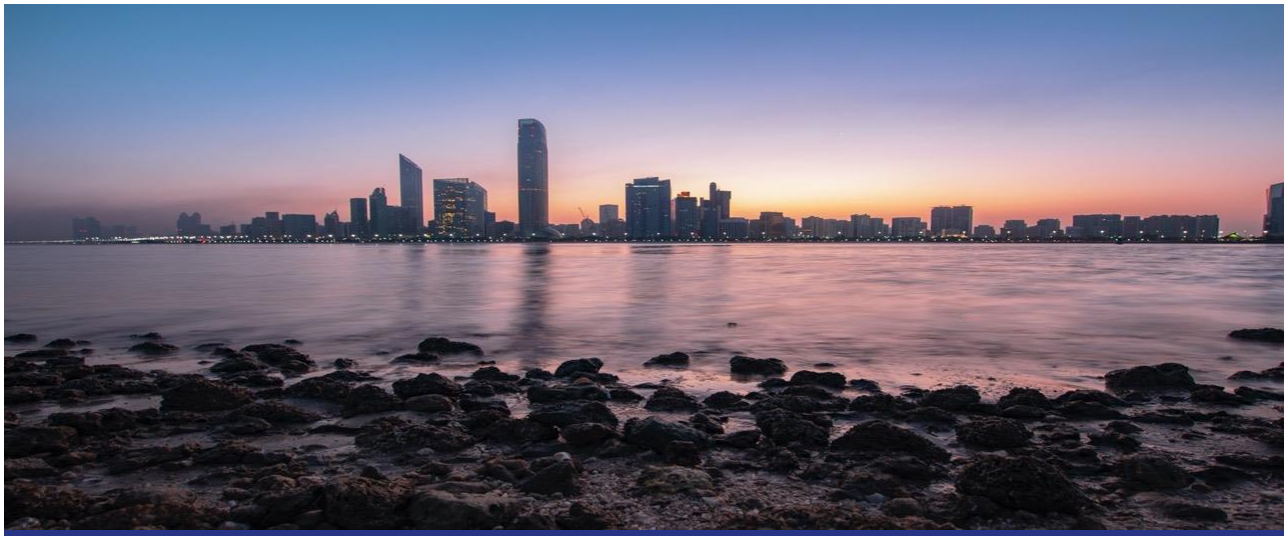


Scripting and Programming

31st October – 1st November 2023

1

Course Agenda

01 Introduction to Scripting and Programming	02 Basic Syntax and Data Types Control Structures	03 Functions and Modules

• What is Scripting?
• What is Programming?
• The history of programming
	• Basic data types
	• Logical Expressions
	  ○True and False
	  ○Boolean Logic Expressions
	  ○Special Properties of and & or
	  ○Conditional Expressions
	• Control of flow
	• While loops
	• Break and continue
	• Assert
	• For Loops / List Comprehensions
	• For loops and the range() function
	• For Loops and Dictionaries
	• String Operations
		• Functions
		• Importing and Modules
		• Commonly Used Modules

2

Introduction to Scripting and Programming

Learning Outcome 1

3

Scripting

Scripting is the act of creating scripts, which are sequences of instructions executed by computers. Scripts, typically written in scripting languages, automate tasks, control software, and perform operations. Scripting languages are often interpreted and are known for their flexibility and quick execution. Common examples include JavaScript for web development and Python for automation.

4

Programming

Programming involves creating computer programs using programming languages. It encompasses designing, coding, testing, and maintaining software applications. Programming languages provide a means to communicate with computers, instruct them, and solve problems. Programmers use algorithms and data structures to achieve specific goals through software development.

5

Programming History

- The history of programming traces back to the early days of computing and includes significant milestones such as:
- Charles Babbage's invention of the first programmable computer in the 19th century.
- The development of assembly language and machine code programming in the mid-20th century.
- The creation of high-level programming languages like Fortran, COBOL, and Lisp in the 1950s and 1960s.
- The emergence of modern programming languages like C, C++, Java, and Python in the late 20th century.
- The rise of the World Wide Web and web programming languages like JavaScript and PHP in the 1990s.

6

Basic Syntax and Data Types Control Structures

Learning Outcome 2

7

Basic Data Types

- Integers (default for numbers)

  ```
  z = 5 / 2   # Answer is 2, integer division.
  ```

- Floats

  ```
  x = 3.456
  ```

- Strings
  - Can use "" or '' to specify.

    ```
    "abc"  'abc'  (Same thing.)
    ```

  - Unmatched can occur within the string.

    ```
    "matt's"
    ```

  - Use triple double-quotes for multi-line strings or strings than contain both ' and " inside of them:

    ```
    """a'b"c"""
    ```

8

Logical Expressions - True and False

- True and False are constants in Python.
- Other values equivalent to True and False:
  - False: zero, None, empty container or object
  - True: non-zero numbers, non-empty objects
- Comparison operators: ==, !=, <, <=, etc.
  - X and Y have same value: X == Y
  - Compare with X is Y :
    - X and Y are two variables that refer to the *identical same object*.

9

Logical Expressions - Boolean Logic Expressions

- You can also combine Boolean expressions.
  - True if a is True and b is True: a and b
  - True if a is True or b is True: a or b
  - True if a is False: not a
- Use parentheses as needed to disambiguate complex Boolean expressions.
- Actually, evaluation of expressions is *lazy*…

10

Logical Expressions - Special Properties of and & or

- Actually and and or *don't* return True or False.
- They return the value of one of their sub-expressions (which may be a non-Boolean value).
- X and Y and Z
  - If all are true, returns value of Z.
  - Otherwise, returns value of first false sub-expression.
- X or Y or Z
  - If all are false, returns value of Z.
  - Otherwise, returns value of first true sub-expression.
- and and or use *lazy evaluation*, so no further expressions are evaluated

11

Logical Expressions - Conditional Expressions

```
x = true_value if condition else false_value
```

- Evaluation rule:
  - First, condition is evaluated.
  - If *True*, true_value is evaluated and returned.
  - If *False*, false_value is evaluated and returned.

12

Control of Flow

if Statements

```
if x == 3:
    print "X equals 3."
elif x == 2:
    print "X equals 2."
else:
    print "X equals something else."
print "This is outside the 'if'."
```

Be careful! The keyword if is also used in the syntax of filtered list comprehensions.

Note:

- Use of indentation for blocks
- Colon (:) after boolean expression

13

While Loops

```
>>> x = 3
>>> while x < 5:
    print x, "still in the loop"
    x = x + 1
3 still in the loop
4 still in the loop
>>> x = 6
>>> while x < 5:
    print x, "still in the loop"
>>>
```

14

break and continue

- You can use the keyword *break* inside a loop to leave the *while* loop entirely.
- You can use the keyword *continue* inside a loop to stop processing the current iteration of the loop and to immediately go on to the next one.

15

assert

- **An *assert* statement will check to make sure that something is true during the course of a program.**
  - If the condition if false, the program stops

    (more accurately: the program throws an exception)

```
assert(number_of_players < 5)
```

16

Exercise 1: Deposit/Loan Income calculator

- Description
  - Build a financial calculator that helps users estimate their income from deposits or loans.

17

Exercise 2: Exam Score Calculator

- Description
  - Develop a program to calculate exam scores, grade averages, or determine pass/fail status.

18

For Loops / List Comprehensions

- Python's list comprehensions provide a natural idiom that usually requires a for-loop in other programming languages.
  - As a result, Python code uses many fewer for-loops
  - Nevertheless, it's important to learn about for-loops.
- *Caveat*! The keywords *for* and *in* are also used in the syntax of list comprehensions, but this is a totally different construction.

19

For Loops 1

- For-each is Python's *only* for construction
- A for loop steps through each of the items in a collection type, or any other type of object which is "iterable"

```
for <item> in <collection>:
  <statements>
```

- If <collection> is a list or a tuple, then the loop steps through each element of the sequence.
- If <collection> is a string, then the loop steps through each character of the string.

```
for someChar in "Hello World":
  print someChar
```

20

For loops and the range() function

- Since a variable often ranges over some sequence of numbers, the range() function returns a list of numbers from 0 up to but not including the number we pass to it.
- range(5) returns [0,1,2,3,4]
- So we could say:
  for x in range(5):
    print x
- (There are more complex forms of range() that provide richer functionality…)
- xrange() returns an iterator that provides the same functionality here more efficiently

21

For Loops and Dictionaries

```
>>> ages = { "Sam" : 4, "Mary": 3, "Bill": 2 }
>>> ages
{'Bill': 2, 'Mary': 3, 'Sam': 4}
>>> for name in ages.keys():
        print name, ages[name]
Bill 2
Mary 3
Sam 4
>>> for (name, age) in ages.items():
  print name, age
```

22

String Operations

- A number of methods for the string class perform useful formatting operations:

```
>>> "hello".upper()

'HELLO'
```

- Check the Python documentation for many other handy string operations.
- Helpful hint: use <string>.strip() to strip off final newlines from lines read from files

23

24

Functions

- The syntax for a function call is:

```
>>> def myfun(x, y):
        return x * y
>>> myfun(3, 4)
12
```

- Parameters are passed by value (“Call by Value”)
  - Each argument expression is evaluated and the resulting value is bound to the corresponding variable in the function (which is a new one for each invocation)
  - All assignment in Python, including binding function parameters, use reference semantics, since values are all object references.
  - Many web discussions on this are simply confused and call it “call by assignment” or “call by sharing”.

25

Functions

- All functions in Python have a return value
  - even if no *return* line inside the code.
- Functions without a *return* return the special value *None*.
  - None is a special constant in the language.
  - None is used like NULL, void, or nil in other languages.
  - None is also logically equivalent to False.
  - The interpreter doesn't print None

26

Functions

- Functions can be used as any other data type
- They can be
  - Arguments to function
  - Return values of functions
  - Assigned to variables
  - Parts of tuples, lists, etc

```
>>> def myfun(x):
    return x*3
>>> def apply(q, x):
    return q(x)
>>> apply(myfun, 7)
21
```

27

Exercise 3: # Coffee Mugs to Cover Expenses

- Description
  - Create a financial planner to calculate how many coffee mugs need to be sold to cover business expenses of a coffeeshop.

28

Exercise 4: Financial Budget

- Description
  - Build a budgeting tool that allows users to track income, expenses, and savings goals.

29

Importing and Modules

- Use classes & functions defined in another file.
- A Python module is a single file with the same name (plus the *.py* extension)
- Like Java *import*

  Where does Python look for module files?

  - The list of directories where Python looks: sys.path
  - To add a directory of your own to this list, append it to this list.

    sys.path.append('/my/new/path')

30

Import (1)

import somefile

- Everything in somefile.py can be referred to by:

somefile.className.method("abc")

somefile.myFunction(34)

- from somefile import *
  - Everything in somefile.py can be referred to by:

className.method("abc")

myFunction(34)

- Caveat! This can easily overwrite the definition of an existing function or variable!

31

Commonly Used Modules

- **os**: This module provides functions for interacting with the operating system, such as opening and closing files, creating and deleting directories, and managing processes.
- **sys**: This module provides information about the Python system, such as the version of Python being used, the operating system, and the environment variables.
- **math**: This module provides mathematical functions, such as trigonometric functions, exponential functions, and statistical functions.
- **re**: This module provides functions for working with regular expressions, which are patterns that can be used to search, edit, or manipulate text.
- **random**: This module provides functions for generating random numbers and sequences.

32

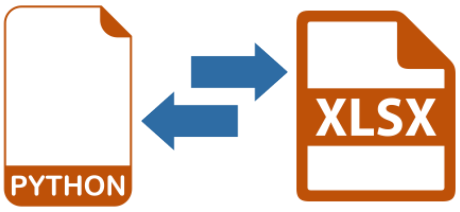
Commonly Used Modules

- **numpy**: This module provides functions for working with numerical data, such as arrays and matrices.
- **pandas**: This module provides a high-level data structure for working with tabular data, such as data frames and series.
- **matplotlib**: This module provides functions for creating data visualizations, such as charts, plots, and histograms.
- **requests**: This module provides functions for making HTTP requests.
- **flask**: This module is a lightweight web framework for building Python web applications.
- **django**: This module is a full-featured web framework for building Python web applications.

33

Exercise 5: Data Generator from Excel

- Description
  - Develop a script to extract and manipulate data from Excel spreadsheets.



34

Exercise 6: Age Calculator from ID Number.

- Description
  - Create a program that calculates a person's age based on their identification number, such as a national ID.

35

Exercise 7: Automated Banking Report

- Description
  - Design a tool for automating the generation of banking reports, including account balances and transaction history.

36

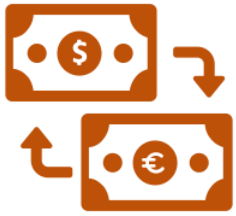
Exercise 8: Medical Report

- Description
  - Build a system for generating medical reports based on patient data, diagnoses, and treatment information.

37

Exercise 9: Currency Converter

- Description
  - Develop a tool that converts currency values between different exchange rates.



38