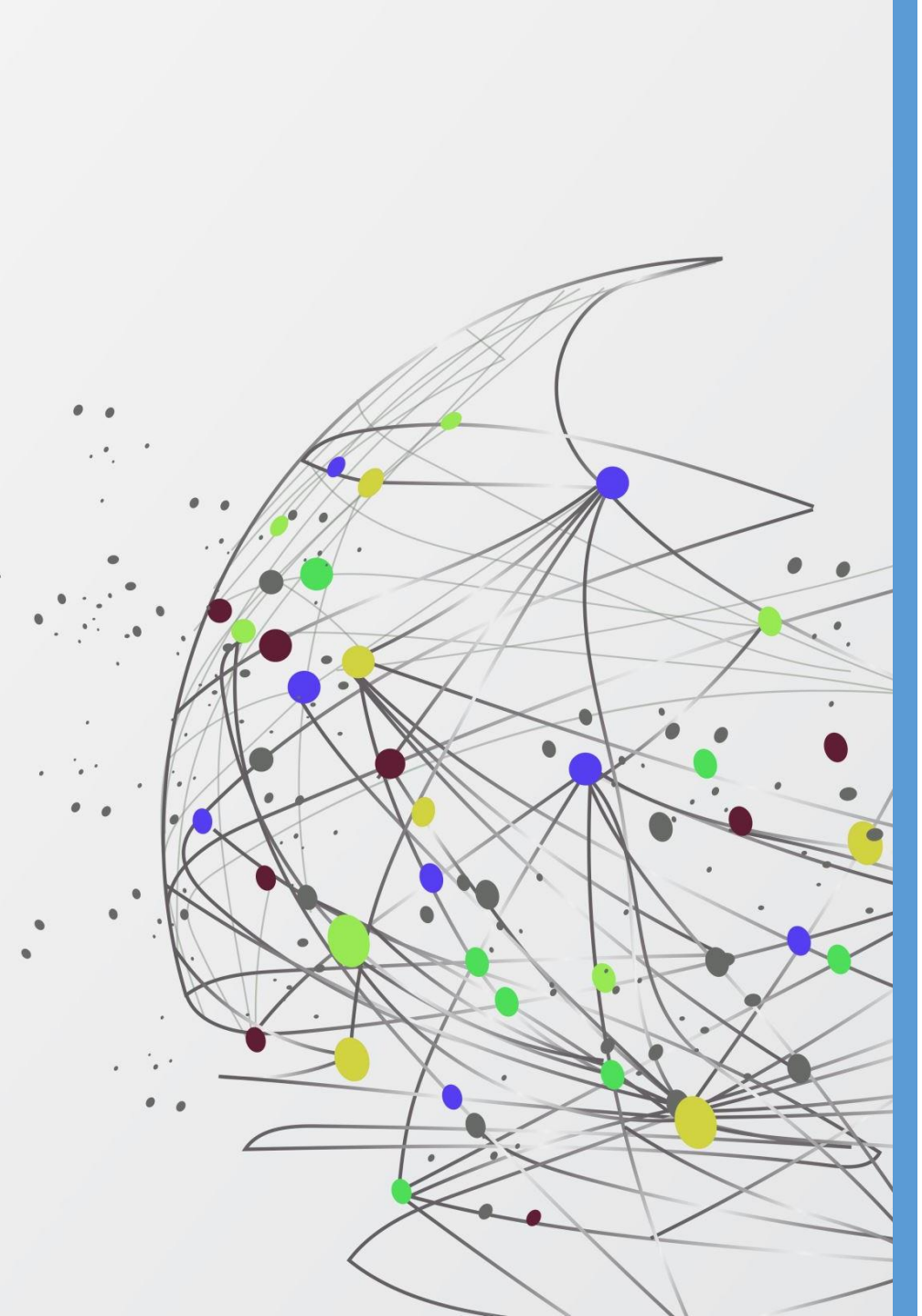
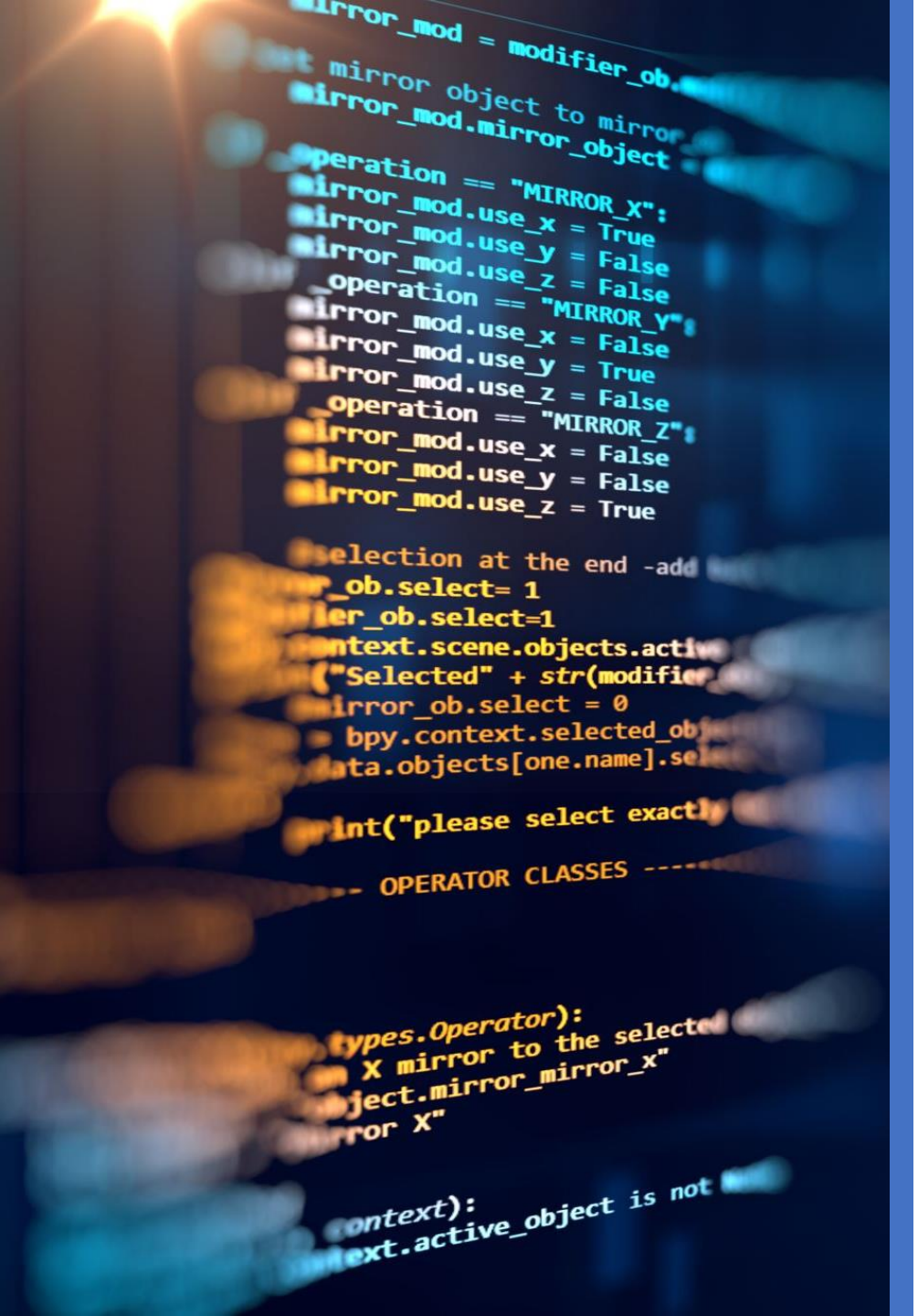
Computer Programing for Spatial Data



Overview



introduction

- This chapter introduces the basic concepts of computer, hardware, software, and programming, and sets up the context for GIS programming.

Computer Hardware and Software

- Computer Hardware
- Computer Software

(a)

(b)

FIGURE 1.1
(a) NASA supercomputer. (From NASA supercomputer at http://www.nas.nasa.gov/hecc/resources/pleiades.html.) (b) Other computers: personal computer (PC), laptop, pad. (From different computers at http://www.computerdoc.com.au/what-are-the-different-types-of-computers.)

Software (Application) Freeware or requiring purchase

OS

Several Application (Word, Excel, PowerPoint)

GIS software

- *Geographic information system* (GIS) is one type of application software that deals primarily with geographic information (Longley et al. 2001).

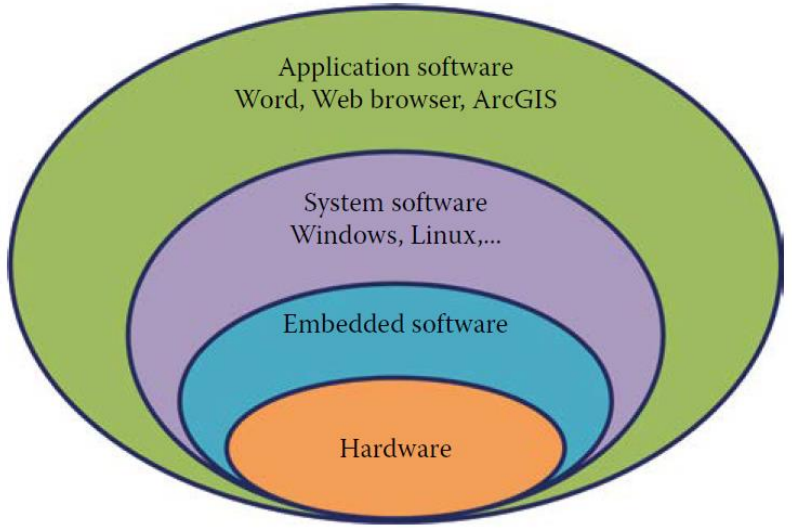
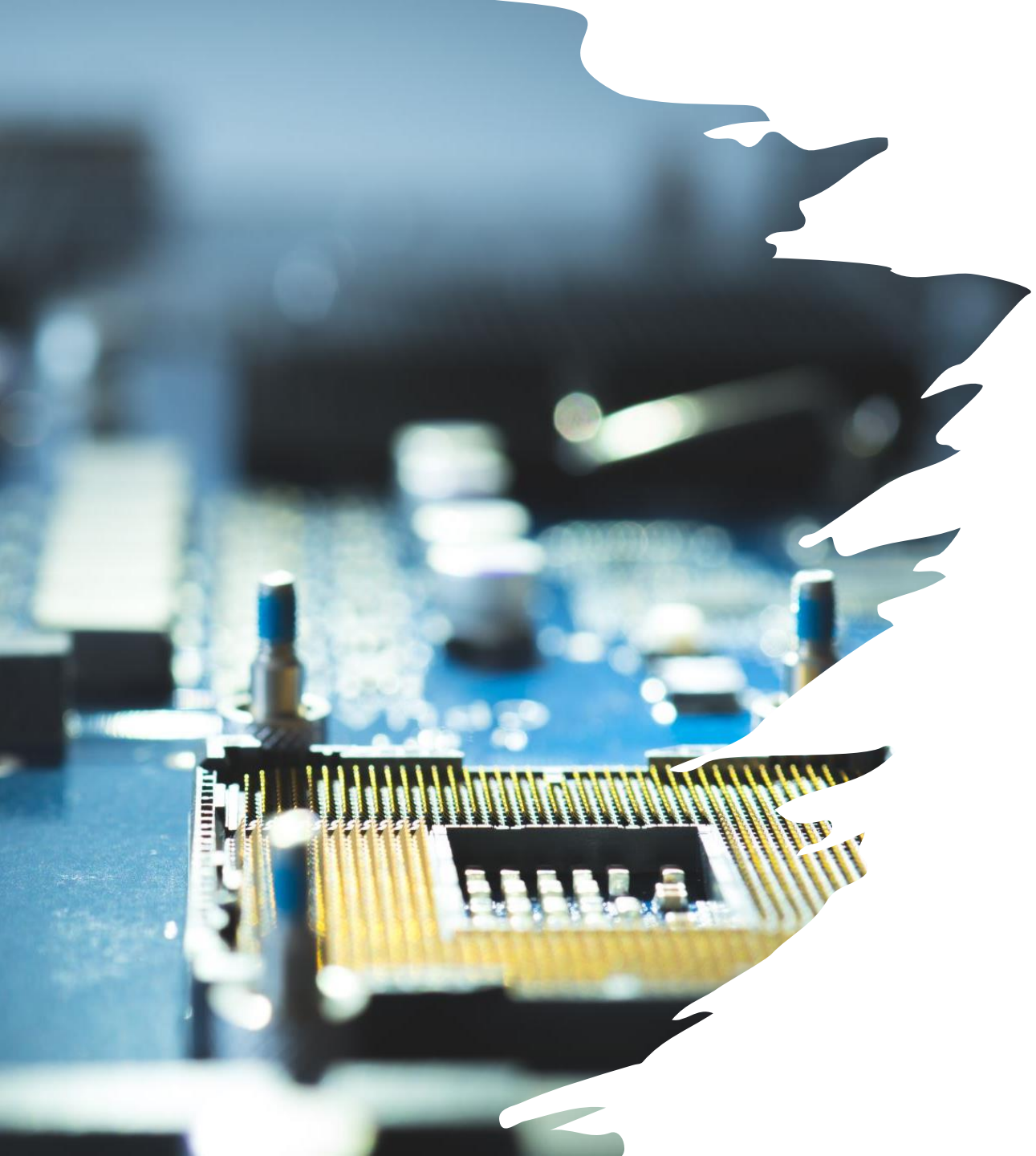


FIGURE 1.2
Different types of software.



GIS and Programming

- GIS originates from several domains and refers to the system designed to capture, observe, collect, store, and manage geographic data, and to provide tools for spatial analyses and visualization (Longley et al. 2001).

- The process of developing software is called programming. Programming instructs the computer to accomplish a task based on the orders. There are many different types of programming levels (Mitchell 1996).

Why do we need GIS programming?

Customizing	Customizing software for application:
Automating	Automating a process
Satisfying	Satisfying simple GIS need
Cultivating	Cultivating advanced GIS professionals

Python

It is excellent for programming beginners, yet superb for experts.

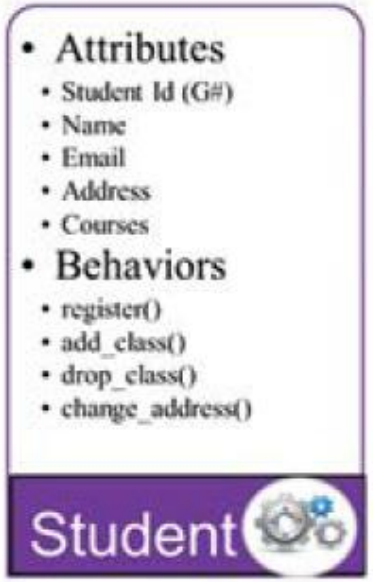
The syntax of Python is very simple and easy to learn. When you become familiar with them, you will feel that it is really very handy.

It is portable cross-platform. This means that a program written in Windows can be run using the Linux or Mac operating systems.

It is a fully object-oriented language, simple yet elegant, and stable and mature.

Class and Object

Class uses a set of attributes and behaviors to represent a category of real-world phenomena.



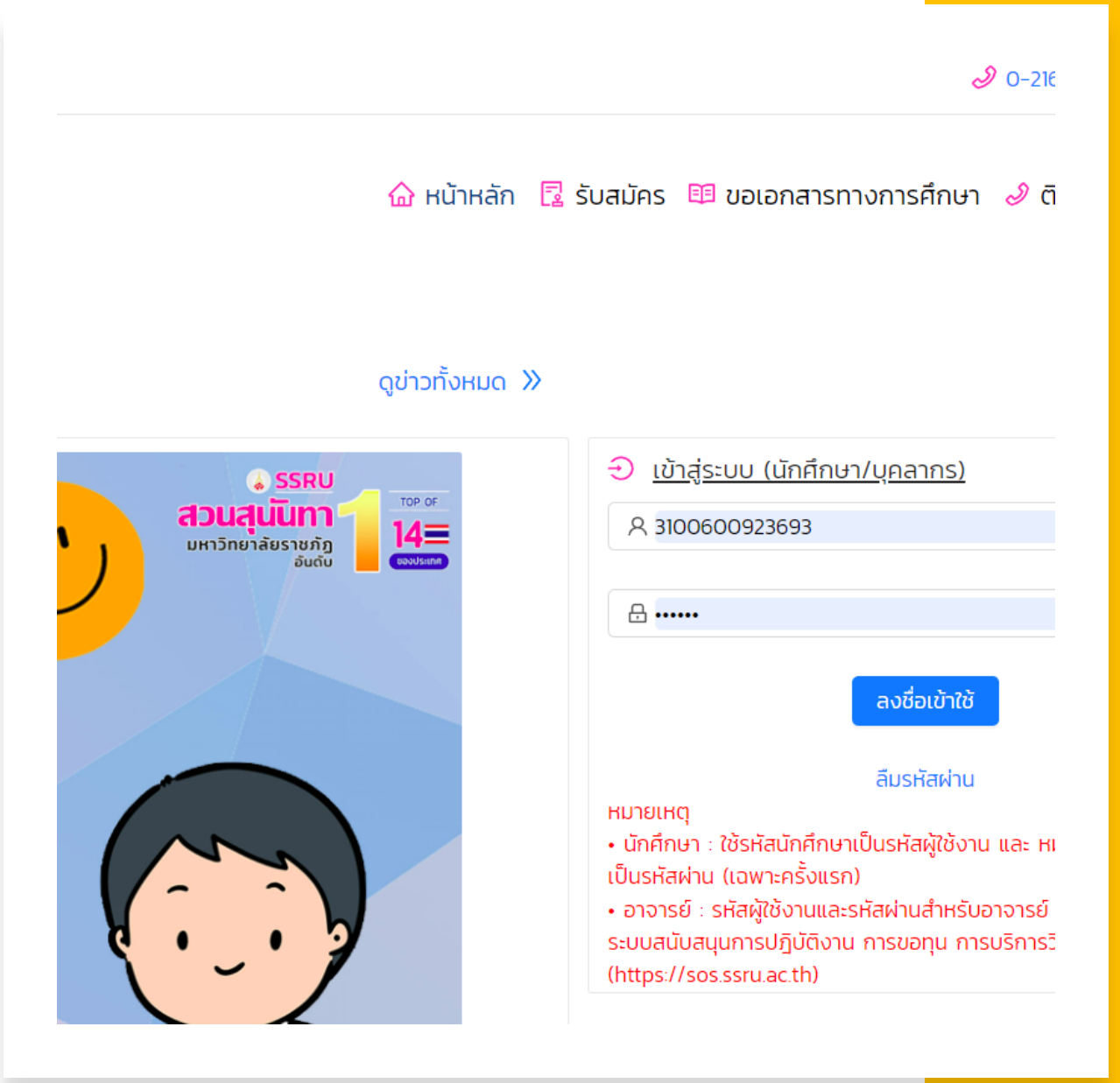
Attributes: A student will have a set of attributes, including student ID, name, email, current home address, courses the student have registered for each semester.

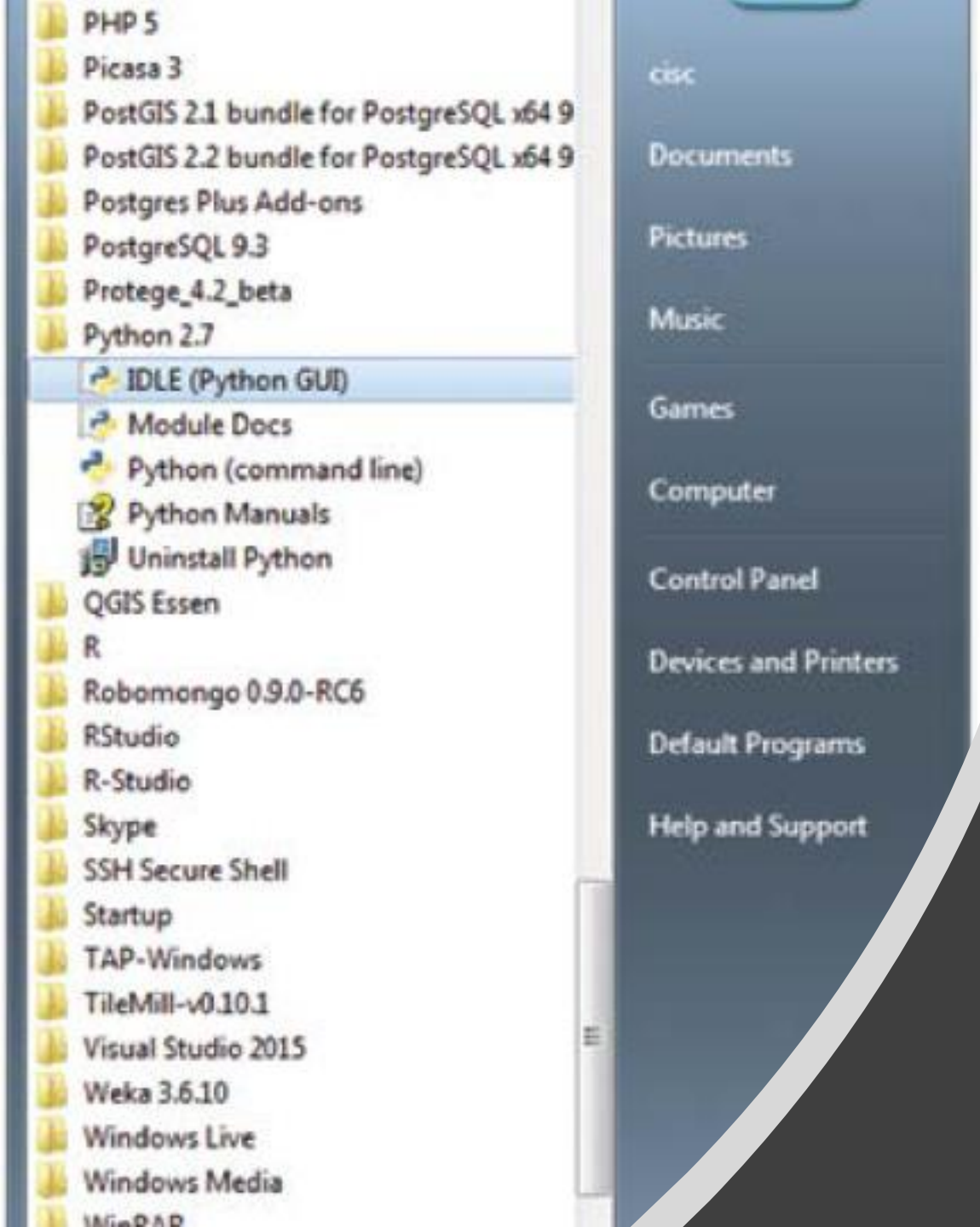
Behaviors: Behaviors could include register the current semester, change address if the student moves to a different place, add or drop a class.

FIGURE 1.3
An example of representing students with the Student class.

Object

- An object is a specific instance of a class. We can consider objects as instances of classes by assigning values to their attributes.





Hands-On Experience with Python

reference

1

2.1 Introduction

In this chapter we will return to the Hello World program and look at what it is doing. We will also modify it to become more interactive and will explore the concept of Python variables

2

2.2Hello World

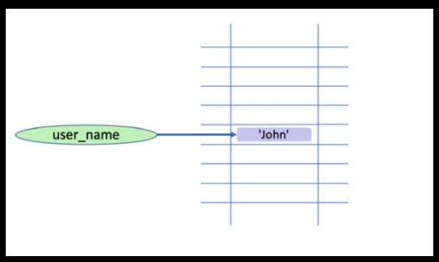
- `print('Hello World')`

  For example, we can create a file called hello.py containing the above print().

3

2.3 Interactive Hello World

- `print('Hello, world')`
- `user_name = input('Enter your name: ')`
- `print('Hello ', user_name)`



4

2.4 Variables

```
print('Hello, world')
name = input('Enter your name: ')
print('Hello', name)
name = input('What is the name of
your best friend: ')
print('Hello Best Friend', name)
```

```
my_variable = 'John'
print(my_variable)
my_variable = 42
print(my_variable)
my_variable = True
print(my_variable)
```

5

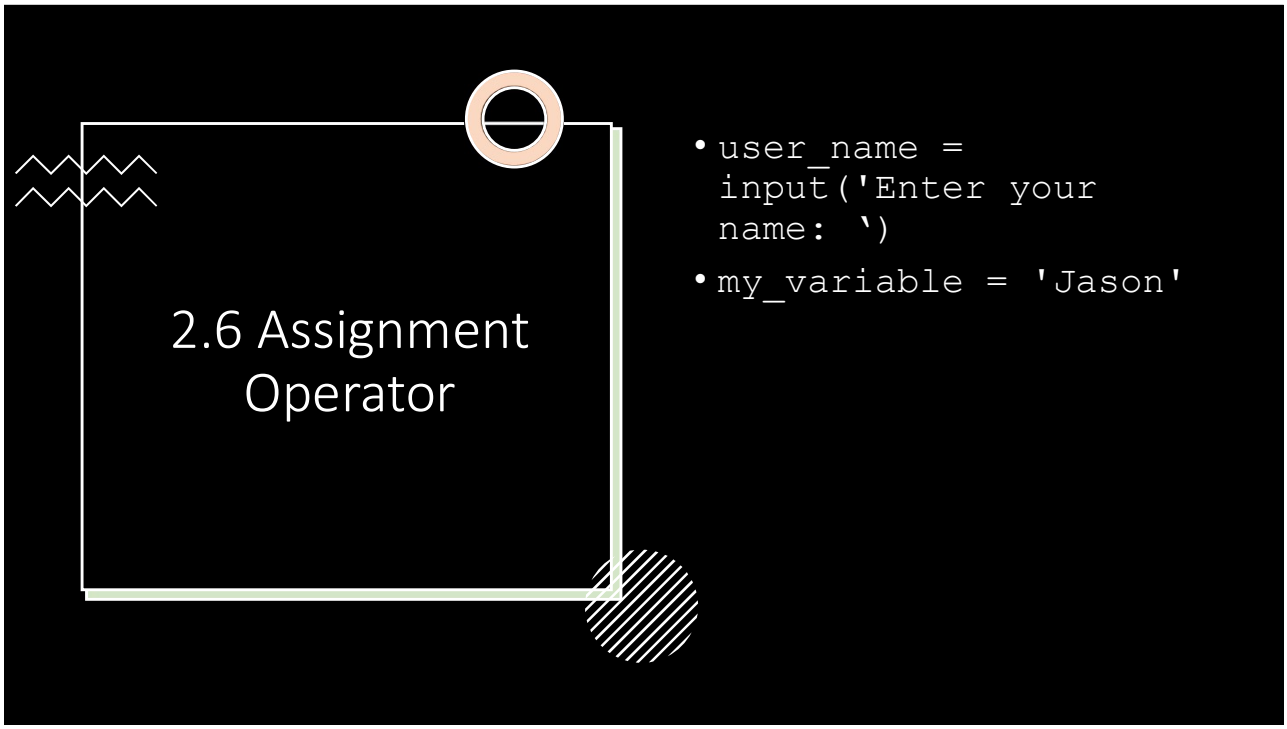
2.5 Naming Conventions

You may have noticed something above some of the variable names we have introduced above such as user_name and my_variable. Both these variable names are formed of a set of characters with an underbar between the 'words' in the variable name.

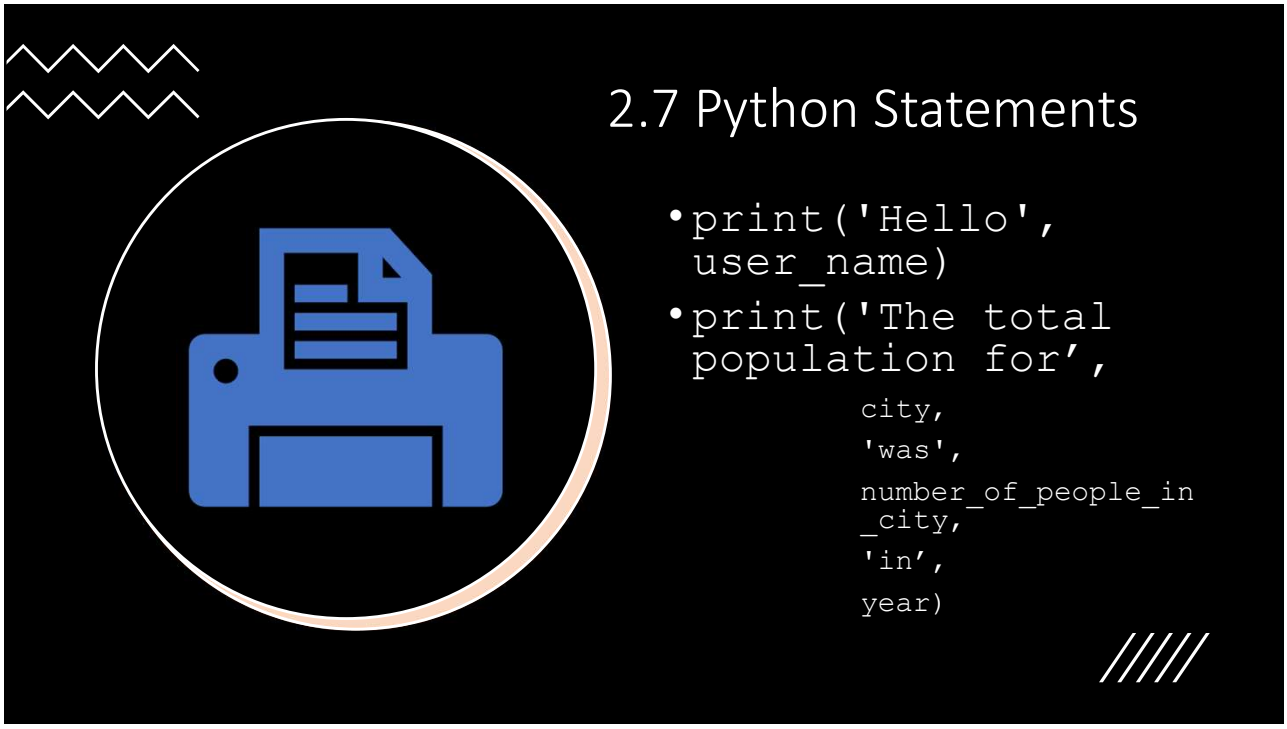
Both these variable names highlight a very widely used naming convention in Python, which is that variable names should:

- be all lowercase,
- be in general more descriptive than variable names such as a or b (although there are some exceptions such as the use of variables i and j in looping constructs).
- with individual words separated by underscores as necessary to improve readability.

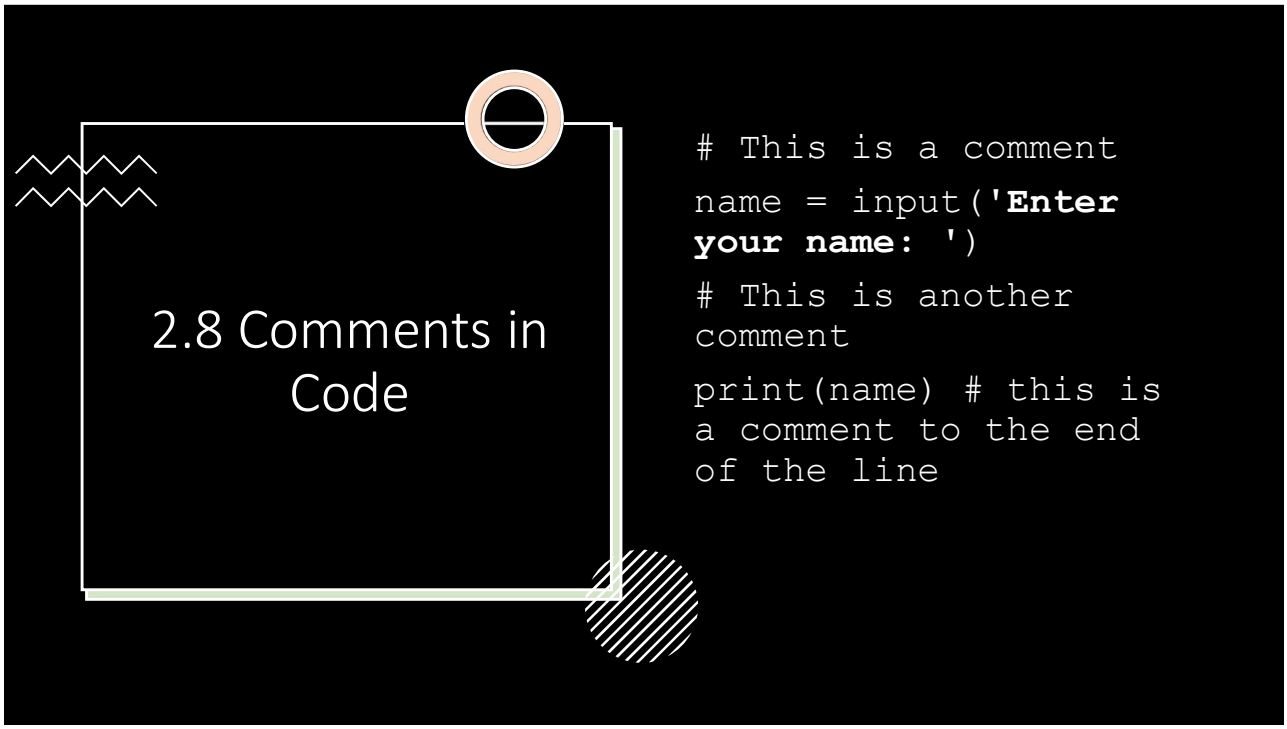
6


2.6 Assignment Operator
user_name = input('Enter your name: ‘)
my_variable = 'Jason'

7


2.7 Python Statements
print('Hello', user_name)
print('The total population for’, city, 'was', number_of_people_in_city, 'in’, year)

8



9

Exercises

1. Try creating your own variables and storing values into those instead of the

variable user_name.

2. Add a print() function to the program with your own prompt.

3. Include an assignment that will add two numbers together (for example 4 + 5) and then assign the result to a variable.

4. Now print out that variables value once it has been assigned a value.

5. Make sure you can run the program after each of the above changes. If there is an error reported attempt to fix that issue before moving on.

10

3. Python Strings

11

3.1Introduction

In the previous chapter we used strings several times, both as prompts to the user and as output from the print() function. We even had the user type in their name and store it in a variable that could be used to access this name at a later point in time. In this chapter we will explore what a string is and how you can work with and manipulate them.

12

3.2 What Are Strings?

During the description of the Hello World program we referred to Python strings several times, but what is a string? In Python a string is a series, or sequence, of characters in order. In this definition a character is anything you can type on the keyboard in one keystroke, such as a letter 'a', 'b', 'c' or a number '1', '2', '3' or a special characters such as '\', '[', '$' etc. a space is also a character ' ', although it does not have a visible representation. It should also be noted that strings are immutable. Immutable means that once

- `'Hello'`
- `'Hello World'`
- `'Hello Andrea2000'`
- `'To be or not to be that is the question!'`
- `some_string = ''`

13

3.3 Representing Strings

- `'Hello World'`
- `"Hello World"`

You should note however, that you cannot mix the two styles of start and end strings, that is you cannot start a string with a single quote and end a string with a double quote, thus the following are both illegal in Python:

- 'Hello World'' # This is illegal
- ''Hello World' # So is this

14

3.4 What Type Is String

```
my_variable = 'Bob'
print(type(my_variable))
```

- The result of executing these two lines of code is the output:
  - <class 'str'>

15

3.5 What Can You Do with Strings?

- **3.5.1 String Concatenation**

```
string_1 = 'Good'
string_2 = " day"
string_3 = string_1 + string_2
print(string_3)
print('Hello ' + 'World')
```

- The output from this is

Good day
Hello World

16

3.5.2 Length of a String

- `print(len(string_3))`

17

3.5.3 Accessing a Character

- `my_string = 'Hello World'`
- `print(my_string[4])`

18

3.5.4 Accessing a Subset of Characters

```
my_string = 'Hello World'
print(my_string[4]) # characters at position 4
print(my_string[1:5]) # from position 1 to 5
print(my_string[:5]) # from start to position 5
print(my_string[2:]) # from position 2 to the end
```

Will generate

```
o
ello
Hello
llo World
```

19

3.5.5 Repeating Strings

```
print('*' * 10)
print('Hi' * 10)
```

- Will generate

  ****

  HiHiHiHiHiHiHiHiHiHi

20

3.5.6 Splitting Strings

```
title = 'The Good, The Bad, and the Ugly'
print('Source string:', title)
print('Split using a space')
print(title.split(' '))
print('Split using a comma')
print(title.split(',’)
```

This produces as output

```
Source string: The Good, The Bad, and the Ugly
Split using a space
['The', 'Good,', 'The', 'Bad,', 'and', 'the', 'Ugly']
Split using a comma
['The Good', ' The Bad', ' and the Ugly']
```

21

3.5.7 Counting Strings

my_string = 'Count, the number of spaces'

print("my_string.count(' '):", my_string.count(' ‘))

Which has the output

my_string.count(' '): 8

22

3.5.8 Replacing Strings

- welcome_message = 'Hello World!'
- print(welcome_message.replace("Hello", "Goodbye"))

The output produced by this is thus

Goodbye World!

23

3.6 Hints on Strings

- Python Strings Are Case Sensitive
  - some_string.lower().startswith('h')
- Function/Method Names

  Be very careful with capitalisation of function/method names; in Python isupper() is a completely different operation to isUpper().
- Function/Method Invocations

```
print(some_string.isupper)
print(some_string.isupper())
```

produces the output:

```
<built-in method isupper of str object at
0x105eb19b0>
False
```

Notice that the first print out tells you that you are referring to the built-in method called isupper defined on the type String; while the second actually runs isupper() for you and returns either True or False.

24

3.7 String Formatting

```
format_string = 'Hello {}!'
```

This can be used with the format() string method to provide a value (or

populate) the placeholder, for example:

```
print(format_string.format('Phoebe'))
```

The output from this is:

```
Hello Phoebe!
```

25

3.7 String Formatting

- # Allows multiple values to populate the string
- name = "Adam"
- age = 20
- print("{} is {} years old".format(name, age))

- In this case the output is:

Adam is 20 years old

26

3.7 String Formatting

- # Can specify an index for the substitution
- format_string = "Hello {1} {0}, you got {2}%"
- print(format_string.format('Smith', 'Carol', 75))

Hello Carol Smith, you got 75%

27

3.7 String Formatting

- # Can use named substitutions, order is not significant
- format_string = "{artist} sang {song} in {year}"
- print(format_string.format(artist='Paloma Faith',
- song='Guilty', year=2017))

Paloma Faith sang Guilty in 2017

28

3.7 String Formatting

- print('|{:25}|'.format('25 characters width'))

|25 characters width |

- `print('|{:<25}|'.format('left aligned')) # The default`
- `print('|{:>25}|'.format('right aligned'))`
- `print('|{:^25}|'.format('centered'))`

Which produces:

```
|left aligned |
| right aligned|
| centered |
```

29

3.7 String Formatting

```
# Can format numbers with comma as thousands separator
print('{:,}'.format(1234567890))
print('{:,}'.format(1234567890.0))
```

Which generates the output:

```
1,234,567,890
1,234,567,890.0
```

30

3.8 String Templates

```
import string
# Initialise the template with ¢variables that
# will be substitute with actual values
template = string.Template('$artist sang $song in $year')

print(template.substitute(artist='Freddie Mercury', song='The
Great Pretender', year=1987))

'Freddie Mercury sang The Great Pretender in 1987'
```

31

Exercises

1. Explore replacing a string

Create a string with words separated by ',' and replace the commas with spaces; for example replace all the commas in 'Denyse,Marie,Smith,21,London,UK' with spaces. Now print out the resulting string.

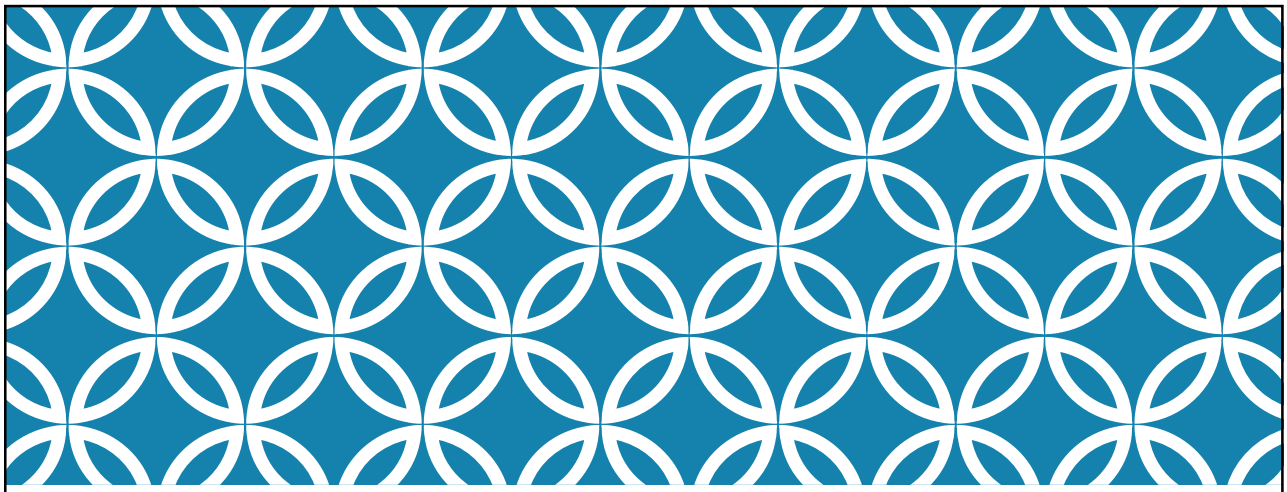
2. Handle user input

The aim of this exercise is to write a program to ask the user for two strings and concatenate them together, with a space between them and store them into a new variable called new_string.

Next:

- Print out the value of new_string.
- Print out how long the contents of new_string is.
- Now convert the contents of new_string to all upper case.
- Now check to see if new_string contains the string 'Albus' as a substring.

32



NUMBERS, BOOLEANS AND NONE

1

CONTENT

Introduction

Types of Numbers

Integers

Floating Point Numbers

Complex Numbers

Boolean Values

Arithmetic Operators

None Value

2

5.1 INTRODUCTION

In this chapter we will explore the different ways that numbers can be represented by the built-in types in Python. We will also introduce the Boolean type used to represent True and False. As part of this discussion we will also look at both numeric and assignment operators in Python. We will conclude by introducing the special value known as None

3

5.2 TYPES OF NUMBERS

There are three types used to represent numbers in Python; these are integers (or integral) types, floating point numbers and complex numbers.

4

5.3 INTEGERS

```
x = 1
print(x)
print(type(x))
x =
10000000000000000000000000000000000000000000000000000000000001
print(x)
print(type(x))
```

```
1
<class 'int'>
10000000000000000000000000000000000000000000000000000000000001
<class 'int'>
```

5

5.3.1 CONVERTING TO INTS

```
total = int('100')
age = int(input('Please enter your age:'))
print(type(age))
print(age)
```

Running this gives:

```
Please enter your age: 21
<class 'int'>
21
```

6

5.4 FLOATING POINT NUMBERS

```
exchange_rate = 1.83
print(exchange_rate)
print(type(exchange_rate))
```

This produces output indicating that we are storing the number 1.83 as a floating point number:

```
1.83
<class 'float'>
```

7

5.4.1 CONVERTING TO FLOATS

```
int_value = 1
string_value = '1.5'
float_value = float(int_value)
print('int value as a float:', float_value)
print(type(float_value))
float_value = float(string_value)
print('string value as a float:', float_value)
print(type(float_value))
```

The output from this code snippet is:

```
int value as a float: 1.0
<class 'float'>
string value as a float: 1.5
<class 'float'>
```

8

5.4.2 CONVERTING AN INPUT STRING INTO A FLOATING-POINT NUMBER

exchange_rate = float(input("Please enter the exchange rate to use: "))

print(exchange_rate)

print(type(exchange_rate))

Using this we can input the string 1.83 and convert it to a floating-point number:

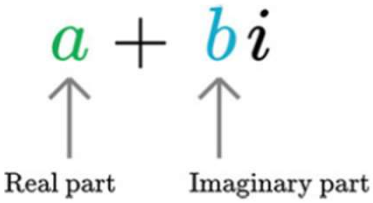
Please enter the exchange rate to use: 1.83

1.83

<class 'float'>

9

5.5 COMPLEX NUMBERS




```
c1 = 1j
c2 = 2j
print('c1:', c1, ', c2:', c2)
print(type(c1))
print(c1.real)
print(c1.imag)
```

We can run this code and the output will be:

```
c1: 1j , c2: 2j
<class 'complex'>
0.0
1.0
```

10

5.6 BOOLEAN VALUES

```
all_ok = True
print(all_ok)
all_ok = False
print(all_ok)
print(type(all_ok))
```

The output of this is

```
True
False
<class 'bool'>
```

```
print(int(True))
print(int(False))
print(bool(1))
print(bool(0))
```

Which produces

```
1
0
True
False
```

11

5.7 ARITHMETIC OPERATORS

Operator	Description	Example
+	Add the left and right values together	`1 + 2`
−	Subtract the right value from the left value	`3 − 2`
*	Multiple the left and right values	`3 * 4`
/	Divide the left value by the right value	`12/3`
//	Integer division (ignore any remainder)	`12//3`
%	Modulus (aka the remainder operator)—only return any remainder	`13%3`
**	Exponent (or power of) operator—with the left value raised to the power of the right	`3 ** 4`

12

5.7.1 INTEGER OPERATIONS

```
home = 10
away = 15
print(home + away)
print(type(home + away))
print(10 * 4)
print(type(10*4))
goals_for = 10
goals_against = 7
print(goals_for - goals_against)
print(type(goals_for - goals_against))
```

The output from this is 25

```
<class 'int'>
        40
<class 'int'>
        3
<class 'int'>
```

```
print(100 / 20)
print(type(100 / 20))
```

The output is

```
5.0
<class 'float'>
```

13

5.7.2 NEGATIVE NUMBER INTEGER DIVISION

```
print('True division 3/2:', 3 / 2)
print('True division 3//2:', -3 / 2)
print('Integer division 3//2:', 3 // 2)
print('Integer division 3//2:', -3 // 2)
```

The output from this is:

True division 3/2: 1.5

True division 3//2: -1.5

Integer division 3//2: 1

Integer division 3//2: -2

14

5.7.3 FLOATING POINT NUMBER OPERATORS

```
print(2.3 + 1.5)
print(1.5 / 2.3)
print(1.5 * 2.3)
print(2.3 - 1.5)
print(1.5 - 2.3)
```

These statements produce the output given below:

```
3.8
0.6521739130434783
3.4499999999999997
0.7999999999999998
-0.7999999999999998
```

15

5.7.4 INTEGERS AND FLOATING POINT OPERATIONS

```
i = 3 * 0.1
print(i)
```

Executing this we get

```
0.30000000000000004
```

16

5.7.5 COMPLEX NUMBER OPERATORS

```
c1 = 1j
c2 = 2j
c3 = c1 * c2
print(c3)
```

We can run this code and the output will be:

```
(-2+0j)
```

17

5.8 ASSIGNMENT OPERATORS

```
x = 0
x += 1 # has the same behaviour as x = x + 1
```

Operator	Description	Example	Equivalent
`+=`	Add the value to the left-hand variable	`x += 2`	`x = x + 2`
`-=`	Subtract the value from the left-hand variable	`x -= 2`	`x = x - 2`
`*=`	Multiple the left-hand variable by the value	`x *= 2`	`x = x * 2`
`/=`	Divide the variable value by the right-hand value	`x /= 2`	`x = x/2`
`//=`	Use integer division to divide the variable's value by the right-hand value	`x //= 2`	`x = x//2`
`%=`	Use the modulus (remainder) operator to apply the right-hand value to the variable	`x %= 2`	`x = x % 2`
`**=`	Apply the power of operator to raise the variable's value by the value supplied	`x **= 3`	`x = x ** 3`

18

5.9 NONE VALUE

winner = None

You can then test for the presence of None using 'is' and 'is not', for

example:

print(winner is None)

This will print out True if and only if the variable winner is currently set to

None.

Alternatively you can also write:

print(winner is not None)

19

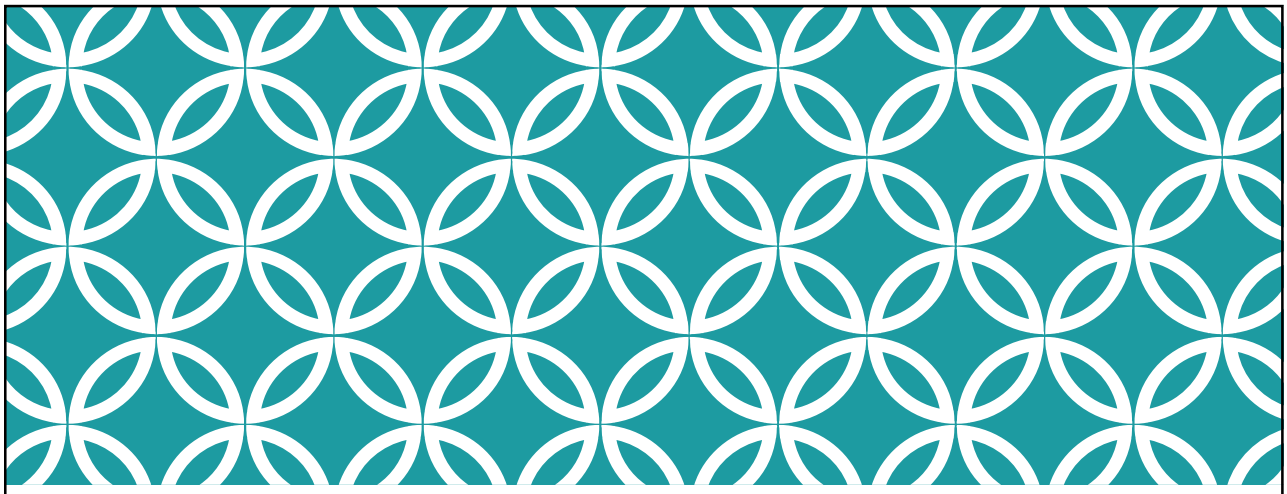
5.9 NONE VALUE

```
winner = None
print('winner:', winner)
print('winner is None:', winner is None)
print('winner is not None:', winner is not None)
print(type(winner))
print('Set winner to True')
winner = True
print('winner:', winner)
print('winner is None:', winner is None)
print('winner is not None:', winner is not None)
print(type(winner))
```

The output of this code snippet is:

```
winner: None
winner is None: True
winner is not None: False
<class 'NoneType'>
Set winner to True
winner: True
winner is None: False
winner is not None: True
<class 'bool'>
```

20



FLOW OF CONTROL USING IF STATEMENTS

21

6.1 INTRODUCTION

In this chapter we are going to look at the if statement in Python. This statement is used to control the flow of execution within a program based on some condition. These conditions represent some choice point that will be evaluated to True or False. To perform this evaluation, it is common to use a comparison operator (for example to check to see if the temperature is greater than some threshold).

In many cases these comparisons need to take into account several values and in these situations logical operators can be used to combine two or more comparison expressions together.

22

6.2 COMPARISON OPERATORS

In Python there are a range of comparison operators represented by typically one or two characters. These are:

Operator	Description	Example
==	Tests if two values are equal	3 == 3
!=	Tests that two values are *not* equal to each other	2 != 3
<	Tests to see if the left-hand value is less than the right-hand value	2 < 3
>	Tests if the left-hand value is greater than the right-hand value	3 > 2
<=	Tests if the left-hand value is less than *or* equal to the right-hand value	3 <= 4
>=	Tests if the left-hand value is greater than or equal to the right-hand value	5 >= 4

23

LOGICAL OPERATORS

Operator	Description	Example
`and`	Returns True if both left and right are true	`(3 < 4) and (5 > 4)`
`or`	Returns two if either the left or the right is truce	`(3 < 4) or (3 > 5)`
`not`	Returns true if the value being tested is False	`not 3 < 2`

24

6.4 THE IF STATEMENT

An if statement is used as a form of conditional programming; something you probably do every day in the real world. That is, you need to decide whether you are going to have tea or coffee or to decide if you will have toast or a muffin for breakfast etc. In each of these cases you are making a choice, usually based on some information such as I had coffee yesterday, so I will have tea today. In Python such choices are represented programmatically by the if condition statement. In this construct if some condition is true some action is performed, optionally if it is not true some other action may be performed instead

25

6.4.1 WORKING WITH AN IF STATEMENT

```
num = int(input('Enter a number: '))
if num < 0:
        print(num, 'is negative')
```

For example,

Enter a number: -1

-1 is negative

```
num = int(input('Enter another number: '))
if num > 0:
    print(num, 'is positive')
    print(num, 'squared is ', num * num)
print('Bye')
```

Enter another number: 2

2 is positive

2 squared is 4

Bye

26

6.4.2 ELSE IN AN IF STATEMENT

```
num = int(input('Enter yet another number: '))
if num < 0:
        print('Its negative')
else:
        print('Its not negative')
```

For example, in run 1 if we enter the value 1:
Enter yet another number: 1
Its not negative
And in run 2 if we enter the value −1:
Enter yet another number: -1
Its negative

27

6.4.3 THE USE OF ELIF

```
savings = float(input("Enter how much you have in savings: "))
if savings == 0:
        print("Sorry no savings")
elif savings < 500:
        print('Well done')
elif savings < 1000:
        print('Thats a tidy sum')
elif savings < 10000:
    print('Welcome Sir!')
else:
    print('Thank you')
```

If we run this:
Enter how much you have in savings: 500
Thats a tidy sum

28

6.5 NESTING IF STATEMENTS

```
snowing = True
temp = -1
if temp < 0:
        print('It is freezing')
        if snowing:
                print('Put on boots')
        print('Time for Hot Chocolate')
print('Bye')
```

```
snowing = True
temp = -1
if temp < 0:
        print('It is freezing')
        if snowing:
                print('Put on boots')
                print('Time for Hot Chocolate')
print('Bye')
```

29

6.6 IF EXPRESSIONS

```
age = 15
status = None
if (age > 12) and age < 20:
        status = 'teenager'
else:
        status = 'not teenager'
print(status)
```

****<result1> if <condition-is-met> else <result2>***

30

6.7 A NOTE ON TRUE AND FALSE

Python is actually quite flexible when it comes to what actually is used to represent True and False, in fact the following rules apply

- 0, '' (empty strings), None equate to False
- Non zero, non empty strings, any object equate to True.

However, we would recommend sticking to just True and False as it is often a cleaner and safer approach.

31

ITERATION/LOOPING

1

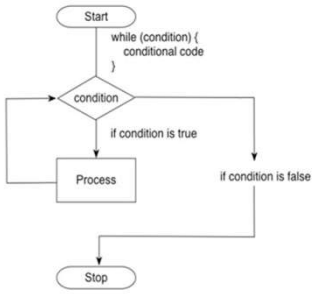
7.1 INTRODUCTION

In this section we will look at the while loop and the for loop available in Python. These loops are used to control the repeated execution of selected statements.

2

7.2 WHILE LOOP

The while loop exists in almost all programming languages and is used to iterative (or repeat) one or more code statements as long as the test condition (expression) is True. This iteration construct is usually used when the number of times we need to repeat the block of code to execute is not known. For example, it may need to repeat until some solution is found or the user enters a particular value. The behaviour of the while loop is illustrated in below.

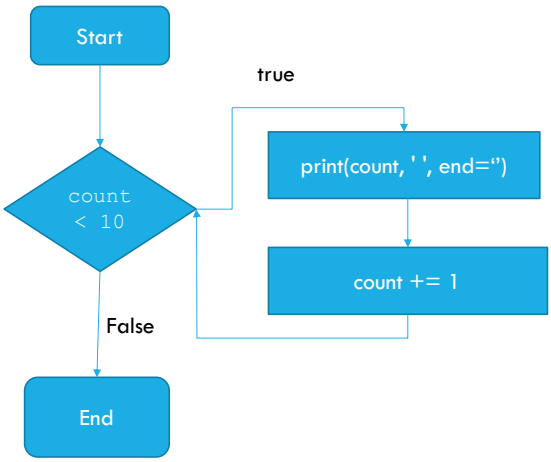


J. Hunt, *A Beginners Guide to Python 3 Programming*, Undergraduate Topics in Computer Science, https://doi.org/10.1007/978-3-030-20290-3_7

3

WHILE LOOP

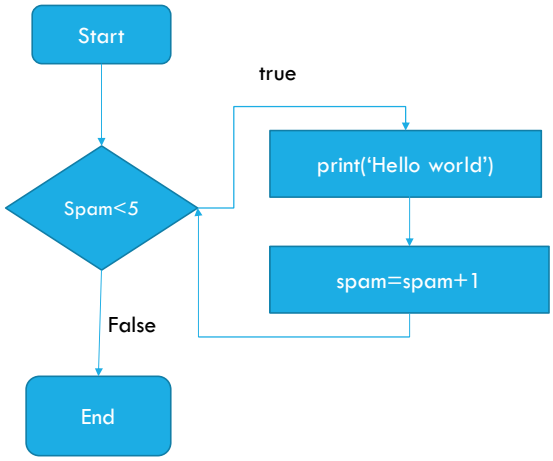
```
count = 0
print('Starting')
while count < 10:
    print(count, ' ', end='')
    count += 1
print()
print('Done')
```



4

WHILE LOOP

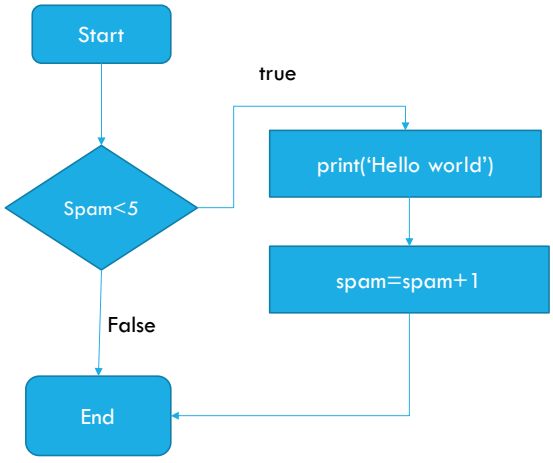
```
spam=0
while spam<5:
  print('Hello world')
  spam=spam+1
```



5

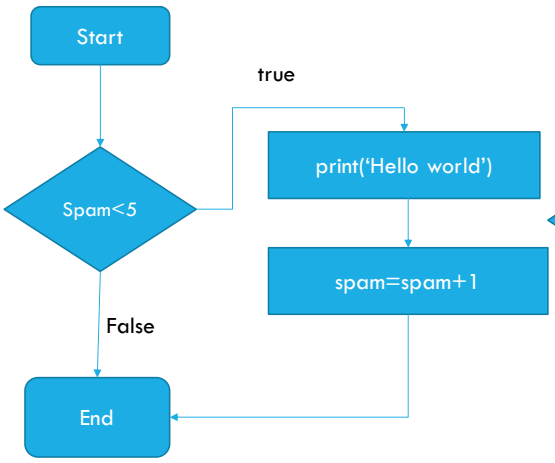
IF

```
If spam<5:
  print('Hello world')
  spam=spam+1
```

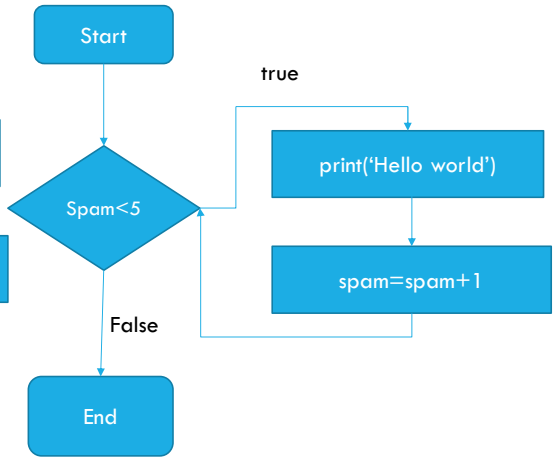


6

If spam<5:
print('Hello world')
spam=spam+1



spam=0
while spam<5:
print('Hello world')
spam=spam+1

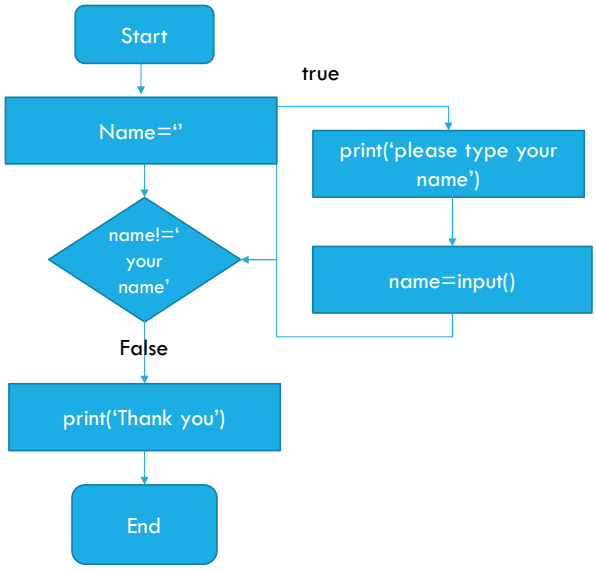


7

WHILE LOOP

name=''

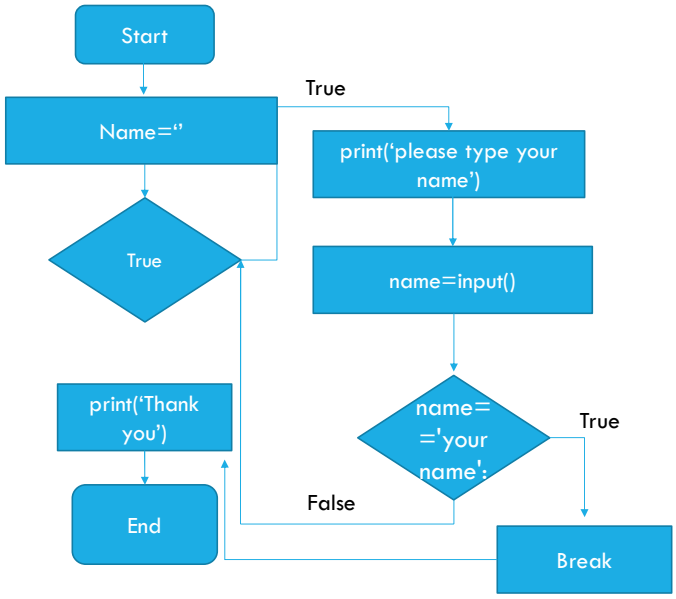
while name!='your name'
print('please type your name')
name=input()
print('Thank you')



8

WHILE LOOP/ BREAK

```
name=''
while True:
    print('Please type your name')
    name = input()
    if name=='your name':
        break
print('Thank you')
```



9

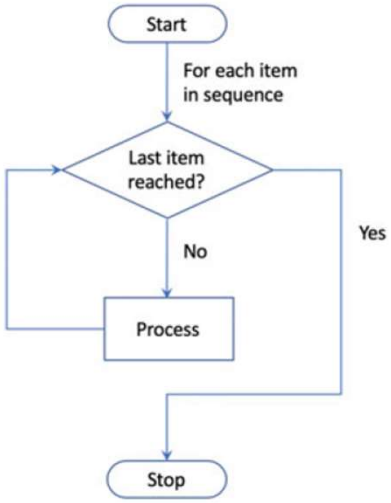
WHILE LOOP/ CONTINUE

```
spam=0
while spam<5:
        spam=spam+1
        if spam==3:
                Continue
        Print('spam is  '+ str(spam))
```

10

7.3 FOR LOOP

The for loop is used to step a varic through a series of values until a gi test is met. The behaviour of the fo loop is illustrated in below.

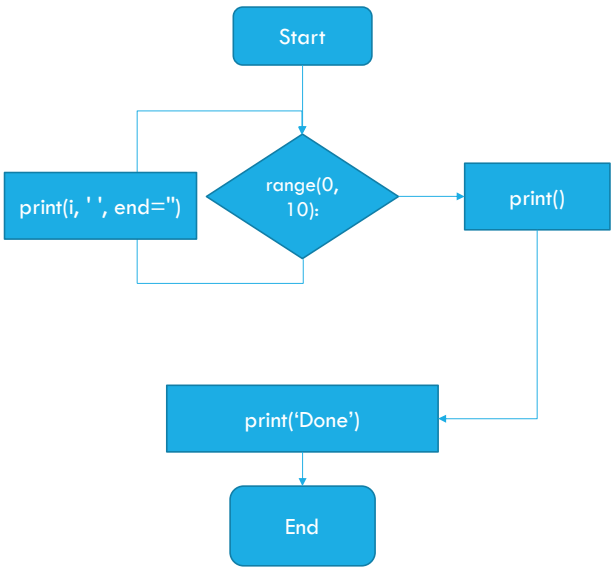


11

FOR LOOP

```
# Loop over a set of values in a range
for i in range(0, 10):
        print(i, ' ', end='')
print()
print('Done')
```

```
Print out values in a range
0 1 2 3 4 5 6 7 8 9
Done
```



12

BREAK LOOP STATEMENT

```
print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
        if i == num:
                break
        print(i, ' ', end='')
print('Done')
```

Enter a number to check for: *7*
0 1 2 3 4 5 Done
Enter a number to check for: *3*
0 1 2 Done

13

CONTINUE LOOP STATEMENT

```
for i in range(0, 10):
        print(i, ' ', end='')
        if i % 2 == 1:
                continue
        print('hey its an even number')
        print('we love even numbers')
print('Done')
```

0 hey its an even number
we love even numbers
1 2 hey its an even number
we love even numbers
3 4 hey its an even number
we love even numbers
5 6 hey its an even number
we love even numbers
7 8 hey its an even number
we love even numbers
9 Done

14

FOR LOOP WITH ELSE

```
# Only print code if all iterations completed over a list
print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
        if i == num:
            break
        print(i, ' ', end='')
else:
        print()
        print('All iterations successful')
```

```
Only print code if all iterations completed
Enter a number to check for: 7
0 1 2 3 4 5
All iterations successful
```

15

END

16

LIST TUPLE SET DICTIONARY AND STRING

1

LIST

A list, in Python, is a collection of objects. As per Lutz "It is the most general sequence provided by the language." Unlike strings, lists are mutable. That is, an element at a particular position can be changed in a list. A list is useful in dealing with homogeneous and heterogeneous sequences.

A list can be one of the following:

A list can be a collection of similar elements (homogeneous), for example

[1, 2, 3]

It can also contain different elements (heterogeneous), like [1, "abc," 2.4]

A list can also be empty ([])

A list can also contain a list

2

LIST

Listing

```
authors = ['Harsh Bhasin', 'Mark Lutz', 'Shiv']
print(authors)
combined =[1, 'Harsh', 23.4, 'a']
print(combined)
list3= []
print(list3)
listoflist = [1, [1,2], 3]
print(listoflist)
```

```
['Harsh bhasin', 'Mark Lutz', 'Shiv']
[1, 'Harsh', 23.4, 'a']
[]
[1, [1, 2], 3]
```

3

LIST

Listing

```
list1 = [1, 2, 3]
print(list1[1])
print(list1[-1])
>>>
```

Output

```
========== RUN
C:/Python/Chapter 2/list2.py
==========
2
3 >>>
```

4

LIST METHOD

Append

Extend(list)

Count()

Index(x)

Insert(index, value)

Remove(value)

Revere()

Sort()

5

LIST FUCNTION

Comp(list1,list2)

Len(list)

List(data)

max(list)

Min(list)

6

EXAMPLE

```
a=[1,4,7,6,5,9,18]  #list
print('number of a', len(a)) #function
print('a :', a)
print('max value :', max(a)) #function
a.sort() #method
print('sorting a:',a)
```

7

EXAMPLE

```
m=[]
for i in range(10):
        x=int(input('input number:'))
        m.append(x)
print(m)
```

```
a=list(range(0,22,3))
print(a)
```

8

TUPLES

tuple contains elements which can be treated individually or as a group. A tuple (say (x, y)) can be printed using the standard print( ) function. The elements of a tuple can be accessed by assigning it to a tuple, as shown in the following listing. A tuple may also contain heterogeneous elements. For example, in the following listing, tup2 and tup3 contain a string and an integer.

9

TUPLES

Listing

```
tup1= (2, 3)
print(tup1)
(a, b) = tup1
print('The first element is ',a)
print('The second element is ',b)
tup2=(101, 'Hari')
tup3=(102,'Shiv')
(code1, name1)=tup1
(code2, name2)=tup2
print('The code of ', name1,' is ',code1,'\nThe code of ',name2, ' is ',code2)
```

10

TUPLES

========== RUN C:/Python/Chapter 2/tuple.py ==========

(2, 3)

The first element is 2

The second element is 3

The code of 3 is 2

The code of Hari is 101

>>>

11

TUPLES

```
print('Enter the first number\t:')
num1= int(input())
print('Enter the second number\t:')
num2= int(input())
print('\nThe numbers entered are ',num1,' & ', num2)
(num1, num2) = (num2, num1)
print('\nThe numbers now are ',num1,' & ', num2)
>>>
```

Output

========== RUN C:/Python/Chapter 2/swap.py ==========

Enter the first number :

2

Enter the second number :

3

The numbers entered are 2& 3

The numbers now are 3& 2

>>>

12

TUPLE METHOD

Count(x)

Index(x)

Cmp(tup1,tup2)

Len(tp)

Max(tp)

Min(tp)

Tuple(st)

13

EXAMPLE

d= (4,5,6,7,8,9,1,3)

d.index(3)

14

SET

เป็นการจัดเก็บข้อมูลคล้ายกับ List และ Tuple แต่ข้อมูลจะไม่มีการซ้ำกันและไม่สนใจลำดับ

```
a={1,2,2,2,3,4,4,5,6,8,8,7,7,9}
a
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

15

SET FUNCTION AND METHOD

Add(x)	Set1.issubset(set2)
Clear()	Set1.issuperset(set2)
Discard(x)	Set1.union(set2)
Len(set)	Set1.update({x,y,z,…})
Pop()	Set1. intersection(set2)
Remove()	Set1. difference(set2)
Set(list)	Set1. symmetric_difference(set2)

16

EXAMPLE

```
s=set("geography")
m={'g','o'}
print(m.issubset(s))
True
print(s)
{'y', 'o', 'h', 'p', 'r', 'g', 'a', 'e'}
```

```
a={1,2,3,4,5,6}
b={2,4,6,8}
a.add(9)
a
{1, 2, 3, 4, 5, 6, 9}
c=b.intersection(a)
c
{2, 4, 6}
```

```
c=b.union(a)
c
{1, 2, 3, 4, 5, 6, 8, 9}
c=b.difference(a)
c
{8}
```

17

DICTIONARY

key	value
1	cat
2	dog
3	bear
4	rat

```
a={1:'cat',2:'dog',3:'bear',4:'rat'}
a
{1: 'cat', 2: 'dog', 3: 'bear', 4: 'rat'}
print(a[2])
dog
print(a[4])
rat
print(a.key())
```

18

DICTIONARY FUNCTION AND METHOD

Clear()

Cmp(dict1,dict2)

Copy()

Fromkeys(seq,v)

Len(dict)

Type()

Get(key)

Keys()

Value()

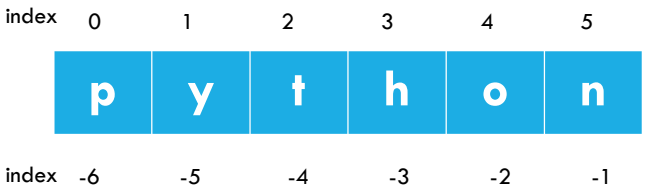
19

STRING

A='computer'

B='geography'

C='123'




```
Print(a[0])
Print(a[0:4])
Print(a[0:7:2])
```

20

STRING METHODS

capitalize()	Converts the first character to upper case
lower()	Converts a string into lower case
upper()	Converts a string into upper case
count()	Returns the number of times a specified value occurs in a string
index()	Searches the string for a specified value and returns the position of where it was found
replace()	Returns a string where a specified value is replaced with a specified value
join()	Joins the elements of an iterable to the end of the string

https://www.w3schools.com/python/python_strings_methods.asp

21

EXAMPLE

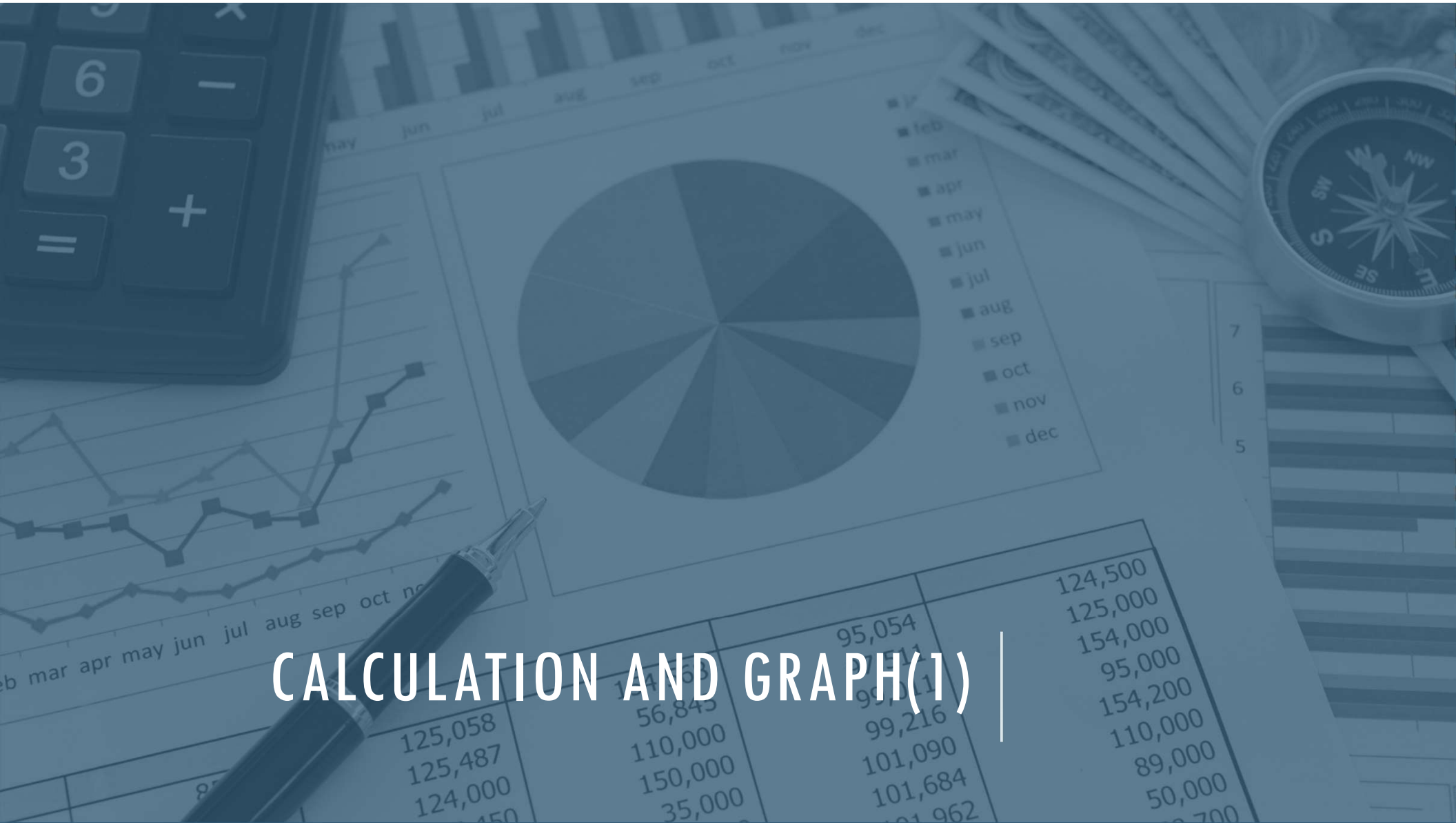
```
str1=input('input str1:')
str2=input('input str2:')
str1=set(str1)
str2=set(str2)
print('str1 same str2:', str1.intersection(str2))
```

22

EXAMPLE

```
book={1: 'GIS', 2: 'Remote', 3: 'Python'}
print("my book", len(book), "items")
for v in book.values():
    print(v)
print("Book list")
for d in book:
    print(d, book[d])
```

23



CALCULATION AND GRAPH(1)

LIST

```
A=[1,4,6,7]
M=A

A[0]=8
A
B
```

```
A=[1,4,6,7]
M=[x for x in A]
M
A[0]=8
A
B
```

LIST COMPREHENSIONS

Newlist= [expression(x) for x in old list]

A=list(range(0,10,2))

b = [x**2for x in A]

[0,4,16,36,64]

Import math

A=list(range(4,25,4))

B= [math.sqrt(x) for x in A]

A

B

LIST 2 D

A=[[1,2,3], [4,5,6], [7,8,9]]

A

A[0]

A[1]

A[2]

NUMPY

```
import numpy as np
A=[1,3,5,7,9]
n_a= np.array(A)
print(A)
print(n_a)
```

```
import numpy as np
a= np.array(range(1,10,2))
b= np.array([[2,4,6],[1,3,5]])
print(a)
print(b)
print(a.shape)
print(a.ndim)
```

NUMPY CREATING ARRAYS METHOD

From existing data

array(object[, dtype, copy, order, subok, ...])	Create an array.
asarray(a[, dtype, order, like])	Convert the input to an array.
asanyarray(a[, dtype, order, like])	Convert the input to an ndarray, but pass ndarray subclasses through.
ascontiguousarray(a[, dtype, like])	Return a contiguous array (ndim >= 1) in memory (C order).
asmatrix(data[, dtype])	Interpret the input as a matrix.
copy(a[, order, subok])	Return an array copy of the given object.
frombuffer(buffer[, dtype, count, offset, like])	Interpret a buffer as a 1-dimensional array.
from_dlpack(x, /)	Create a NumPy array from an object implementing the __dlpack__ protocol.
fromfile(file[, dtype, count, sep, offset, like])	Construct an array from data in a text or binary file.
fromfunction(function, shape, *[, dtype, like])	Construct an array by executing a function over each coordinate.
fromiter(iter, dtype[, count, like])	Create a new 1-dimensional array from an iterable object.
fromstring(string[, dtype, count, like])	A new 1-D array initialized from text data in a string.
loadtxt(fname[, dtype, comments, delimiter, ...])	Load data from a text file.

https://numpy.org/doc/stable/reference/routines.array-creation.html

NUMPY CREATING ARRAYS METHOD

From shape or value

empty(shape[, dtype, order, like])	Return a new array of given shape and type, without initializing entries.
empty_like(prototype[, dtype, order, subok, ...])	Return a new array with the same shape and type as a given array.
eye(N[, M, k, dtype, order, like])	Return a 2-D array with ones on the diagonal and zeros elsewhere.
identity(n[, dtype, like])	Return the identity array.
ones(shape[, dtype, order, like])	Return a new array of given shape and type, filled with ones.
ones_like(a[, dtype, order, subok, shape])	Return an array of ones with the same shape and type as a given array.
zeros(shape[, dtype, order, like])	Return a new array of given shape and type, filled with zeros.
zeros_like(a[, dtype, order, subok, shape])	Return an array of zeros with the same shape and type as a given array.
full(shape, fill_value[, dtype, order, like])	Return a new array of given shape and type, filled with *fill_value*.
full_like(a, fill_value[, dtype, order, ...])	Return a full array with the same shape and type as a given array.

https://numpy.org/doc/stable/reference/routines.array-creation.html

NUMPY CREATING ARRAYS METHOD

Numerical ranges

arange([start,] stop[, step,][, dtype, like])	Return evenly spaced values within a given interval.
linspace(start, stop[, num, endpoint, ...])	Return evenly spaced numbers over a specified interval.
logspace(start, stop[, num, endpoint, base, ...])	Return numbers spaced evenly on a log scale.
geomspace(start, stop[, num, endpoint, ...])	Return numbers spaced evenly on a log scale (a geometric progression).
meshgrid(*xi[, copy, sparse, indexing])	Return coordinate matrices from coordinate vectors.
mgrid	*nd_grid* instance which returns a dense multi-dimensional "meshgrid".
ogrid	*nd_grid* instance which returns an open multi-dimensional "meshgrid".

https://numpy.org/doc/stable/reference/routines.array-creation.html

NUMPY CREATING ARRAYS

```
import numpy as np
A = np.array([1, 2, 3, 4, 5])
print(A)
```

```
import numpy as np
A = np.array([1, 2, 3, 4, 5])
```

NUMPY CREATING ARRAYS METHOD

```
import numpy as np
a=np.ones(5)
print(a)

b=np.ones((5,), dtype=int)
print(b)
```

```
import numpy as np
a=np.ones((5,), dtype=int)
print(a)

b=np.full((2, 2), 10)
print(b)
```

NUMPY CREATING ARRAYS METHOD

```
import numpy as np
a=np.linspace(2.0, 3.0, num=5)
print(a)
```

```
b=np.eye(2,2, dtype=int)
print(b)
```

```
import  numpy as np
x = np.random.randint(10,size=(3,3))
print(x)
```

ACCESS ARRAY

```
b= np.array(([[2,4,6], [1, 3,5]]))
print(b[0][0])
print(b[0][1])
print(b[1][2])
```

```
import numpy as np
b=np.array(([[2,4,6],[1,3, 5]]))
print(b[0,0])
print(b[0,1])
print(b[1,2])
print(b[1][:])
print(b[1, :])
```

PROCESS ARRAY

```
import numpy as np
A=[1,3,5,7,9]
B= np.array([1,2,3,4,5])
print(A*2)
print(A+B)
```

ARRAY METHODS

Array methods

An `ndarray` object has many methods which operate on or with the array in some fashion, typically returning an array result. These methods are briefly explained below. (Each method's docstring has a more complete description.)

For the following methods there are also corresponding functions in `numpy`: `all`, `any`, `argmax`, `argmin`, `argpartition`, `argsort`, `choose`, `clip`, `compress`, `copy`, `cumprod`, `cumsum`, `diagonal`, `imag`, `max`, `mean`, `min`, `nonzero`, `partition`, `prod`, `ptp`, `put`, `ravel`, `real`, `repeat`, `reshape`, `round`, `searchsorted`, `sort`, `squeeze`, `std`, `sum`, `swapaxes`, `take`, `trace`, `transpose`, `var`.

Array conversion

`ndarray.item`(*args)	Copy an element of an array to a standard Python scalar and return it.
`ndarray.tolist`()	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
`ndarray.itemset`(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
`ndarray.tostring`([order])	A compatibility alias for *tobytes*, with exactly the same behavior.

https://numpy.org/doc/stable/reference/arrays.ndarray.html#array-methods

ARRAY METHODS

```
import numpy as np

a = np.arange(6).reshape((3, 2))

print(a)
```

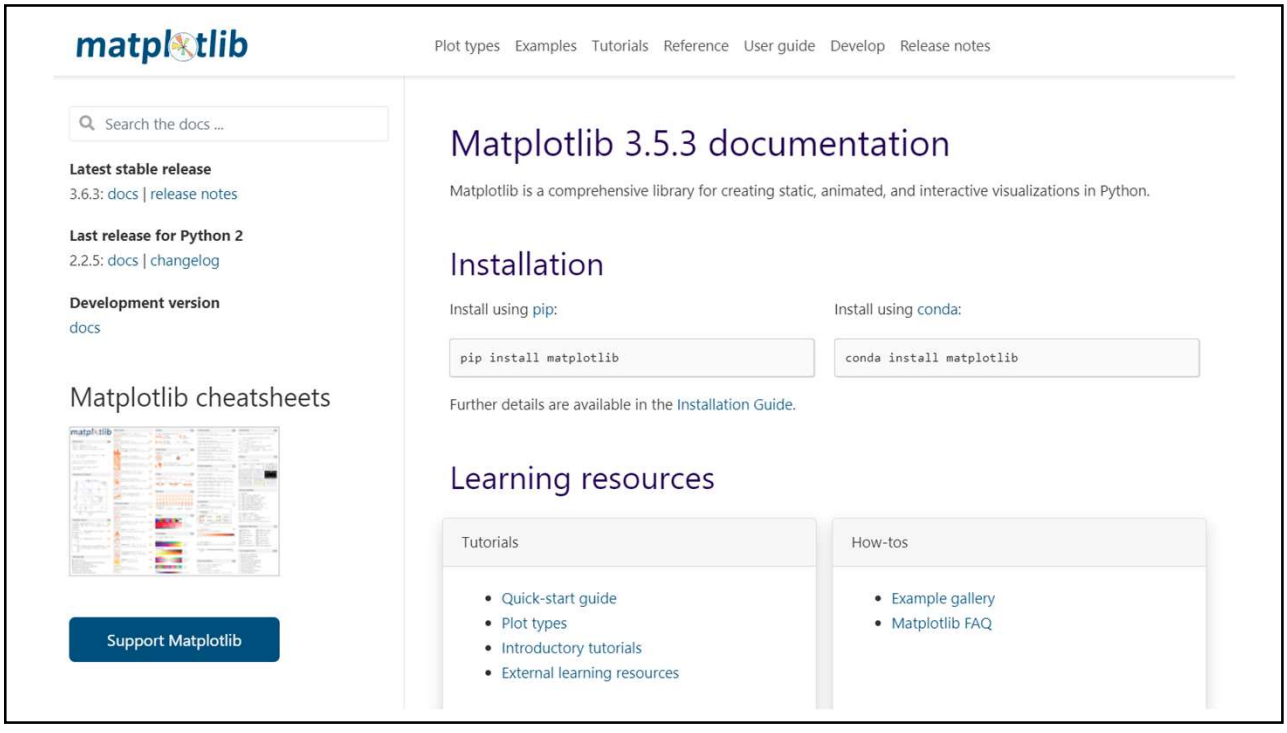
```
print("sum a =", a.sum())

print("mean a =", a.mean())

print("min a =", a.min())

print("std a =", a.std())
```

ARRAY METHODS

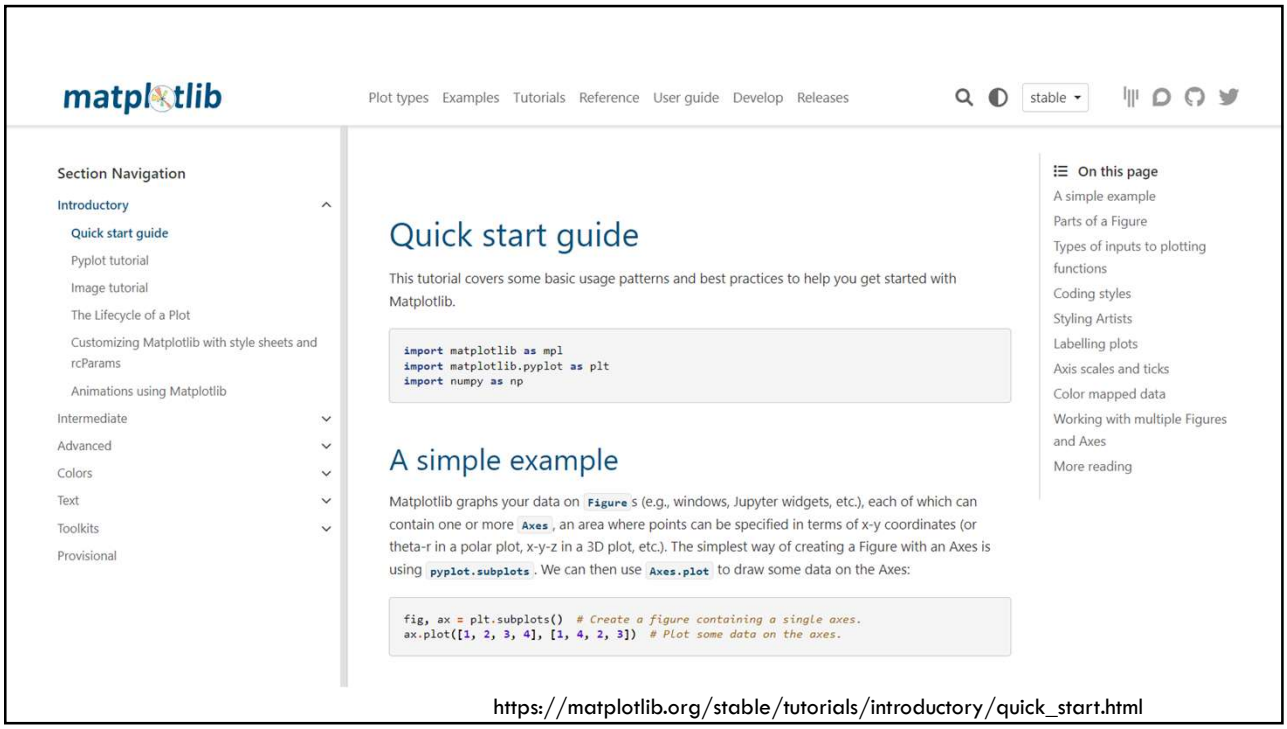
```
import numpy as np
A= np.random.randint(1,10,20)
print(A)
print(np.sort(A))
A=A.reshape(4,5)
print("sum column= ", A.sum(axis=0))
print("sum row= ", A.sum(axis=1))
```

CALCULATION AND GRAPH(2)

17



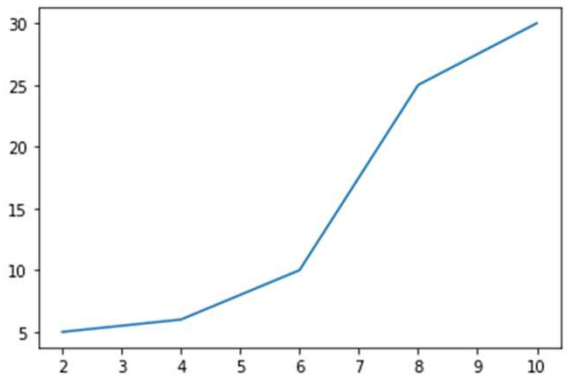
18



https://matplotlib.org/stable/tutorials/introductory/quick_start.html

19

```
import matplotlib.pyplot
 as pt
x=[2,4,6,8,10]
y=[5,6,10,25,30]
pt.plot(x,y)
pt.show()
```



20

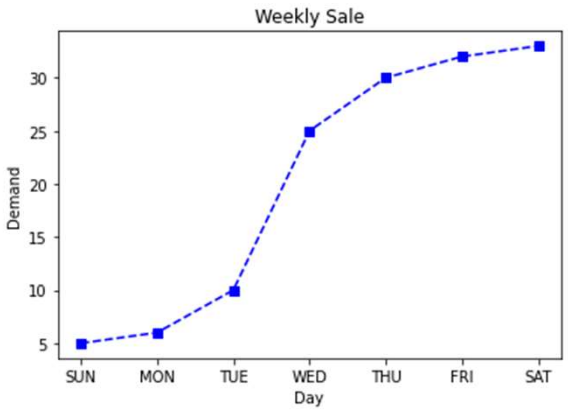
Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker

https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.plot.html

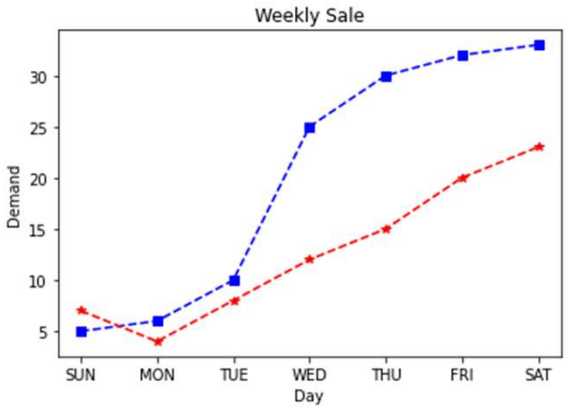
21

```
import matplotlib.pyplot as pt
x=['SUN','MON','TUE','WED','THU'
,'FRI','SAT']
y=[5,6,10,25,30,32,33]
pt.plot(x,y1, 's--b')
pt.title("Weekly Sale")
pt.xlabel("Day")
pt.ylabel("Demand")
pt.show()
```



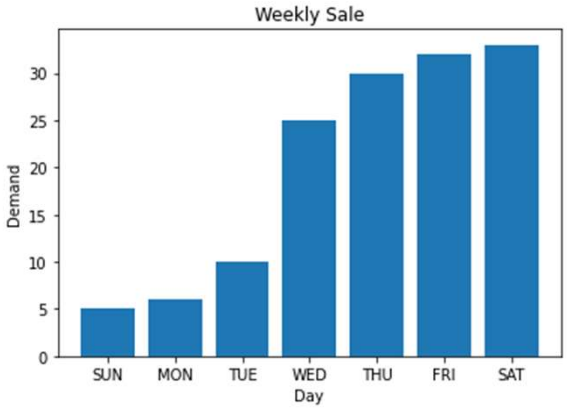
22

```
import matplotlib.pyplot as pt
x=['SUN','MON','TUE','WED','THU','FRI',
'SAT']
y1=[5,6,10,25,30,32,33]
y2=[7,4,8,12,15,20,23]
pt.plot(x,y1, 's--b')
pt.plot(x,y2, '*--r')
pt.title("Weekly Sale")
pt.xlabel("Day")
pt.ylabel("Demand")
pt.show()
```



23

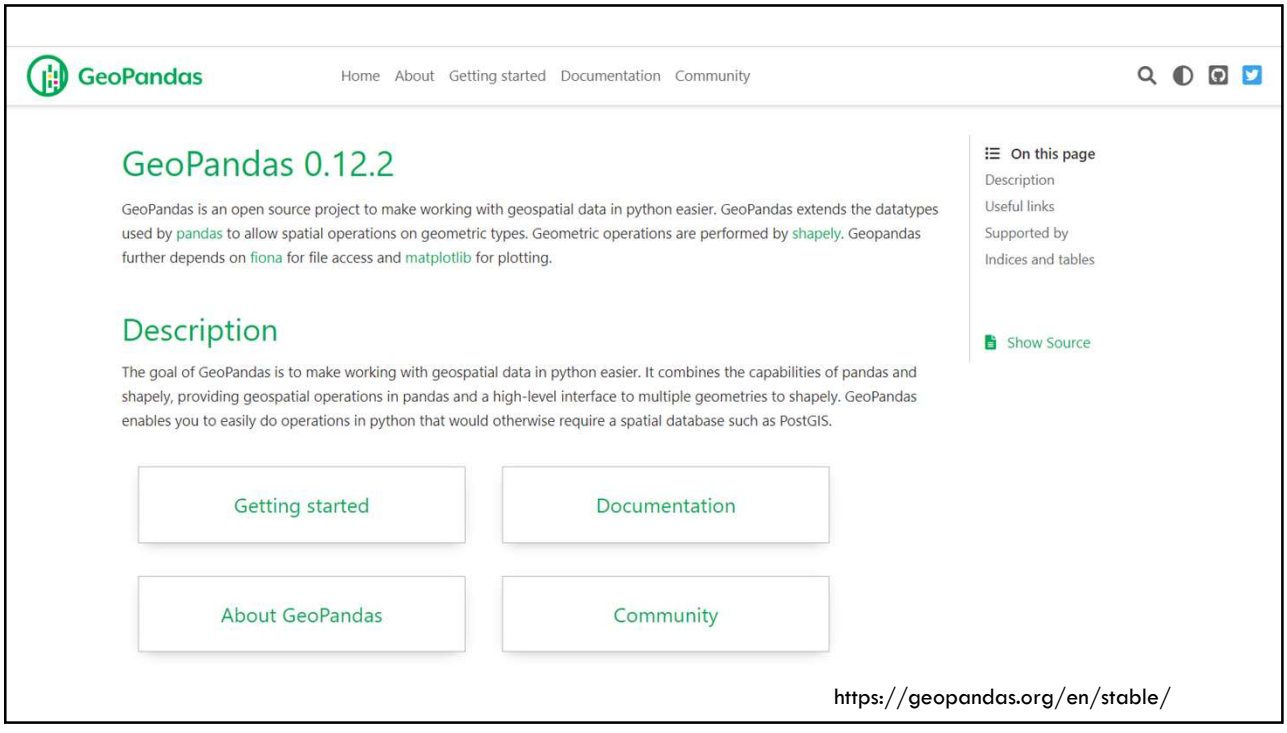
```
import matplotlib.pyplot as pt
x=['SUN','MON','TUE','WED','THU'
,'FRI','SAT']
y=[5,6,10,25,30,32,33]
pt.bar(x,y)
pt.title("Weekly Sale")
pt.xlabel("Day")
pt.ylabel("Demand")
pt.show()
```



24

GEOPANDAS

25



26

INSTALL GEOPANDAS

```
!pip install git+git://github.com/geopandas/geopandas.git
!pip install pyproj
!pip install Shapely
!pip install pysal
!pip install descartes
!pip install geopy
!pip install geojson
```

27

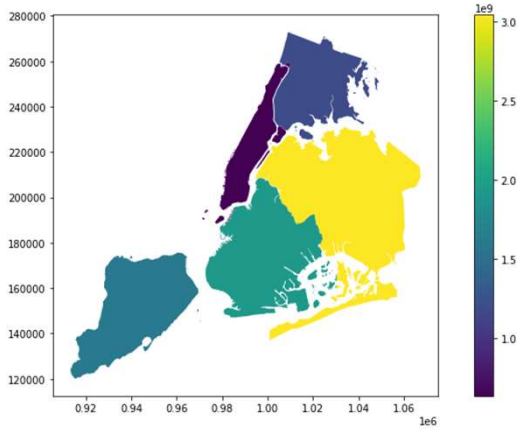
```
import geopandas

path_to_data = geopandas.datasets.get_path("nybb")
gdf = geopandas.read_file(path_to_data)
```

gdf

	BoroCode	BoroName	Shape_Leng	Shape_Area	geometry
0	5	Staten Island	330470.010332	1.623820e+09	MULTIPOLYGON (((970217.022 145643.332, 970227....
1	4	Queens	896344.047763	3.045213e+09	MULTIPOLYGON (((1029606.077 156073.814, 102957...
2	3	Brooklyn	741080.523166	1.937479e+09	MULTIPOLYGON (((1021176.479 151374.797, 102100...
3	1	Manhattan	359299.096471	6.364715e+08	MULTIPOLYGON (((981219.056 188655.316, 980940....
4	2	Bronx	464392.991824	1.186925e+09	MULTIPOLYGON (((1012821.806 229228.265, 101278...

28

```
gdf.plot("Shape_Area", legend=True)
```



29

```
gdf.plot("Shape_Area", legend=True)
```

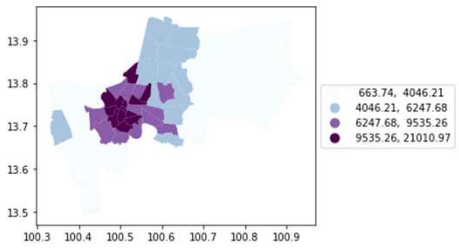
30

GEOPANDAS2

31

CHORO LEGENDS

```
import geopandas
zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"
map_bkk=geopandas.read_file(zipfile)
map_bkk.plot(column="Density", scheme='QUANTILES', k=4, \
             cmap='BuPu', legend=True,
             legend_kwds={'loc': 'center left', 'bbox_to_anchor':(1,0.5)})
```

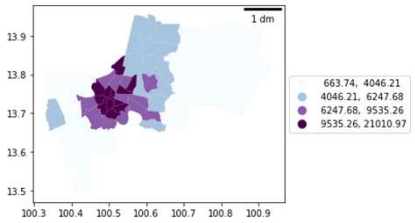


32

ADDING A SCALE BAR TO A MATPLOTLIB PLOT

pip install matplotlib-scalebar

```
import geopandas as gpd
from matplotlib_scalebar.scalebar import ScaleBar
zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"
map_bkk=gpd.read_file(zipfile)

mp=map_bkk.plot(column="Density", scheme='QUANTILES', k=4, \
            cmap='BuPu', legend=True,
            legend_kwds={'loc': 'center left', 'bbox_to_anchor':
(1,0.5)})
mp.add_artist(ScaleBar(1))
```

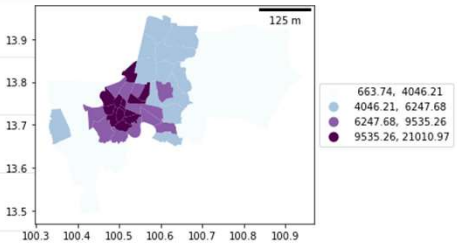


33

SCALE BAR UNIT

mp.add_artist(ScaleBar(1,dimension="si-length", units="km"))

dimension	units
si-length	km, m, cm, um
imperial-length	in, ft, yd, mi
si-length-reciprocal	1/m, 1/cm
angle	deg



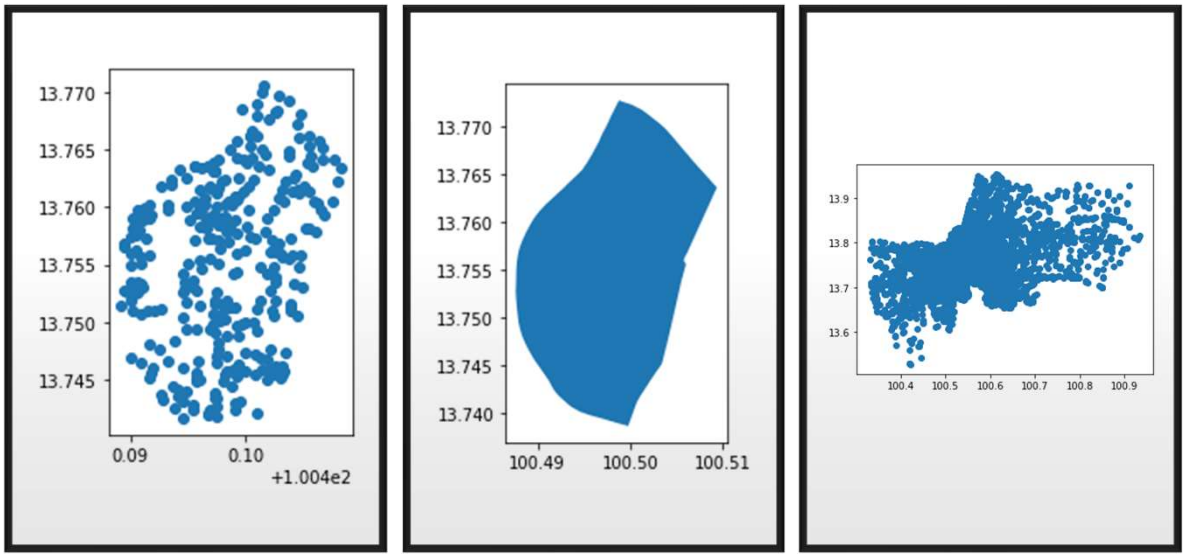
34

CLIP

```
import matplotlib.pyplot as plt
import geopandas as gpd
from shapely.geometry import Polygon

zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"
map_bkk=gpd.read_file(zipfile)

PHRA_NAKHON=map_bkk[map_bkk['AMP_NAME']=="PHRA NAKHON"]
PHRA_NAKHON.plot()
```

```
ld_zipfile= "/content/sample_data/landmark_BKK.zip"
ld_map=gpd.read_file(ld_zipfile)
ld_map.plot()

ld_clip=ld_map.clip(PHRA_NAKHON)
ld_clip.plot()
ld_clip
```

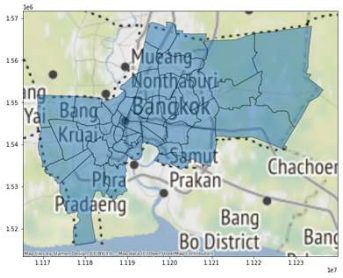
35

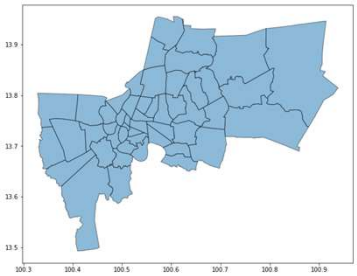


36

ADDING A BACKGROUND MAP TO PLOTS

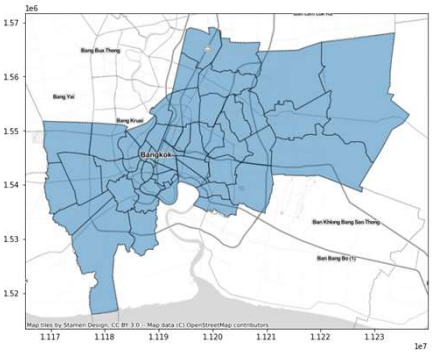
```
import geopandas

import contextily as cx

zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"

map_bkk=gpd.read_file(zipfile)

Layout = map_bkk.plot(figsize=(10, 10), alpha=0.5, edgecolor='k
')

map_bkk.crs



map_bkk_wm=map_bkk.to_crs(epsg=3857)

Layout=map_bkk_wm.plot(figsize=(10, 10), alpha=0.5, edgecolor='
k')

cx.add_basemap(Layout, zoom=9)
```



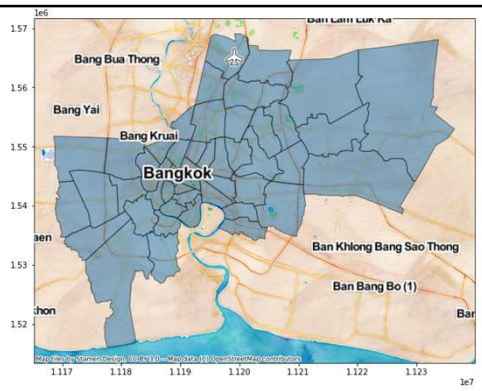


37

```
import geopandas
import contextily as cx
zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"
map_bkk=gpd.read_file(zipfile)
Layout = map_bkk.plot(figsize=(10, 10), alpha=0.5, edgecolor='k')
map_bkk.crs


map_bkk_wm=map_bkk.to_crs(epsg=3857)
Layout=map_bkk_wm.plot(figsize=(10, 10), alpha=0.5, edgecolor='k')
cx.add_basemap(Layout, source=cx.providers.Stamen.TonerLite)
cx.add_basemap(Layout, source=cx.providers.Stamen.TonerLabels)
```



38




```
import geopandas
import contextily as cx
zipfile = "/content/sample_data/Admin_Amphoe_BKK.zip"
map_bkk=gpd.read_file(zipfile)
Layout = map_bkk.plot(figsize=(10, 10), alpha=0.5, edgecolor='k')
map_bkk.crs
```

```
map_bkk_wm=map_bkk.to_crs(epsg=3857)
Layout=map_bkk_wm.plot(figsize=(10, 10), alpha=0.5, edgecolor='k')
cx.add_basemap(Layout, source=cx.providers.Stamen.Watercolor, zoom=12)
cx.add_basemap(Layout, source=cx.providers.Stamen.TonerLabels, zoom=10)
```

39