compare each of the 16 digits, one by one, using the XOR truth table, where 0 is false and 1 is true. 5 in binary is 0000000000000101 6 in binary is 0000000000000110 Using the XOR logic operator for each bit: 0000000000000101 0000000000000110 --- 0000000000000011 Therefore, result is Results 3 In Contrast to using the OR logic operator for each bit: 0000000000000101 0000000000000110 --- 0000000000000111 Results 7

Bitwise Operators		
Bitwise operator	**Meaning**	**Example**
~	Not (for bits), that is, Converting the value into bits (binary) before performing said logical operator for each digit.	x = 5 print(~x) Results -6 This will compare the 16 digits, one by one, using the NOT truth table, where 0 is false and 1 is true. 5 in binary is 0000000000000101 As a NOT, 0 converts to 1, while 1 becomes 0, like so: 111111111111010 is Results -6

Bitwise Operators		
Bitwise operator	**Meaning**	**Example**
<<	After Converting the value into bits (binary) Move all the digits to the left equal to the number specified after <<, replacing all displaced digits with 0.	x = 5 print(x << 2) Results 20 Convert x into 16 digit binary. 5 in binary is 0000000000000101 then move 2 digits to the left: 0000000000010100 Results 20
>>	After Converting the value into bits (binary) Move all the digits to the right equal to the number specified after >>, replacing all displaced digits with 0.	x = 5 print(x >> 2) Results 1 Convert x into 16 digit binary. 5 in binary is 0000000000000101
		then move 2 digits to the right: 00000000000001 is Results 1

Assignment operators

Assignment operators as shown in Table 4-13.

Table 4-13: Assignment Operators

Assignment Operators		
Assignment	**Example**	**Equivalent Result**
=	x=7	x=7
+=	x+=7	x=x+7
-=	x-=7	x=x-7
*=	x*=7	x=x*7
/=	x/=7	x=x/7
//=	x//=7	x=x//7
**=	x**=7	x=x**7
%=	x%=7	x=x%7
&=	x&=7	x=x&7
|=	x|=7	x=x|7
^=	x^=7	x=x^7
>>=	x>>=7	x=x>>7
<<=	x<<=7	x=x<<7

Assignment Examples

```
num=0
num +=1
print(num)
```

Results

```
1
```

```
num=0
num +=50
print(num)
```

Results

```
50
```

Deduct values

```
num=0
num -=50
```

Results

```
-50
```

display the variable using f-String

```
num=0
num -=50
Logic = True
print(f"num is {num} Logic is {Logic}")
```

Results

```
num is -50 Logic is True
```

Exercises

1. print(15+2**3+9/4*10%3%3) gives what result?
2. Using the formula for BMIin this chapter, Calculate your BMI.
3. If Tourist = [“ant”,”bee”,”kai”]
   print("banana" in Tourist) gives what result?
4. If x = 40 y = 35
   print(not(x == y and x>y)) gives what result?
   print(not(x == y and x&y)) gives what result?
   print(x ^ y and x|y) gives what result?
   print(x += y) gives what result?
   print(x //= y) gives what result?
   print(x <<= y) gives what result?

CHAPTER 5 CONDITIONAL STATEMENTS

When writing a conditional statement, The program checks whether the statement is true or false. If true, it executes 1.1, then 1.2 (if available) then 1.3 (if available) until all commands are executed. if false, it executes 2.1, then 2.2 (if available) then 2.3 (If available) until all commands are executed. In python, there are additional statements that help us write conditional statements, namely elif with...as and pass accordingly.

Conditional statements in python are as follows:

if..else
while..loop
for..loop
elif

Commands that make writing conditional statements more convenient are:

with...as, continue and pass

if

An **if** statement has the following syntax:

Syntax 5-1: if statement

```
if condition :
    command1
    command2
    ...
    commandN

Note
    command1 to commandN are executed when condition is True.
    N is the total number of commands executed
```

if..else

An **if..else** statement has the following syntax:

Syntax 5-2: if..else statement

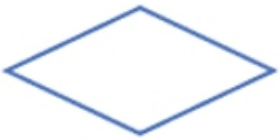
```
if condition :
    command1.1
    command1.2
    ...
    command1.N
else:
    command2.1
    command2.2
    ...
    command2.M

Note
    command1.1 to command1.N are executed when condition is True.
    command2.1 to command2.M are executed when condition is False.

    N is the total number of commands executed when condition is True.
    M is the total number of commands executed when condition is False.
```

Programming conditional statements is easier if you organize your thoughts with a **flow chart,** which is useful for debugging and software testers checking if the program is running properly. Symbols used in drawing Flow Charts are as Table 5-1.

Table 5-1: Symbols for Flow Chart

Symbol	Meaning
	**Start Program** or **Stop Program**
	**Connect**
	Direction - Show work in the direction of arrow
	**Process**
	**Input Data**
	**Decisions** that require conditional statements. Program by stating if, followed by condition. If true, executes the following set of commands, if false, execute commands following else:
	**Display**
	**Print**

Example Program

Programming to find BMI values and display results begins with designing and drawing a flow chart before coding the program, shown as Figure 5-1.

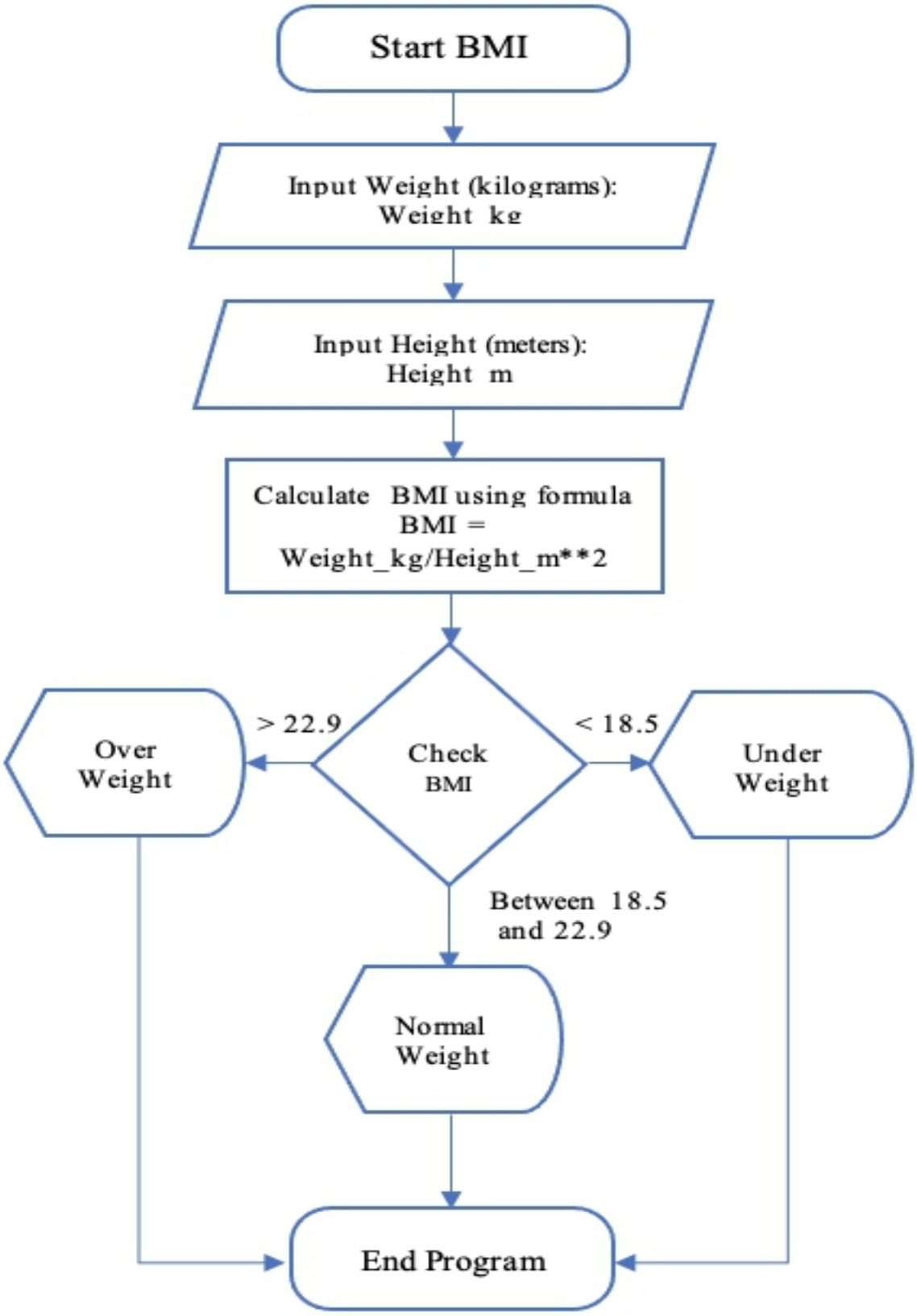

Start BMI
Input Weight (kilograms):
Weight_kg
Input Height (meters):
Height_m
Calculate BMI using formula
BMI =
Weight_kg/Height_m**2
Check
BMI
> 22.9
Over
Weight
< 18.5
Under
Weight
Between 18.5
and 22.9
Normal
Weight
End Program

Figure 5-1: Flow Chart for finding BMI values and displaying results.

Whether a person is over, under or normal weight, when inputting their weight and height, write a flow chart first to make programming easier.

Comparison operators, shown as Table 5-2 : Comparison operators

Table 5-2: Comparison operators

Comparison Operators	
Comparison Operator	**Meaning**
>	more than
<	less than
==	equal to
>=	greater than or equal to
<=	less than or equal to
!=	not equal to

To calculate the BMI value and know whether the result is over, under or normal weight, write a program using additional conditions of BMI table values to find the result, that is, using if or else, which after if must be a condition that gives the result TRUE or FALSE (FALSE) and when the result is true, it will do command 1 and if it is false, it will do command 2 as follows.

```
Weight_kg        = float(input('Your Weight(Kg) is '))
Height_m         = float(input('Your Height(M) is '))
BMI              = round(float(Weight_kg/Height_m**2),2)
if BMI > 22.9:
  print("You are overweight")
```

Results

```
Your Weight(Kg) is 60
Your Height(M)is 1.4
You are overweight
Weight_kg        = float(input('Your Weight(Kg) is '))
Height_m         = float(input('Your Height(M) is '))
```

```
BMI             = round(float(Weight_kg/Height_m**2),2)
if BMI > 22.9:
   print("You are overweight")
else: print("You are normal or underweight")
```

Results

```
Your Weight(Kg) is 40
Your Height(M)is 1.60
You are normal or underweight
```

if..elif..else

In the case of overlapping conditions, an **if..elif..else** command has the following syntax:

Syntax 5-3: if..elif..else statement

```
if condition1:
    command1.1
    command1.2
    ...
    command1.L
elif condition2 :
    command2.1
    command2.2
    ...
    command2.M
else:
    command3.1
    command3.2
    ...
    command3.N
```

Note

- commands are executed from 1.1 to 1.L when condition$_1$ is true.
- L is the total number of commands executed when condition1 is true.
- commands are executed from 2.1 to 2.M when condition$_1$ is false and condition$_2$ is true.
- M is the total number of commands executed when condition$_1$ is false and condition$_2$ is true.
- commands are executed from 3.1 to 3.N when condition$_1$ is false and condition$_2$ is false.
- N is the total number of commands executed when condition$_1$ is false and condition$_2$ is false.

Commands can be programmed as if_else as well.

```
Weight_kg         = float(input('Your Weight(Kg) is '))
Height_m          = float(input('Your Height(M) is '))
BMI               = round(float(Weight_kg/Height_m**2),2)
if BMI > 22.9:
   print("You are overweight")
elif BMI < 18.5:
    print("You are underweight")
else:
   print("You are normal weight")
```

Results

```
Your Weight(Kg) is 50
Your Height(M)is 1.5
You are normal weight
```

Results

```
Your Weight(Kg) is 60
Your Height(M)is 1.4
You are overweight
```

Results

```
Your Weight(Kg) is 40
```

```
Your Height(M)is 1.60
You are underweight
```

You can also write the program as nested if_else chains. For example, if condition 1 is this:

```
 if BMI < 18.5:
   print("You are underweight")
else:
    print("You are normal")
```

Therefore, you can write the BMI calculation program in this manner:

```
Weight_kg         = float(input('Your Weight(Kg) is '))
Height_m          = float(input('Your Height(M) is '))
BMI               = round(float(Weight_kg/Height_m**2),2)
if BMI < 22.9:
   if BMI < 18.5:
      print("You are underweight")
   else:
     print("You are normal")
else:
   print("You are overweight")
```

Results

```
Your Weight(Kg) is 50
Your Height(M)is 1.5
You are normal weight
Your Weight(Kg) is 60
Your Height(M)is 1.4
You are overweight
Your Weight(Kg) is 40
Your Height(M)is 1.60
You are underweight
```

while..loop

The syntax of the **while..loop** conditional statement has the following syntax:

Syntax 5-4: while..loop statement

```
i = 0
j= K
while i<j:
    command1.1
    command1.2
    …
    command1.N
    i += 1
```

where i is the variable used to count the cycles.
j is a numeric variable that defines a cycle equal to K cycles.

Note :

- commands are executed from 1.1 to 1.L when i<j
- N is the total number of commands executed when i<j.

while..loop Is used in case of when you want a program to run for a certain number of cycles until a number of determined cycles is reached.

Example

Programming to display numbers 0-10 begins with designing and drawing a flow chart before coding the program, shown as Figure 5-2.

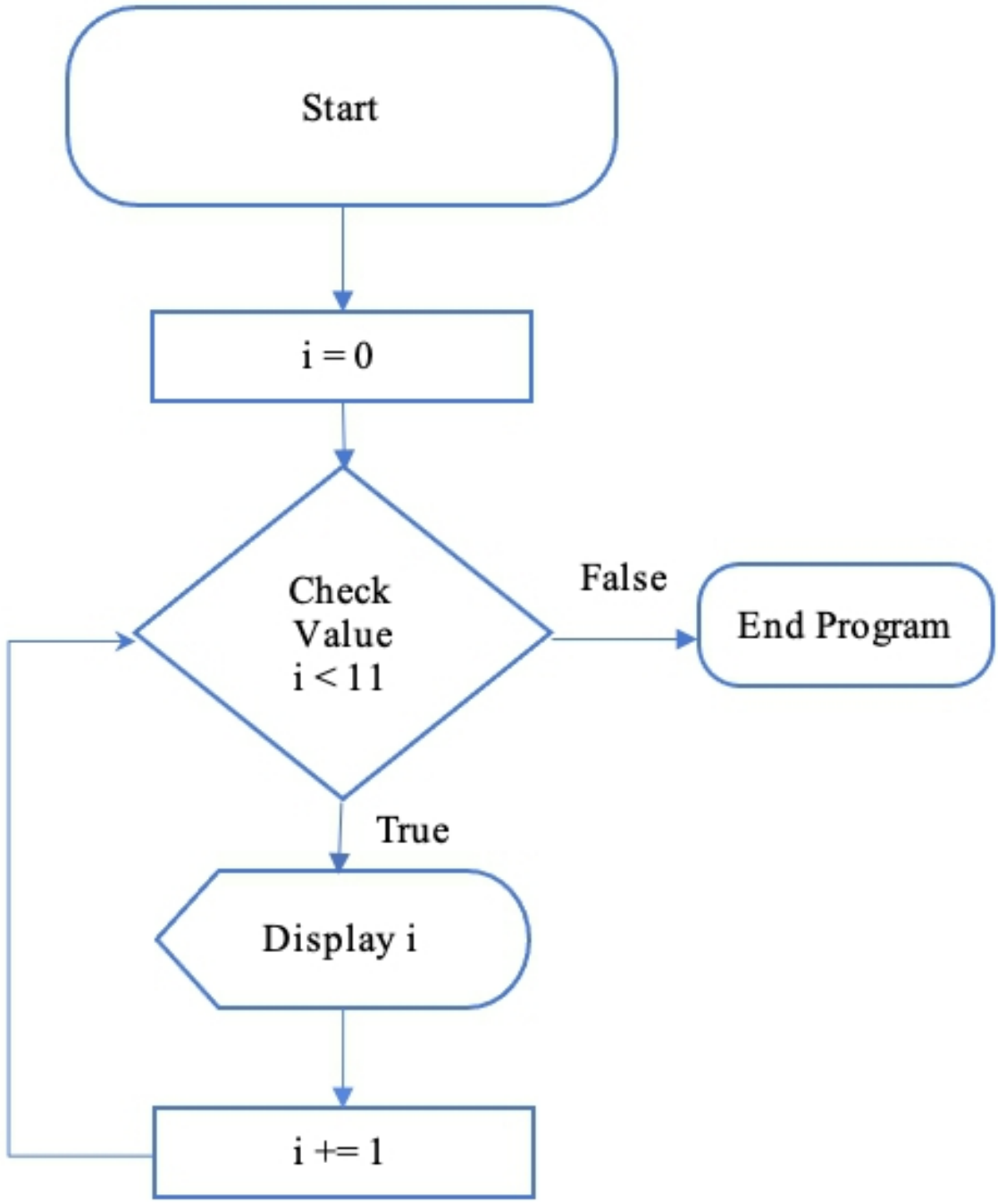


Figure 5-2: Flow Chart for Displaying number 0 to10

Program for displaying the numbers 0-10

```
i = 0
while i < 11:
  print(i)
```

```
 i += 1
```

Results

```
0
1
2
3
4
5
6
7
8
9
10
```

We can use **break** during the process if there is a condition that requires a break. The syntax of the **while..loop** with **break** as shown in Syntax 5-5

Syntax 5-5: while..loop with break statement

```
i = 0
j= K
while i<j:
    command1.1
    command1.2
    …
    command1.M
    if condition :
      command2.1
      command2.2
      …
      command2.N
      break
    i += 1
```

where i is the variable used to count the cycles.
j is a numeric variable that defines a cycle equal to K cycles.

Note :

- commands are executed from 1.1 to 1.M when i<j
- M is the total number of commands executed when i<j.
- commands are executed from 2.1 to 2.N and exit after executed when i<j and condition is True.
- N is the total number of commands executed when i<j and condition is True..

Example program:

Programming to display index of “durian” with designing and drawing a flow chart before coding the program, shown as Figure 5-3.

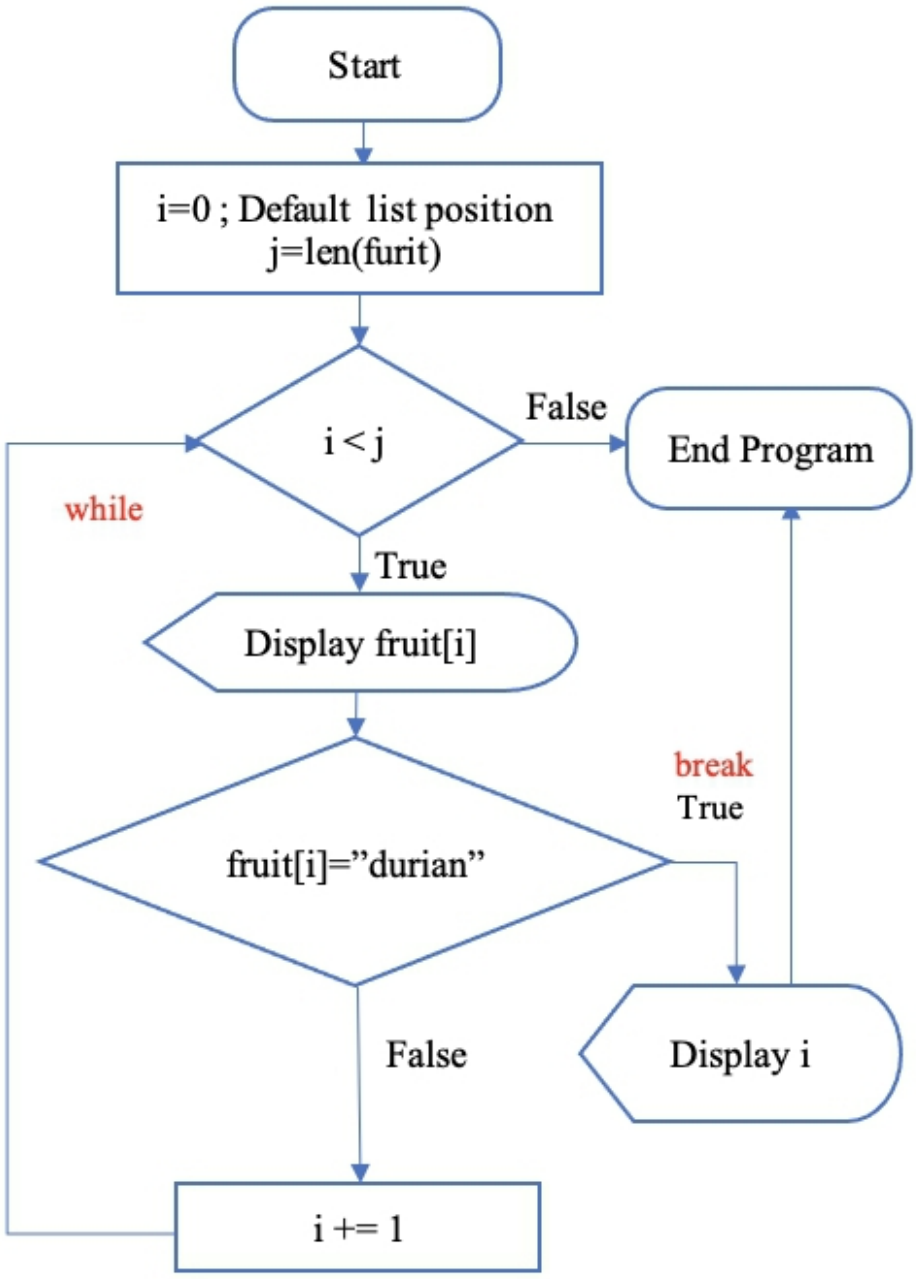


Figure 5-3: Flow Chart for finding index of “durian” in list

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
i = 0
```

```
j = len(fruit)
while i < j:
  print(fruit[i])
  if (fruit[i] == "durian"):
    print("position of durian is "+str(i))
    break
  i += 1
```

Results
orange
mango
 banana
rambutan
durian
position of durian is 4

with…as

with...as is a command that is similar to defining a variable with a different way to code, but yields the same results. Examples of its use could be found in Chapter 11: with…as.

continue

Or if you want to show all the items in the list along with a certain item's position, you can use **continue**, but remember to use **break** to not go beyond the total values in the list, which can cause errors in the display.

We can use **break** and **continue** during the process if there is a condition that requires **break** and **continue**. The syntax of the **while..loop** with **break** and **continue** as shown in Syntax 5-6

Syntax 5-6: while..loop with break and continue statement

```
i=0
j=K
while  i<j :
    command1.1
    command1.2
    …
    command1.L
    i+=1
   if condition1:
     command2.1
     command2.2
     …
     command2.M
     break
   if condition2:
     command3.1
     command3.2
     …
     command3.N
     continue
```

where i is the variable used to count the cycles.
j is a numeric variable that defines a cycle equal to K cycles.

Note :

- commands are executed from 1.1 to 1.L when i<j
- L is the total number of commands executed when i<j.
- commands are executed from 2.1 to 2.M and exit after executed when i<j and $condition_1$ is True.
- M is the total number of commands executed and exit the loop when i<j and condition is True.
- commands are executed from 3.1 to 3.N and continue the loop when i<j and $condition_2$ is True.
- N is the total number of commands executed and continue the loop when i<j and $condition_2$ is True.

Example Program:

Programming to display index of “durian” and display all of fruit in list with designing and drawing a flow chart before coding the program, as shown in Figure 5-4.

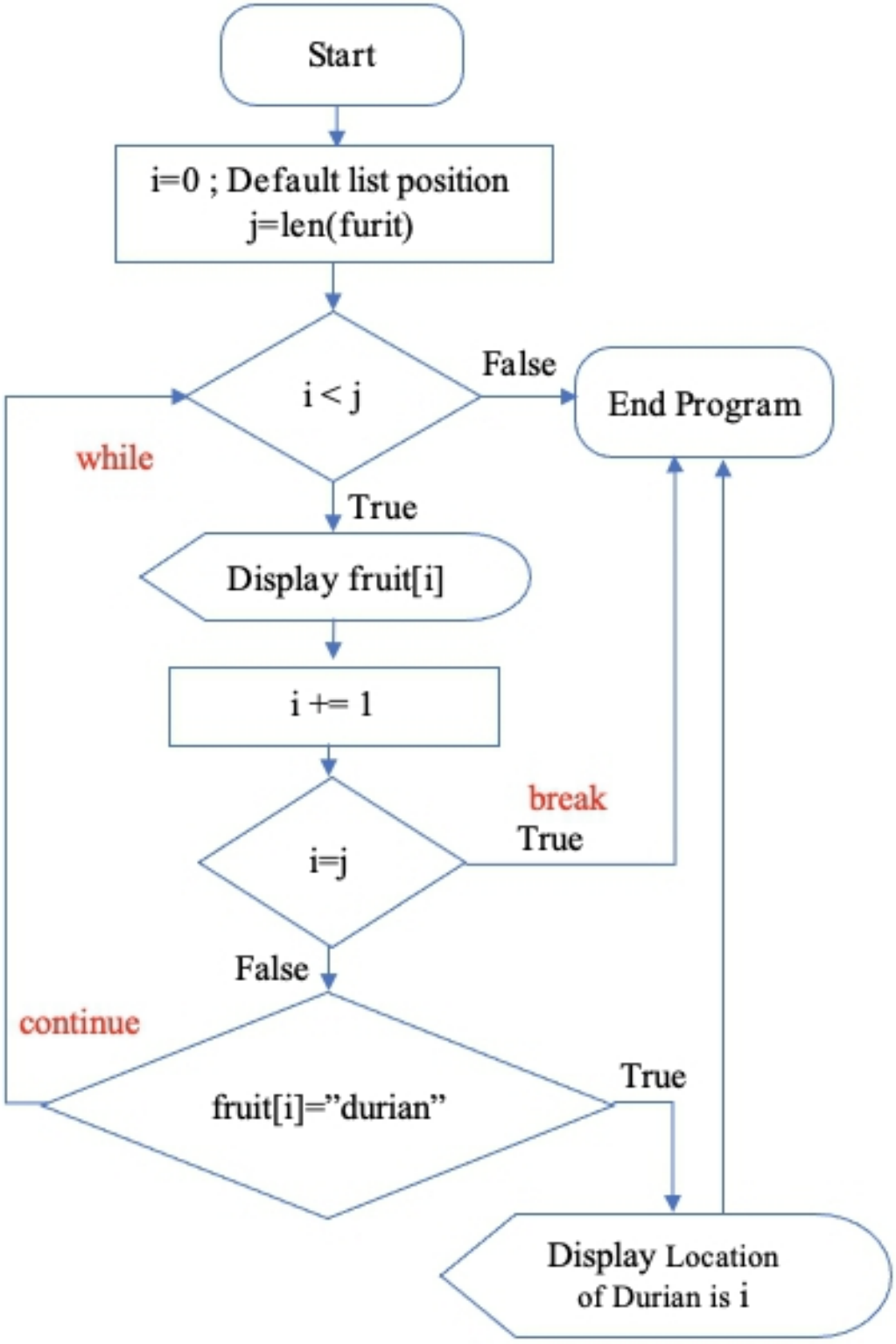

Start
i=0 ; Default list position
j=len(furit)
i < j
False
End Program
while
True
Display fruit[i]
i += 1
break
i=j
True
False
continue
fruit[i]="durian"
True
Display Location
of Durian is i

Figure 5-4: Flow Chart to display index of “durian” and display all of fruit in list

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
i = 0
j = len(fruit)
while i < j:
  print(fruit[i])
  i += 1
  if (i == j): break
  if (fruit[i] == "durian"):
    print("Location of Durian is "+str(i))
    continue
```

Result

```
orange
mango
banana
rambutan
Location of Durian is 4
durian
pineapple
papaya
```

while..else

We can use **else** during the process if there is a condition that requires **else**. With this, after completing the designated number of loops, will execute commands after **else**.The syntax of the **while** with **else**, as shown in Syntax 5-7:

Syntax 5-7: while..else statement

```
i = 0
j= K
while i<j:
    command1.1
    command1.2
    …
    command1.M
    i += 1
else :
    command2.1
    command2.2
    …
    command2.N
```

where i is the variable used to count the cycles.
j is a numeric variable that defines a cycle equal to K cycles.

Note :

- commands are executed from 1.1 to 1.M when i<j
- M is the total number of commands executed when i<j.
- commands are executed from 1.1 to 1.N when i>=j
- N is the total number of commands executed when i>=j.

Example

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
i = 0
j = len(fruit)
```

```
while i < j:
  print(fruit[i])
  i += 1
else :
print("End of fruit list")
```

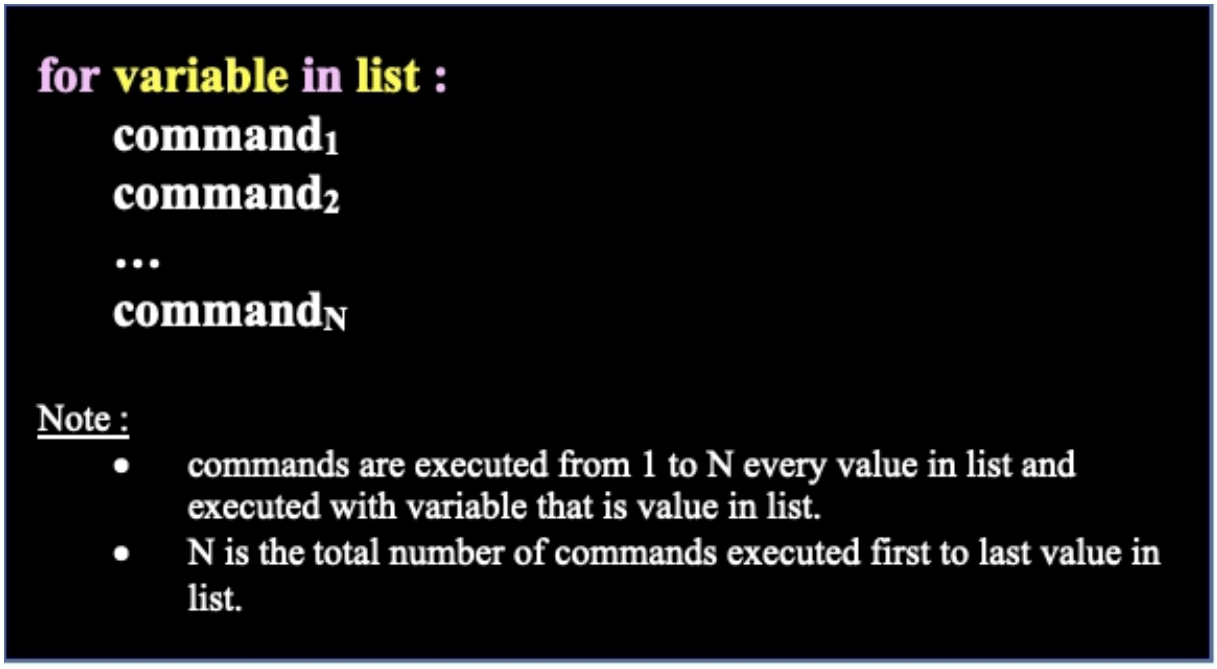
Results

```
orange
mango
 banana
rambutan
durian
pineapple
papaya
End of fruit list
```

for..loop

The syntax of **for..loop** is as follows:

Syntax 5-8: for..loop statement



For..loop is for when you want to write a command to cycle through the desired command from the first value in the list (position 0) until the

last value in the list (position len(list)-1).

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
for n in fruit:
  print(n)
```

Results

```
orange
mango
  banana
rambutan
durian
pineapple
  papaya
```

We can use a break during processing if there are conditions that require a break.

The syntax of **for..loop with break** is as follows:

Syntax 5-9:for..loop with break statement

```
for variable in list :
    command1.1
    command1.2
    …
    command1.N
    if condition
      command2.1 :
      command2.2
      …
      command2.M
      break
```

Note :

- commands are executed from 1.1 to 1.N from value the first to the last value in list.Unless the condition is True the loop will stop.
- N is the total number of commands executed from value the first to the last value in list.Unless the condition is True the loop will stop.
- commands are executed from 2.1 to 2.M when condition is True and stop looping.

Example Programs:

If you wish to break once you find "mango" in the list:

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
for n in fruit:
  print(n)
  if n == "mango":
    print(len(fruit))
    break
```

Results

orange
mango
7

If you wish to break once you find "mango", as well as display its position in the list:

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
p = 0
for n in fruit:
  print(n)
  p=p+1
  if n == "mango":
    print(p)
    break
```

Results

orange
mango
2

Caution: Indentations

Caution: Indentations matter for commands, and will have different effects on how statements are executed. For example:

For if…else

Example A

```
m = 2
n = m = 2
n = 33
if m > n:
    print("m>n")
```

```
   print("m is more n")
print("results as stated")
```

Results **orange**

results as stated

Example B

```
m = 2
n = 33
if m > n:
    print("m>n")
print("m is more n")
print("results as stated")
```

Results

m is more than n

results as stated

Because in example A, print("m is more n") **is within** if**, whereas example B has** print("m is more n") **outside** if.

For while..loop

Example A

```
i = 0
while i < 11:
  i += 1
  print(i)
```

Results

```
1
2
3
4
5
6
7
8
9
```

```
10
11
```

Example B

```
i = 0
while i < 11:
  i += 1
print(i)
```

Results

```
11
```

Because in example A, print(n) **is within** while**, whereas example B has** print(n) **outside** while.

For for..loop

Example A

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
p = 0
for n in fruit:
  if n == "mango":
    continue
   print(n)
```

Results

```
orange
mango
  banana
rambutan
durian
pineapple
papaya
```

Example B

```
fruit =
list(("orange","mango","banana","rambutan","durian","pineapple","papaya"
))
p = 0
for n in fruit:
  if n == "mango":
    continue
print(n)
```

Results

papaya

Because in example A, print(n) **is within** for**, whereas example B has** print(n) **outside** for.

elif

elif checks if the condition preceding **if** is not true, executes **elif** commands instead.

Example

```
num1 = 2
num2 = 10
if num2>num1:
  print("num2 is more than num1")
elif num1 == num2:
  print("num2 is equal to num 1")
else:
  print("num1 is more than num2")
```

Results num2 is more than num1

pass

pass check if the condition is true, will not display anything.

Example

```
num1 = 2
num2 = 10
if num2>num1:
  pass
elif num1 == num2:
```

```
  print("num2 is equal to num 1")
else:
  print("num1 is more than num2")
```

Results

Nothing is displayed

Exercises

1. Draw a flowchart, then write a program to check the conditions for variable values with positive integers displays '+', negative integers displays '-' and displays '0' when variable values have a value of 0.
2. Repeat Exercise 1, but only use the comparison operator > in programming.
3. Repeat Exercise 1, but only use the comparison operator >= in programming.
4. Repeat Exercise 1, but only use the comparison operator != in programming.
5. Repeat Exercise 1, but use while…loop in programming, where it accepts 5 values to display.
6. Repeat Exercise 1, but use while…loop in programming to only allow input of values less than 100 in the display. If the variable has a value greater than 100, exit the program and display the variable value.
7. Repeat Exercise 1, but use while…else in programming to only allow input of values less than 100 in the display. If the variable has a value greater than 100, exit the program and display the variable value.
8. Repeat Exercise 1, but use for…loop in programming to only allow input of values less than 100 in the display. If the variable has a value greater than 100, exit the program and display the variable value.

CHAPTER 6 FUNCTIONS

Creating a function for Python is creating a block of program code that runs when it is called and can pass data called parameters into the function. The function can return data as a result.

define function

The following is the syntax for defining a function in Python:

Syntax 6-1: defining a function

```
def functionName(parameter1,…,parameterM ) :
    command1
    …
    commandN
```

Call Function

```
functionName(argument1, argument2,…, argumentM)
```

Note :

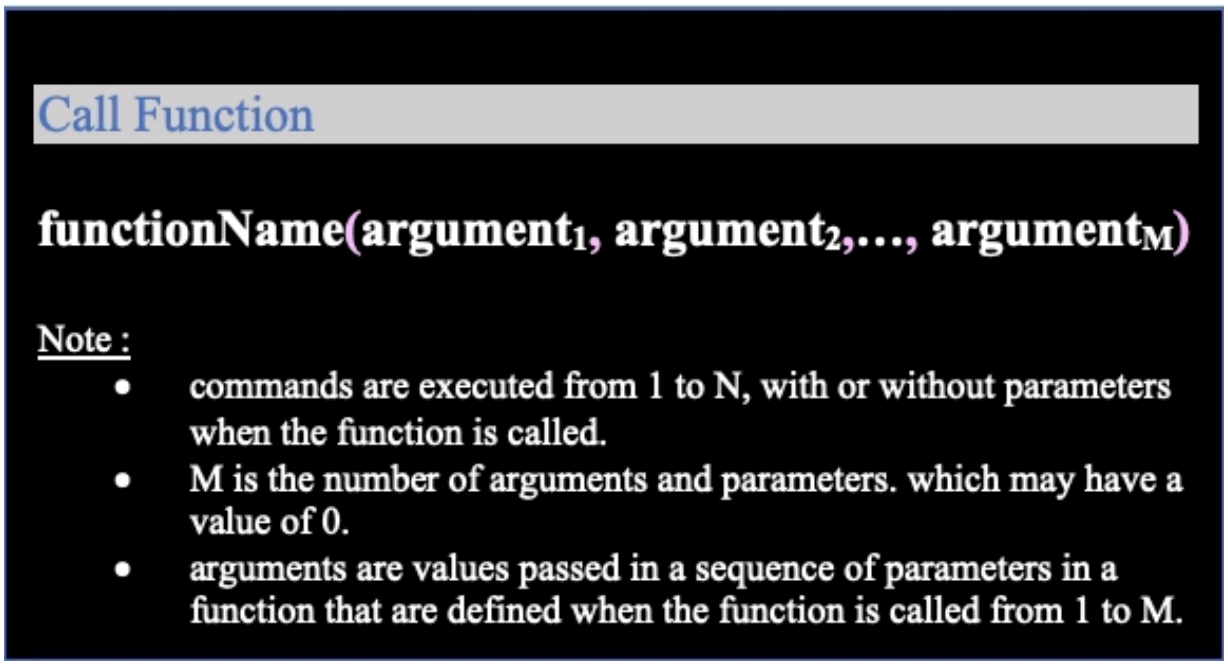
- commands are executed from 1 to N, with or without parameters when the function is called.
- M is the number of parameters. which may have a value of 0.
- N is the total number of commands executed when the function is called.
- arguments are values passed in a sequence of parameters in a function that are defined when the function is called from 1 to M.

Where $parameter_1,\ldots,parameter_N$ are **arguments**. Whether arguments are present or not depends on whether perimeters are needed.

calling a function

To call a function, use the following syntax:

Syntax 6-2: calling a function



Where value$_1$,…,value$_M$ are **arguments**. Whether arguments are present or not depends on whether perimeters are needed to execute that function.

Example Function Programs

```
def favoriteFruit():
  print("My favorite fruit is "+"orange")
favoriteFruit()
```

Results

```
My favorite fruit is orange
```

when the argument parameter$_1$ is fFruit

```
def favoriteFruit(fFruit):
  print("My favorite fruit is "+fFruit)

favoriteFruit("apple")
```

Results

```
My favorite fruit is apple

favoriteFruit("orange")
```

Results

```
My favorite fruit is orange
```

If you want to add a *parameter* to include the name of the person as well.

```
def favoriteFruit(fName,fFruit):
  print("The fruit "+fName+" likes most is "+fFruit)

favoriteFruit(“Som”,"orange")
```

Results

```
The fruit Som likes most is orange
```

```
favoriteFruit("Sansuay","mango")
```

Results

```
The fruit Sansuay likes most is mango
```

Caution: include all *parameters* so it can function without errors.

define a function with *parameter

For Python, in cases where we don't know the exact number of arguments to be passed to our function, place a asterisk (*) in front of parameter. The *parameter must be a tuple.

The following is the syntax for defining a function with *parameter in Python:

Syntax 6-3: defining a function with *parameter

```
def functionName(parameter1,...,parameterL,
                 *parameter1,...,*parameterM) :
    command1
    ...
    commandN
```

Call Function

```
functionName(argument1, argument2,..., argumentL ,
*argument1, *argument2,..., *argumentM)
```

Note :

- commands are executed from 1 to N, with or without parameters and with or without *parameters when the function is called.
- L is the number of parameters. which may have a value of 0.
- M is the number of *parameters. which may have a value of 0.
- *parameter must be tuple.
- N is the total number of commands executed when the function is called.

Example Program

```
def favoriteFruit(*fFruit):
  print("At least 2 of my favorite fruits are "+fFruit[0]+" "+fFruit[1])
favoriteFruit("orange","banana","mango",
"watermelon")
```

Results

```
At least 2 of my favorite fruits are orange banana
```

```
def favoriteFruit(fName,lName,*fFruit):
  print("At least 2 fruits that"+fName+" "+lName+" likes most are
"+fFruit[0]+" "+fFruit[1])
favoriteFruit("Sansuay","Sakunsaendeengam","Orange",
"Banana","Mango", "Watermelon")
```

Results

At least 2 fruits thatSansuay Sakunsaendeengam likes most are Orange
Banana

To display every item of fruit, write a program like this:

```
def favoriteFruit(fName,lName,*fFruit):
  print("The fruits that "+fName+" "+lName+" likes most are ")
  for name in fFruit:
        print(name)
favoriteFruit("Sansuay","Sakunsaeneengam","Orange",
"Banana","Mango","Watermelon")
```

Results

```
The fruits that Sansuay Sakunsaeneengam likes most are
Orange
Banana
Mango
Watermelon
```

defining a function with **parameter

If we don't know the number of parameters to pass to the function, use double-asterisks ** before the parameter name.

The following is the syntax for defining a function with **parameter in Python:

Syntax 6-4: defining a function with **parameter

```
def functionName(**parameter) :
    command1
    ...
    commandL
```

Note 1:

- When using parameters, you must refer to parameters and variables as follows in commands:

```
parameter["variable"]
```

Call Function

```
functionName(variable1=argument1,
             variable2=argument2,...,
             variableM=argumentM)
```

Note 2 :

- commands are executed from 1 to L, with parameters when the function is called.
- L is the number of commands are executed, when the function is called.
- M is the number of parameters.

Example Program

```
def favoriteFruit(**fFruit):
    print("The fruits that "+fFruit["fName"]+" "+
    fFruit["lName"]+" likes most are "    +fFruit["fFruit1"]
    +" "+fFruit["fFruit2"]+" "+fFruit["fFruit3"]
    +fFruit["fFruit4"])
favoriteFruit(fName ="Sansuay",lName
="Sakunsaendeengam",fFruit1="orange",fFruit2="banana",fFruit3="mango
",
```

```
fFruit4="watermelon")
```

Results

```
The fruits that Sansuay Sakunsaendeengam likes most are orange banana mangowatermelon
```

defining a function with default values to parameters

The following is the syntax for defining a function with default values to parameters in Python:

Syntax 6-5: defining a function with default values to parameters

```
def functionName(parameter1 = value1.1,
                 parameter2 = value1.2,...,
                 parameterM = value1.M) :
    command1
    ...
    commandN

Call Function

functionName()
functionName(parameter1 = value2.1,
             parameter2 = value2.2,...,
             parameterM = value2.M)
```

Note :

- commands are executed from 1 to N, with default values (1.1 to 1.M) assigned to each parameter(1 toM). If no value is passed in the function call but if in calling the function there is a value (2.1 to 2.M) set in any parameterm that value will be used instead.
- N is the number of commands are executed, when the function is called.
- M is the number of parameters, whose values range from 0.

Example Program

```
def favoriteFruit(fName="Sansuay",lName
    ="Sakunsaedeengam",fFruit="orange"):
    print("The fruit that "+fName+" "+lName+" likes is
   "+fFruit)
  favoriteFruit(fName ="Ma",lName
  ="Boonlaimakmay",fFruit="papaya")
```

```
  favoriteFruit()
```

Results

```
The fruit that Ma Boonlaimakmay likes is papaya

The fruit that Sansuay Sakunsaedeengam likes is orange
```

In python, we can send list data as follows:

```
def favoriteFruit(fFruit):
  for n in fFruit:
    print("My favorite fruit is "+n)
fruit = ["orange","banana","mango","watermelon"]
favoriteFruit(fruit)
```

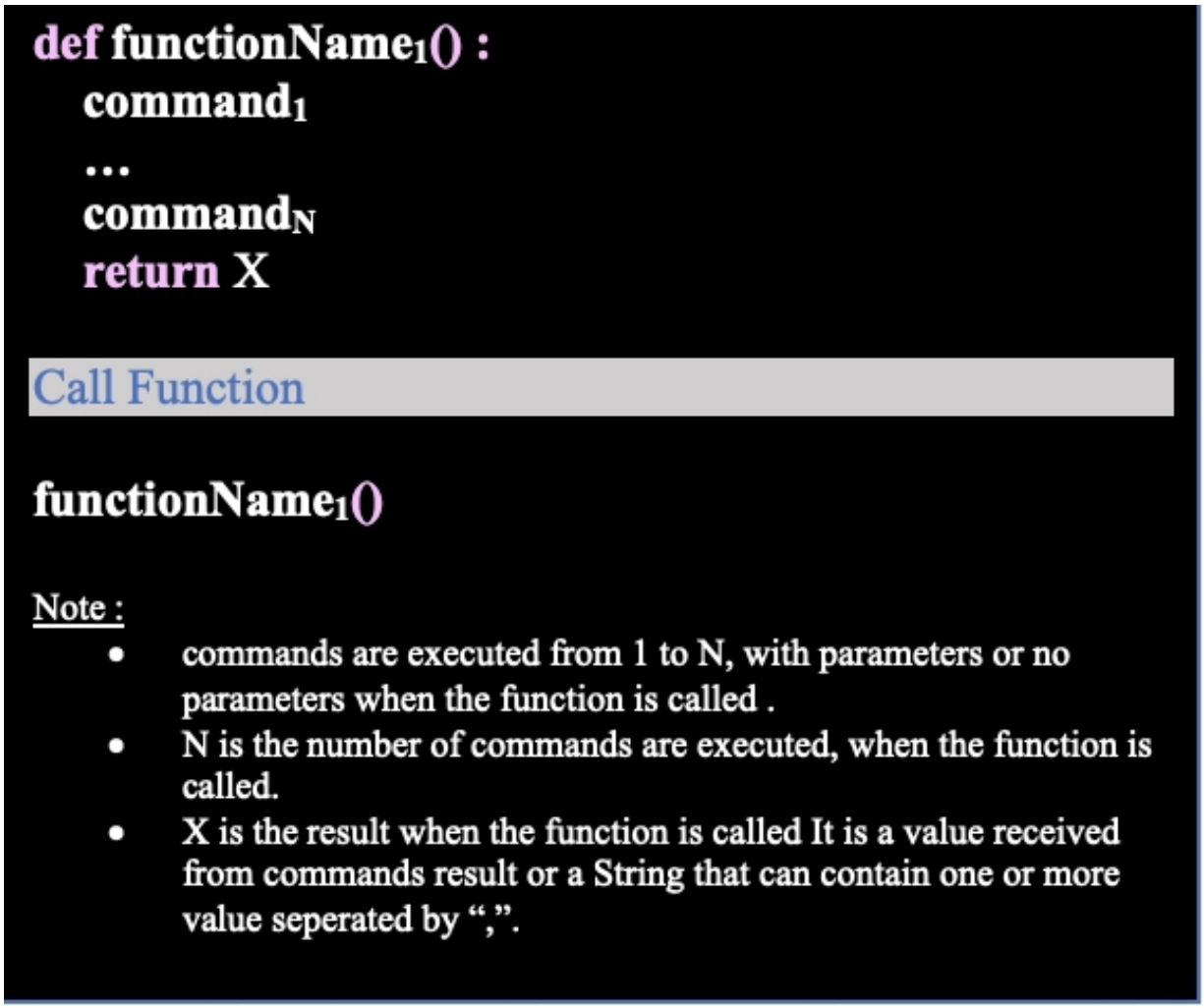
Results

```
My favorite fruit is orange
My favorite fruit is banana
My favorite fruit is mango
My favorite fruit is watermelon
```

defining a function with return

Using **return** returns a value when calling a function. The following is the syntax for defining a function with return in Python:

Syntax 6-6: defining a function with return



Example

```
def my_function(n):
  print("The input value is "+str(n))
  return "The return value is "+str(10 * n)
print(my_function(20))
print(my_function(5))
print(my_function(34))
```

Results

```
The input value is 20
The return value is 200
The input value is 5
The return value is 50
```

```
The input value is 34
The return value is 340
```

Using the returned value as a variable can be done as follows:

Example Program

```
def my_function(n):
  print(“The input value is “+str(n))
  return “The return value is “+str(10 * n)
m = my_function(20)
print(m)
```

Results

```
The input value is 20
The return value is 200
```

Example Program

```
def cal(a, b):
    result = a + b
    result2 = a -b
    return "result a+b=",result,"result a+b=",result2
def nocal():
    return "no cal"
print(cal(2,2))
print(nocal())
```

Results

```
('result a+b=', 4, 'result a+b=', 0)
no cal
```

Example Program

```
def factorial(x):
    if x == 1:
        return 1
```

```
    else:
        return (x * factorial(x-1))
n = 5
print("Factorial value of", n, "is", factorial(n))
```

Results

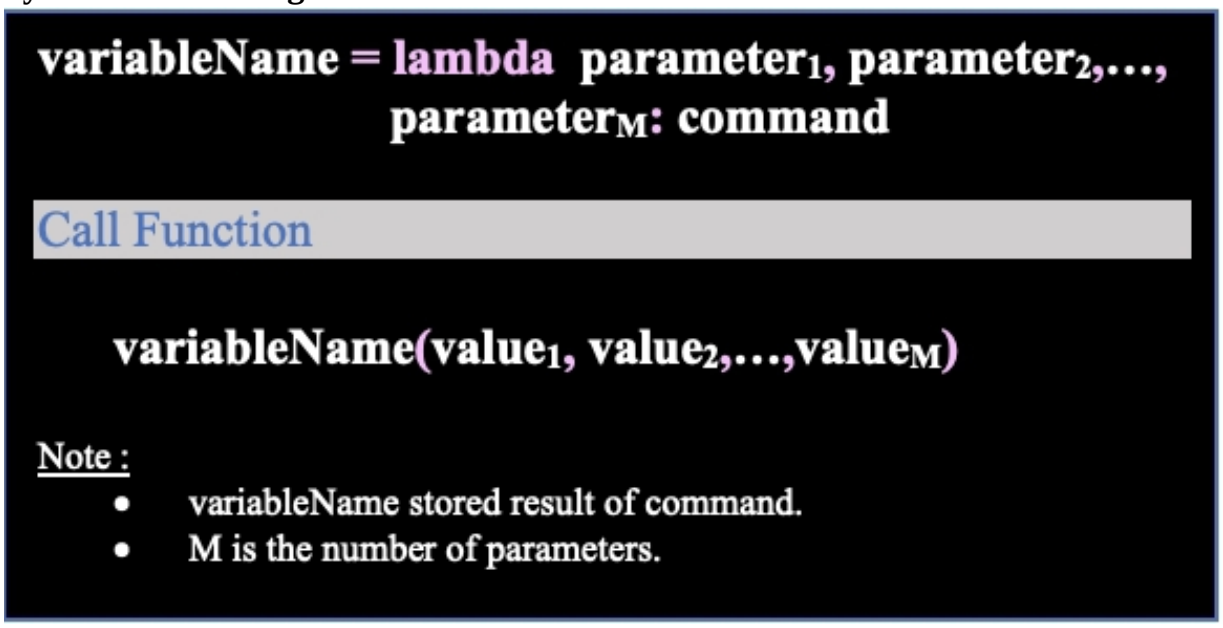
```
Factorial value of 5 is 120
```

Lambda Functions

A Lambda function in Python is a block of code that runs when executed and can pass data called parameters to the function. A function can return data as a result as a small function written in one line.

The following is the syntax for defining a Lambda function in Python:

Syntax 6-7: defining a Lambda function



where $parameter_1$,…,$parameter_N$ are **arguments** which may or may not be present. while the command can use $parameters_1$,…, $parameter_N$.

Example Program

```
sawasdee = lambda : print("Hi")
sawasdee()
```

Results

```
Hi
```

```
sawasdee = lambda name: print("Hi "+name)
sawasdee("Sansuay")
```

Results

```
Hi Sansuay
```

```
add5 = lambda n : n + 5
print(add5(5))
```

Results

```
10
```

```
multi = lambda n,m : n * m
print(multi(5,6))
```

Results

```
30
```

defining a function with return value of the lambda function

The following is the syntax for defining a function with return value of the lambda function in Python:

Syntax 6-8: defining a function with return value of the lambda function

```
def functionName(parameter1.1,
                 parameter1.2,
                 ...,parameter1.L,
                 parameter2.1= value2.1,
                 parameter2.2=value2.2,
                 ...,parameter2.M= value2.M) :
    command1.1
    ...
    command1.N
    return lambda  parameter3.1, parameter3.2,
                   ...,parameter3.P: command2

variable = functionName(argument1.1, argument1.2,
                        ..., argument1.L,
                        parameter2.1= value3.1,
                        parameter2.2=value3.2,
                        ...,parameter2.M= value3.M)
```

Note :

- commands are executed from 1.1 to 1.N, with parameters (1.1 to 1.L and 2.1 to 2.M) or without parameters (1.1 to 1.L and 2.1 to 2.M) when the function is called and $command_2$ is excuted with parameters (3.1 to 3.P) or without parameters (3.1 to 3.P).
- arguments are actual value that are passed to the function in the order their parameters are defined from 1.1 to 1.L.
- The values from 3.1 to 3.M are used instead of the values from 2.1 to 2.M respectively if they were defined then the function was called.
- L is the number of parameters of the function without default value .
- M is the number of parameters of the function with default value.
- N is the number of commands of the function
- P is the number of parameters of the lambda for $command_2$.

Call Function that return lambda

```
variable(argument3.1, argument3.2,..., argument3.P)
```

Note :

- $command_2$ are executed when using variable.
- arguments are actual value that are passed to $command_2$ in the order their parameters are defined from 3.1 to 3.P.

The lambda command can use function parameters as demonstrated in the following

Example programs

```
def funcandlambda(n):
  return lambda m : m * n
callfunclamb = funcandlambda(10)
print(callfunclamb(50))
```

Results

```
500
```

Finding exponents

```
def powerVal(n):
  return lambda m : m ** n
power2 = powerVal(2)
power3 = powerVal(3)
print(power2(5))
print(power3(5))
```

Results

```
25
125
```

```
def funcandlambda(n,q=5):
  q=4
  n=6
  return lambda m : m * n * q
callfunclamb = funcandlambda(10)
print(callfunclamb(50))
```

Results

```
1200
```

```
def funcandlambda(n,q=5):
  q=4
  n=6
```

```
  return lambda m,p : m * n * q * p
callfunclamb = funcandlambda(10)
print(callfunclamb(50,10))
```

Results

```
12000
```

Exercises

1. Create a function called sumX to get the values of 5 variables and sum them up and display the result.
2. Create a lambda function to take the values of 5 variables and combine them and display a result.

CHAPTER 7 MODULES

To create a module, go to https://replit.com/ which lets you write Python code through a browser.Once there, you will find a previously created module called “main.py”. Let’s create a new file and name a new .py module with the following syntax:

create module

Syntax 7-1: create python file

```
moduleName.py
```

using module

By naming our module my_module.py, we will get a module named my_module. When we want to use it, use the following syntax:

Syntax 7-2: use a module

```
import moduleName
```

For example, write the my_module.py module as follows:

```
x = 5
```

After saving my_module.py in the main.py program, use the **import** my_module command. After that, we can call the defined variable or function saved in that module. The format of the command to call the variables saved in the module is as follows.

using variable in module

Syntax 7-3: use a variable in module

For example when you need variable x,

```
import my_module

print(my_module.x)
```

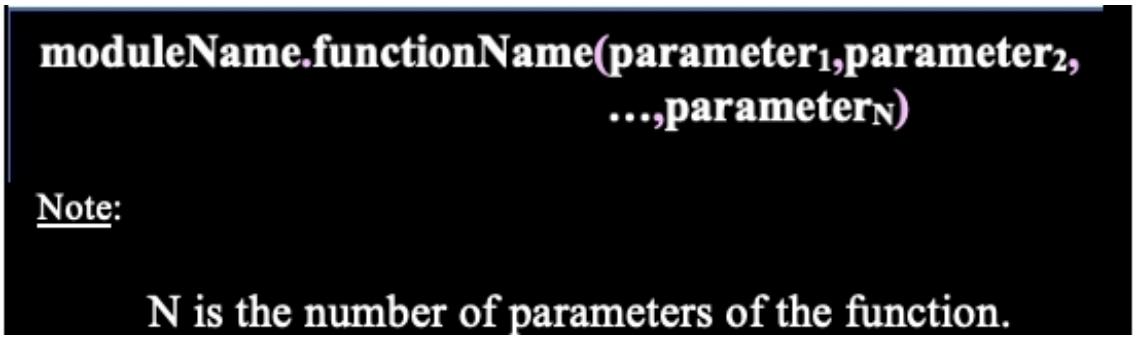
Results

```
5
```

using function in module

We can add and use functions in a module, and can call the variables saved in the module with the following syntax:

Syntax 7-4: use a function in module



Note: the presence or absence of parameters depends on the functions in the module.

For example, when we program a function to my_module.py as follows:

```
def sawasdee(name):
  print("Hi" + name)
```

When we run my_module and call this function in the program called by writing the program as follows.

```
import my_module
my_module.sawasdee("Sansuay")
```

Results

```
Hi Sansuay
```

using dictionary in module

We can also create dictionaries in modules, and call it with the following syntax:

Syntax 7-5: use a dictionary in module

```
moduleName.dictionaryName['key']
```

Example Program

Add a dictionary to the my_module module:

```
Tourist_gender =
  {'Ammy':'Female','Jame':'Male','Mike':'Male',
   'Minnie':'Female'}
```

Then, run my_module and call this function as follows:

```
import my_module
print(my_module.Tourist_gender['Jame'])
```

Results

```
Male
```

rename module

To rename the module, use the following syntax:

Syntax 7-6: rename a module

```
import moduleName as renamedModule
```

Once the name is changed, the new module name must be used throughout.

Example Program

```
import my_module as mm
print(mm.Tourist_gender['Jame'])
```

Results

```
Male
```

Python has built-in modules that can be viewed at

https://docs.python.org/3/py-modindex.html

using random module

To use the random module to randomly pick an integer value between 1 to 10, import random and write the program as follows:

```
import random
random_int = random.randint(1,10)
print(random_int)
```

When run, it displays a random value between 1 to 10.

First executed result could be:

```
2
```

Second executed result could be:

```
4
```

python has created a Python Random Module that has functions that can be used without having to write a new program. You can go see more at:

https://docs.python.org/3/library/random.html

Frequently Python Random Module is shown in Table 7-1

Table 7-1: Frequently Used Functions of Random Module Provided by Python

Frequently Used Functions of Random Module Provided by Python		
Function	**Description**	**Example**
seed()	Generate a random number based on the bracketed seed value. Will always return the same random number.	To generate a random number with 8 as the seed value with random.seed(8), it will always return 0.2267058593810488 as the same random number Example import random random.seed(8) print(random.random()) Results 0.2267058593810488

Frequently Used Functions of Random Module Provided by Python		
Function	**Description**	**Example**
randint()	Returns a random number between the given range, including the given integers.	To get a random integer value between 50 and 60: import random print(random.randint(50, 60)) Results 52 print(random.randint(50, 60)) Results 59
randrange()	Returns a random number between the given range, including the first integer but not the last.	To get a random integer value between 50 and 59: import random print(random.randrange(50, 60)) Results 52 print(random.randrange(50, 60)) Results 59
choice()	Returns a random element from the given list (see chapter 3)	To return 1 random element from the fruit list: import random fruit = ["orange","mango","banana","rambutan","durian", "pineapple","papaya"] print(random.choice(fruit)) Results pineapple

Frequently Used Functions of Random Module Provided by Python		
Function	**Description**	**Example**
sample()	Returns a given sample of a list (see chapter 3)	To return a sample of 3 random elements from the fruit list: import random fruit = ["orange","mango","banana","rambutan","durian", "pineapple","papaya"] print(random.sample(fruit,3)) Results ['mango', 'rambutan', 'orange']
random()	Returns a random float number between 0.00000000000000000 and 1.00000000000000000	Returns a random float number between 0.0000000000000000 and 1.0000000000000000 import random print(random.random()) Results 0.8236315525555947
uniform()	Returns a random float number between two given parameters	Returns a random float number between 0.0000000000000000 and 10.0000000000000000 import random print(random.uniform(1, 10)) Results 3.351523796647222

While using programming applications, various functions available in imported modules will be displayed, such as in the case of **import random**. When the function is called, it will be displayed as shown in Fiqure 7-1:

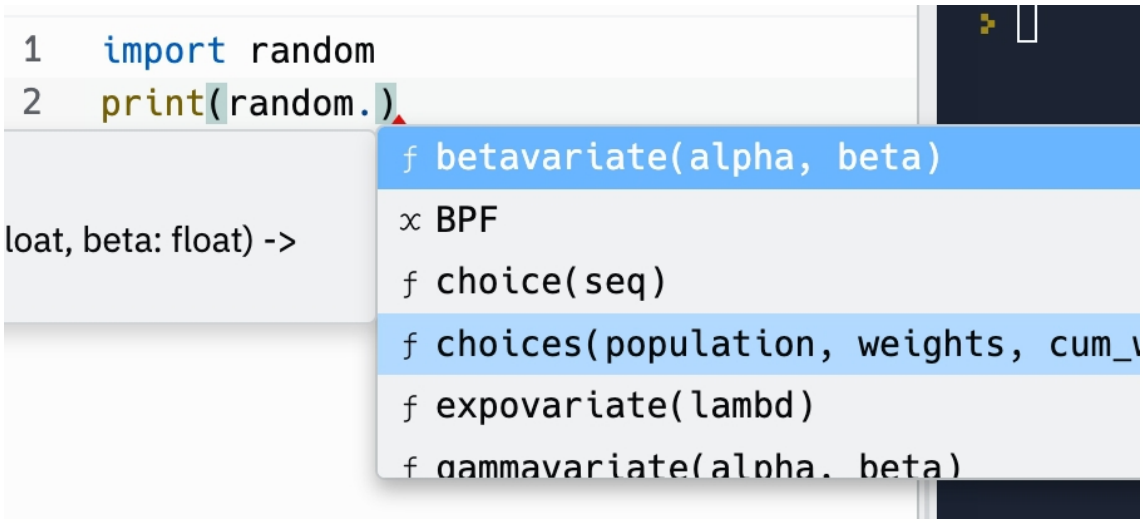


Figure 7-1: Visual Studio Code displaying all modules available of python for programmer

To use generate a random integer value between 1 and 10, navigate the menu to **randint()** as shown:

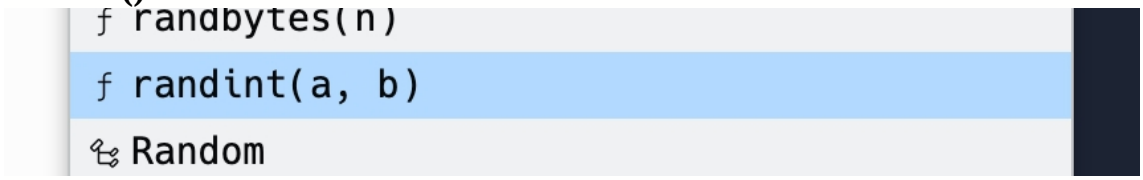


Figure 7-2: module randint() is used

When clicked, the function will appear in the program.

```
import random
print(random.randint(1,10))
```

Figure 7-3:random. randint() will appear in program

If you want to get a random float value, use **random.random()** instead.

using datetime module
The datetime Module
import datetime now = datetime.datetime.now() print(now)
Results
2024-04-27 19:00:18.380691

This Module pulls the current date and time to display in a format we define, provided by Python, when calling the strftime() or strptime() functions is as follows:

datetime module formatting when calling the strftime() or strptime() function.

Command Format enclosed in brackets strftime() or strptime() of datetime Module is shown in Table 7-2

Table 7-2: Command Format enclosed in brackets strftime() or strptime()

Command Format enclosed in brackets strftime() or strptime()	Displayed result	Example
%d	Date in 2 digits: '01'-'31'	import datetime now = datetime.datetime.now() print(now.strftime("%d")) Results 07
%m	Month in 2 digits: '01'-'12'	print(now.strftime("%m")) Results 02
%y	Year in 2 digits: For example, 2023 displays '23'.	print(now.strftime("%y")) Results 23
%Y	Year in 4 digits: For example, 2023 displays '2023'.	print(now.strftime("%Y")) Results 2023

%H	Time : hours in 12h, value ‘00’-‘12’	print(now.strftime("%H")) Results 07
%I	Time: hours in 24h, value '00'-'23'	print(now.strftime("%I")) Results 07
%M	Time: minutes, value '00'-'59'	print(now.strftime("%M")) Results 38
%S	Time: seconds, value '00'-'59'	print(now.strftime("%S")) Results 36
%p	AM or PM	print(now.strftime("%p")) Results PM
%f	Time: microseconds, ‘000000’-‘999999’	print(now.strftime("%f")) Results 784460
%A	Days: 'Monday'-'Sunday'	print(now.strftime("%A")) Results Tuesday
%a	Days: 'Mon'-'Sun'	print(now.strftime("%a")) Results Tue
%B	Months 'January'-'December'	print(now.strftime("%B")) Results February
%b	Months 'Jan'-'Dec'	print(now.strftime("%b")) Results Feb

%j	Order of days in 366 days 001-366	print(now.strftime("%j")) Results 038
%U	Order of weeks in 53 weeks 001-53	print(now.strftime("%U")) Results 06

For Thai dates and times, you must use the replace() command to convert Arabic numbers to Thai numbers. And convert full days, abbreviated days, full months, abbreviated months from English to Thai, as well as converting A.D. to B.E. by adding 543 years.
Download source code and consult at

https://www.variitsris.org/learning-python/

or scan QR Code

Exercises

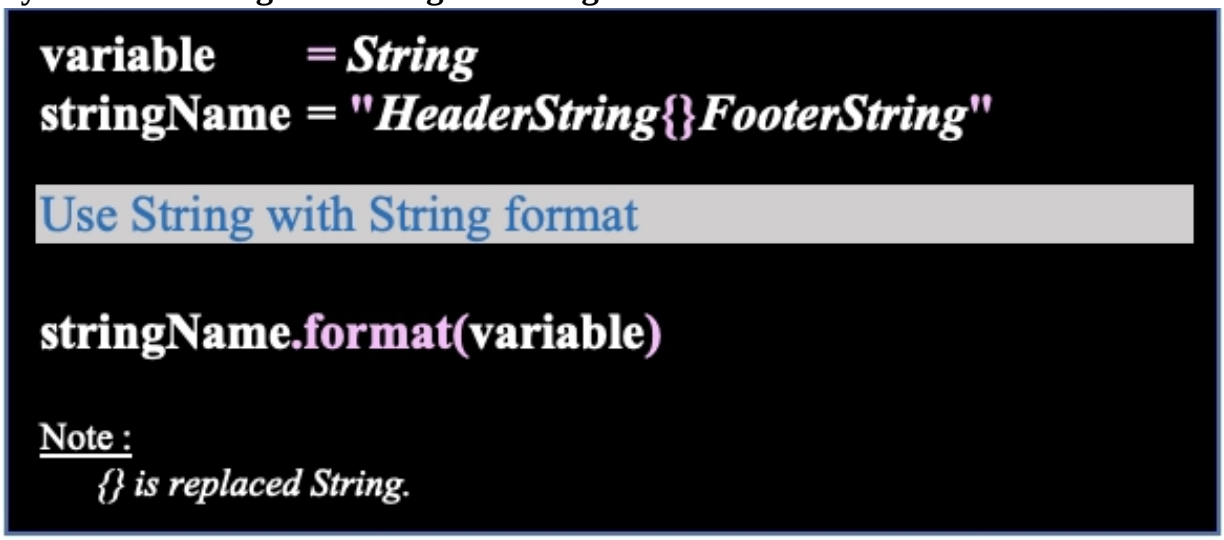
1. Create a module to input 5 variables and display their sum.
2. Use the module from step 1 in 3 different programs that lets you input 5 variables and display their sum.
3. From step 1, generate 5 random numbers and display their sum.

4. Write a program to display the current date and time in hours and minutes, in English and Thai.

CHAPTER 8 STRING FORMATTING

Python string formatting is a method that lets programmers make sure the string will display variables as expected, using the following syntax:

Syntax 8-1: String formatting for Strings



Example

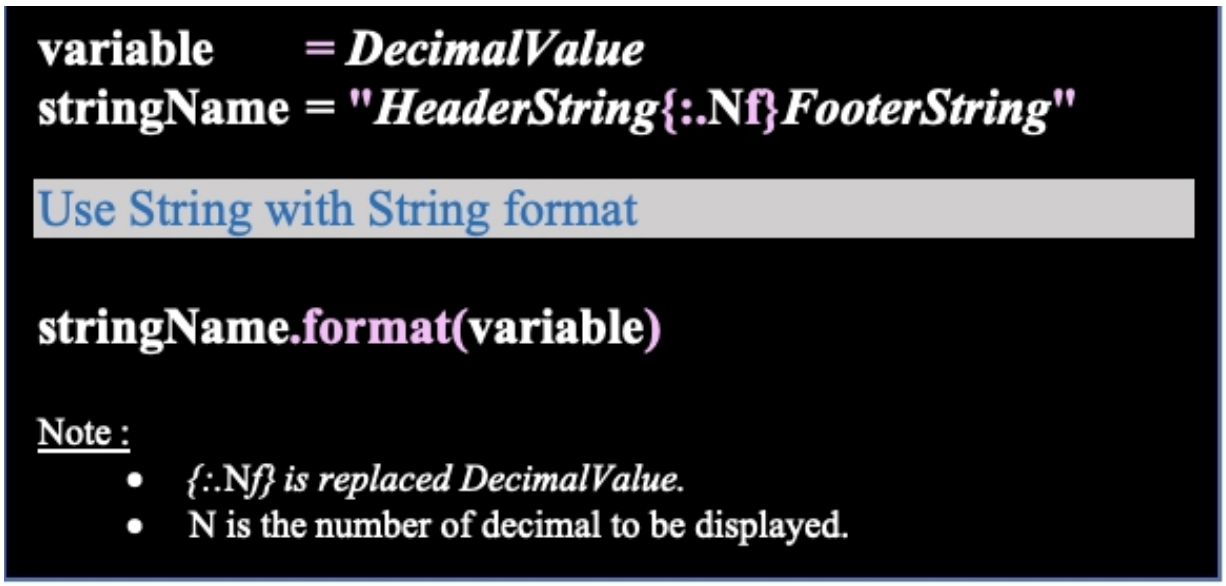
```
animal1 = 'lions'
zoo = "There are {} at The Beauty Zoo."
print(zoo.format(animal1))
```

Results

```
There are lions at The Beauty Zoo.
```

If the variable to be displayed in the string has the decimal value to be displayed. The programmer can specify using the following syntax:

Syntax 8-2: String format for decimal values



Example

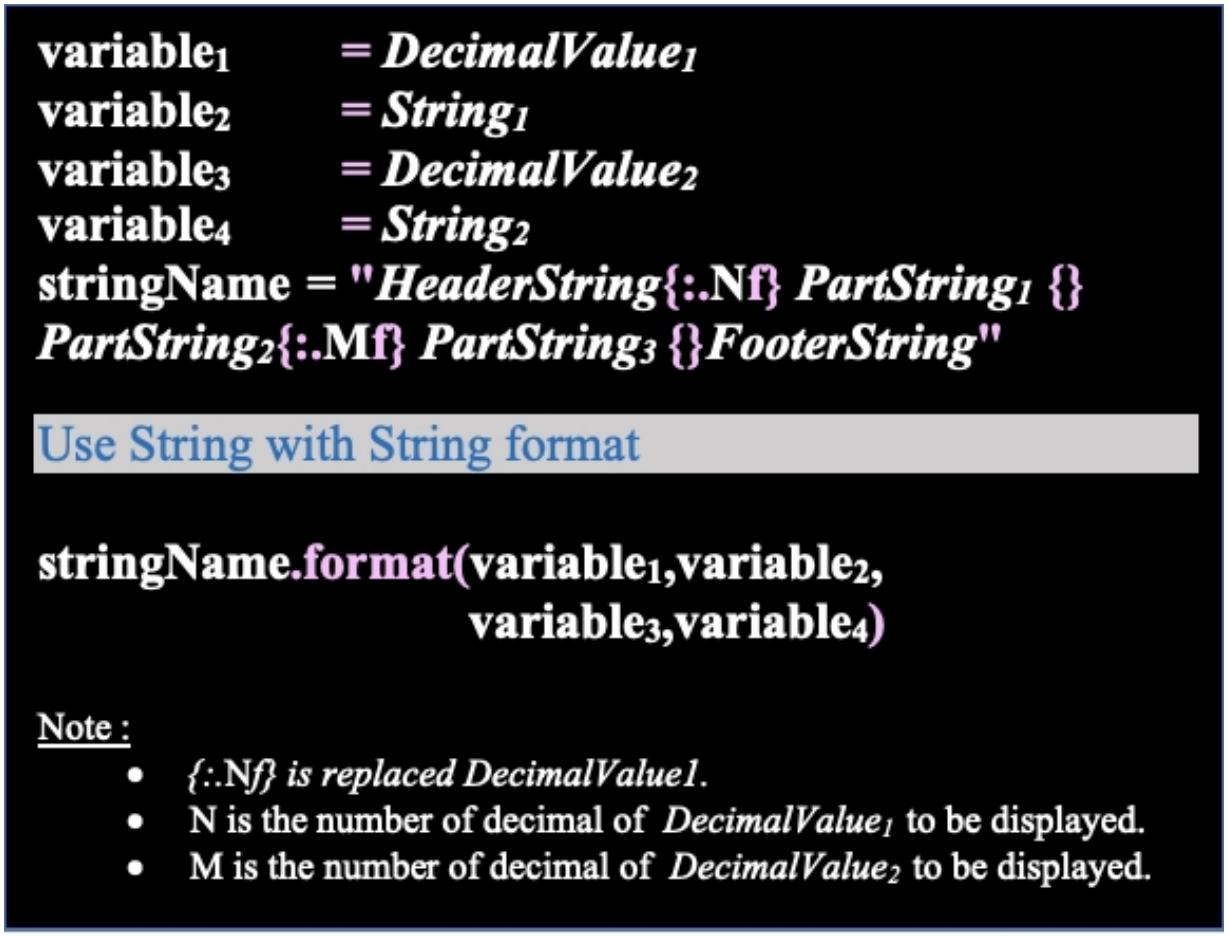
```
foodPrice1 = 199.50
zoo = "Beauty Zoo sells animal feed for {:.2f} THB per bag."
print(zoo.format(foodPrice1))
```

Results

```
Beauty Zoo sells animal feed for 199.50 THB per bag.
```

For multiple variables to display in a string, use the following syntax:

Syntax 8-3: format to display multiple values inside a string



Example

```
animal1 = 'lion'
foodPrice1 = 199.50
animal2 = 'fish'
foodPrice2 = 19.50
zoo = "Beauty Zoo sells {} feed for {:.2f} THB per bag and {} feed for
{:.2f} THB per bag."
print(zoo.format(animal1,foodPrice1,animal2,foodPrice2))
```

Results

```
Beauty Zoo sells lion feed for 199.50 THB per bag and fish feed for 19.50
THB per bag.
```

Example

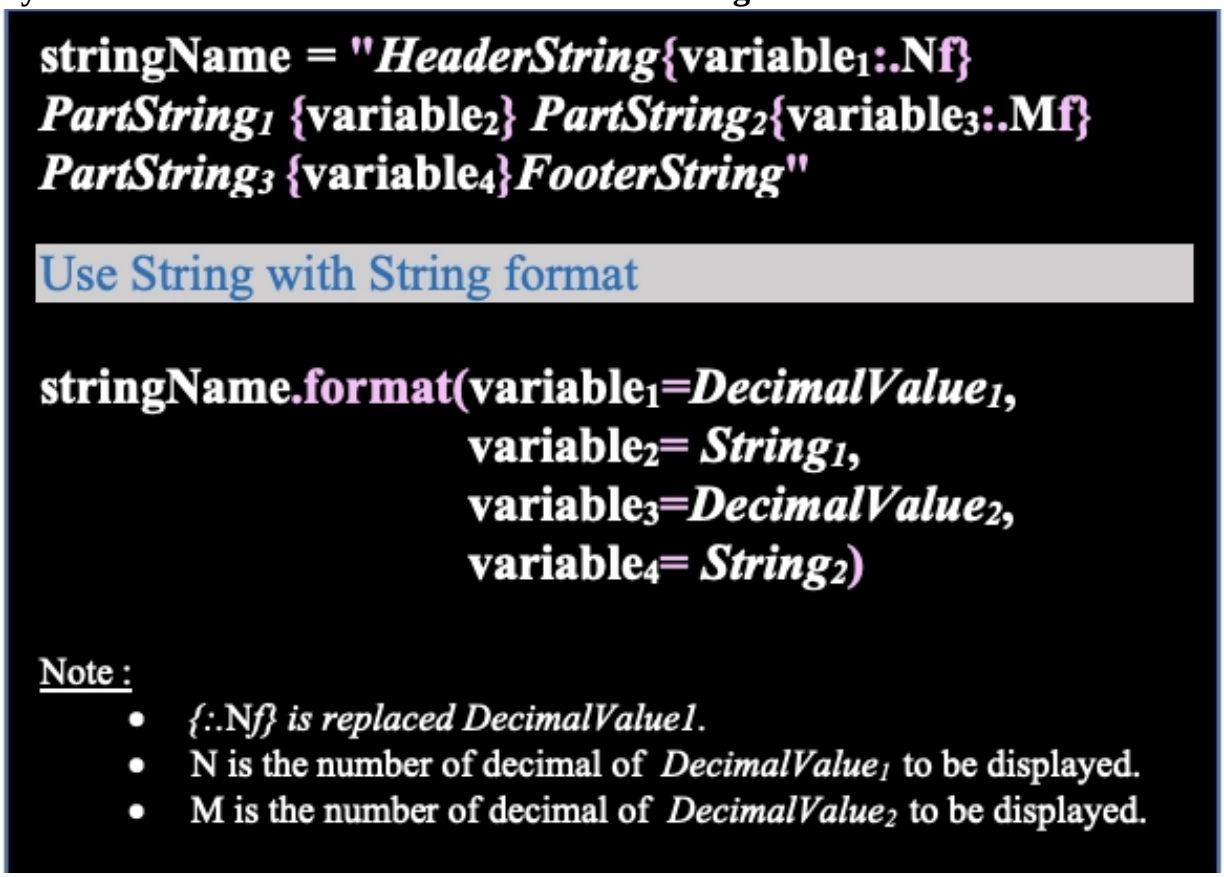
```
animal1 = 'lion'
foodPrice1 = 199.50
animal2 = 'fish'
foodPrice2 = 19.50
zoo = "Beauty Zoo sells {0} feed for {1:.2f} THB per bag and {2} feed for {3:.2f} THB per bag. The {0} feeding time is from 10am to 3pm, while the {2} feeding time is from 9am to 4pm."
print(zoo.format(animal1,foodPrice1,animal2,foodPrice2))
```

Results

Beauty Zoo sells lion feed for 199.50 THB per bag and fish feed for 19.50 THB per bag. The lion feeding time is from 10am to 3pm, while the fish feeding time is from 9am to 4pm.

Alternatively, you may use the following syntax:

Syntax 8-4: format with variables inside the string



Example

```
zoo = "Beauty Zoo sells {animal1} feed for {foodprice1:.2f} THB per bag and {animal2} feed for {foodprice2:.2f} THB per bag. The {animal1} feeding time is from 10am to 3pm, while the {animal2} feeding time is from 9am to 4pm."
print(zoo.format(animal1= 'lion',foodPrice1 = 199.50,animal2 = 'fish', foodPrice2= 19.50))
```

Results

Beauty Zoo sells lion feed for 199.50 THB per bag and fish feed for 19.50 THB per bag. The lion feeding time is from 10am to 3pm, while the fish feeding time is from 9am to 4pm.

Exercises

1. Write a program to display member names in sentences. “Welcome, Member name” greeting them every time the member comes to use the program.
2. Write a program to display product names with prices with decimals and the unit of product in the sentence “Product name price Product price baht per unit” and design a product trading system shown as a class diagram and write the program as designed.

CHAPTER 9 OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming, discovered in 1960, is a method of writing code to be consistent with the analysis and design of real-world object-oriented systems by creating abstract Classes and creating identifiable Objects within those classes. You can then change the state of the class according to the environment with a State Chart Diagram, create relationships between classes with a Class Diagram, show details of the relationship between classes or objects in Sequence Diagrams or Collaboration Diagrams and show work steps in an Activity Diagram to visualize how the whole system works before coding a program. System design analysts will make these diagrams to review the understanding of the system and use it as a summary before bringing it to the programmer to write a program, as well as use them to check the programmer's work before delivering it to the user. Since object-oriented programming is also compatible with C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP, using object-oriented programming in Python will be helpful for coders in making programs consistent with system design analysis using object-oriented principles. More importantly, object-oriented language programmers must be able to understand the diagrams mentioned above. For more information refer to the Advanced Python manual.

Create Class and Object

Class is the classification of things: types of animals, organizational positions, tools, equipment, machinery, furniture etc. When creating classes, create only ones we will use. For example, if we are going to create a system for a zoo, we must create animal-related classes, such as animals, pens, cages, feed, staff, etc. to be used in the system. As programmers, we

must understand the diagrams, and if we are acting as both programmers and system analysts, we need to create diagrams to easily communicate to users.By using Unified Modeling Language(UML) class diagrams, it is a standard that ensures every programmer understands how to code object-oriented programs according to the user's needs. Therefore, we should use language in diagrams that is easy to understand. Creating a class consists of three parts,as shown in Figure 9-1.

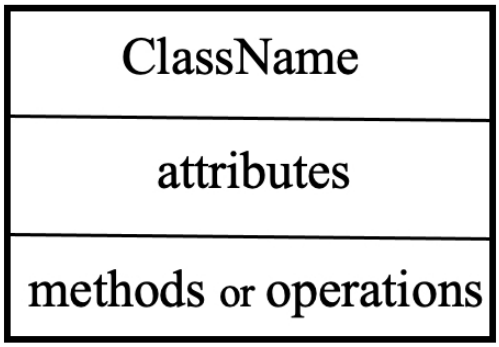


Figure 9-1: UML representation of a class

Class names must be consistent with names used in the real world for ease of understanding between programmers, colleagues in other roles and users involved with this program.

To create class in a program, use the following syntax:

Syntax 9-1: Creating a class in python

```
class ClassName :
   variable1 = value1
   variable2 = value2
   ...
   variableL = valueL
    attributes properties
   def __init__(do, attribute1, attribute2,..., attributeN)
      do.attribute1 = attribute1
      do.attribute2 = attribute2
      ...
      do.attributeN = attributeN
    display attributes properties
    def __str__(do):
      return f"HeaderString {do.attribute1}partString1
{do.attribute2} partString2...{do.attributeN}
FooterString"
```

```
    method properties
  def method1(parameter1.1,parameter1.2,
      ...,parameter1.P)
    command1.1
    command1.2
    ...
    command1.Q
  def method2(parameter2.1,parameter2.2,
      ...,parameter2.R)
    command2.1
    command2.2
    ...
    command2.S
    ...
  def methodV(parameterV.1,parameterV.2,
      ...,parameterV.T)
    commandV.1
    commandV.2
    ...
    commandV.U
```

Note:

L is the number of variables of Class
N is the number of attributes of Class some Class no attribute.
V is the number of methods of Class some Class no method.

Command of method ranging from 0 and have parameters or no parameter.

To create object in PYTHON program, use the following syntax:

Syntax 9-2: Creating an object in python

ObjectName = ClassName(value$_1$,value$_2$,…,value$_N$)

Note :

- N is the number of attributes of Class which has a value ranging from 0.
- values are passed in a sequence of attributes in Class when create Object from Class..

Here we will use an example to create an animal class,as shown in Figure 9-2.

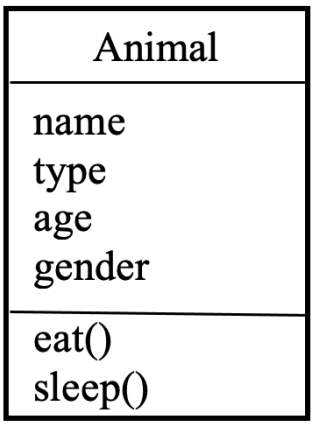


Figure 9-2: UML representation of the "animal" class

To display an object, use the following syntax:

Syntax 9-3: Display an object in python

display an Object

print(ObjectName)

use method of Object

ObjectName.method$_1$(parameter$_{1.1}$,parameter$_{1.2}$,
…,parameter$_{1.L}$)

ObjectName.method$_2$(parameter$_{2.1}$,parameter$_{2.2}$,
…,parameter$_{2.M}$)

…

ObjectName.method$_V$(parameter$_{V.1}$,parameter$_{V.2}$,
…,parameter$_{V.N}$)

Note:

- V is the number of methods of Class some Class no method.

Command of method ranging from 0 and have parameters or no parameter.

For example, to create the animal class:

```
class Animal:
 place = 'Beauty Zoo'
 def __init__(do,name,type, age, gender):
  do.name = name
  do.type = type
  do.age = age
  do.gender = gender
 def __str__(do):
  return f"A {do.age} year-old {do.gender} {do.type} named {do.name}."
 def eat(animal):
  print(animal.name+"is feeding.")
 def sleep(animal):
```

```
    print(animal.name+"is sleeping.")
A1 = Animal("Sudsuay","lion", "3", "female")
print('At '+A1.place+' are these animals:')
print(A1)
A1.eat()
A1.sleep()
```

When running the program, you will get
At Beauty Zoo are these animals:
A 3 year-old female lion named Sudsuay.
Sudsuay is feeding.
Sudsuay is sleeping.

parent and child

In the case of classes, there are also properties that help make object-oriented programming different from non-object-oriented programming, namely **inheritance**, which requires the creation of a **parent** or **generalization**. For this textbook, we will use the term **parent** class, which is inherited by a **child** class (or **specialization.**) This method combines a commonly used program into its parent to keep programs short and easily corrected in the parent class, so all its child classes can be called immediately. Therefore, this is very useful for when designing or drawing class diagrams, as shown in Figure 9-3:

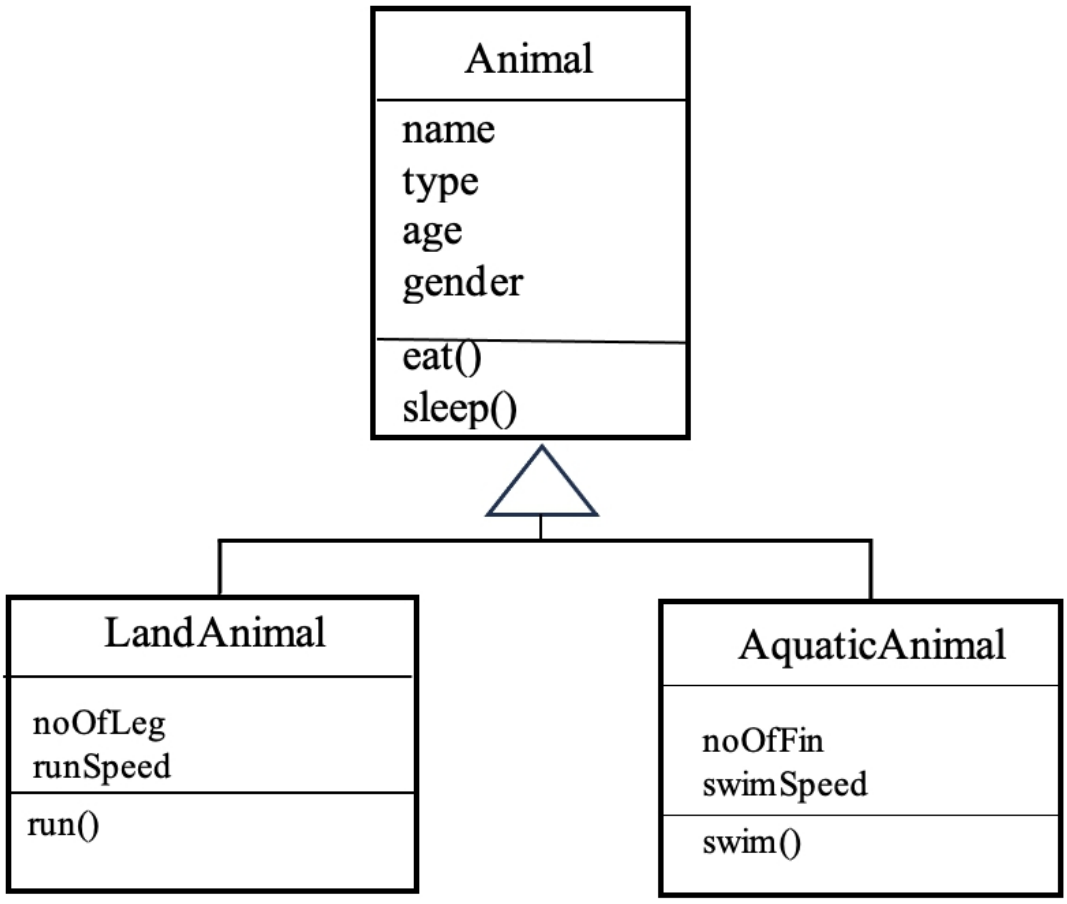


Figure 9-3:UML representation of an inheritance relationship.

This class diagram shows a parent and its children, where the parent is the Animal class. The base of the is divided into 2 child classes: LandAnimal and AquaticAnimal, which contain the same attributes in the Animal class, while different methods and attributes are that the child class, i.e. land animals having number of legs and top running speed, whereas aquatic animals have number of fins and top swimming speed.

To program Parent/Child classes in Python, Using only its parent's attributes and methods, without defining attributes and method for the child class, use the following syntax:

Syntax 9-4: Creating a child class in python

```
class ChildName(ParentName):
    pass
```

Create an and object of child class in python with the following syntax:

Syntax 9-5: Creating an object in a child class in python

```
ChildObjectName = ChildName(value1,value2,…,
                            valueN)
```

Note :

- N is the number of attributes of Parent Class which has a value ranging from 0.
- values are passed in a sequence of attributes in Parent Class when create Object from Child Class.

Example

```
class LandAnimal(Animal):
    pass
LA1 = LandAnimal("Sudsuay","lion", "3", "female")
print(LA1)
```

Results

```
A 3 year-old female lion named Sudsuay.
```

To add attributes or methods to a Child ,use the following syntax:

Syntax 9-6: Adding attributes or methods to a child class

```
class ChildName(ParentName):
   variable1 = value1
   variable2 = value2
   ...
   variableL = valueL
    attributes properties of Child
   def __init__(do, attribute1, attribute2,..., attributeM)
      do.attribute1 = attribute1
      do.attribute2 = attribute2
      ...
      do.attributeM = attributeM
    attributes properties of Parent
      ParentName._init_(do, attribute1.1, attribute1.2,...,
attribute1.N)
    display attributes properties of Child
   def __str__(do):
      return f"HeaderString {do.attribute1}partString1
{do.attribute2} partString2...{do.attributeM}
FooterString"
    method properties of Child
   def method1(parameter1.1,parameter1.2,
         ...,parameter1.P)
      command1.1
      command1.2
       ...
      command1.Q
```

```
def method2(parameter2.1,parameter2.2,
        ...,parameter2.R)
    command2.1
    command2.2
     ...
    command2.S
    ...

def methodV(parameterV.1,parameterV.2,
        ...,parameterV.T)
    commandV.1
    commandV.2
     ...
    commandV.U
```

Note :

- L is the number of variables of Child Class.
- M is the number of attributes of Child Class which has a value ranging from 0.
- N is the number of attributes of Parent Class which has a value ranging from 0.
- P is the number of parameters of Child Class $method_1$ ranging from 0.
- R is the number of parameters of Child Class $method_2$ ranging from 0.
- T is the number of parameters of Child Class $method_V$ ranging from 0.
- Q is the number of commands $method_1$ ranging from 0.
- S is the number of commands $method_2$ ranging from 0.
- U is the number of commands $method_V$ ranging from 0.

Example

```
class LandAnimal(Animal):
    def __init__(do,name,type, age, gender,noOfLeg,runSpeed):
        do.noOfLeg = noOfLeg
        do.runSpeed = runSpeed
```

```
      Animal.__init__(do,name,type, age, gender)
LA1 = LandAAnimal("Sudsuay","lion", "3", "female", "4", "74")
print(LA1)
   def intro(do):
      print('A land animal '+'with  '+LA1.noOfLeg+'  legs'+' and a top speed
of ' —--+LA1.runSpeed+' km/h')
LA1 = LandAnimal("Sudsuay","lion", "3", "female", "4", "74")
LA1.intro()
```

Results

A 3 year-old female lion named Sudsuay.
A land animal with 4 legs and a top speed of 74 km/h

Overrides

To **Override** an **Attribute** or **Method** of a parent class with one in a child class, you use an **Attribute** or **Method** with the same name as the parent class. For example:

```
class LandAnimal(Animal):
   def __init__(do,name,type, age, gender,noOfLeg,runSpeed):
     Animal.__init__(do,name,type, age, gender)
     do.noOfLeg = noOfLeg
     do.runSpeed = runSpeed
     do.type = 'land animal, the '+type
   def __str__(do):
     return f"A {do.age} year-old {do.gender} {do.type} named {do.name}
has —--{do.noOfLegs} legs and a top speed of {do.runSpeed} km/h.”
LA1 = LandAnimal("Sudsuay","lion", "3", "female", "4", "74")
print(LA1)
```

This overrides the attribute of the parent class: **type**.This also overrides the method of the parent class: def __str__(do)

Results

A 3 year-old female land animal, the lion named Sudsuay has 4 legs and a top speed of 74 km/h.

Polymorphism

Polymorphisms, meaning “many forms,” allows coders to program and manage data types and function using a **single interface.** For example, A Tourist's GPA can have multiple statuses, such as “normal” with a GPA of not less than 2.00, “Under probation” with a GPA under 2.00, or “dropped” with a GPA below 1.50. Thus, creating the **Tourist** class that changes according to their context and impacts execution. This in turn lets us **overload** methods, meaning create a method with the same name but behaves differently depending on the input data. An easy example of this in Python are methods that are not class-based for programmers, such as using the '+'.

```
int1 = 22
int2 = 145
float1 = 19.1
str1 = "hi
str2 = "Python"
print(int1 + int2)
print(type(int1 + int2))
print(float1 + float1)
print(type (float1 + float1))
print(str1 + str2)
print(type(str1 + str2))
```

Results

```
167
<class 'int'>
38.2
<class 'float'>
hiPython
<class 'str'>
```

or len()

```
str = 'Fruits starting with M'
fruit = ('Mango','Mangosteen','Melon','Mandarin')
fruit_list = ['Mango','Mangosteen','Melon','Mandarin']
fruit_dict = {'1':'Mango','2':'Mangosteen','3':'Melon','4':'Mandarin'}
print(len(str))
```

```
print(len(fruit))
print(len(fruit_list))
print(len(fruit_dict))
```

Results

```
21
4
4
4
```

Despite using the same len() method, Different values could be found depending on the type of data inputted.

For example, when animals of different ages receive different types of feed, such as lions over one years old be fed meat instead of milk, do as follows:

```
def eat(animal):
    if animal.age>=1:
        print(animal.name+ 'is over one years old and must be fed meat.')
    else :
      print(animal.name+ "is less than a year old and must be fed milk.')
LA1 = LandAnimal("Sandsuay","lion", 4, "wife", "4", "74")
LA1.eat()
```

Results

```
Sandsuay is over one years old and must be fed meat.
```

Example: polymorphism of multiple classes using a method with the same name.

```
class Cat:
 def __hot__(do, name, moveSpeed):
   do.name = name
   do.moveSpeed = moveSpeed
 def move(animal):
   print("The "+animal.name+" runs at a speed of "+animal.moveSpeed)
class Fish:
 def __hot__(do, name, moveSpeed):
   do.name = name
   do.moveSpeed = moveSpeed
 def move(animal):
```

```
    print("The "+animal.name+" swims at a speed of "+animal.moveSpeed)
class Bird:
  def __hot__(do, name, moveSpeed):
    do.name = name
    do.moveSpeed = moveSpeed
  def move(animal):
    print("The "+animal.name+" flies at a speed of"+animal.moveSpeed)
tiger1 = Cat("leopard", "60 kilometers per hour")
bird1 = Bird("parrot", "5 kilometers per hour")
fish1 = Fish("Goby", "20 kilometers per hour")
for x in (tiger1, bird1, fish1):
  x.move()
```

Results

```
The Leopard runs at a speed of 60 kilometers per hour.
The parrot flies at a speed of 5 kilometers per hour.
The goby swims at a speed of 20 kilometers per hour.
```

Inheriting Multiple Classes

Multi-class inheritance is when multiple parent classes are inherited to a child class, as shown in Figure 9-4:

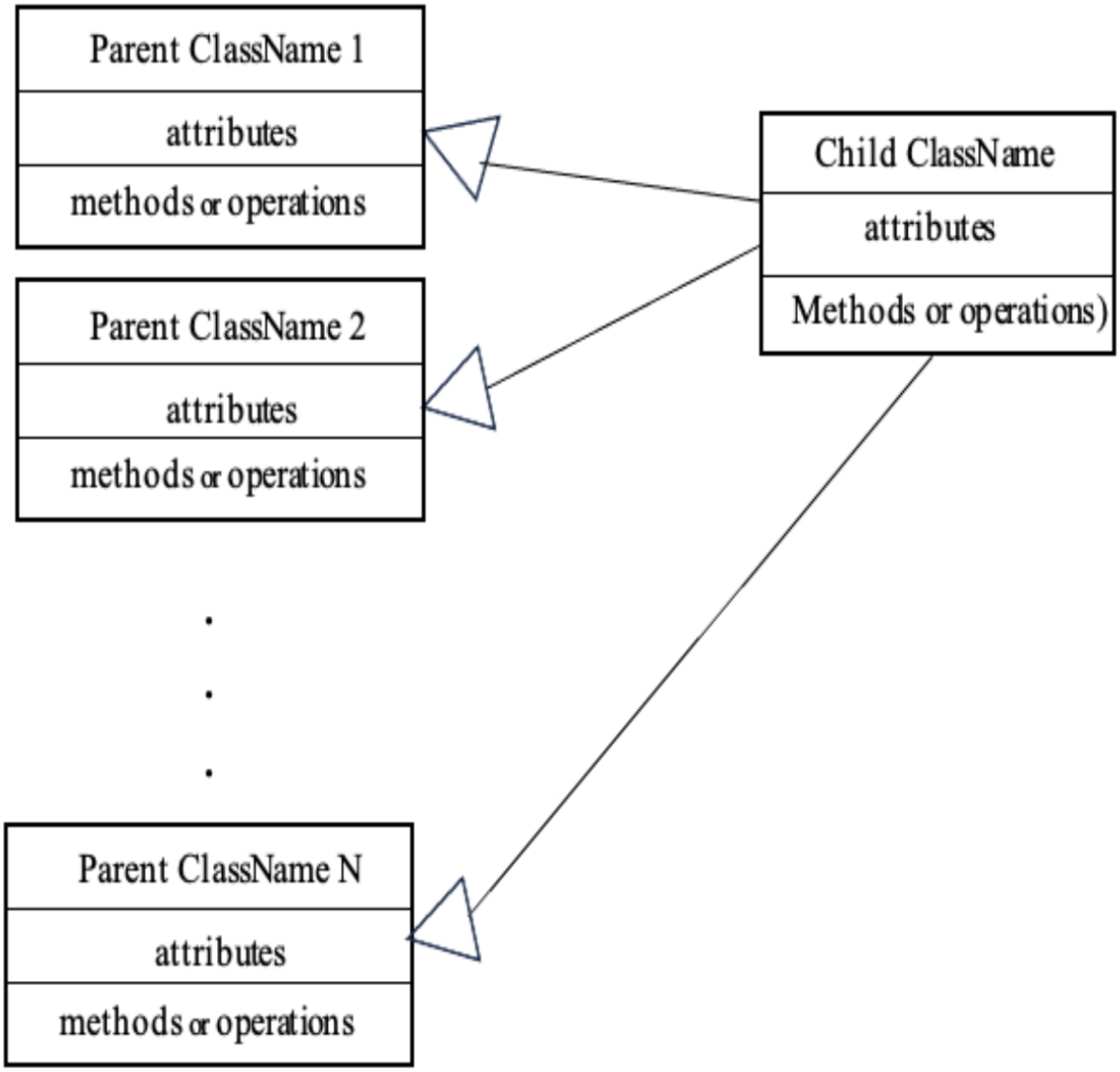


Figure 9-4: UML representation an inheritance relationship with multiple classes.

Example

Diagram of animals in the zoo to show how lions are land animals and mammals.

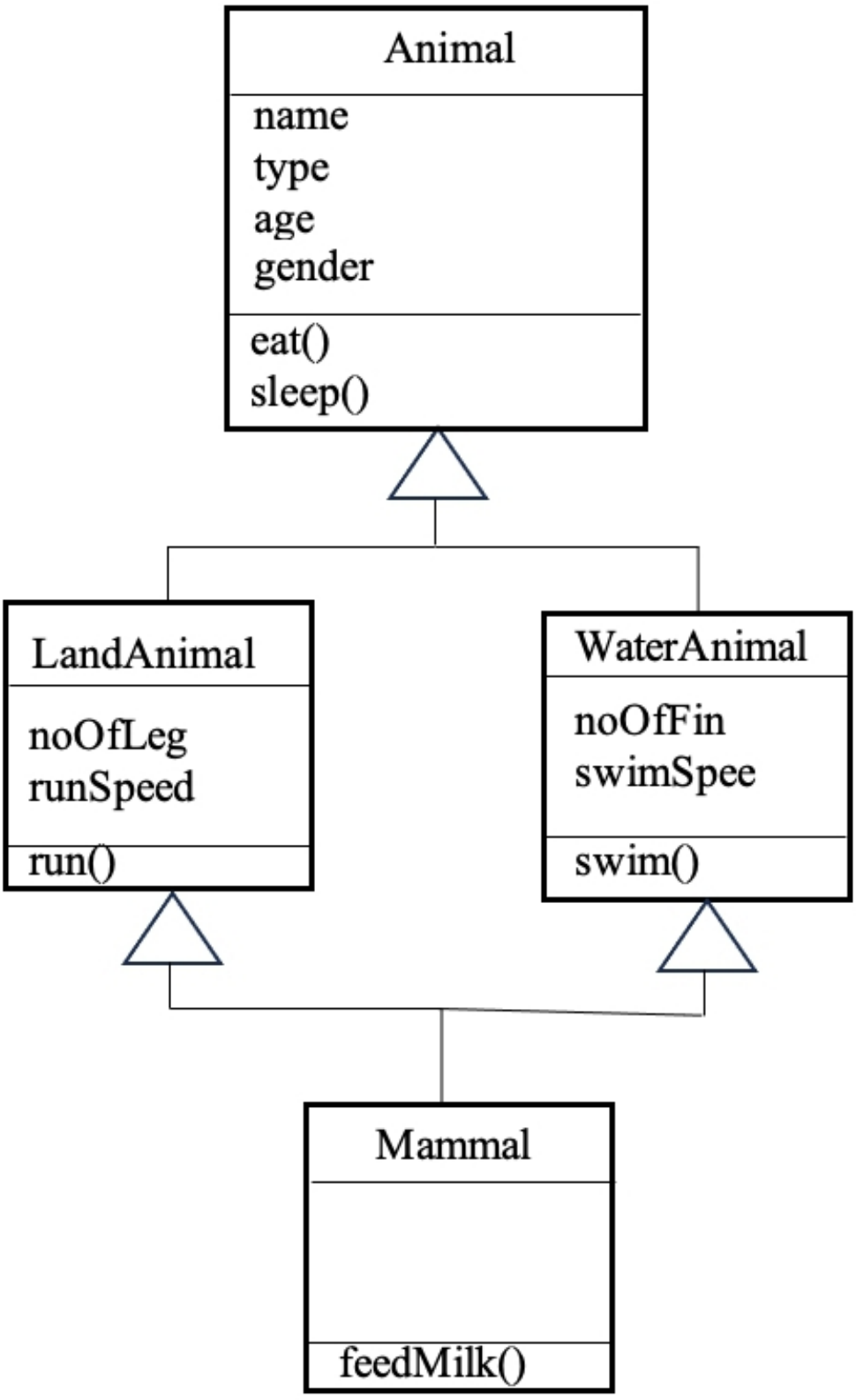

Animal
name
type
age
gender
eat()
sleep()
LandAnimal
noOfLeg
runSpeed
run()
WaterAnimal
noOfFin
swimSpee
swim()
Mammal
feedMilk()

Figure 9-5: Class diagram of animals in the zoo that show an inheritance relationship with multiple classes.

The syntax used to program Python child classes that have multiple parents.

To create a child class that have multiple parents, using the following syntax:

Syntax 9-7: Create a child class that have multiple parent classes in python

```
class ChildName(ParentName1,ParentName2, …,
                ParentNameP):
  variable1 = value1
  variable2 = value2
  ...
  variableQ= valueQ
   attributes properties of Child
  def __init__(do, attribute1, attribute2,..., attributeR)
    do.attribute1 = attribute1
    do.attribute2 = attribute2
    ...
    do.attributeR= attributeR
   attributes properties of Parent
    ParentName1._init_(do, attribute1.1, attribute1.2,...,
                       attribute1.L)
    ParentName2._init_(do, attribute2.1, attribute2.2,...,
                       attribute2.M)
    …
    ParentNameP._init_(do, attributeP.1,
                       attributeP.2,..., attributeP.N)
   display attributes properties of Child
  def __str__(do):
    return f"HeaderString {do.attribute1}partString1
{do.attribute2} partString2…{do.attributeR}
FooterString"
```

method properties of Child

```
def method1(parameter1.1,parameter1.2,
      ...,parameter1.T)
    command1.1
    command1.2
     ...
    command1.U
    ...

def method2(parameter2.1,parameter2.2,
      ...,parameter2.V)
    command2.1
    command2.2
     ...
    command2.W
    ...

def methodX(parameterX.1,parameterX.2,
      ...,parameterX.Y)
    commandX.1
    commandX.2
     ...
    commandX.Z
```

Note :

- Q is the number of variables of Child Class.
- R is the number of attributes of Child Class which has a value ranging from 0.
- L is the number of attributes of $ParentName_1$ Class which has a value ranging from 0.
- M is the number of attributes of $ParentName_2$ Class which has a value ranging from 0.
- N is the number of attributes of $ParentName_P$ Class which has a value ranging from 0.
- T is the number of parameters of Child Class $method_1$ ranging from 0.
- Vis the number of parameters of Child Class $method_2$ ranging from 0.
- Yis the number of parameters of Child Class $method_V$ ranging from 0.
- U is the number of commands $method_1$ ranging from 0.
- W is the number of commands $method_2$ ranging from 0.
- X is the number of commands $method_X$ ranging from 0.

Example program:

```
class Animal:
  place = 'Beauty Zoo'
  def __init__(do,name,type, age, gender):
    do.name = name
    do.type = type
    do.age = age
    do.gender = gender
  def __str__(do):
    return f"A {do.age} year-old {do.gender} {do.type} named {do.name}"
  def eat(animal):
     if animal.age>=1:
         print(animal.name+ ' is over one years old and must be fed meat.')
     else :
       print(animal.name+ ' is less than a year old and must be fed milk.')
  def sleep(animal):
    print(animal.name+" is sleeping.")
A1 = Animal("Rashun","lion", 3, "female")
print('At '+A1.place+' are these animals:')
print(A1)
A1.eat()
A1.sleep()
class LandAnimal(Animal):
    def __init__(do,name,type, age, gender,noOfLeg,runSpeed):
      Animal.__init__(do,name,type, age, gender)
      do.noOfLeg = noOfLeg
      do.runSpeed = runSpeed
      do.type = type+', a land animal'
      do.legFin = 'legs,'
      do.moveMethod = 'and runs with a top speed of'
    def __str__(do):
      return f"A {do.age} year-old {do.gender} {do.type} named {do.name}
has {do.noOfLeg} {do.legFin} {do.moveMethod} {do.runSpeed} km/h. "
    def run(landAnimal):
```

```
    print(landAnimal.name+" is running at the top speed of "
+landAnimal.runSpeed+' km/h.')
LA1 = LandAnimal("Sudsuay","lion", 4, "female", "4", "74")
LA1.run()
print(LA1)
class AquaticAnimal(Animal):
   def __init__(do,name,type, age, gender,noOfFin,swimSpeed):
     Animal.__init__(do,name,type, age, gender)
     do.noOfFin = noOfFin
     do.swimSpeed = swimSpeed
     do.type = type+', an aquatic animal'
     do.legFin = 'fins,'
     do.moveMethod = 'and swims with a top speed of'
   def __str__(do):
     return f"A {do.age} year-old {do.gender} {do.type} named {do.name}
has {do.noOfFin} {do.legFin} {do.moveMethod} {do.swimSpeed} km/h. "
   def swim(AquaticAnimal):
     print(AquaticAnimal.name+" is swimming at a top speed of "
+AquaticAnimal.swimSpeed+' km/h')
AA1 = AquaticAnimal("Bee","goby", 2, "male", "5", "23")
print(AA1)
AA1.swim()
class Mammal(AquaticAnimal,LandAnimal):
   def __init__(do,name,type, age,
gender,noOfFin,swimSpeed,noOfLeg,runSpeed):
     AquaticAnimal.__init__(do,name,type, age,
gender,noOfFin,swimSpeed)
     LandAnimal.__init__(do,name,type, age, gender,noOfLeg,runSpeed)
     do.type = type+', a mammal'
   def __str__(do):
     return f"A {do.age} year-old {do.gender} {do.type} named
{do.name}."
   def feedMilk(mammal):
      print(mammal.name+" is breastfeeding.")
M1 = Mammal("BB","Dolphin", 2, "female", "5", "0", "0", "0")
print(M1)
```

```
M1.feedMilk()
```

Results

```
At Beauty Zoo are these animalsL
A 3 year-old female lion named Rashun.
Rashun is over one years old and must be fed meat.
Rashun is sleeping.
Sudsuay is running at the top speed of 74 km/h.
A 4 year-old female lion, a land animal named Sudsuay has 4 legs, and runs with a top speed of 74 km/h.
A 2 year-old male goby, an aquatic animal named Bee has 5 fins, and swims with a top speed of 23 km/h.
Bee is swimming at a top speed of 23 km/h.
A 2 year-old female Dolphin, a mammal named BB.
BB is breastfeeding.
```

Association

Association is when a class is related to another class, such as the driver class being associated with a car class. To create associations, use the following Diagram:

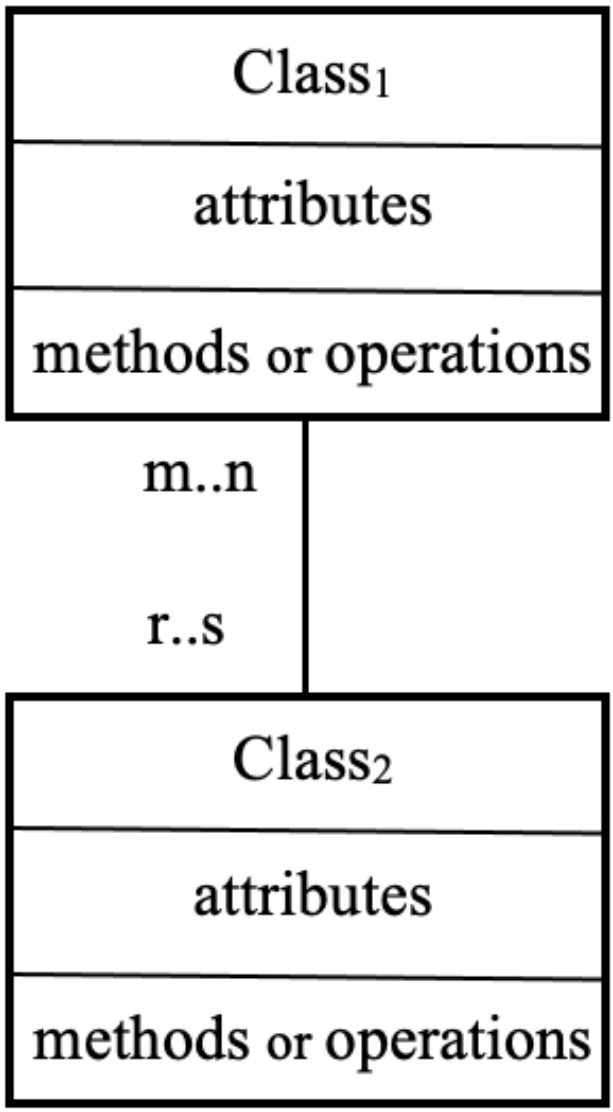


Figure 9-6: UML representation of an association relationship.

To create an association between $class_1$ and $class_2$ with method in python, using following syntax:

Syntax 9-8: Create an association between $class_1$ and $class_2$ with method in python

```
Class1.method(Class2.attribute)
```

Aggregation

Aggregation is when a class has other classes join in, and will remain even if the joined classes no longer exist. For example, the "Company" class has aggregated "department" classes inside. Diagram as shown:

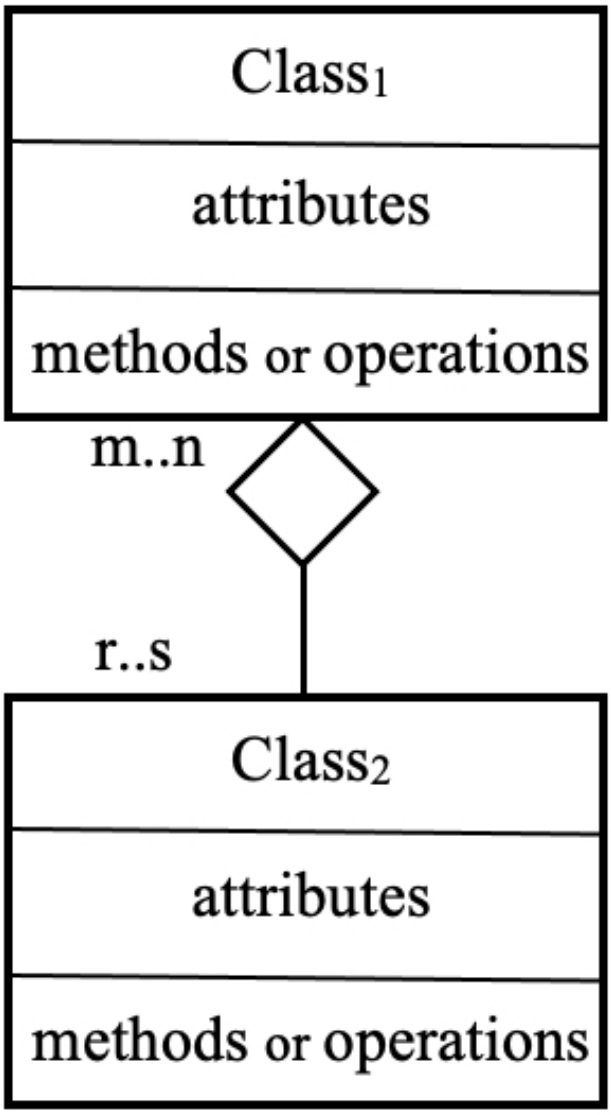


Figure 9-7: UML representation an aggregation relationship.

Note: m is the minimum and n is the maximum number of attributes in class 1, which is the aggregate class, within class 2. (This could be more than class, depending on the aggregate class.) r is the minimum and s is the maximum number of attributes in class 2, which is the subclass, within class 1.

Composition

Compositions are classes composed by the classes, such as the 'Cars' class is composed of wheels, doors, roofs, seats, etc., i.e. its components. diagram as shown:

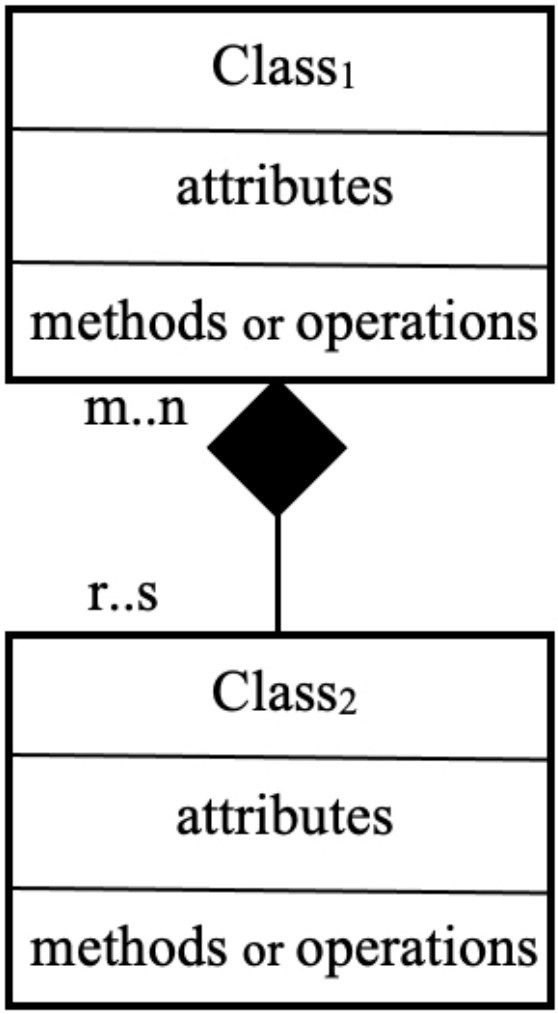


Figure 9-8: UML representation a composition relationship.

Examples

1. **Association:** **Animals** have **habitats** of various **sizes** and in different **locations** within the zoo. Each animal can only belong in 1 enclosure.
2. **Aggregation**: in the **zoo** class, there are **habitats**, one **ticket office**, one **staff office**, and one **Gift Shop**.

3. **Composition:** A Land mammal is composed of body parts, such as a **head**, a **body** and **legs**

Example Diagram of a class Association

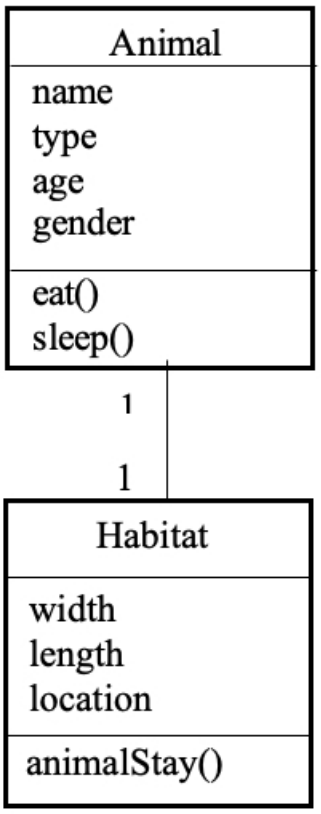


Figure 9-9: Class diagram of animals in the zoo with an association relationship.

Example Program

```
class Animal:
  place = 'Beauty Zoo'
  def __init__(do,name,type, age, gender):
    do.name = name
    do.type = type
    do.age = age
    do.gender = gender
  def __str__(do):
```

```
   return f"A {do.age} year-old {do.gender} {do.type} named {do.name}."
 def eat(animal):
    if animal.age>=1:
       print(animal.name+ ' is over one years old and must be fed meat.')
    else :
      print(animal.name+ ' is less than a year old and must be fed milk.')
 def sleep(animal):
   print(animal.name+" is sleeping.")
class Habitat():
   def __init__(do, location, width, length):
      do.location = location
      do.width = width
      do.length = length
   def __str__(do):
    return f"{'Located '+do.location+', the enclosure is '+str(do.width)+'
meters wide and '+str(do.length)+' meters long.'}"
   def animalStay(do, animal):
      print(animal+' is located '+do.location+'. The enclosure has an area of
'+str(do.width * do.length)+' meters squared.')
Habitat1 = Habitat("across the pool", 2, 6)
print(Habitat1)
A1 = Animal("Rashun","Lion", 3, "Female")
print(A1)
Habitat1.animalStay(A1.name)
```

Results

```
Located across the pool, the enclosure is 2 meters wide and 6 meters long.
A 3 year-old Female Lion named Rashun.
Rashun is located across the pool. The enclosure has an area of 12 meter
squared.
```

Example

diagram of a class Aggregation

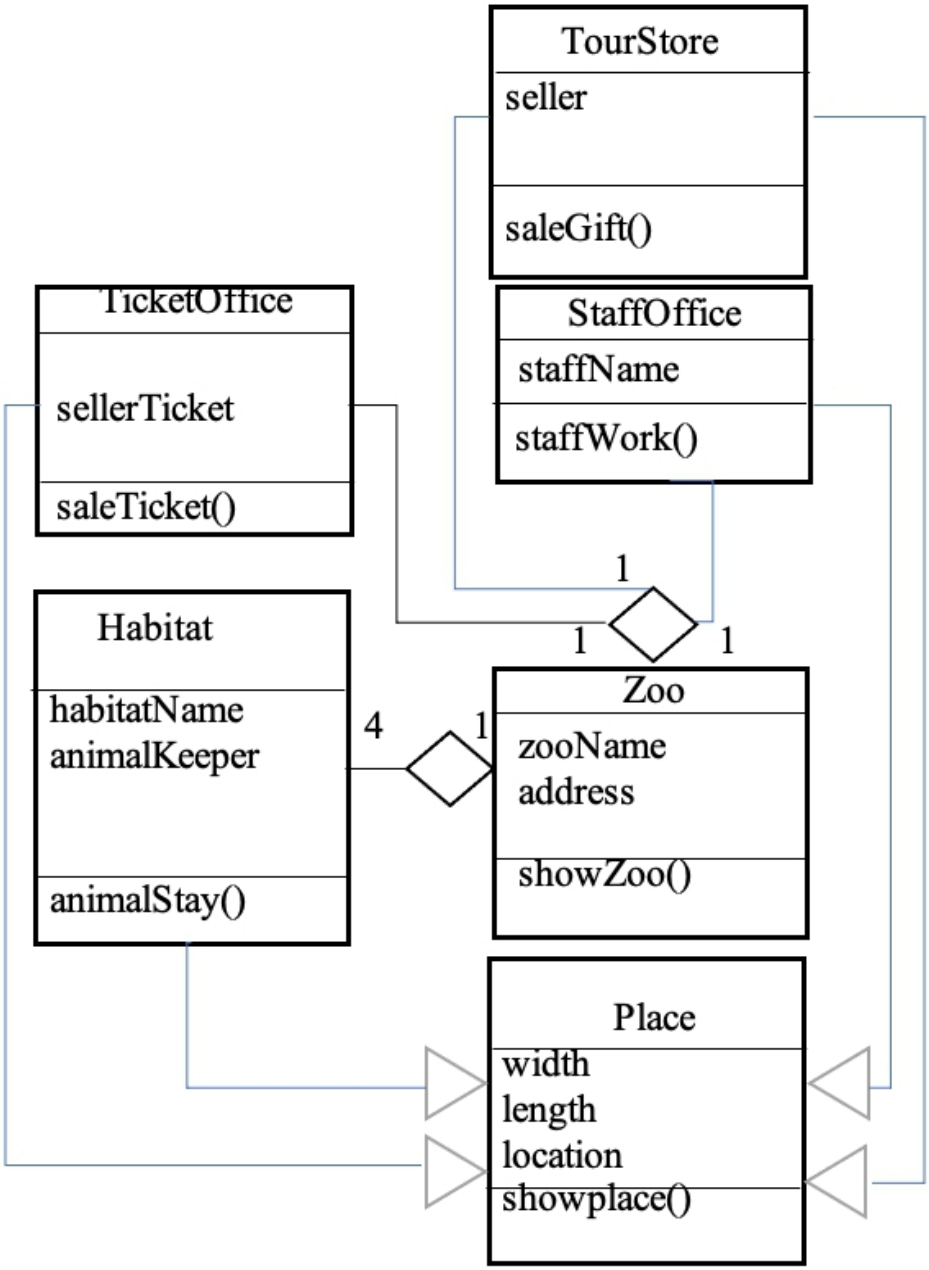


Figure 9-10: Class diagram of animals in the zoo with an aggregation relationship.

Example program

```
class Habitat():
    def __init__(do, location, width, length):
        do.location = location
        do.width = width
        do.length = length
    def __str__(do):
     return f"{'Located '+do.location+', the enclosure is '+str(do.width)+'
meters wide and '+str(do.length)+' meters long.'}"
    def animalStay(do, animal):
        print(animal+' is located '+do.location+'. The enclosure has an area of
'+str(do.width * do.length)+' meters squared.')
class TicketOffice():
    def __init__(do,location, width, length, sellerTicket):
        do.location = location
        do.width = width
        do.length = length
        do.sellerTicket = sellerTicket
    def __str__(do):
     return f"{'Ticket Office, staffed by '+do.sellerTicket}"
class StaffOffice():
    def __init__(do,location, width, length, staffName):
        do.location = location
        do.width = width
        do.length = length
        do.staffName = staffName
    def __str__(do):
     return f"{'Staffed by'+do.staffName}"
class TourStore():
    def __init__(do,location, width, length, seller):
        do.location = location
        do.width = width
        do.length = length
        do.seller = seller
    def __str__(do):
     return f"{'Staffed by'+do.seller}"
class Zoo(object):
```

```
    def __init__(do, name, address,
Habitat1,Habitat2,Habitat3,Habitat4,TicketOffice,StaffOffice,TourStore):
        do.name = name
        do.address = address
        do.Habitat1 =  Habitat1
        do.Habitat2 =  Habitat2
        do.Habitat3 =  Habitat3
        do.Habitat4 =  Habitat4
        do.TicketOffice=TicketOffice
        do.StaffOffice=StaffOffice
        do.TourStore=TourStore
    def __str__(do):
     return f"{do.name+' is located at '+do.address+'. Habitat 1 is located
'+do.Habitat1.location+'. Habitat 2 is located '+do.Habitat2.location+'.
Habitat 3 is located '+do.Habitat3.location+'. The Aquarium is located '
+do.Habitat4.location+'. The ticket office is staffed by '
+do.TicketOffice.sellerTicket+'. The staff office is staffed by '
+do.StaffOffice.staffName+'. The gift shop is staffed by '
+do.TourStore.seller}."
H1 = Habitat("across the pool", 2, 6)
H2 = Habitat("right of the entrance", 5, 6)
H3 = Habitat("right of the entrance", 5, 6)
H4 = Habitat("at the end of the pool", 2, 3)
print(H1)
T1 = TicketOffice("at the entrance", 2, 6,"Smiley Mcwelcome")
S1 = StaffOffice("end of zoo ", 2, 6,"Manny Jerial")
TS1 = TourStore("near exit door ", 3, 4,"Kassho Credit")
Z1 = Zoo("Beauty Zoo", "999, Bangkok,
Thailand",H1,H2,H3,H4,T1,S1,TS1)
print(Z1)
```

Results

Beauty Zoo is located at 999, Bangkok, Thailand. Habitat 1 is located across the pool. Habitat 2 is located right of the entrance. Habitat 3 is located right of the entrance. The Aquarium is located at the end of the

pool. The ticket office is staffed by Smiley Mcwelcome. The staff office is staffed by Manny Jerial. The gift shop is staffed by Kassho Credit.

Example diagram of a composition

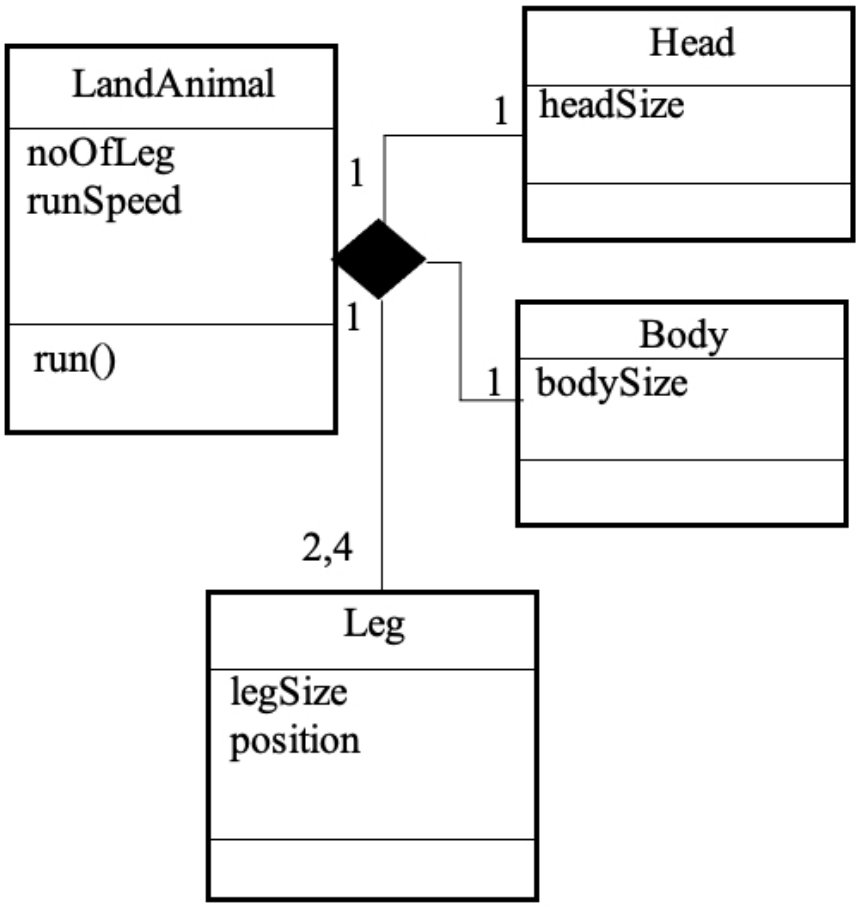


Figure 9-11: Class diagram of animals in the zoo with a composition relationship.

Example program

```
class Head:
   def __init__(do, headSize):
      do.headSize = headSize
   def __str__(do):
    return f"{str(do.headSize)}"
class Body:
   def __init__(do, bodySize):
      do.bodySize = bodySize
   def __str__(do):
```

```
    return f"{str(do.bodySize)}"
class Leg:
    def __init__(do, legSize, position):
        do.legSize = legSize
        do.position = position
    def __str__(do):
     return f"{do.position+' '+str(do.legSize)+' cm. '}"
class LandAnimal:
    def
__init__(do,headSize,bodySize,leg1Size,position1,leg2Size,position2,leg3S
ize,position3,leg4Size,position4):
        do.headSize = Head(headSize)
        do.bodySize = Body(bodySize)
        do.leg1Size = Leg(leg1Size,position1)
        do.leg2Size = Leg(leg2Size,position2)
        do.leg3Size = Leg(leg3Size,position3)
        do.leg4Size = Leg(leg4Size,position4)
    def __str__(do):
     return f"{'Head Length: '+str(do.headSize)+' cm. '+'Body Length: '
+str(do.bodySize)+' cm. '
+str(do.leg1Size)+str(do.leg2Size)+str(do.leg3Size)+str(do.leg4Size)}"
A1 = LandAnimal(80,120,60,'Front Left:',60,'Front Right:',60,'Back
Left:',60,'Back Right:')
print(A1)
print('The head is '+str(Head(A1.headSize))+' cm long.')
print('The body is '+str(Body(A1.bodySize))+' cm long.')
A2 = LandAnimal(5,8,3,'Front Left:',3,'Front Right:',0,'',0,'')
print(A2)
print('The head is '+str(Head(A2.headSize))+' cm long.')
print('The body is '+str(Body(A2.bodySize))+' cm long.')
```

Results

```
Head Length: 80 cm. Body Length: 120 cm. Front Left: 60 cm. Front Right:
60 cm. Back Left: 60 cm. Back Right: 60 cm.
The head is 80 cm long.
The body is 120 cm long.
```

Head Length: 5 cm. Body Length: 8 cm. Front Left: 3 cm. Front Right: 3 cm. 0 cm. 0 cm.
The head is 5 cm long.
The body is 8 cm long.

Exercises

1. From exercise 1 in Chapter 8, add an association as part of its design.
2. From exercise 1 in Chapter 8, add an aggregation as part of its design.
3. From exercise 1 in Chapter 8, add a composition as part of its design.

CHAPTER 10 TRY… EXCEPTION

When a program is being used, there's always a chance of malfunction, either from users, the hardware, network, equipment or power. This affects the operation of the program, so errors must be pre-written as follows:

try: A set of instructions used to find errors, and if found, to perform any action.

except : A set of instructions that, if an error is encountered, make an exception and do nothing.

else : A command that executes when no errors are found.

finally : Command when the error check has been completed.

Example

```
while True:
    try:
        x = int(input("Please enter a number: "))
    except ValueError:
        print("Sorry, that is not a number. Please enter again...")
```

```
  else :
    print("We've got the numbers.")
  finally :
    print("Thank you very much.")
```

Result in the case where numbers are not entered

Please enter numbers: !

Sorry, that is not a number. Please enter again...

Thank you very much

Result in case of entering numbers

Please enter number: 1

We've got the numbers.

Thank you very much.

Note: The ‘finally’ command is executed every time.

Exercises

1. From exercise 1 in Chapter 8, create an **exception** for 3 user errors.

CHAPTER 11 FILE MANAGEMENT WITH...AS AND JSON

File management includes

1. Open
2. Create
3. Edit/Write
4. Delete

Open File

To open a file, use the **open()** function with two parameters: the name of the file you want to open and the mode, as follows:

Mode 1: "r"– **reads** the specified value in the file instead of opening it. If that file doesn't exist, it will show an error.

Example
Create a txt file.

Type 'Test file command' and save the file name as 'test.txt' in the same folder where the program is stored.

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

Results

```
Test file command
```

If the file is not found

```
f = open("test2.txt")
print(f.read())
```

Results

```
FileNotFoundError: [Errno 2] No such file or directory: 'test2.txt'
```

Create

Mode 2: "a"-**Appends** (creates) a file.

Example

Create test2.txt file.

```
f = open("test2.txt","a")
```

A new file, test2.txt, is created.

If this file already exists, no error will be displayed.

Write File

Mode 3: "w"-**Writes** into a file.

Example

Write the file Sawasdee.txt

```
Sawasdee= open("Sawasdee.txt", "w")
Sawasdee.write("Hi, this is Thailand.\n")
Sawasdee.write("A country with delicious food,\n")
Sawasdee.write(“everyone has a smile.\n")
Sawasdee.write("A beautiful, safe place.\n")
```

with...as

with...as is a command that is similar to defining a variable with a different way to code, but yields the same results.

Example 1

In case the file you want to open is in the folder '/ExamplePythonCode/FileForPythonTest/’

```
f = open("/ExamplePythonCode/FileForPythonTest/Open.txt", "r")
```

```
print(f.read())
```

The command used to open the open.txt file is long since it is located in a very deep folder. Therefore, to make the programmer's work easier, it can be done by specifying it as a variable, f, which the programmer can write using with...as.

Example 2

```
with open("/ExamplePythonCode/FileForPythonTest/Open.txt", "r") as The dead one:
    print(The dead one.read())
```

Both examples have the same effect, meaning the files can be opened in the same way.

If the programmer wants to use a command to read each line of a file, write the command as follows:

```
with open("Sawasdee2.txt") as Sawasdee3:
    for line in Sawasdee3:
        print(line)
```

Or it could be written like this:

```
Sawasdee= open("Sawasdee.txt")
with  Sawasdee:
    for line in Sawasdee:
        print(line)
```

Results

```
Hi, this is Thailand.
A country with delicious food,
everyone has a smile.
A beautiful, safe place.
```

Create File

Mode 4: "x"-Creating a file, but if an **extra** file with the same name already exists, an error will be displayed.

Example

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

Results
have file


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

Results

```
FileExistsError: [Errno 17] File exists: 'Sawasdee.txt'
```

If not there will be a file created.

Delete File

If you want to delete a file, **import os** and use the **remove** command as in the example.

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

If the file to delete is not found, the following will be displayed:

```
FileNotFoundError: [Errno 2] No such file or directory: 'S.txt'
```

```
import os
os.remove("Hello3.txt")
```

The file named Hello3.txt will be deleted.
To create an exception, use the following:

```
import os
if os.path.exists("Hello3.txt"):
  os.remove("Hello3.txt")
else:
  print("File hello3.txt not found")
```

Results

```
The file Hello3.txt was not found.
```

Delete Folder

if we want to delete a folder, use the following:

```
import os
```

```
os.is rm("myfolder")
```

Results

delete folder myfolder

If this folder is not found

Results

```
FileNotFoundError: [Errno 2] No such file or directory: 'myfolder'
```

If the folder is found, it will be deleted.

JSON

JSON, short for JavaScript Object Notation, is a standard format for representing structured data based on the JavaScript Object syntax, typically used for sending data in web applications. (e.g. sending some data from a server to a client so that it can be displayed on a web page or vice versa.) Python itself has commands that can be used with JSON files, but you must import JSON first, where specific data type in Python can be converted to JSON as shown in the table.

Table 11-1:Python to JSON conversion table

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

loads

Using the **loads** command to convert the **person** value in Python to json, namely **person2**, then **print** to display the **person2** values, and to display only the **Marriage** value of **person2['Marriage']**

Example**:**

```
import json
```

```
person = '{"name": "Somsak", "Marriage":
true,"age":87,"Country":"Thai"}'
person2 = json.loads(person)
print( person2)
print(person2['Marriage'])
```

Results

```
{'name': 'Somsak', 'Marriage': True, 'age': 87, 'Country': 'Thai'}
True
```

You can also use commands to use the values in the json file, for example:

fileperson.json

```
{"name": "Somsak", "Marriage": true,"age":87,"Country":"Thai"
}
```

open

Open a json file and read the value, then create a variable that loads the value, named **data**.

Example**:**

```
with open('person.json', 'r') as f:
  data = json.load(f)
print(data)
```

Results

```
{'name': 'Somsak', 'Marriage': True, 'age': 87, 'Country': 'Thai'}
```

convert Python to JSON

You can convert values from Python to JSON using the **dumps** command and improve readability using the **indent**, **separators** and **sort_keys** commands as in the example.

```
import json
print(json.dumps({"name": "Nest", "age": 30}))
print(json.dumps(["orange", "pineapple"]))
print(json.dumps(("orange", "pineapple")))
print(json.dumps("orange"))
print(json.dumps(241))
print(json.dumps(50.23))
```

```
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
person = {"name": "Somsak",
"marriage": True,"age": 87,"country":"Thai",
"children": ("Odd","Nee","Also","Jar","Small"),
  "pets": None,
  "cars": [
    {"model": "Benz S-Class", "price": 3.5},
    {"model": "BMW 500", "price": 2.5}
  ]
}
# Convert person to JSON:
person_json = json.dumps(person)
print(person_json)
# Make it easier to read by using the paragraph indent :
print(json.dumps(person, indent=4))
# By using separators with specified values:
print(json.dumps(person, indent=4, separators=(". ", " = ")))
# Sorting
print(json.dumps(person, indent=4, sort_keys=True))
```

Results

{"name": "Manee", "age": 30}
["orange", "pineapple"]
["orange", "pineapple"]
"orange"
241
50.23
true
false
null
{"name": "Somsak", "marriage": true, "age": 87, "country": "Thai", "children": ["Odd", "Nee", "Pun", "Jar", "Yar"], "pets": null, "cars": [{"model": "Benz S-Class", "price": 3.5}, {"model": "BMW 500", "price": 2.5}]}

```
{
   "name": "Somsak",
   "marriage": true,
   "age": 87,
   "country": "Thai",
   "children": [
      "Odd",
      "Nee",
      "Also",
      "Jar",
      "Small"
   ],
   "pets": null,
   "cars": [
      {
         "model": "Benz S-Class",
         "price": 3.5
      },
      {
         "model": "BMW 500",
         "price": 2.5
      }
   ]
}
{
   "name" = "Somsak".
    "marriage" = true.
    "age" = 87.
    "country" = "Thai".
   "children" = [
         "Odd".
         "Nee".
      "Also".
         "Jar".
      "Small"
    ].
```

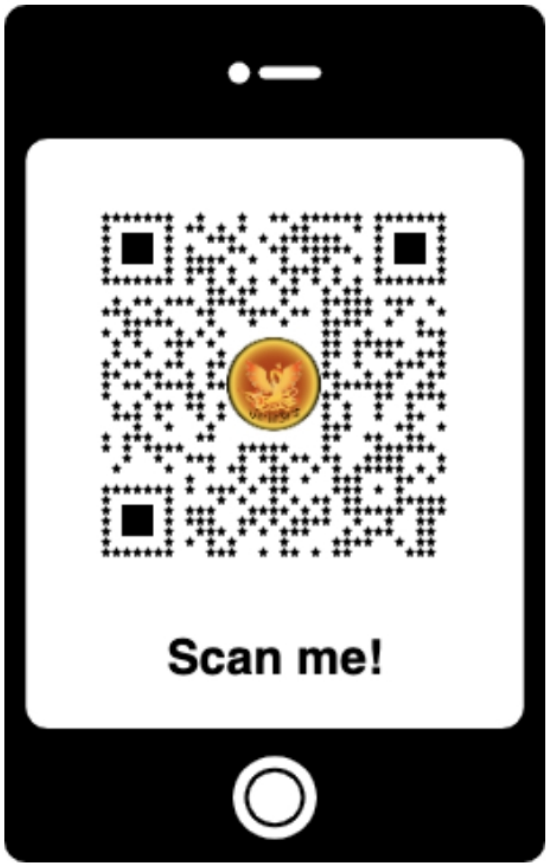
```
    "pets" = null.
   "cars" = [
      {
            "model" = "Benz S-Class".
         "price" = 3.5
        }.
      {
         "model" = "BMW 500".
         "price" = 2.5
      }
   ]
}
{
   "age": 87,
   "cars": [
      {
```

Exercises

1. Write a program to create, open, and edit a file to introduce yourself.
2. Create a json file introducing yourself and write a program to display the json file.

Files used in projects/Exercise Solutions

Go to : https://www.variitsris.org/learning-python/ Or Scan QR Code



About The Author

Having written several bestselling textbooks, the author has experience in software development, research, teaching, and consulting in:

- Artificial Intelligence
- Customer relationship management
- Data mining
- Object-Oriented Technology (UML).
- Human-computer interaction.
- Software engineering
- System analysis and design.
- Data warehouse
- Image processing
- Big data
- Business intelligence

Experience in program and database development and giving advice :

- Node.js
- JavaScript
- Oracle
- Python

- HTML5

- MongoDB

Author certifications associated with this textbook are as follows:

1. Cybersecurity: Managing Risk in the Information Age, HARVARD University.
2. Digital Transformation: From AI and IoT to Cloud, Blockchain and Cybersecurity, MIT.
3. Data Science and Big Data Analytics : Making Data-Driven Decisions, MIT.