SUSS
SINGAPORE UNIVERSITY
OF SOCIAL SCIENCES

ANL251
Python Programming

STUDY GUIDE

School of Business

Course Development Team

Head of Programme	: Dr James Tan Swee Chuan
Course Developer(s)	: Dr Xiao Shu
Technical Writer	: Lynn Lim, ETP
Video Production	: Mohd Jufrie Bin Ramli, ETP

ISBN 978-981-478-747-5

Educational Technology & Production
Singapore University of Social Sciences
463 Clementi Road
Singapore 599494

Release V1.1

Build S1.0.5, T1.5.21

Table of Contents

Course Guide

Study Unit 1: Python Basics

Study Unit 2: Control Flow and Lists

Study Unit 3: Tuples and Dictionaries

Study Unit 4: Functions, Methods and Packages

Study Unit 5: Scientific Computing and Plotting with Python

Study Unit 6: Data Analysis with Python

List of Tables

List of Figures

Course Guide

Python Programming

1. Welcome

Presenter: Dr Xiao Shu

This streaming video requires Internet connection.
Access it via Wi-Fi to avoid incurring data charges on your personal mobile plan.

Click here to watch the video. [i]

Welcome to the course *ANL251 Python Programming*, a 5 credit unit (CU) course.

This Study Guide will be your personal learning resource to take you through the course learning journey. The guide is divided into two main sections – the Course Guide and Study Units.

The Course Guide describes the structure for the entire course and provides you with an overview of the Study Units. It serves as a roadmap of the different learning components within the course. This Course Guide contains important information regarding the course learning outcomes, learning materials and resources, assessment breakdown and additional course information.

[i] https://d2jifwt31jjehd.cloudfront.net/ANL251/IntroVideo/ANL251_Intro_Video.mp4

2. Course Description and Aims

This course provides you with fundamental knowledge and skills to use the Python programming language to write programs for problem solving. At the end of this course, students will be competent in using Python to perform tasks such as data acquiring, manipulation, visualization and analysis. Since this course is designed to help students with little prior exposure to programming, it will focus on breadth rather than depth.

Course Structure

This course is a 5-credit unit course presented over 6 weeks.

There are six Study Units in this course. The following provides an overview of each Study Unit.

Study Unit 1 – Python Basics

This unit takes the first steps Python programming, including variables, data types, operators, formatted printing and user input.

Study Unit 2 – Control Flow and Lists

This unit will focus on the two types of control flow to dynamically change how our Python program behaves, the conditional statement and loops. This unit also starts to introduce the compound built-in data type in Python, lists.

Study Unit 3 – Tuples and Dictionaries

This unit continues to introduce another two types of compound built-in data types in Python, tuples and dictionaries.

Study Unit 4 – Functions, Methods and Packages

This unit covers the import mechanisms in Python for reuse, distribute and manage code.

Study Unit 5 – Scientific Computing and Plotting with Python

This unit introduces two important fundamental packages – the NumPy package to efficiently store and do calculations with huge amounts of data, and the matplotlib package to create customized plots.

Study Unit 6 – Data Analysis with Python

This unit introduces the Pandas package, the key data structure and data analysis tools for Python.

3. Learning Outcomes

Knowledge & Understanding (Theory Component)

By the end of this course, you should be able to:

- Describe and compare the basic data types, operators, strings, lists, tuples and dictionaries in Python.
- Discuss the concepts of functions, methods, packages and modules, and how Python manages and imports packages/modules
- Explain the vectorized operations on NumPy arrays

Key Skills (Practical Component)

By the end of this course, you should be able to:

- Write Python programs to apply appropriate programming concepts, Python built-in functions, user-defined functions and standard libraries for problem solving.
- Develop logic control flows in Python programs.
- Prepare and analyse datasets using NumPy and pandas.
- Construct customized plots using matplotlib.

4. Learning Material

The following is a list of the required learning materials to complete this course.

Required Textbook(s)

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

"Register your textbook and then download the included videos at informit.com/title/9780134692883."

Other recommended study material (Optional)

The following learning materials may be required to complete the learning activities:

Website(s):

The Python documentation. (n.d.). Retrieved from https://docs.python.org/3/

NumPy User Guide. (n.d.). Retrieved from https://docs.scipy.org/doc/numpy/user/index.html

The Pyplot API. (n.d.). Retrieved from https://matplotlib.org/api/pyplot_summary.html

pandas documentation. (n.d.). Retrieved from http://pandas.pydata.org/pandas-docs/stable/

5. Assessment Overview

The overall assessment weighting for this course is as follows:

Assessment	Description	Weight Allocation
PCOQ	Pre-Course Quiz	2%
PCQ1	Pre-Class Quiz 1	2%
PCQ2	Pre-Class Quiz 2	2%
PCT	Participation	6%
GBA	Group-Based Assignment	19%
TMA	Tutor-Marked Assignment	19%
ECA	End-of-Course Assignment	50%
TOTAL		100%

The following section provides important information regarding Assessments.

Continuous Assessment:

There will be continuous assessment in the form of participation, three quizzes, one group-based assignment and one tutor-marked assignment. In total, this continuous assessment will constitute 50 percent of overall student assessment for this course.

End-of-Course Assignment:

The end-of-course assignment will constitute the other 50 percent of overall student assessment. All topics covered in the course outline will be covered.

Passing Mark:

To successfully pass the course, you must obtain a minimum passing mark of 40 percent for the continuous assessment. That is, students must obtain at least a mark of 40 percent for the combined assessments and also at least a mark of 40 percent for the final exam. For detailed information on the Course grading policy, please refer to The Student Handbook ('Award of Grades' section under Assessment and Examination Regulations). The Student Handbook is available from the Student Portal.

6. Course Schedule

To help monitor your study progress, you should pay special attention to your Course Schedule. It contains study unit related activities including quizzes and assignments. Please refer to the Course Timetable in the Student Portal for the updated Course Schedule.

Note: You should always make it a point to check the Student Portal for any announcements and latest updates.

7. Learning Mode

The learning process for this course is structured along the following lines of learning:

a. Self-study guided by the study guide units. Independent study will require ***at least 3 hours per week.***

b. Working on assignments, either individually or in groups.

c. Classroom Seminar sessions (3 hours each session, 6 sessions in total).

iStudyGuide

You may be viewing the iStudyGuide version, which is the mobile version of the Study Guide. The iStudyGuide is developed to enhance your learning experience with interactive learning activities and engaging multimedia. Depending on the reader you are using to view the iStudyGuide, you will be able to personalise your learning with digital bookmarks, note-taking and highlight sections of the guide.

Interaction with Instructor and Fellow Students

Although flexible learning – learning at your own pace, space and time – is a hallmark at SUSS, you are encouraged to actively participate the seminar sessions. Sharing of ideas through meaningful debates will help broaden your learning and crystallise your thinking.

Academic Integrity

As a student of SUSS, it is expected that you adhere to the academic standards stipulated in The Student Handbook, which contains important information regarding academic policies, academic integrity and course administration. It is necessary that you read and understand the information stipulated in the Student Handbook, prior to embarking on the course.

Study Unit 1

Python Basics

Learning Outcomes

By the end of this unit, you should be able to:

1. Execute Python program in the Python interpreter or the PowerShell/Terminal command line
2. Use comments in Python scripts
3. Solve problems using Python scripts with appropriate variable names, types and operations
4. Construct formatted printing using format strings, the format() method and escape sequences
5. Create user input and implement appropriate operations based on the input

Overview

This study unit describes your first steps to Python programming. We will learn ways to write and execute Python programs. In order to compose Python programs, we must understand how to create variables in the various data types, so that we can store, retrieve and calculate information. Input and output are important to programing. We learn how to create user input, implement appropriate operations on the input, and construct formatted printing as well.

Chapter 1: Python Programming Environment

We will need to install two software, Python 3 and Atom. Atom is to be used to edit the Python program.

Note: We are learning Python 3 in this course. When going through Python Documentation, make sure that you are reading about Python 3.x, and not Python 2.x.

1.1 Installing Python 3

Download the latest Python 3 version at this URL https://www.python.org/downloads/. Note that when installing it on Windows, be sure to check the box that adds Python to PATH as in Figure 1.1.

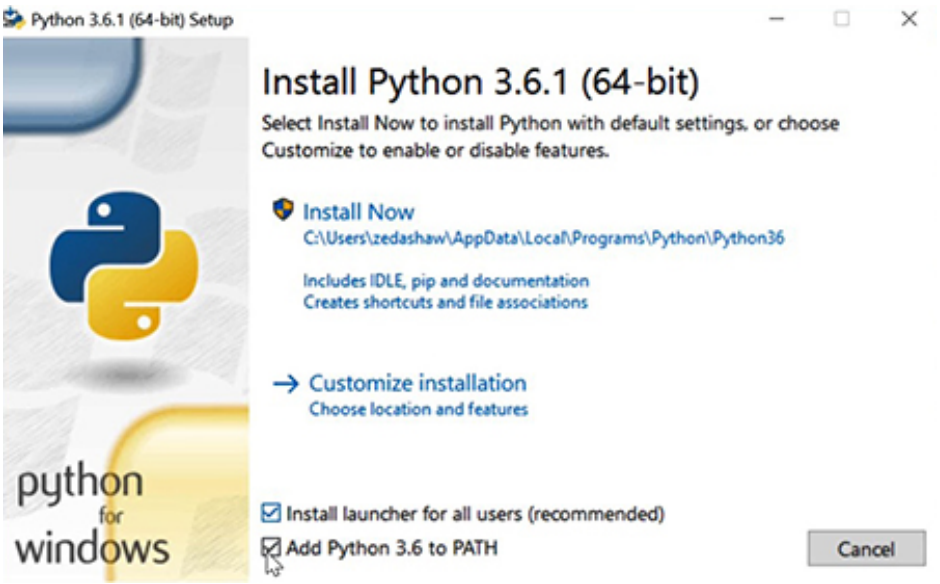

Figure 1.1 Installing Python 3 on windows: check the box to add Python to PATH

(Source: screenshot from the textbook video for Exercise 0: The Setup (Windows))

Watch

Watch the following two videos of the textbook for step-by-step instructions on installing Python 3:

Video for Exercise 0 The Setup (Windows)

Video for Exercise 0 The Setup (macOS)

Read

Read the following two sections of the textbook on installing Python 3 on mac OS or Windows:

Exercise 0 The Setup (Windows)

Exercise 0 The Setup (maxOS)

1.2 Python Interpreter

Let's start with the Python interpreter, where you can type Python code and immediately see the output. On Windows, you may follow the steps below to open Python interpreter and run simple Python code in it.

1. Search for PowerShell from the Start menu on Windows. Press Enter to run it.
2. Type python at the prompt.
3. Type 2 + 7 and hit Enter. Python interprets what you typed and prints the result 9.
4. Type quit() and press Enter to exit Python shell. You will return to PowerShell at a prompt similar to what you had before you run Python.

Note: If you don't use Windows, Step 1 will be different. Refer to the textbook video Exercise 0: The Setup (macOS) about opening Python shell in terminal.

1.3 Python Scripts

Apart from interactively working with the Python interpreter, we can also have Python to run scripts. These scripts are simply text files with the extension .py, in which we put all the Python codes to be executed in batch, whereas in contrast in the interpreter we type and execute the code line by line.

We will use Atom as the editor to compose Python scripts, and then execute the scripts using python command in the PowerShell as in Figure 1.2. Download Atom text editor at https://atom.io and then install it.

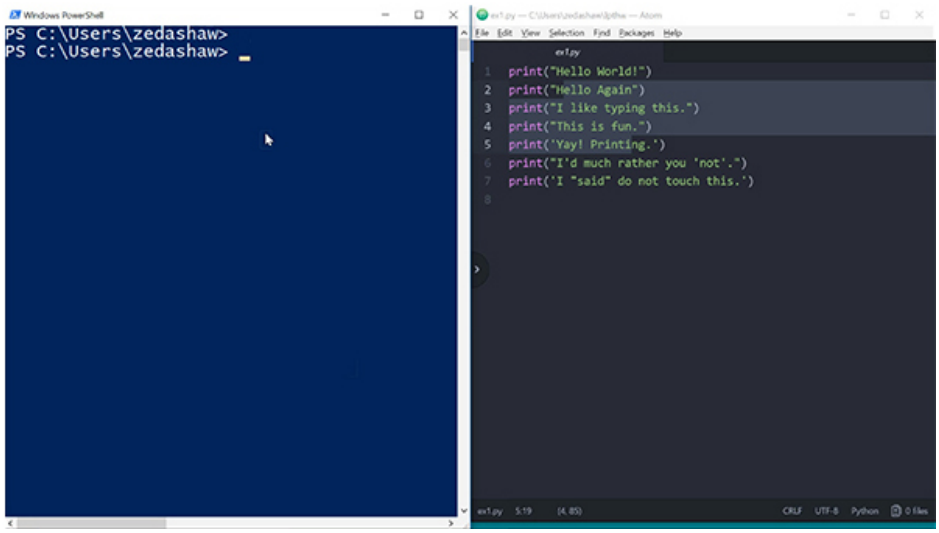

Figure 1.2 Composing scripts in Atom and Executing in PowerShell

(Source: screenshot from the textbook video for Exercise 1: A Good First Program)

Follow the steps below to edit and run your Python script.

1. Let's put the Python command in a script using Atom text editor. Save your script with the extension .py. Different from Python interpreter, you have to explicitly use print() inside scripts if you want to generate output during execution of Python scripts.

2. In the PowerShell, change into the directory where you save the Python script in Step 1.
3. Run the script by executing the command. The same output as before in the Python interpreter is generated.

Watch

Watch the following video of the textbook for examples of composing and executing Python scripts:

Video for Exercise 1 A Good First Program (Windows)

Video for Exercise 1 A Good First Program (macOS)

Read

Read the following section of the textbook on examples of composing and executing Python scripts:

Exercise 1 A Good First Program (Windows)

Exercise 1 A Good First Program (macOS)

Note: You've learned two ways of working with Python: use the Python interpreter for experimentation, and use the Python script editor to compose the program to avoid retyping everything over and over again if you want to make a change. You simply make and save the change to the script in Atom, and rerun Step 3.

1.4 Comments

Comments are disabled parts of your program, i.e. those lines of comments are not part of the executable program. Comments in your programs are usually used to tell information to whoever read your program. Comments in Python program are those lines starting with a pound character # as in Figure 1.3:

```
# A comment, this is so you can read your program later.
# Anything after the # is ignored by python.

print("I could have code like this.") # and the comment after is ignored

# You can also use a comment to "disable" or comment out code:
# print("This won't run.")

print("This will run.")
print("Hi # there.")
```

Figure 1.3 Comments in Python scripts

Note: The # in the last line of code in Figure is inside a string, so it will be put into the string until the ending "character is hit. Pound characters in string are just considered characters, not comments.

Watch

Watch the following video of the textbook for explanations and examples of comments in Python scripts as in Figure 1.3:

Video for Exercise 2 Comments and Pound Characters

Read

Read the following section of the textbook on examples of comments in Python scripts as in Figure 1.3:

Exercise 2 Comments and Pound Characters

Chapter 2: Variables and Types

If we want to do complex tasks, we will want to store values while coding along. We can do this by defining a variable with a specific and case-sensitive name. Once we create such a variable with its assigned value, we can later retrieve its value by using the variable name.

2.1 Variables

As in the examples in Figure 1.4, we create variables and assign values to them. Every time we type the variables' names, Python changes them with the actual values of the variables.

```
cars = 100
space_in_a_car = 4.0
drivers = 30
passengers = 90
cars_not_driven = cars - drivers
cars_driven = drivers
carpool_capacity = cars_driven * space_in_a_car
average_passengers_per_car = passengers / cars_driven

print("There are", cars, "cars available.")
print("There are only", drivers, "drivers available.")
print("There will be", cars_not_driven, "empty cars today.")
print("We can transport", carpool_capacity, "people today.")
print("We have", passengers, "to carpool today.")
print("We need to put about", average_passengers_per_car, "in each car.")
```

Figure 1.4 Creating and using Python variables

Note: In Figure 1.4, the = (single-equal) assigns the value on the right to a variable on the left. The == (double-equal) tests whether two things have the same value.

Note: In Figure 1.4, we put the _ ,underscore character, to join words in variable names.

Note: We cannot create a variable like this: 1a = 'Happy New Year', because 1a is not a valid variable name. A valid variable name needs to start with a character, so a1 would work, but 1a will not.

Watch

Watch the following video of the textbook for creating and using variables in Python as in Figure 1.4:

Video for Exercise 4 Variables and Names

Read

Read the following section of the textbook on examples of creating and using variables in Python as in Figure 1.4:

Exercise 4 Variables and Names

2.2 Types

So far, we've worked with numerical values or strings. Numerical values can be a float (python's way of representing a real number) or int (python's way of representing integers).

The type str (short for string) is Python's way to represent text. You can use both double and single quotes to build a string, as you can see from the previous examples. We will learn more about string type in Study Unit 4.

The type bool (short for Boolean) is a type that can either be True or False. We will learn more about boolean type in Study Unit 2.

Read

Read the Python documentation (https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex) for more about the different operations on numeric data types.

2.3 Operators

Python is also able to act as calculators. Expression syntax is straightforward: the operators +, -, * and /; parentheses () for grouping as the examples in Figure 1.5.

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the modulus % operator returns the remainder of the division
2
>>> 2 ** 7  # 2 to the power of 7
128
>>> "Happy" + "New Year"
'HappyNew Year'
```

Figure 1.5 Math operations in Python

Note: As in the example in Figure 1.5, we can apply the plus + operator to two integers or two strings. For the integers, the values were summed, while for the strings, the strings were pasted together. The plus operator behaves differently for different data types. In Python, how the operator behaves depends on the types applied to.

Watch

Watch the following video of the textbook for math calculation in Python:

Video for Exercise 3 Numbers and Math

Read

Read the following section of the textbook on math calculation in Python:

Exercise 3 Numbers and Math

Read

Read the Python official documentation (https://docs.python.org/3/reference/expressions.html#operator-precedence) for more about the operator precedence in Python.

Chapter 3: Printing

Every time we put " (double-quotes) around a piece of text we have been making a string. We have learned how to print strings. In this chapter, we will learn how to make strings that have variables embedded in them. It is called a "format string" as in Figure 1.6.

```
my_age = 35 # years
my_height = 74 # inches
my_weight = 180 # lbs
my_eyes = 'Blue'
my_hair = 'Brown'

print(f"He's {my_height} inches tall.")
print(f"He's {my_weight} pounds heavy.")
print(f"He's got {my_eyes} eyes and {my_hair} hair.")

total = my_age + my_height + my_weight
print(f"If I add {my_age}, {my_height}, and {my_weight} I get {total}.")
```

Figure 1.6 Printing format strings

Note: We usually use format strings for printing and seldom break a string into pieces separated by a comma as the way in Figure 1.6.

Watch

Watch the following video of the textbook for printing format strings as in Figure 1.6:

Video for Exercise 5 More Variables and Printing

Read

Read the following section of the textbook on printing format strings as in Figure 1.6:

Exercise 5 More Variables and Printing

Python also has another kind of formatting strings using the .format() method as in Figure 1.7.

```
hilarious = False
print("Isn't that joke so funny?! {}".format(hilarious))

print("Mary had a little lamb. Its fleece was white as {}.".format('snow'))
```

Figure 1.7 Printing format strings using the format() method

Watch

Watch the following two videos of the textbook for printing format strings using the format() method as in Figure 1.7:

Video for Exercise 6 Strings and Text

Video for Exercise 7 More Printing

Read

Read the following two sections of the textbook on printing format strings using the format() method as in Figure 1.7:

Exercise 6 Strings and Text

Exercise 7 More Printing

Escape sequences are also import for formatting strings' printing as in Figure 1.8.

```
print("Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug")
print("I am 6'2\" tall.")  # escape double-quote inside string
print('I am 6\'2" tall.')  # escape single-quote inside string
```

Figure 1.8 Escape sequences in Python scripts

Watch

Watch the following two videos of the textbook for escape sequences in printing format strings as in Figure 1.8:

Video for Exercise 9 Printing, Printing, Printing

Video for Exercise 10 What Was That?

Read

Read the following two sections of the textbook on using escape sequence in printing format strings as in Figure 1.8:

Exercise 9 Printing, Printing, Printing

Exercise 10 What Was That?

Chapter 4: Input

Most of what software does is to take some kind of input from a user, working on it and print out some results. In this chapter, we will learn how to get any input from a user during the execution of the Python script as in the examples in Figure 1.9.

```
name = input("Name? ")
age = input("How old are you? ")
height = input("How tall are you in metres? ")
weight = input("How much do you weigh in kilograms? ")

print(f"{name} is {age} old, {height} tall and {weight} heavy.")

bmi = float(weight)/float(height)**2
print(f"Your BMI is {bmi}.")
```

Figure 1.9 User input in Python

Note: The user's inputs all come in as strings, even if they are typed as numbers. How to get a number from input for math calculation? As in Figure 1.9, use int(input()) or float(input()), which gets the number as a string from input(), then converts it to an integer using int(), or to a float using float().

Watch

Watch the following two videos of the textbook on getting input from a user during the execution of Python scripts as in Figure 1.9:

Video for Exercise 11 Asking Questions

Video for Exercise 12 Prompting People

Read

Read the following two sections of the textbook on getting users' input as in Figure 1.9:

Exercise 11 Asking Questions

Exercise 12 Prompting People

Summary

We have learned the basics of writing and executing Python program. We covered the various data types and appropriate operations applied to the types. On the numeric types, there's the float to represent a real number, and the int to represent an integer. We also have str, short for string to represent text, and bool which can be either True or False. Lastly, we learned how to create user input, implement appropriate operations on the input, and construct formatted printing.

Formative Assessment

1. Which file extension is used for Python scripts?
    a. .exe
    b. .pys
    c. .python
    d. .py

2. Which line of code creates a variable x with the value 1?
    a. x equals 1
    b. x == 1
    c. 1 = x
    d. x = 1

3. Which of the following statements is correct about the following two lines of code?

    x = “test”

    y = False
    a. x is an integer, and y is a boolean.
    b. x is a string, and y is a float.
    c. x is an integer, and y is a float.
    d. x is a string, and y is a boolean.

4. What is the proper way to terminate the Python interpreter and get back to the PowerShell/Terminal prompt?
    a. exit
    b. stop()
    c. end
    d. quit()

5. Which of the following is a bad Python variable name?
    a. happy_10
    b. _happy
    c. #happy
    d. HAPPY10

6. Which of the following elements of a mathematical expression in Python is evaluated first?
    a. *
    b. +
    c. **
    d. ()

7. What is the value of 7 % 10 in Python 3?
    a. *
    b. 1
    c. 10
    d. 7

8. What is the value of 1 + 2 * 3 - 8 / 4 in Python 3?
    a. 4
    b. 2.0
    c. 5.0
    d. 8

9. What is the value of int(6.9) in Python 3?
    a. 6
    b. 7
    c. 6.5

d. It has a syntax error

10. What does the Python input() function do?
    a. Read data from a remote web page
    b. Read data from a file in hard disk
    c. Read the memory of the running program
    d. Pause the program and read data from the user

Solutions or Suggested Answers

Formative Assessment

1. Which file extension is used for Python scripts?

   a. .exe

      Incorrect. Refer to Section 1.3 on Python scripts.

   b. .pys

      Incorrect. Refer to Section 1.3 on Python scripts.

   c. .python

      Incorrect. Refer to Section 1.3 on Python scripts.

   d. .py

      Correct.

2. Which line of code creates a variable x with the value 1?

   a. x equals 1

      Incorrect. Refer to Section 2.1 on variables.

   b. x == 1

      Incorrect. Refer to Section 2.1 on variables.

   c. 1 = x

      Incorrect. Refer to Section 2.1 on variables.

   d. x = 1

      Correct.

3. Which of the following statements is correct about the following two lines of code?

   x = “test”

y = False

a. x is an integer, and y is a boolean.
   Incorrect. Refer to Section 2.2 on Types.

b. x is a string, and y is a float.
   Incorrect. Refer to Section 2.2 on Types.

c. x is an integer, and y is a float.
   Incorrect. Refer to Section 2.2 on Types.

d. x is a string, and y is a boolean.
   Correct.

4. What is the proper way to terminate the Python interpreter and get back to the PowerShell/Terminal prompt?

   a. exit
      Incorrect. Refer to Section 1.2 on the Python interpreter.

   b. stop()
      Incorrect. Refer to Section 1.2 on the Python interpreter.

   c. end
      Incorrect. Refer to Section 1.2 on the Python interpreter.

   d. quit()
      Correct.

5. Which of the following is a bad Python variable name?

   a. happy_10
      Incorrect. Refer to Section 2.1 on variables.

   b. _happy
      Incorrect. Refer to Section 2.1 on variables.

c. #happy

Correct.

d. HAPPY10

Incorrect. Refer to Section 2.1 on variables.

6. Which of the following elements of a mathematical expression in Python is evaluated first?

a. *

Incorrect. Refer to Section 2.3 on math operators in Python.

b. +

Incorrect. Refer to Section 2.3 on math operators in Python.

c. **

Incorrect. Refer to Section 2.3 on math operators in Python.

d. ()

Correct.

7. What is the value of 7 % 10 in Python 3?

a. *

Incorrect. Refer to Section 2.3 on math operators in Python.

b. 1

Incorrect. Refer to Section 2.3 on math operators in Python.

c. 10

Incorrect. Refer to Section 2.3 on math operators in Python.

d. 7

Correct.

8. What is the value of 1 + 2 * 3 - 8 / 4 in Python 3?

a. 4

Incorrect. Refer to Section 2.3 on math operators in Python.

b. 2.0

Incorrect. Refer to Section 2.3 on math operators in Python.

c. 5.0

Correct.

d. 8

Incorrect. Refer to Section 2.3 on math operators in Python.

9. What is the value of int(6.9) in Python 3?

a. 6

Correct.

b. 7

Incorrect. Refer to Section 2.3 on math operators in Python.

c. 6.5

Incorrect. Refer to Section 2.3 on math operators in Python.

d. It has a syntax error

Incorrect. Refer to Section 2.3 on math operators in Python.

10. What does the Python input() function do?

a. Read data from a remote web page

Incorrect. Refer to Chapter 4 on the input() function.

b. Read data from a file in hard disk

Incorrect. Refer to Chapter 4 on the input() function.

c. Read the memory of the running program

Incorrect. Refer to Chapter 4 on the input() function.

d. Pause the program and read data from the user

Correct.

References

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

Numeric types – int, float, complex. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Operator precedence. (n.d.). Retrieved from https://docs.python.org/3/reference/expressions.html#operator-precedence

Study Unit 2

Control Flow and Lists

Learning Outcomes

By the end of this unit, you should be able to:

1. Compose appropriate Boolean expressions for given scenarios
2. Construct conditional control flow
3. Use while-loop for repeated tasks
4. Select lists as the appropriate data structures for given scenarios, and implement subsetting on Lists
5. Solve problems using Python lists and for-loop

Overview

The if, while and for statements implement control flow constructs. These are compound statements which contain groups of other statements and control the execution of those other statements. In addition to Boolean expressions and control flow, this unit also starts to discuss one of the compound data structures in Python, Lists. We will learn subsetting lists and iterating over elements of Lists.

Chapter 1: Control Flow

We can use relational operators and logical operators to compose conditions for dynamically changing how our program behaves. The two important structures for control flow in Python are conditional statements and loop statements. These are compound statements which contain groups of other statements and control the execution of those other statements.

1.1 Boolean Expressions

Boolean expressions are used as conditions in the control flow structures for dynamically changing our program's behaviours. A Boolean expression can be the result of a relational operation, or combining the results of multiple relational operations by logical operators.

Watch

Watch the following two videos of the textbook:

Video for Exercise 27 Memorizing Logic: This video demonstrates the basics of relational operators and logic operators.

Video for Exercise 28 Boolean Practice: This video mainly explains how to combine the results of multiple relational operations by logical operators.

Read

Read the following two sections of the textbook on Boolean expressions:

Exercise 27 Memorizing Logic

Exercise 28 Boolean Practice

1.2 Conditional Statements

We can use Boolean expression as condition to dynamically change our program's behaviour. We can do this with three major conditional statements in Python:

- if
- if-else
- if-elif-else.

Watch

Watch the following three videos of the textbook:

Video for Exercise 29 What If: This video demonstrates how to use the basic if-statement.

Video for Exercise 30 Else and If and video for Exercise 31 Making Decisions: The two videos explain how to use the if-else and if-elif-else blocks for more complicated logic flow control as in Figures 2.1 and 2.2.

```
people = 30
cars = 40
trucks = 15

if cars > people:
    print("We should take the cars.")
elif cars < people:
    print("We should not take the cars.")
else:
    print("We can't decide.")

if people > trucks:
    print("Alright, let's just take the trucks.")
else:
    print("Fine, let's stay home then.")
```

Figure 2.1 Creating sequential if-elif-else blocks

Reflect

1. Change the numbers of cars, people, and trucks, and then trace through each if-statement to see what will be printed.
2. Try some more complex Boolean expressions like *cars > people or trucks < cars.*

```
print("""You enter a dark room with two doors.
Do you go through door #1 or door #2?""")
door = input("> ")

if door == "1":
    print("There's a giant bear here eating a cheese cake.")
    print("What do you do?")
    print("1. Take the cake.")
    print("2. Scream at the bear.")
    bear = input("> ")
    if bear == "1":
        print("The bear eats your face off. Good job!")
    elif bear == "2":
        print("The bear eats your legs off. Good job!")
    else:
        print(f"Well, doing {bear} is probably better.")
        print("Bear runs away.")
elif door == "2":
    print("You stare into the endless abyss at Cthulhu's retina.")
    print("1. Blueberries.")
    print("2. Yellow jacket clothespins.")
    print("3. Understanding revolvers yelling melodies.")
    insanity = input("> ")
    if insanity == "1" or insanity == "2":
        print("Your body survives powered by a mind of jello.")
        print("Good job!")
    else:
        print("The insanity rots your eyes into a pool of muck.")
        print("Good job!")
else:
    print("You stumble around and fall on a knife and die. Good job!")
```

Figure 2.2 Creating nested if-elif-else blocks

Read

Read the following three sections of the textbook on conditional statements for control flow:

Exercise 29 What If

Exercise 30 Else and If

Exercise 31 Making Decisions

1.3 Loops

The while-loop is used for repeated execution as long as a condition as Boolean expression is true. A while-loop can do any kind of iteration. However, a for-loop can only iterate over collections of things. Hence, we will introduce for-loop in Chapter 2 after we have learned the basics of the data structure List.

Watch

Watch the following video of the textbook:

Video for Exercise 33 While Loops: This video demonstrates how to use the while-statement and Boolean expressions for program flow control as in Figure 2.3.

```
i = 0
print(f"At the top i is {i}")
while i < 6:
    i = i + 1
    print("i now is: ", i)
```

Figure 2.3 Creating a while-loop block

Read

Read the following section of the textbook on while loops:

Exercise 33 While Loops

Note: while-loop is a type of unbounded loop, i.e. it does not stop if we set an always True condition.

Chapter 2: Lists

We have learned different data types in SU1. However, we will often want to work with many data points. It would be inconvenient to create a new python variable for each data point. What we can do instead is to store all data points in a python list.

2.1 Creating Lists

We can build a Python list with square brackets. For example, scores = [62, 50, 78] is a Python list with three integers, stored in the variable named scores.

- We can see that a list is a way to give a single name to a collection of values. These values can be in any type, floats, integer, booleans, strings, or more advanced Python types like lists.
- It's also possible for a list to contain different types of values like ['Jack', 1.78, 'Tom', 1.75] or contain lists themselves like ['Jack', [1.78, 70]].

2.2 Subsetting Lists

After creating a Python list, we want to index or slice elements in the list. Like strings, lists can be indexed and sliced. All slice operations return a new list containing the requested elements.

Watch

Watch the following video of the textbook:

Video for Exercise 34 Accessing Elements of Lists: This video demonstrates how to access elements of lists using index as in Figure 2.4.

Video for Exercise 38 Doing Things to Lists: This video explains how to add, remove and join elements in Python list as in Figure 2.5.

```
>>> animals = ['bear', 'tiger', 'penguin', 'zebra']
>>> animals[1]
'tiger'
>>> animals[1:]
['tiger', 'penguin', 'zebra']
>>> animals[1:2]
['tiger']
>>> animals[:2]
['bear', 'tiger']
>>> animals[-1]
'zebra'
>>> animals[-2]
'penguin'
>>> animals[::2]
['bear', 'penguin']
```

Figure 2.4 Creating and subsetting Python list

```
stuff = ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light', 'Sugar']
print(stuff)

more_stuff = ["Day", "Night", "Song", "Frisbee","Corn", "Banana","Girl", "Boy"]
while len(stuff) != 10:
    next_one = more_stuff.pop()
    print("Adding: ", next_one)
    stuff.append(next_one)
    print(f"There are {len(stuff)} items now.", stuff)

print("We are removing the last element", stuff.pop())
print("We are joining all the elements", ' '.join(stuff))
print("We are joining the 4th and 5th elements with a #" , '#'.join(stuff[3:5]))
```

Figure 2.5 Adding, removing and joining elements in Python list

Read

Read the Python official documentation (https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types) for more details on operations applicable to lists.

Read

Read the following section of the textbook on accessing, adding, removing and joining elements of lists:

Exercise 34 Accessing Elements of Lists

Exercise 38 Doing Things to Lists

2.3 Lists and Loops

We have learned while-loop in Section 1.3. A for-loop is another way of looping which can only iterate over collections of things. We will learn in this chapter how to iterate elements of lists using for-loop.

Watch

Watch the following video of the textbook:

Video for Exercise 32 Loops and Lists: This video demonstrates how to iterate elements of lists using for-loop as in Figure 2.6.

```
the_count = [1, 2, 3, 4, 5]
fruits =['apples', 'oranges', 'pears', 'apricots']

for number in the_count:
    print(f"This is count {number}")

for fruit in fruits:
    print(f"A fruit of type: {fruit}")
```

Figure 2.6 Iterating elements in Python list using for-loop

Read

Read the following official Python documentation: (https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range) for more details and examples on lists.

Read

Read the following section of the textbook on looping over elements of lists:

Exercise 32 Loops and Lists

Summary

We have learned three types of compound statements for control flow. The if statement is used for conditional execution. The while statement is used for repeated execution as long as an expression is True. The for statement is used to iterate over the elements of a sequence, for instance, lists in this unit. It is a new Python type, in addition to the strings, Booleans, integers and floats we have already learned. We have learned how to subset and iterate over elements of lists.

Formative Assessment

1. What is the result of 5>=5?
    a. Incorrect syntax. It should be 5 > = 5
    b. None
    c. True
    d. False

2. What is the result of 4 != 4?
    a. Incorrect syntax.
    b. None
    c. False
    d. True

3. Which statement is correct about the and operator?
    a. The and operator returns True if neither operands is True.
    b. The and operator returns True if at least one of the operands is True.
    c. The and operator returns True if exactly one of the operands is True.
    d. The and operator returns True only if both operands are True.

4. Which statement is correct about the or operator?
    a. The or operator returns True only if both operands are True.
    b. The or operator returns True if exactly one of the operands is True.
    c. The or operator returns True if neither operands is True.
    d. The or operator returns True if at least one of the operands is True.

5. What will be printed out if you execute the following code?

```
x = 5
if x > 6 :
    print("high")
else :
    print("low")
```

a. Nothing is printed.

b. low

c. high

d. The code has syntax errors.

6. What will be printed out if you execute the following code?

```
x = 7
if x > 6 :
    print("high")
elif x > 3 :
    print("ok")
else :
    print("low")
```

a. ok

b. low

c. high

d. The code has syntax errors.

7. Which of the following is correct about Python list?

a. It allows to store and refer to the same values as two variables with different names

b. It is a data structure to hold three and above elements.

c. It can only be used for integers and floats.

d. It is a way to name a collection of values, instead of creating separate variables for each element.

8. Which statement below is invalid to create a Python list x?
    a. x = [“this” + “is”, 1, True, “color”]
    b. x = [“this”, “is”, 1, True, “color”]
    c. x = [“this”, “is”, “a”, True, “color”]
    d. x = [“this”, “is”, “a” True “color”]

9. Given a Python list y = [“this”, “is”, True, “color”], which statement correctly extracts the Boolean value from this list?
    a. y[2]
    b. y[3]
    c. y’4’
    d. y{-1}

10. Given the general syntax to slice a Python list x[begin:end], which of the following is correct?
    a. Both the begin and end indexes are included in the slice.
    b. The begin index is not included in the slice, the end index is.
    c. Neither the begin nor the end index is included in the slice.
    d. The begin index is included in the slice, the end index is not.

Solutions or Suggested Answers

Formative Assessment

1. What is the result of 5>=5?

    a. Incorrect syntax. It should be 5 > = 5

    Incorrect. Refer to Section 1.1 on Boolean expressions.

    b. None

    Incorrect. Refer to Section 1.1 on Boolean expressions.

    c. True

    Correct.

    d. False

    Incorrect. Refer to Section 1.1 on Boolean expressions.

2. What is the result of 4 != 4?

    a. Incorrect syntax.

    Incorrect. Refer to Section 1.1 on Boolean expressions.

    b. None

    Incorrect. Refer to Section 1.1 on Boolean expressions.

    c. False

    Correct.

    d. True

    Incorrect. Refer to Section 1.1 on Boolean expressions.

3. Which statement is correct about the and operator?

    a. The and operator returns True if neither operands is True.

Incorrect. Refer to Section 1.1 on Boolean expressions.

b. The and operator returns True if at least one of the operands is True.
Incorrect. Refer to Section 1.1 on Boolean expressions.

c. The and operator returns True if exactly one of the operands is True.
Incorrect. Refer to Section 1.1 on Boolean expressions.

d. The and operator returns True only if both operands are True.
Correct.

4. Which statement is correct about the or operator?
   a. The or operator returns True only if both operands are True.
   Incorrect. Refer to Section 1.1 on Boolean expressions.

   b. The or operator returns True if exactly one of the operands is True.
   Incorrect. Refer to Section 1.1 on Boolean expressions.

   c. The or operator returns True if neither operands is True.
   Incorrect. Refer to Section 1.1 on Boolean expressions.

   d. The or operator returns True if at least one of the operands is True.
   Correct.

5. What will be printed out if you execute the following code?

```
x = 5
if x > 6 :
    print("high")
else :
    print("low")
```

   a. Nothing is printed.

Incorrect. Refer to Section 1.2 on conditional statements.

b. low

Correct.

c. high

Incorrect. Refer to Section 1.2 on conditional statements.

d. The code has syntax errors.

Incorrect. Refer to Section 1.2 on conditional statements.

6. What will be printed out if you execute the following code?

```
x = 7
if x > 6 :
    print("high")
elif x > 3 :
    print("ok")
else :
    print("low")
```

a. ok

Incorrect. Refer to Section 1.2 on conditional statements.

b. low

Incorrect. Refer to Section 1.2 on conditional statements.

c. high

Correct.

d. The code has syntax errors.

Incorrect. Refer to Section 1.2 on conditional statements.

7. Which of the following is correct about Python list?

a. It allows to store and refer to the same values as two variables with different names
Incorrect. Refer to Section 2.1 on Python lists.

b. It is a data structure to hold three and above elements.
Incorrect. Refer to Section 2.1 on Python lists.

c. It can only be used for integers and floats.
Incorrect. Refer to Section 2.1 on Python lists.

d. It is a way to name a collection of values, instead of creating separate variables for each element.
Correct.

8. Which statement below is invalid to create a Python list x?

a. x = [“this” + “is”, 1, True, “color”]
Incorrect. Refer to Section 2.1 on lists.

b. x = [“this”, “is”, 1, True, “color”]
Incorrect. Refer to Section 2.1 on lists.

c. x = [“this”, “is”, “a”, True, “color”]
Incorrect. Refer to Section 2.1 on lists.

d. x = [“this”, “is”, “a” True “color”]
Correct.

9. Given a Python list y = [“this”, “is”, True, “color”], which statement correctly extracts the Boolean value from this list?

a. y[2]
Incorrect. Refer to Section 2.2 on subsetting lists.

b. y[3]

Correct.

c. y'4'
Incorrect. Refer to Section 2.2 on subsetting lists.

d. y{-1}
Incorrect. Refer to Section 2.2 on subsetting lists.

10. Given the general syntax to slice a Python list x[begin:end], which of the following is correct?

a. Both the begin and end indexes are included in the slice.
Incorrect. Refer to Section 2.2 on subsetting lists.

b. The begin index is not included in the slice, the end index is.
Incorrect. Refer to Section 2.2 on subsetting lists.

c. Neither the begin nor the end index is included in the slice.
Incorrect. Refer to Section 2.2 on subsetting lists.

d. The begin index is included in the slice, the end index is not.
Correct.

References

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

Mutable sequence types. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

Sequence types – list, tuple and range. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

Study Unit 3

Tuples and Dictionaries

Learning Outcomes

By the end of this unit, you should be able to:

1. Use the Python tuple data structure and its operations appropriately, including indexing, subsetting, unpacking and iterating elements in tuples
2. Explain the immutable nature of the Python tuple
3. Create Python dictionary and implement operations including indexing, getting, adding or removing elements in dictionary
4. Implement sorting on Python dictionaries based on the keys
5. Solve problems using Python dictionaries and for-loop

Overview

Python provides a number of compound data types to group together a collection of values. Compound data structures are a way of organising and storing data so that they can be accessed and worked with efficiently. They define the relationship between the data, and the operations that can be performed on the data. In addition to the Python list, we will learn about Python tuple and dictionary and how to create them, when to use them, what operations to perform on them.

Chapter 1: Tuples

Python provides a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated items between square brackets. There is also another standard sequence data type: the tuple.

Tuples are a limited version of lists. The code in Figure 3.1 gives examples of creating and manipulating tuples.

```
>>> t = 12345, 54321, 'hello!'
>>> t
(12345, 54321, 'hello!')
>>> t[0]
12345
>>> x, y, z = t
>>> x
12345
>>> y
54321
>>> z
'hello!'
>>> t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> for i in t:
...     print(i)
...
12345
54321
hello!
```

Figure 3.1 Creating and manipulating tuples

1. Tuples may be constructed by separating items with commas as in 12345, 54321, 'hello!'.
2. Similar to lists, we can subset tuples using the index operator [].
3. As opposed to creating a tuple for packing items, we can unpack a tuple “t” and assign each item to individual variables “x”, “y” and “z”.

4. Note that tuples are immutable. If we try to change t[0] to 88888 with an assignment statement, we will encounter an error.
5. Similar to lists, we can use a for-loop to iterate the items of a tuple.

Note: If you recall, strings are immutable also.

Note: Because tuples are immutable, they are more efficient in terms of performance and memory use. Tuples are useful in situations where you want to share the data with others but not allow them to modify the data. They can use the data values, but no change is reflected in the original data shared.

Note: The official Python documentation (https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types) states the operations who is valid to apply to mutable sequence types, e.g. lists, not able to immutable sequence types, e.g. tuples.

Reflect

The official Python documentation (https://docs.python.org/3/library/stdtypes.html#typesseq) explains the common operations between Python list and tuple, as well as the differences between these two sequence types. Practise the examples given.

Chapter 2: Dictionaries

Another useful built-in data type is the dictionary. It is best to think of a dictionary as an unordered set of key:value pairs, with the requirement that the keys are unique (within one dictionary). For example, tel = {'jack': 4098, 'sape': 4139, 'irv': 4127} creates a dictionary, using a pair of braces with a comma-separated list of key:value pairs within the braces.

Note: Unlike lists and tuples, which are indexed by a range of numbers, dictionaries are indexed by keys, which are usually strings or numbers as in Figure 3.2.

```
>>> stuff = {'name': 'Zed', 'age': 39,'height': 6 * 12 + 2}
>>> print(stuff['name'])
Zed
>>> stuff[1] = "Wow"
>>> print(stuff[1])
Wow
```

Figure 3.2 Comparing indexes of lists and dictionaries

Watch

Watch the following video of the textbook:

Video for Exercise 39 Dictionaries, Oh Lovely Dictionaries: This video demonstrates how to

- create dictionaries as in Figure 3.3
- access key:value pairs of dictionaries as in Figure 3.4
- add or delete key:value pairs in dictionaries as in Figure 3.4
- iterate over key:value pairs in dictionaries using for-loop as in Figure 3.5

```
# create a mapping of state to abbreviation
states = {
    'Oregon': 'OR',
    'Florida': 'FL',
    'California': 'CA',
    'New York': 'NY',
    'Michigan': 'MI'
}

# create a basic set of states and some cities in them
cities = {
    'CA': 'San Francisco',
    'MI': 'Detroit',
    'FL': 'Jacksonville'
}
```

Figure 3.3 Creating dictionaries

```
# add some more cities
cities['NY'] = 'New York'
cities['OR'] = 'Portland'

# print out some cities
print('-' * 10)
print("NY State has: ", cities['NY'])
print("OR State has: ", cities['OR'])

# print some states
print('-' * 10)
print("Michigan's abbreviation is: ", states['Michigan'])
print("Florida's abbreviation is: ", states['Florida'])

# do it by using the state then cities dict
print('-' * 10)
print("Michigan has: ", cities[states['Michigan']])
print("Florida has: ", cities[states['Florida']])

print('-' * 10)
# safely get a abbreviation by state that might not be there
state = states.get('Texas')
if not state:
    print("Sorry, no Texas.")
# get a city with a default value
city = cities.get('TX', 'Does Not Exist')
print(f"The city for the state 'TX' is: {city}")
```

Figure 3.4 Adding items into dictionaries and two ways of accessing items in dictionaries

```python
# print every state abbreviation
print('-' * 10)
for state, abbrev in list(states.items()):
    print(f"{state} is abbreviated {abbrev}")

# print every city in state
print('-' * 10)
for abbrev, city in list(cities.items()):
    print(f"{abbrev} has the city {city}")

# now do both at the same time
print('-' * 10)
for state, abbrev in list(states.items()):
    print(f"{state} state is abbreviated {abbrev}")
    print(f"and has city {cities[abbrev]}")
```

Figure 3.5 Iterating over items in dictionaries using for-loop

Read

Read the following section of the textbook on creating and manipulating dictionaries:

Exercise 39 Dictionaries, Oh Lovely Dictionaries

Read

Read the following official Python documentation for more details and examples on dictionaries:

Mapping types -- dict.
https://docs.python.org/3/library/stdtypes.html#typesmapping

Note: A dictionary is an unordered set of key: value pairs, with the requirement that the keys are unique within one dictionary. Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in arbitrary order. Use sorted(d.keys()) to produce a list of the sorted keys. Take a look at the collections.

OrderedDict (https://docs.python.org/3/library/collections.html#collections.OrderedDict) data structure in Python if you need a dictionary where the elements are in order.

Summary

In this unit, we have learned two types of built-in compound data structures of the Python, tuples and dictionaries. Similar to the Python list we learned in Study Unit 2, we learned about how to create tuples and dictionaries, the common operations we can do with tuples and dictionaries. There are many various kinds of data structures. Hence, we highlighted the major differences among the Python list, tuple and dictionary, and explain what data structures to use for given scenarios.

Formative Assessment

1. What is the difference between a Python tuple and a Python list?
    a. Lists are mutable and tuples are immutable.
    b. Lists are indexed by integers and tuples are indexed by strings.
    c. Tuples can be expanded after they are created and lists cannot.
    d. Lists maintain the order of the items and tuples do not maintain order.

2. What will be in the variable y after this code x, y = 3, 4 is executed?
    a. 4
    b. 3
    c. A dictionary with the key 3 mapped to the value 4
    d. A two-item tuple

3. What does the following Python code accomplish?

```
x = {'chuck' : 1 , 'fred' : 42, 'jan': 100}
tmp = list()
for k, v in x.items() :
    tmp.append( (k, v) )
```

    a. It sorts the dictionary based on its key values.
    b. It creates a list of tuples where each tuple is (key, value).
    c. It computes the average of all of the values in the dictionary.
    d. It counts all the items in the dictionary.

4. How to access the 'Wed' of the tuple, days = ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun')?
    a. days(2)
    b. days[2]
    c. days[3]

d. days(3)

5. In the following Python loop, what are stored in the two iteration variables (k and v)?

```
x = {'chuck' : 1 , 'fred' : 42, 'jan': 100}
tmp = list()
for k, v in x.items() :
    tmp.append( (k, v) )
```

a. the key and value returned by the items() method in each iteration

b. a list of all the keys and a list of all the values

c. the previous and current key for each item

d. the current and next key for each item

6. When might you prefer to use a tuple over a list?

a. For a variable that you will use without modifying it

b. For a list of key:value pairs

c. For a list of items that will be extended as new items are found

d. For a list of items you intend to sort in place

7. How are Python dictionaries different from Python lists?

a. Python lists are indexed using integers and dictionaries can use strings as indexes.

b. Python lists are a collection of items but dictionaries are not.

c. Python lists are mutable but dictionaries are not.

d. Python lists can store any data types but dictionaries can only store strings.

8. What would the following Python code print out?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
print(tel['candy'])
```

a. The program would fail with an error.

b. None

c. -1

d. candy

9. What would the following Python code print out?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
print(tel.get('candy',-1))
```

a. -1

b. None

c. The program would fail with an error.

d. candy

10. What is stored in the variable y in the following for-loop?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
for y in tel:
    print(y)
```

a. It loops through the values in the dictionary.

b. It loops through the integers in the range from zero through the length of the dictionary.

c. It loops through the keys in the dictionary.

d. It loops through the key:value pairs in the dictionary.

Solutions or Suggested Answers

Formative Assessment

1. What is the difference between a Python tuple and a Python list?

    a. Lists are mutable and tuples are immutable.

    Correct.

    b. Lists are indexed by integers and tuples are indexed by strings.

    Incorrect. Refer to Chapter 1 on tuples.

    c. Tuples can be expanded after they are created and lists cannot.

    Incorrect. Refer to Chapter 1 on tuples.

    d. Lists maintain the order of the items and tuples do not maintain order.

    Incorrect. Refer to Chapter 1 on tuples.

2. What will be in the variable y after this code x, y = 3, 4 is executed?

    a. 4

    Correct.

    b. 3

    Incorrect. Refer to Chapter 1 on tuples.

    c. A dictionary with the key 3 mapped to the value 4

    Incorrect. Refer to Chapter 1 on tuples.

    d. A two-item tuple

    Incorrect. Refer to Chapter 1 on tuples.

3. What does the following Python code accomplish?

```
x = {'chuck' : 1 , 'fred' : 42, 'jan': 100}
tmp = list()
for k, v in x.items() :
    tmp.append( (k, v) )
```

a. It sorts the dictionary based on its key values.

Incorrect. Refer to Chapter 2 on dictionaries.

b. It creates a list of tuples where each tuple is (key, value).

Correct.

c. It computes the average of all of the values in the dictionary.

Incorrect. Refer to Chapter 2 on dictionaries.

d. It counts all the items in the dictionary.

Incorrect. Refer to Chapter 2 on dictionaries.

4. How to access the 'Wed' of the tuple, days = ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun')?

a. days(2)

Incorrect. Refer to Chapter 1 on tuples.

b. days[2]

Correct.

c. days[3]

Incorrect. Refer to Chapter 1 on tuples.

d. days(3)

Incorrect. Refer to Chapter 1 on tuples.

5. In the following Python loop, what are stored in the two iteration variables (k and v)?

```
x = {'chuck' : 1 , 'fred' : 42, 'jan': 100}
tmp = list()
for k, v in x.items() :
    tmp.append( (k, v) )
```

a. the key and value returned by the items() method in each iteration
Correct.

b. a list of all the keys and a list of all the values
Incorrect. Refer to Chapter 2 on dictionaries.

c. the previous and current key for each item
Incorrect. Refer to Chapter 2 on dictionaries.

d. the current and next key for each item
Incorrect. Refer to Chapter 2 on dictionaries.

6. When might you prefer to use a tuple over a list?

a. For a variable that you will use without modifying it
Correct.

b. For a list of key:value pairs
Incorrect. Refer to Chapter 1 on tuples.

c. For a list of items that will be extended as new items are found
Incorrect. Refer to Chapter 1 on tuples.

d. For a list of items you intend to sort in place
Incorrect. Refer to Chapter 1 on tuples.

7. How are Python dictionaries different from Python lists?

a. Python lists are indexed using integers and dictionaries can use strings as indexes.

Correct.

b. Python lists are a collection of items but dictionaries are not.
Incorrect. Refer to Chapter 2 on dictionaries.

c. Python lists are mutable but dictionaries are not.
Incorrect. Refer to Chapter 2 on dictionaries.

d. Python lists can store any data types but dictionaries can only store strings.
Incorrect. Refer to Chapter 2 on dictionaries.

8. What would the following Python code print out?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
print(tel['candy'])
```

a. The program would fail with an error.
Correct.

b. None
Incorrect. Refer to Chapter 2 on dictionaries.

c. -1
Incorrect. Refer to Chapter 2 on dictionaries.

d. candy
Incorrect. Refer to Chapter 2 on dictionaries.

9. What would the following Python code print out?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
print(tel.get('candy',-1))
```

a. -1
Correct.

b. None

Incorrect. Refer to Chapter 2 on dictionaries.

c. The program would fail with an error.

Incorrect. Refer to Chapter 2 on dictionaries.

d. candy

Incorrect. Refer to Chapter 2 on dictionaries.

10. What is stored in the variable y in the following for-loop?

```
tel = {'jack': 4098, 'sape': 4139, 'irv': 4127}
for y in tel:
    print(y)
```

a. It loops through the values in the dictionary.

Incorrect. Refer to Chapter 2 on dictionaries.

b. It loops through the integers in the range from zero through the length of the dictionary.

Incorrect. Refer to Chapter 2 on dictionaries.

c. It loops through the keys in the dictionary.

Correct.

d. It loops through the key:value pairs in the dictionary.

Incorrect. Refer to Chapter 2 on dictionaries.

References

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

Mapping types – dict. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#typesmapping

Mutable sequence types. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

Tuples. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#tuples

Study Unit 4

Functions, Methods and Packages

Learning Outcomes

By the end of this unit, you should be able to:

1. Apply the Python built-in functions.
2. Compose and use user-defined functions
3. Use the Python built-in types and the associated methods
4. Explain the concepts of packages and modules, and how Python manages and imports packages/modules
5. Solve problems using appropriate Python standard libraries

Overview

In this unit, we will learn about what functions are and how we can use the Python built-in functions. We will also learn about the Python built-in types and the associated methods. Specially, we see examples of the methods of string and file objects, which allow our programs process much larger amounts of data by reading and writing files on the storage on our computer. Lastly, we introduce the Python packages and modules which are, together with functions and methods, very useful for reusing and organising our program.

Chapter 1: Functions

Functions aren't entirely new for us actually. A function is a piece of reusable code for solving a particular task. You can call functions instead of writing all the codes from scratch. We have already used a few Python built-in functions. For example, print() is a built-in function that prints objects to the text stream file, the screen by default.

1.1 Built-in Functions

The code in Figure 4.1 gives an example of using the built-in function len().

```
>>> marks = [56, 77, 80, 60]
>>> len(marks)
4
>>> count = len(marks)
>>> count
4
```

Figure 4.1 Using the built-in function len()

1. Suppose we want to count the number of items in the list marks. Instead of writing code that goes through the list and finds the total number of items, we can use Python's len() function. This is one of Python's built-in functions. We pass the list marks to len() inside parentheses. The output returns the total number of items in the list.
2. We can also assign the return of a function call to a new variable, for example, count. The variable count stores the returned value and we can use it just like any other variable.

Note: Whenever we need to do a rather standard task in Python, it is very likely there's already a function for it. Do a quick Python documentation search.

Read

Refer to the link below for more details and examples on the Python built-in functions:

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

1.2 User-defined Functions

We can make our own functions and use them in our Python scripts. We use the user-defined functions the way we use the built-in functions.

SU4-user-definedFunction.py

```
1  def cheese_and_crackers(cheese_count, boxes_of_crackers):
2      print(f"You have {cheese_count} cheeses!")
3      print(f"You have {boxes_of_crackers} boxes of crackers!")
4
5  print("Listing the stock")
6
7  print("We can just give the function numbers directly:")
8  cheese_and_crackers(20, 30)
9
10 print("OR, we can use variables from our script:")
11 amount_of_cheese = 20
12 amount_of_crackers = 30
13 cheese_and_crackers(amount_of_cheese, amount_of_crackers)
14
15 print("\nUpdating the stock")
16
17 print("We can do math inside directly:")
18 cheese_and_crackers(amount_of_cheese + 10, amount_of_crackers - 15)
19
20 def add(a, b):
21     return a + b
22
23 def subtract(a, b):
24     return a - b
25
26 print("OR, we can just give the variables updated by the function:")
27 amount_of_cheese_updated = add(amount_of_cheese, 10)
28 amount_of_crackers_updated = subtract(amount_of_crackers, 15)
29 cheese_and_crackers(amount_of_cheese_updated, amount_of_crackers_updated)
30
31 print("OR, we can call functions inside another function directly:")
32 cheese_and_crackers(add(amount_of_cheese, 10), subtract(amount_of_crackers, 15))
33
```

Figure 4.2 Defining and calling user-defined functions

The example in Figure 4.2 defines three functions as in Lines 1-3 and Lines 20-24.

1. Lines 1-3 define a function named cheese_and crackers.
    a. Start the function definition with *def*, followed by the function name, and parameters in parentheses separated by commas.
    b. We end this line with a : (colon) and start indenting.
    c. After the colon, all the lines that are indented four spaces will become attached to the defined function

2. Lines 8, 13 and 18 demonstrate how to use this user-defined function. We call the function by its name and supply the values into the parentheses. We can give values as straight numbers, variables or mathematical expressions.
3. Lines 20-24 define two functions. Different from the first function, these two functions return values, the result of a mathematical calculation in this example.
    a. We can assign the returned values to variables, as in Lines 27 and 28.
    b. Or use the value returned by a function as the argument value into another function as in Line 32.
4. Execute the program SU4-user-definedFunction.py in PowerShell/Terminal. The output is like below.

$ python3 SU4-user-definedFunction.py

Listing the stock

We can just give the function numbers directly:

You have 20 cheeses!

You have 30 boxes of crackers!

OR, we can use variables from our script:

You have 20 cheeses!

You have 30 boxes of crackers! Updating the stock

We can do math inside directly:

You have 30 cheeses!

You have 15 boxes of crackers!

OR, we can just give the variables updated by the function:

You have 30 cheeses!

You have 15 boxes of crackers!

OR, we can call functions inside another function directly:

You have 30 cheeses!

You have 15 boxes of crackers!

Note: Same as variable names, a function name only consists of letters, numbers, and underscores and doesn't start with a number.

Note: The function definition does not execute the function body; this gets executed only when the function is called.

Note: The variables *cheese_count*, *boxes_of_crackers*, *a* and *b* are called local variables. They are temporary variables made just for the function's run. When the function exits, these temporary variables go away.

Watch

Watch the following three videos of the textbook for more details and examples on defining and using our own functions:

Video for Exercise 18: Names, Variables, Code, Functions

Video for Exercise 19: Functions and Variables

Video for Exercise 21: Functions Can Return Something

Read

Read the following three exercises of the textbook for more details and examples on user-defined functions:

Exercise 18: Names, Variables, Code, Functions

Exercise 19: Functions and Variables

Exercise 21: Functions Can Return Something

Chapter 2: Methods

When we need to do a rather standard task in Python, built-in functions are one way we may look for. But some basic tasks, we won't find built-in functions for them. Instead they are implemented as objects' methods.

In the past study units, we have seen variables in various data types or data structures. Each one is a so-called Python object. These objects have a specific type, such as str, float, or list. An object's type defines the possible values for objects of that type.

Python objects also come with their so-called "methods". You can think of methods as functions that belong to Python objects. Each object has specific methods associated, depending on the type of the object.

2.1 Methods of String Objects

The code in Figure 4.3 gives five examples of using the string method split().

```
>>> 'a,b,c'.split(',')
['a', 'b', 'c']
>>> 'a,b,c'.split(',', maxsplit=1)
['a', 'b,c']
>>> 'a,,b,c,'.split(',')
['a', '', 'b', 'c', '']
>>> 'a b c'.split()
['a', 'b', 'c']
>>> ' a b  c   '.split()
['a', 'b', 'c']
```

Figure 4.3 Using the string method split()

1. The example shows that the split() method returns a list of components in the string 'a,b,c', using the specified delimiter ','.

2. If the parameter maxsplit is given a value, at most maxsplit splits are done. If maxsplit is not specified, all splits are made as in the first example. The second example gives a value of 1 to maxsplit. Hence only one split is done in the output.
3. The third example shows that consecutive delimiters are not grouped together and are deemed to delimit empty strings.
4. The last two examples that if there is no delimiter specified, the leading and trailing whitespaces will be first stripped, and then consecutive whitespaces are regarded as a single separator to split the remaining string for output.

Read

Refer to the link below for more details and examples on the Python string methods:
https://docs.python.org/3/library/stdtypes.html#string-methods

2.2 Methods of File Objects

A web server log maintains a history of page requests, including client IP address, request date/time, page requested, HTTP code and bytes served. A HTTP code of 401 as highlighted in Figure 4.4 indicates an unauthorised access error, meaning the client request has not been applied because it lacks valid authentication credentials for the target resource.

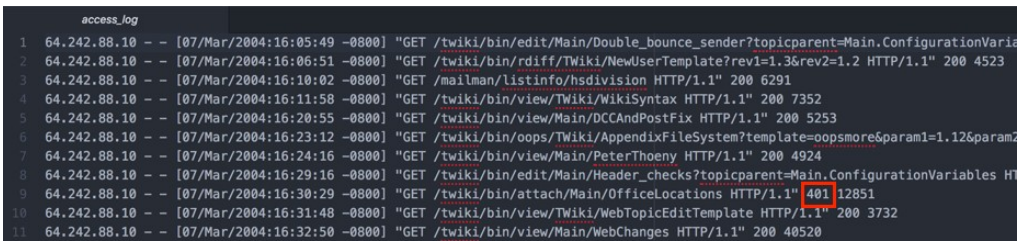

Figure 4.4 A sample of Apache web log

(Source: http://www.monitorware.com/en/logsamples/apache.php)

The code in Figure 4.5 reads the full original log file named “access_log” and write the entries with the HTTP code 401 into a created file named “access_log_401”.

SU4-file.py

```
1  from_file = input("Type the filename of the full access log: ")
2  to_file = input("Type the filename to store the 401 error log: ")
3  in_file = open(from_file)
4  out_file = open(to_file, 'w')
5
6  for line in in_file:
7      if line.find(' 401 ') != -1:
8          out_file.write(line)
9
10 print("Checking 401 error done.")
11 out_file.close()
12 in_file.close()
```

Figure 4.5 Reading and writing text files

(Source: http://www.monitorware.com/en/logsamples/apache.php)

1. The first two lines of code use the built-in function input() to read a line from input. The strings provided in the input() functions are the prompt messages written to standard output for the users. After the user types the two filenames, the two strings “access_log” and “access_log_401” will be stored in the two variables “from_file” and “to_file” respectively.
2. Lines 3 and 4 then use the built-in function open() to open the two files with names specified in the variables “from_file” and “to_file” respectively. The open() functions return corresponding file objects to be stored in the two variables “in_file” and “out_file” respectively. The argument ‘w’ means opening the text file for writing, truncating the file first.
3. Lines 6-8 go through the “in_file” file object line by line. If a single line contains a string “ 401 ”, the HTTP code of the target error, this line is written into the file object “out_file” using its write() method.
4. Lines 11-12 close two file objects lastly using the file objects’ close() method. It is important to close files when we are done with them.

5. Execute the program SU4-file.py in PowerShell/Terminal. The output is like below. Note that the program waits for the two inputs of filenames “access_log” and “access_log_401” respectively.

```
$ python3 SU4-file.py
Type the filename of the full access log: access_log
Type the filename to store the 401 error log: access_log_401
Checking 401 error done.
```

Watch

Watch the following three videos of the textbook for more details and examples on working with files:

Video for Exercise 15: Reading Files

Video for Exercise 16: Reading and Writing Files

Video for Exercise 17: More Files

Read

Read the following three exercises of the textbook for more details and examples on working with files:

Exercise 15: Reading Files

Exercise 16: Reading and Writing Files

Exercise 17: More Files

Chapter 3: Packages

To help organise modules and provide a naming hierarchy, Python has a concept of packages. You can think of package as a directory of Python scripts. Each such script is a so-called module. These modules specify functions, methods and new Python types for solving particular tasks. Like file system directories, packages are organised hierarchically, and packages may themselves contain sub-packages, as well as regular modules.

3.1 The Standard Library

The packages of the standard library are installed along with the Python environment. But before we can use the modules in our program, we should first import the package or a specific module of the package.

The code in Figure 4.6 gives an example of using the datetime module, which is mainly for manipulating dates and times.

```
>>> import datetime
>>> now = datetime.date.today()
>>> print(now.strftime("Today is %d %b %Y, %A."))
Today is 21 Mar 2018, Wednesday.
>>> birthday = datetime.date(1964, 7, 31)
>>> age = now - birthday
>>> print(f"You are {age.days//365} years old.")
You are 53 years old.
```

Figure 4.6 Using the standard library datetime

1. We must first import the datetime module.
2. We then use the method date.today() to return the current local date. Simply calling the method like date.today() will generate an error. We must refer to the used function or method from its package and/or modules: datetime.date.today().

3. Next is to use the date object's strftime() method to print a string representing the date, controlled by an explicit format string provided in the parenthesis. %d %b %Y %A are formatting directives with their meanings explained in Table 4.1.
4. The difference between two dates as in "now - birthday" is a timedelta object, stored in the variable age. It has an attribute "days" for representing this duration in days. We further divide it by 365 to get the age in years.

Table 4.1 The meaning of formatting directives used in Figure 4.6

Directive	Meaning	Example
%d	Day of the month as a zero-padded decimal number.	01, 02, …, 31
%b	Month as locale's abbreviated name.	an, Feb, …, Dec (en_US)
%Y	Year with century as a decimal number.	0001, 0002, …, 2013, 2014, …, 9998, 9999
%A	Weekday as locale's full name.	Sunday, Monday, …, Saturday (en_US)

Read

Refer to the link below for more details and examples on the basic types and methods in the datetime module: https://docs.python.org/3/library/datetime.html#module-datetime

Note: There are cases in which we only need one specific function of a package/module. Suppose we only want to use the date() function from the datetime module. You can

use “from datetime import date” and simply call like date.today(). No need to use the “datetime.” prefix.

Note: You may extend the import statement by given an alias to the imported package/module. For example, “import numpy as np” will require your code to use the alias “np” in your code.

3.2 Managing Packages with pip3

There are many Python packages available from the internet but not installed along with the Python environment. To use those Python packages, you'll first have to install them on your system. Then you will be able to import them in the same as the standard library.

The packages numpy, matplotlib and pandas are to be used in Study Unit 5 and Study Unit 6. We will use pip3, a package maintenance system for Python 3, to install them. Suppose we want to install the numpy package. Go to the PowerShell/terminal, and execute `pip3 install numpy`. Then you will be able to import them in the same as the standard library.

Summary

We have learned about functions, methods and packages in Python. Methods are actually functions specific to Python objects. Depending on the type of the Python object, the methods applicable are different and behave differently. Packages and modules organise the program for standard tasks into so-called Python library. We practise with examples of applying the Python built-in functions, user-defined functions, the Python built-in types and associated methods, and the Python standard library.

Formative Assessment

1. What is a Python function?
    a. an alternative to Python program
    b. an alternative to Python variables
    c. a data type similar to dictionary
    d. a piece of reusable Python code for a particular task

2. Which Python code should you use to find the minimum value in a Python list named x?
    a. min x
    b. min[x]
    c. min = x
    d. min(x)

3. What Python built-in function opens up the documentation from inside the Python interpreter for the min() function?
    a. input(min)
    b. help(min())
    c. input(min())
    d. help(min)

4. The built-in function round() has two arguments. Check the relevant Python documentation and select the correct statement.
    a. number is a required argument. ndigits is an optional argument.
    b. number is an optional argument. ndigits is a required argument
    c. number is a required argument. ndigits is a required argument.
    d. number is an optional argument. ndigits is an optional argument.

5. Which Python code should you use to capitalize the string x = "happy new year!"?
    a. capitalize(x)
    b. capitalize x
    c. x.capitalize
    d. x.capitalize()

6. Study the Python documentation about methods of list objects. What is the output after the following code is executed?

```
x = [4, 9, 5, 7]
x.append(6)
x
```

    a. [6,4,9,5,7]
    b. [6]
    c. [4,9,5,7]
    d. [4,9,5,7,6]

7. Which of the following is the command for installation and maintenance system for Python 3?
    a. pip
    b. pip3
    c. module
    d. install.

8. Which statement is the most common way to import a package or module?
    a. as
    b. from
    c. class
    d. import

9. Which Python code below is valid to use the array() function from the NumPy package, if it is imported using "import numpy as ny"?
    a. numpy.array([1, 2, 3])
    b. ny.numpy.array([1, 2, 3])
    c. numpy.ny.array([1, 2, 3])
    d. ny.array([1, 2, 3])

10. You want to use the datetime module's date class. You need to decide the way to import it. Search and study the necessary Python documentation and select the correct statement.
    a. The "from datetime import date" way will make the code less clear that you're using datetime's date().
    b. Using "from datetime import date" will require you to use datetime.date() for creating a date object.
    c. Using "import datetime as dt" will enable you to directly use date() for creating a date object.
    d. Using "import datetime" will enable you to directly use date() for creating a date object.

Solutions or Suggested Answers

Formative Assessment

1. What is a Python function?

    a. an alternative to Python program

    Incorrect. Refer to Chapter 1 on functions.

    b. an alternative to Python variables

    Incorrect. Refer to Chapter 1 on functions.

    c. a data type similar to dictionary

    Incorrect. Refer to Chapter 1 on functions.

    d. a piece of reusable Python code for a particular task

    Correct.

2. Which Python code should you use to find the minimum value in a Python list named x?

    a. min x

    Incorrect. Refer to Chapter 1 on built-in functions.

    b. min[x]

    Incorrect. Refer to Chapter 1 on built-in functions.

    c. min = x

    Incorrect. Refer to Chapter 1 on built-in functions.

    d. min(x)

    Correct.

3. What Python built-in function opens up the documentation from inside the Python interpreter for the min() function?

a. input(min)
   Incorrect. Refer to Chapter 1 on built-in functions.

b. help(min())
   Incorrect. Refer to Chapter 1 on built-in functions.

c. input(min())
   Incorrect. Refer to Chapter 1 on built-in functions.

d. help(min)
   Correct.

4. The built-in function round() has two arguments. Check the relevant Python documentation and select the correct statement.

   a. number is a required argument. ndigits is an optional argument.
      Correct.

   b. number is an optional argument. ndigits is a required argument
      Incorrect.

   c. number is a required argument. ndigits is a required argument.
      Incorrect.

   d. number is an optional argument. ndigits is an optional argument.
      Incorrect.

5. Which Python code should you use to capitalize the string x = “happy new year!”?

   a. capitalize(x)
      Incorrect. Refer to Chapter 2 on methods.

   b. capitalize x
      Incorrect. Refer to Chapter 2 on methods.

   c. x.capitalize

Incorrect. Refer to Chapter 2 on methods.

d. x.capitalize()

Correct.

6. Study the Python documentation about methods of list objects. What is the output after the following code is executed?

```
x = [4, 9, 5, 7]
x.append(6)
x
```

a. [6,4,9,5,7]

Incorrect. Refer to Chapter 2 on methods.

b. [6]

Incorrect. Refer to Chapter 2 on methods.

c. [4,9,5,7]

Incorrect. Refer to Chapter 2 on methods.

d. [4,9,5,7,6]

Correct.

7. Which of the following is the command for installation and maintenance system for Python 3?

a. pip

Incorrect. Refer to Section 3.2 on managing packages.

b. pip3

Correct.

c. module

Incorrect. Refer to Section 3.2 on managing packages.

d. install.

Incorrect. Refer to Section 3.2 on managing packages.

8. Which statement is the most common way to import a package or module?

   a. as

      Incorrect. Refer to Section 3.2 on managing package.

   b. from

      Incorrect. Refer to Section 3.2 on managing package.

   c. class

      Incorrect. Refer to Section 3.2 on managing package.

   d. import

      Correct.

9. Which Python code below is valid to use the array() function from the NumPy package, if it is imported using "import numpy as ny"?

   a. numpy.array([1, 2, 3])

      Incorrect. Refer to Section 3.2 on managing package.

   b. ny.numpy.array([1, 2, 3])

      Incorrect. Refer to Section 3.2 on managing package.

   c. numpy.ny.array([1, 2, 3])

      Incorrect. Refer to Section 3.2 on managing package.

   d. ny.array([1, 2, 3])

      Correct.

10. You want to use the datetime module's date class. You need to decide the way to import it. Search and study the necessary Python documentation and select the correct statement.

a. The “from datetime import date” way will make the code less clear that you're using datetime's date().
Correct.

b. Using “from datetime import date” will require you to use datetime.date() for creating a date object.
Incorrect. Refer to Section 3.2 on managing package.

c. Using “import datetime as dt” will enable you to directly use date() for creating a date object.
Incorrect. Refer to Section 3.2 on managing package.

d. Using “import datetime” will enable you to directly use date() for creating a date object.
Incorrect. Refer to Section 3.2 on managing package.

References

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

Apache (Unix) log samples. (n.d.). Retrieved from http://www.monitorware.com/en/logsamples/apache.php

Built-in functions. (n.d.). Retrieved from https://docs.python.org/3/library/functions.html

Built-in types. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html

datetime – Basic date and time types. (n.d.). Retrieved from https://docs.python.org/3/library/datetime.html#module-datetime

Reading and writing files. (n.d.). Retrieved from https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

Study Unit 5

Scientific Computing and Plotting with Python

Learning Outcomes

By the end of this unit, you should be able to:

1. Use NumPy arrays and vectorised operations
2. Implement subsetting NumPy arrays using index or Boolean mask
3. Explain the basics of NumPy array attributes
4. Apply NumPy functions for statistics and random sampling
5. Create basic plots and choose appropriate matplotlib.pyplot functions for customisation

Overview

This study unit introduces two Python packages: NumPy and Matplotlib.

- NumPy is the fundamental package for efficient scientific computing with Python. We will learn subsetting and vectorised operations in NumPy arrays. We will also learn the NumPy functions for statistics and random sampling.
- This unit also introduces data visualisation basics, with a focus on charting using the matplotlib.pyplot package. We will learn the functionality available for plotting and customising basic statistical charts.

Chapter 1: NumPy

We have seen in Study Unit 2 that the Python list can hold any type and can hold different types at the same time. We can also change, add and remove elements. But when analysing data, we'll often want to carry out operations over entire collections of values. For example, there is a list dataList = [1.2, 3.5, 5.1], and we want to double every element. With lists, we could solve this by going through each list element one after the other, and doubling each element separately, but this is inefficient. A way more elegant is to use the Python package NumPy. The NumPy array is similar to a Python list, but additionally can perform calculations over entire arrays.

1.1 Creating NumPy Arrays

Note: Open the command line and execute `pip3 install numpy` if we haven't installed the NumPy library.

The code in Figure 5.1 applies arithmetic operations on an array elementwise.

1. Note that the first line of code extends the import statement with the "as". Using this numpy dot prefix all the time can be tiring especially when the package name is long. So, we would prefer importing the package and referring to it with a shorter name. Now, instead of numpy.array(), we can use np.array() to use Numpy's function.
2. We first use the Numpy function array() to create an array from a regular Python list.
3. Next is to double every element in the array. Note that the operation *= acts in place to modify an existing array rather than create a new one.
4. The calculations were performed element-wise as the output in Figure 5.1.

```
>>> import numpy as np
>>> dataList = [1.2, 3.5, 5.1]
>>> dataArray = np.array(dataList)
>>> dataArray *= 2
>>> dataArray
array([ 2.4,  7. , 10.2])
```

Figure 5.1 Applying arithmetic operators on arrays elementwise

Note: First of all, NumPy can do this elementwise operation because it assumes that NumPy arrays can only contain values of a single type. Second, Python list and NumPy array can behave differently when being applied the same operator. For example, if we apply '+' operation to two lists, the list elements are pasted together, generating a list including all the elements from both lists. If we apply '+' operation to two NumPy arrays, Python will do an element-wise sum of the two arrays.

1.2 Subsetting

We can use square brackets to get elements from NumPy array, same as Python List. There are two ways of subsetting as in Figure 5.2.

- Using index: Suppose we want to get the second value, so at index 1. The index starts with 0, same as Python List.
- Using Boolean masking: Suppose we want to get all values that are over 5. A first step is using 'dataArray > 5' to produce a Boolean mask. The result is a NumPy array containing booleans: True if the corresponding value is above 5, False if it's below. Next, we can use this boolean mask inside square brackets to do subsetting. Only the elements that are above 5, so for which the corresponding boolean mask is True, are selected. There are two values above 5, so we end up with a NumPy array with two values.

```
>>> dataArray[1]
7.0
>>> dataArray[dataArray>5]
array([ 7. , 10.2])
```

Figure 5.2 Subsetting NumPy array using index or Boolean masking

1.3 Creating and Subsetting 2-D NumPy Arrays

The array created in Figure 5.1 is a one-dimensional array, but the array() function is able to create two dimensional, three dimensional, …, n dimensional arrays. The array() function transforms sequences of sequences into two-dimensional arrays, sequences of sequences of sequences into three-dimensional arrays, and so on.

We can create a 2D NumPy array from a regular Python list of lists as in Figure 5.3.

1. If we print it out, it is a rectangular data structure: Each sublist in the list, corresponds to a row in the two dimensional NumPy array.
2. Next, we check three attributes of the array:
    - shape: The output shows 2 rows and 5 columns. The shape attribute returns a tuple of array dimensions.
    - ndim: The ndim attribute returns the number of array dimensions.
    - size: The size attribute returns the number of elements in the array.

```
>>> dataArray2d = np.array([[1.5,2,3], [4,5,6]])
>>> dataArray2d
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
>>> dataArray2d.shape
(2, 3)
>>> dataArray2d.ndim
2
>>> dataArray2d.size
6
```

Figure 5.3 2-D Numpy Array and its attributes

Read

Refer to the three links below for more details and examples on the attributes 'shape', 'ndim' and 'size' of Numpy arrays: https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html#numpy.ndarray.shape

https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.ndim.html#numpy.ndarray.ndim

https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.size.html#numpy.ndarray.size

Subsetting on 2-D arrays uses square brackets with indexes for each dimension separated by commas. The intersections of the elements at the specified rows and columns' indexes are returned.

Let's go through the examples in Figure 5.4.

```
>>> dataArray2d
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
>>> dataArray2d[0]
array([1.5, 2. , 3. ])
>>> dataArray2d[:,1]
array([2., 5.])
>>> dataArray2d[0,2]
3.0
>>> dataArray2d[:,1:]
array([[2., 3.],
       [5., 6.]])
```

Figure 5.4 Subsetting 2-D Numpy array

- dataArray[0] only specifies the index 0 for the first dimension. Hence, the output selects all the elements at the first dimension's index 0, namely the entire first row is selected.
- In dataArray2d[:,1], the first dimension uses :, meaning selecting all elements in this dimension, namely all the rows are selected. The second dimension specifies index 1, namely the entire second column is selected to output.
- In dataArray2d[0,2], the first dimension specifies an index 0 and the second dimension's index 2. Hence the element at the intersection is selected to output.
- In dataArray2d[:,1:], the first dimension uses :, meaning selecting all elements in this dimension, namely all the rows are selected. The second dimension specifies from index 1 onwards, namely a new array slicing the entire second and third columns is selected to output.

1.4 Working with NumPy Functions

A typical step in analysing data is generating summary statistics about the data. Let's go through the examples in Figure 5.5.

```
>>> dataArray2d
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
>>> np.mean(dataArray2d, axis=0)
array([2.75, 3.5 , 4.5 ])
>>> np.median(dataArray2d,axis=1)
array([2., 5.])
>>> np.std(dataArray2d,axis=0)
array([1.25, 1.5 , 1.5 ])
```

Figure 5.5 Calculating summary statistics with 2-D Numpy array

- The NumPy's mean() function computes the arithmetic mean along the specified axis. A value of 0 for the axis parameter means the calculation is done along the first dimension/axis, namely along the rows. Because it's a function from the NumPy package, don't forget to start with "np.".
- The NumPy's median() function computes the median along the specified axis. A value of 1 for the axis parameter means the calculation is done along the second dimension/axis, namely along the columns.
- The NumPy's std() function computes the standard deviation along the specified axis. A value of 0 for the axis parameter means the calculation is done along the first dimension/axis, namely along the rows.

Read

Refer to the three links below for more details and examples on the functions mean(), median() and std() of the Numpy package:

https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html#numpy.mean

https://docs.scipy.org/doc/numpy/reference/generated/numpy.median.html#numpy.median

https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html#numpy.std

The numpy.random package provides a list of functions for random sampling. For example, the normal() function draws random samples from a normal (Gaussian) distribution as the example in Figure 5.6.

```
>>> s = np.random.normal(0, 1, 30)
>>> s
array([-1.77874906,  0.79392717,  0.3091446 , -0.45412194, -0.31126316,
       -0.50208389,  0.60512056,  0.50429048,  0.76910564,  0.10605086,
       -0.53457312, -0.73926783,  0.94680667,  0.85416046, -0.36148315,
       -0.67602936, -1.76778911,  2.02217648, -0.47440458, -0.75706189,
       -1.50460605, -0.65006415, -0.28369107, -0.90238697, -0.57307728,
       -0.59031231, -0.74841335, -0.02706934, -0.73075724,  1.62148887])
>>> np.mean(s)
-0.194497768523559
>>> np.std(s)
0.8864549099088929
```

Figure 5.6 Generating 30 random samples from standard normal distribution

1. In the normal() function, the first and second parameters specify the mean and standard deviation respectively. The third parameter specifies the number of samples to be drawn.

2. You may verify the mean and standard deviation of the generated random samples using the mean() and std() functions. They are close to the values of the parameters given in the normal() function.

Read

Refer to the link below for more details and examples on the function normal() of the numpy.random package:

https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.normal.html#numpy.random.normal

Activity 1

Create a Python file to reproduce the codes in this chapter on NumPy. Make sure that you can understand the code and output.

Chapter 2: Matplotlib

This chapter is about data visualisation. The mother of all visualisation packages in python is matplotlib, inside which there's the pyplot subpackage.

2.1 Basic Plots

There are different plot types. For each plot, we can do many customisations, such as change colours, shapes, labels, axes, and so on. Let's start with the code in Figure 5.7 to build a simple line plot with several customisations.

```
>>> import matplotlib.pyplot as plt
>>> x = [1800,1850,1900, 1950, 1970, 1990, 2010]
>>> y = [1.0, 1.262, 1.650, 2.519, 3.692, 5.263, 6.972]
>>> plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x109ec6a58>]
>>> plt.xlabel('Year')
Text(0.5,0,'Year')
>>> plt.ylabel('Population')
Text(0,0.5,'Population')
>>> plt.title('World Population Projections')
Text(0.5,1,'World Population Projections')
>>> plt.yticks([0,2,4,6,8,10], ['0','2B','4B','6B','8B','10B'])
([<matplotlib.axis.YTick object at 0x109eb60b8>, <matplotlib.axis.YTick object a
t 0x109eb1748>, <matplotlib.axis.YTick object at 0x10c7974e0>, <matplotlib.axis.
YTick object at 0x10c7979b0>, <matplotlib.axis.YTick object at 0x10c797e80>, <ma
tplotlib.axis.YTick object at 0x10c7a13c8>], <a list of 6 Text yticklabel object
s>)
>>> plt.show()
```

Figure 5.7 Building a line plot with customised labels, title and axis ticks

1. Let's start with importing matplotlib.pyplot subpackage. By convention we give it the local name `plt`.
2. To plot the two Python lists in a line chart, we call plt.plot() function and use our two lists as arguments. The first list corresponds to the horizontal axis, and the second list to the vertical axis.
3. The xlabel() and ylabel() functions set the x and y axis labels. The title() function then sets a title of the current figure.

4. The yticks() function sets the y axis with the first argument for tick locations and the second argument for tick labels. The tick labels are necessary because it would be clearer to show the letter B in ticks, to show that the numbers are in billions.
5. It will wait for the show() function to actually display the plot as in Figure 5.8 and block until the figure has been closed.

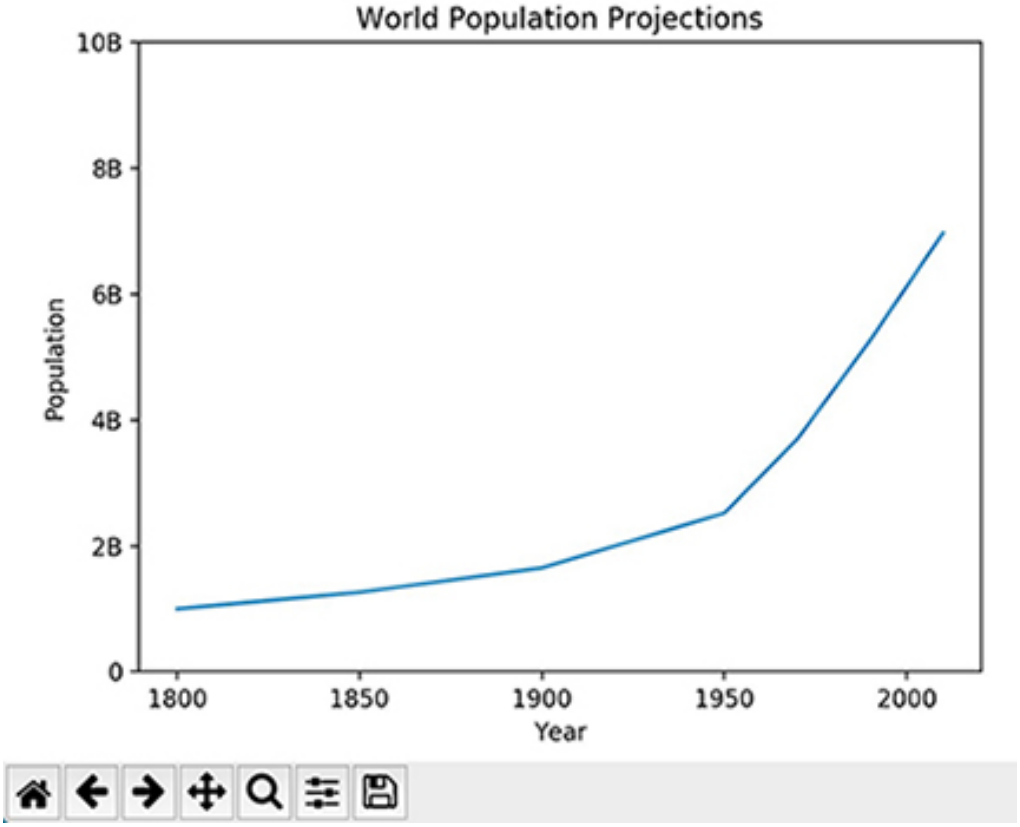

Figure 5.8 A line plot produced with the code in Figure 5.7

Note: Each matplotlib.pyplot function makes some change to a figure: e.g., creating a figure, creating a plotting area in a figure, plotting some lines in a plotting area, decorating the plot with labels, etc. Various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and then the plotting functions are directed to the current part of a figure.

Read

Refer to the three links below for more details and examples on the functions plot(), xticks() and yticks() of the matplotlib.pyplot package:

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

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xticks.html#matplotlib.pyplot.xticks

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.yticks.html#matplotlib.pyplot.yticks

Watch

Legends, titles and labels. A matplotlib tutorial from sentdex

(https://www.youtube.com/watch?v=aCULcv_IQYw)

2.2 Histograms

The histogram can help get an idea about the distribution of the variables. The code in Figure 5.9 shows how it works.

```
>>> import numpy as np
>>> x = np.random.normal(100, 15, 5000)
>>> plt.hist(x, 30)
(array([  2.,   5.,   9.,  17.,  34.,  43.,  66.,  87., 141., 206., 295.,
       301., 358., 418., 490., 417., 439., 406., 332., 281., 196., 168.,
       109.,  76.,  48.,  29.,  13.,   4.,   6.,   4.]), array([ 47.76088346,  5
1.21737856,  54.67387366,  58.13036876,
        61.58686386,  65.04335896,  68.49985406,  71.95634916,
        75.41284426,  78.86933937,  82.32583447,  85.78232957,
        89.23882467,  92.69531977,  96.15181487,  99.60830997,
       103.06480507, 106.52130017, 109.97779528, 113.43429038,
       116.89078548, 120.34728058, 123.80377568, 127.26027078,
       130.71676588, 134.17326098, 137.62975608, 141.08625119,
       144.54274629, 147.99924139, 151.45573649]), <a list of 30 Patch objects>)
>>> plt.xlabel('Values')
Text(0.5,0,'Values')
>>> plt.ylabel('Probability')
Text(0,0.5,'Probability')
>>> plt.title('Histogram of Values')
Text(0.5,1,'Histogram of Values')
>>> plt.show()
```

Figure 5.9 Building a histogram of random samples from a normal distribution

1. We first use the normal() function in the numpy.random to generate 5000 random samples from a normal distribution with mean 100 and standard deviation 15.
2. Next we can use the hist() function. The first argument is the array of values that we want to build a histogram for. The second argument is to specify in how many bins the data should be divided. Based on this number, hist() function will find appropriate boundaries for all the bins and calculate how many values are in each one.
3. Finally we customise the xlabel, ylabel, title and call the show() function. A histogram as in Figure 5.10 is produced.

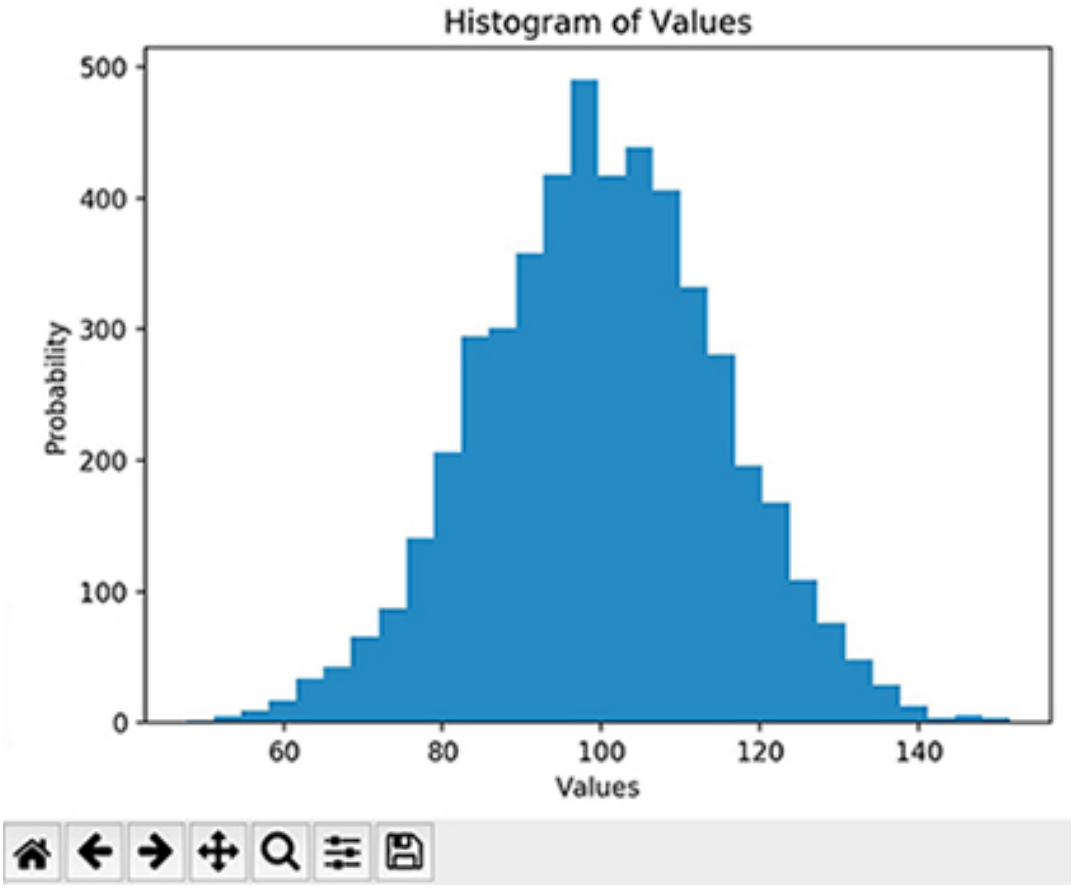

Figure 5.10 A histogram produced with the code in Figure 5.9

Activity 2

Create a Python file to reproduce the codes in this chapter on Matplotlib. Make sure that you can understand the code and output.

Summary

We have seen two Python packages NumPy and Matplotlib in this study unit. NumPy is the fundamental package for efficient scientific computing with Python, and Matplotlib is the fundamental package for visualisation with Python. We have learned the basics of the two packages such as subsetting and vectorised operations in NumPy arrays, the NumPy functions for statistics and random sampling, the matplotlib.pyplot functions for plotting and customising basic statistical charts.

Formative Assessment

1. Which NumPy function do we use to create an array?
    a. arr()
    b. NumPy()
    c. array()
    d. np()

2. What is the resulting array z after executing the following code?

```
import numpy as np
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = x + y
```

    a. array([4, 4, 4])
    b. array([1, 2, 3, 3, 2, 1])
    c. array([2, 4, 6])
    d. array([13, 22, 31])

3. The following code creates an NumPy array, x. Which Python code do we use to select the string "g" from x?

```
import numpy as np
x = np.array([["a", "b", "c", "d"], ["e", "f", "g", "h"]])
```

    a. x[-1, 1]
    b. x[0][1]
    c. x[1, 2]
    d. x[0, 2]

4. What does the resulting array z contain after executing the following code?

```
import numpy as np
x = np.array([[1, 2, 3], [1, 2, 3]])
y = np.array([[1, 1, 1], [1, 2, 3]])
z = x - y
```

a. array([[1, 2, 3],

   [1, 2, 3],

   [1, 1, 1],

   [1, 2, 3]])

b. array([[0, 1, 2],

   [0, 0, 0]])

c. array([[0, 0, 0],

   [0, 1, 2]])

d. A 2-by-3 NumPy array with only zeros.

5. Which Python code do we use to calculate the average of the second column of the array x generated in the following code?

```
import numpy as np
x = np.array([[28, 18],
[34, 14],
[32, 16],
[26, 23],
[23, 17]])
```

a. np.mean(x[:,0])

b. np.mean(x[0,:])

c. np.mean(x[1,:])

d. np.mean(x[:,1])

6. What is the conventional way of importing the pyplot subpackage from the matplotlib package?
    a. import matplotlib.pyplot as plt
    b. import pyplot as plt
    c. import matplotlib as plt
    d. import plt

7. Which option correctly describes the result of the following code?

```
import matplotlib.pyplot as plt
a = [1, 2, 3, 4]
b = [3, 9, 2, 6]
plt.plot(a, b)
plt.show()
```

    a. The values in *a* and *b* are summed in an element-wise manner and mapped onto the vertical axis.
    b. The values in *a* and *b* are summed in an element-wise manner and mapped onto the horizontal axis.
    c. The values in *a* are mapped onto the vertical axis. The values in *b* are mapped onto the horizontal axis.
    d. The values in *a* are mapped onto the horizontal axis. The values in *b* are mapped onto the vertical axis.

8. What is a characteristic of a histogram?
    a. Histogram is only useful if the data is normally distributed.
    b. Histogram is useful for getting a first impression about the distribution of the data.
    c. Histogram has bins that partially overlap, making it possible for a single value to lie in two bins.
    d. Histogram is useful to visualise trend in a series of time.

9. How to extend the following code to specifically set the number of histogram bins to 4?

```
import matplotlib.pyplot as plt
x = [1, 3, 6, 3, 2, 7, 3, 9, 7, 5, 2, 4]
plt.hist(x)
plt.show()
```

   a. Add after plt.hist() and before plt.show(): plt.binsize(4)
   b. Change plt.hist(x) to plt.hist(x, range = max(x) - min(x))
   c. Change plt.hist(x) to plt.hist(x, 4)
   d. No changes. The number of bins are default to 4.

10. Which code to use for customising a plot by labelling its axis? Assume we have executed “import matplotlib.pyplot as plt”.
   a. plt.labelling(x = "x-axis title", y = "y-axis title")
   b. plt.plot(lab = ["x-axis title", "y-axis title"])
   c. plt.title("x-axis title", "y-axis title")
   d. plt.xlabel("x-axis title") and plt.ylabel("y-axis title")

Solutions or Suggested Answers

Formative Assessment

1. Which NumPy function do we use to create an array?

    a. arr()

    Incorrect. Refer to Section 1.1 on creating NumPy arrays.

    b. NumPy()

    Incorrect. Refer to Section 1.1 on creating NumPy arrays.

    c. array()

    Correct.

    d. np()

    Incorrect. Refer to Section 1.1 on creating NumPy arrays.

2. What is the resulting array z after executing the following code?

```
import numpy as np
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = x + y
```

    a. array([4, 4, 4])

    Correct.

    b. array([1, 2, 3, 3, 2, 1])

    Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

    c. array([2, 4, 6])

    Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

    d. array([13, 22, 31])

Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

3. The following code creates an NumPy array, x. Which Python code do we use to select the string "g" from x?

```
import numpy as np
x = np.array([["a", "b", "c", "d"], ["e", "f", "g", "h"]])
```

   a. x[-1, 1]

      Incorrect. Refer to Section 1.2 on subsetting NumPy arrays.

   b. x[0][1]

      Incorrect. Refer to Section 1.2 on subsetting NumPy arrays.

   c. x[1, 2]

      Correct.

   d. x[0, 2]

      Incorrect. Refer to Section 1.2 on subsetting NumPy arrays.

4. What does the resulting array z contain after executing the following code?

```
import numpy as np
x = np.array([[1, 2, 3], [1, 2, 3]])
y = np.array([[1, 1, 1], [1, 2, 3]])
z = x - y
```

   a. array([[1, 2, 3],

      [1, 2, 3],

      [1, 1, 1],

      [1, 2, 3]])

Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

b. array([[0, 1, 2],

[0, 0, 0]])

Correct.

c. array([[0, 0, 0],

[0, 1, 2]])

Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

d. A 2-by-3 NumPy array with only zeros.
Incorrect. Refer to Section 1.1 on applying operations on arrays elementwise.

5. Which Python code do we use to calculate the average of the second column of the array x generated in the following code?

```
import numpy as np
x = np.array([[28, 18],
[34, 14],
[32, 16],
[26, 23],
[23, 17]])
```

a. np.mean(x[:,0])
Incorrect. Refer to Section 1.4 on basic NumPy statistics functions.

b. np.mean(x[0,:])
Incorrect. Refer to Section 1.4 on basic NumPy statistics functions.

c. np.mean(x[1,:])
Incorrect. Refer to Section 1.4 on basic NumPy statistics functions.

d. np.mean(x[:,1])

Correct.

6. What is the conventional way of importing the pyplot subpackage from the matplotlib package?

   a. import matplotlib.pyplot as plt

      Correct.

   b. import pyplot as plt

      Incorrect. Refer to Section 2.1.

   c. import matplotlib as plt

      Incorrect. Refer to Section 2.1.

   d. import plt

      Incorrect. Refer to Section 2.1.

7. Which option correctly describes the result of the following code?

```
import matplotlib.pyplot as plt
a = [1, 2, 3, 4]
b = [3, 9, 2, 6]
plt.plot(a, b)
plt.show()
```

   a. The values in *a* and *b* are summed in an element-wise manner and mapped onto the vertical axis.

      Incorrect. Refer to Section 2.1 on basic plotting.

   b. The values in *a* and *b* are summed in an element-wise manner and mapped onto the horizontal axis.

      Incorrect. Refer to Section 2.1 on basic plotting.

   c. The values in *a* are mapped onto the vertical axis. The values in *b* are mapped onto the horizontal axis.

Incorrect. Refer to Section 2.1 on basic plotting.

d. The values in *a* are mapped onto the horizontal axis. The values in *b* are mapped onto the vertical axis.
Correct.

8. What is a characteristic of a histogram?
   a. Histogram is only useful if the data is normally distributed.
   Incorrect. Refer to Section 2.2 on histograms.

   b. Histogram is useful for getting a first impression about the distribution of the data.
   Correct.

   c. Histogram has bins that partially overlap, making it possible for a single value to lie in two bins.
   Incorrect. Refer to Section 2.2 on histograms.

   d. Histogram is useful to visualise trend in a series of time.
   Incorrect. Refer to Section 2.2 on histograms.

9. How to extend the following code to specifically set the number of histogram bins to 4?

```
import matplotlib.pyplot as plt
x = [1, 3, 6, 3, 2, 7, 3, 9, 7, 5, 2, 4]
plt.hist(x)
plt.show()
```

   a. Add after plt.hist() and before plt.show(): plt.binsize(4)
   Incorrect. Refer to Section 2.2 on histograms.

   b. Change plt.hist(x) to plt.hist(x, range = max(x) - min(x))

Incorrect. Refer to Section 2.2 on histograms.

c. Change plt.hist(x) to plt.hist(x, 4)
Correct.

d. No changes. The number of bins are default to 4.
Incorrect. Refer to Section 2.2 on histograms.

10. Which code to use for customising a plot by labelling its axis? Assume we have executed "import matplotlib.pyplot as plt".

a. plt.labelling(x = "x-axis title", y = "y-axis title")
Incorrect. Refer to Section 2.1 on basic plotting.

b. plt.plot(lab = ["x-axis title", "y-axis title"])
Incorrect. Refer to Section 2.1 on basic plotting.

c. plt.title("x-axis title", "y-axis title")
Incorrect. Refer to Section 2.1 on basic plotting.

d. plt.xlabel("x-axis title") and plt.ylabel("y-axis title")
Correct.

References

Historical stock data for Google. (n.d.). Retrieved from https://www.kaggle.com/szrlee/stock-time-series-20050101-to-20171231

McKinney, W. (n.d.). 10-minute tour of Pandas. Retrieved from https://vimeo.com/59324550

Pandas API reference. (n.d.). Retrieved from http://pandas.pydata.org/pandas-docs/stable/api.html

Worldwide food production. (n.d.). Retrieved from http://www.fao.org/faostat/en/#home

Study Unit 6

Data Analysis with Python

Learning Outcomes

By the end of this unit, you should be able to:

1. Use appropriate Pandas functions to import datasets into DataFrame structures
2. Apply indexing, sorting and selection to DataFrames, and inspect missing values in DataFrames
3. Analyse data by grouping and aggregation
4. Prepare and examine time series data
5. Construct basic statistical charts.

Overview

This unit introduces the Pandas DataFrame, the key data structure for data analysis in Python. You'll learn how to import data into DataFrame structures, how to index, sort and query these structures. You'll then deepen your understanding of the python Pandas library by learning how to group and aggregate data. Lastly, this unit introduces Pandas's efficient functionality for performing time series analysis.

Chapter 1: DataFrame

You have learned about numpy ndarrays in Study Unit 5 to store data in an n-dimensional structure. But numpy arrays can only have data of the same type in there. In practice, you work with data of different types: numerical values, strings, Booleans and so on. The Pandas package is a high-level data manipulation tool to efficiently handle this.

Watch

10-minute tour of Pandas from Wes McKinney (https://vimeo.com/59324550)

Dataset

Worldwide Food Production, 1961-2013 (FAO.csv)

We will be using this dataset in examples throughout this study unit. The dataset's attributes are:

- Area code - Country name abbreviation
- Area - County name
- Item - Food item
- Element - Food or Feed. Food refers to the food item available as human food. Feed refers to the food item available for feeding to the livestock and poultry.
- Latitude - geographic coordinate that specifies the north–south position of a point on the Earth's surface
- Longitude - geographic coordinate that specifies the east-west position of a point on the Earth's surface
- Year - the referred year
- Production - Amount of food item produced in 1000 tonnes during the referred year

This dataset was published by the Food and Agriculture Organization of the United Nations (http://www.fao.org/faostat/en/#home).

1.1 Introduction

In pandas, we store data in a so-called DataFrame. A DataFrame of the data set FAO.csv appears like a table as in Figure 6.1

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	Year	Production
0	AFG	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1961	1928.0
1	AFG	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1962	1904.0
2	AFG	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1963	1666.0
3	AFG	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1964	1950.0
4	AFG	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1965	2001.0

Figure 6.1 A DataFrame of FAO.csv with the first five rows shown

- The rows represent different entries or observations. Each row has unique row label or index.
- The columns represent different attributes, and are identified by their column labels: Area Abbreviation, Area Code, Area, Item Code, etc. The columns can have different types. The Area column is string, while the Production column is numeric, for example.

Note: When using numpy ndarrays to store multiple dimensional data, a burden is placed on the programmer to specify the orientation of the data set, because axes are considered more or less equivalent. In pandas, the axes lend more semantic meaning to the data, and hence reduce the amount of mental effort required to code up data transformation. For example, it is more semantically helpful to think of the index (the rows) and the columns with DataFrame rather than axis 0 and axis 1 as with ndarray.

1.2 Reading CSV Files

Note: Open the command line and execute `pip3 install pandas` if we haven't installed the Pandas library.

You typically don't build a pandas DataFrame manually. Instead, you import data from an external file, for example, a CSV (comma separated values) file. In Figure 6.2, the first line includes the column names and the other lines are the rows of the data in FAO.csv.

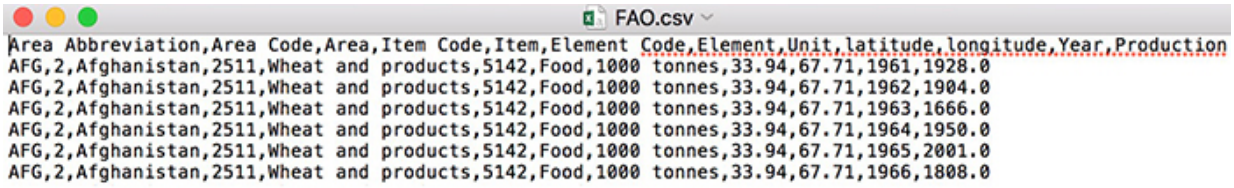

Figure 6.2 FAO.csv opened in a text editor

We start by importing the pandas packages as pd. You then use the read_csv() function with the path to your FAO.csv file as an argument. The head() method displays the first five rows of data. Figure 6.3 shows the code execution in Python interpreter.

```
>>> import pandas as pd
>>> df = pd.read_csv('FAO.csv')
>>> df.head()
  Area Abbreviation  Area Code         Area  Item Code                Item  \
0               AFG          2  Afghanistan       2511  Wheat and products
1               AFG          2  Afghanistan       2511  Wheat and products
2               AFG          2  Afghanistan       2511  Wheat and products
3               AFG          2  Afghanistan       2511  Wheat and products
4               AFG          2  Afghanistan       2511  Wheat and products

   Element Code Element         Unit  latitude  longitude  Year  Production
0          5142    Food  1000 tonnes     33.94      67.71  1961      1928.0
1          5142    Food  1000 tonnes     33.94      67.71  1962      1904.0
2          5142    Food  1000 tonnes     33.94      67.71  1963      1666.0
3          5142    Food  1000 tonnes     33.94      67.71  1964      1950.0
4          5142    Food  1000 tonnes     33.94      67.71  1965      2001.0
```

Figure 6.3 Getting CSV data into a DataFrame

In Figure 6.3, it would be more semantically helpful to set the row labels, also called row indexes, as certain columns in the data. To solve this, the method set_index() as in Figure 6.4 specifies the column 'Item' as the row index. Now the DataFrame df contains semantical row and column labels.

```
>>> df.set_index('Item',inplace=True)
>>> df.head()
                    Area Abbreviation  Area Code         Area  Item Code  \
Item
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511

                    Element Code Element         Unit  latitude  longitude  \
Item
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71

                    Year  Production
Item
Wheat and products  1961      1928.0
Wheat and products  1962      1904.0
Wheat and products  1963      1666.0
Wheat and products  1964      1950.0
Wheat and products  1965      2001.0
```

Figure 6.4 Setting index for a DataFrame

1.3 Sorting

The code in Figure 6.5 shows an example of sorting the column 'Production' in a descending order. From the output in Figure 6.5 we can see the top 5 entries China, mainland's vegetable production from Year 2009 to 2013.

```
>>> df.sort_values(by='Production',ascending=False).head()
           Area Abbreviation  Area Code             Area  Item Code  \
Item
Vegetables               CHN         41  China, mainland       2918
Vegetables               CHN         41  China, mainland       2918
Vegetables               CHN         41  China, mainland       2918
Vegetables               CHN         41  China, mainland       2918
Vegetables               CHN         41  China, mainland       2918

           Element Code Element         Unit  latitude  longitude  Year  \
Item
Vegetables         5142    Food  1000 tonnes     35.86      104.2  2013
Vegetables         5142    Food  1000 tonnes     35.86      104.2  2012
Vegetables         5142    Food  1000 tonnes     35.86      104.2  2011
Vegetables         5142    Food  1000 tonnes     35.86      104.2  2010
Vegetables         5142    Food  1000 tonnes     35.86      104.2  2009

           Production
Item
Vegetables    489299.0
Vegetables    479028.0
Vegetables    462696.0
Vegetables    451838.0
Vegetables    434724.0
```

Figure 6.5 Sorting the column 'Production' in a descending order

Read

Read the Pandas API reference

(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html#pandas.DataFrame.sort_values) for more details and examples on the function sort_values().

Chapter 2: Selection

This chapter introduces three ways to slice columns, rows or elements in a DataFrame:

- by label
- by position
- by Boolean masking

2.1 Column Access

We typically use the indexing operator (square brackets []) directly on a DataFrame for column selection. In a Panda's DataFrame, columns always have a name. So, this selection could always be label-based.

2.1.1 Selecting One Column

The code in Figure 6.6 gives an example of selecting the Area column of a DataFrame df. Notice that the index column 'Item' is also shown beside the selected column 'Area'.

```
>>> df['Area']
Item
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
```

Figure 6.6 Selecting one column using indexing operator

You can also use the dot notation for selecting one column as in Figure 6.7. The selected is treated as an attribute in this way. Notice that in this case, you don't use quotation marks on the column label.

```
>>> df.Area
Item
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
Wheat and products    Afghanistan
```

Figure 6.7 Selecting one column as an attribute

2.1.2 Selecting Multiple Columns

We could include multiple columns in a list and pass it into the indexing operator. Pandas will bring back only the columns asked for. The code in Figure 6.8 specifies three columns as a list ['Area','Year','Production'], which is then passed into the indexing operator [].

```
>>> df[['Area','Year','Production']]
                              Area  Year  Production
Item
Wheat and products  Afghanistan  1961      1928.0
Wheat and products  Afghanistan  1962      1904.0
Wheat and products  Afghanistan  1963      1666.0
Wheat and products  Afghanistan  1964      1950.0
Wheat and products  Afghanistan  1965      2001.0
```

Figure 6.8 Selecting multiple columns

2.1.3 Adding or Setting One Column

The example in Figure 6.9 makes a new column 'Raw Production', based on the column 'Production'. Because pandas is based on numpy, we can simply multiply the column 'Production' by 1000. We multiply by 1000 because the Unit column has a value 1000 tonnes for all the rows.

```
>>> df['Raw Production'] = df['Production'] * 1000
>>> df.head()
                    Area Abbreviation  Area Code         Area  Item Code  \
Item
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511
Wheat and products                AFG          2  Afghanistan       2511

                    Element Code Element         Unit  latitude  longitude  \
Item
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71
Wheat and products          5142    Food  1000 tonnes     33.94      67.71

                    Year  Production  Raw Production
Item
Wheat and products  1961      1928.0       1928000.0
Wheat and products  1962      1904.0       1904000.0
Wheat and products  1963      1666.0       1666000.0
Wheat and products  1964      1950.0       1950000.0
Wheat and products  1965      2001.0       2001000.0
```

Figure 6.9 Adding a new column

In the code in Figure 6.9, if the specified column on the left side of the '=' operator already exists, this column would be updated with the right-side value.

2.2 Row Access

To access rows, we need different ways from accessing columns, since pandas needs to know the specified labels are for rows instead of columns. Rows can be queried by either the index position or the index label. Hence, there are two DataFrame attributes used for row selection.

- loc - To query by the index label.
- iloc - To query by numeric position, starting at zero.

Note: As in Section 1.2 (Figure 6.3), if we don't set an index, the position and the label are of the same values.

2.2.1 loc

We use the DataFrame loc attribute and then the row index label of the row we want to access as a string inside the indexing operator []. The example code in Figure 6.10 finds out that the top 5 productions of 'Coffee and products' are in USA and the years of 2013, 1962, 1968, 1963 and 1964 respectively.

```
>>> df.loc['Coffee and products'].sort_values(by='Production',ascending=False).head()
                    Area Abbreviation  Area Code                      Area  \
Item
Coffee and products               USA        231  United States of America
Coffee and products               USA        231  United States of America
Coffee and products               USA        231  United States of America
Coffee and products               USA        231  United States of America
Coffee and products               USA        231  United States of America

                    Item Code  Element Code Element        Unit  latitude  \
Item
Coffee and products       2630          5142    Food  1000 tonnes     37.09
Coffee and products       2630          5142    Food  1000 tonnes     37.09
Coffee and products       2630          5142    Food  1000 tonnes     37.09
Coffee and products       2630          5142    Food  1000 tonnes     37.09
Coffee and products       2630          5142    Food  1000 tonnes     37.09

                    longitude  Year  Production  Raw Production
Item
Coffee and products     -95.71  2013      1367.0       1367000.0
Coffee and products     -95.71  1962      1356.0       1356000.0
Coffee and products     -95.71  1968      1351.0       1351000.0
Coffee and products     -95.71  1963      1340.0       1340000.0
Coffee and products     -95.71  1964      1338.0       1338000.0
```

Figure 6.10 Selecting rows by label

Note: Keep in mind that iloc and loc are not methods, they are attributes. So, don't use parentheses to query them, but the indexing operator (the square brackets []) instead.

Note: Similar to the column setting in Section 2.1, the loc attribute is also used to add new rows or update existing rows. If the row index label passed in doesn't exist, a new entry is added. Otherwise, the existing row is updated.

2.2.2 iloc

We use the DataFrame iloc attribute and then the row index position of the row we want to access inside the indexing operator []. Be reminded that the position starts at 0. The code in Figure 6.11 selects the third row. The result is the row information, displayed as a

column. The corresponding index label 'Wheat and products' for the index position 2 is also shown.

```
>>> df.iloc[2]
Area Abbreviation                   AFG
Area Code                             2
Area                        Afghanistan
Item Code                          2511
Element Code                       5142
Element                            Food
Unit                         1000 tonnes
latitude                           33.94
longitude                          67.71
Year                               1963
Production                         1666
Raw Production                1.666e+06
Name: Wheat and products, dtype: object
```

Figure 6.11 Selecting a row by position

2.3 Element Access

To select just elements in the DataFrame, you can specify both column and row labels in the loc function as the example in Figure 6.12. The output is consistent with that in Figure 6.10: The top 5 productions of 'Coffee and products' are in USA and the years of 2013, 1962, 1968, 1963 and 1964 respectively.

```
>>> df.loc['Coffee and products',['Area','Production','Year']].sort_values(by='Production',ascending=False)
                                        Area  Production  Year
Item
Coffee and products  United States of America      1367.0  2013
Coffee and products  United States of America      1356.0  1962
Coffee and products  United States of America      1351.0  1968
Coffee and products  United States of America      1340.0  1963
Coffee and products  United States of America      1338.0  1964
```

Figure 6.12 Selecting elements by specifying both column and row labels

2.4 Boolean Masking

A Boolean mask is an array where each of the values are is True or False. This array is overlaid on top of the data structure that we're querying. And any element aligned with a True value will be selected, and any element aligned with a False value will not.

For instance, in Figure 6.13 the code selects only those rows where the production is greater than 400,000.

1. We first build a Boolean mask df['Production']>400000 for this query. We project the Production column using the indexing operator and apply the greater than operator with a comparison value of 400,000.
2. Next is overlay that mask on the DataFrame. The indexing operator [] takes the Boolean mask df['Production']>400000 as a value.

Note: The Boolean mask df['Production']>400000 is essentially broadcasting a comparison operator, greater than. The results are returned where the value of each cell is either True or False.

```
>>> df[df['Production']>400000]
                Area Abbreviation  Area Code            Area  Item Code  \
Item
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918

                Element Code Element        Unit  latitude  longitude  \
Item
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2

                Year  Production  Raw Production
Item
Vegetables, Other  2011    402338.0     402338000.0
Vegetables, Other  2012    419262.0     419262000.0
Vegetables, Other  2013    426850.0     426850000.0
Vegetables         2007    402975.0     402975000.0
Vegetables         2008    425537.0     425537000.0
Vegetables         2009    434724.0     434724000.0
Vegetables         2010    451838.0     451838000.0
Vegetables         2011    462696.0     462696000.0
Vegetables         2012    479028.0     479028000.0
Vegetables         2013    489299.0     489299000.0
```

Figure 6.13 Using one column for Boolean masking

Two Boolean masks being compared with bitwise logical operators result in another Boolean mask. This means that you can chain together several and/or statements in order to create more complex queries.

For instance, in Figure 6.14 the code selects only those rows where the production is greater than 400,000 and from the Year 2010 onwards.

Note: Remember that each Boolean mask needs to be encased in parentheses because of the order of operations.

```
>>> df[(df['Production']>400000) & (df['Year']>=2010)]
                 Area Abbreviation  Area Code             Area  Item Code  \
Item
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables, Other              CHN         41  China, mainland       2605
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918
Vegetables                     CHN         41  China, mainland       2918

                 Element Code Element         Unit  latitude  longitude  \
Item
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables, Other         5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2
Vegetables                5142    Food  1000 tonnes     35.86      104.2

                 Year  Production  Raw Production
Item
Vegetables, Other  2011    402338.0     402338000.0
Vegetables, Other  2012    419262.0     419262000.0
Vegetables, Other  2013    426850.0     426850000.0
Vegetables         2010    451838.0     451838000.0
Vegetables         2011    462696.0     462696000.0
Vegetables         2012    479028.0     479028000.0
Vegetables         2013    489299.0     489299000.0
```

Figure 6.14 Using multiple columns for Boolean masking

Chapter 3: Missing Values

In Python, we have the None type to indicate a lack of data. Pandas uses a special floating-point value for missing values, the NaN in Numpy package which stands for not a number.

Note: When we use statistical functions on DataFrames, these functions typically ignore missing values. This is usually what we want but we should be aware that values are being excluded.

The function read_csv() introduced in Section 1.2 for loading from comma separated files provides control for missing values using two function parameters.

- the na_values parameter: It is to indicate other strings which could refer to missing values. By default, the following values are interpreted as NaN: '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan', '1.#IND', '1.#QNAN', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan', 'null'.
- the na_filter parameter: It is to turn off white space filtering, if white space is an actual value of interest. The default is True.

Read

Read the Pandas API reference (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv) for more details and examples on read_csv() function.

Chapter 4: Grouping

By grouping, we are referring to a process involving one or more of the following steps:

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

For instance, we want to build a summary data frame for the average production per area as in Figure 6.15.

1. We first specify the Area column for splitting using the DataFrame's method groupby().
2. The GroupBy object has a method called agg which is short for aggregate. This method applies a function to the column or columns in each group, and returns the results. With agg, you pass in a dictionary of the column names, and the function to apply to the columns. As in Figure 6.15, we give agg a dictionary with the Production column and the numpy average function.

```
>>> df.groupby('Area').agg({'Production': np.average})
                                                Production
Area
Afghanistan                                     156.663333
Albania                                          36.386256
Algeria                                         232.899117
Angola                                          122.211528
Antigua and Barbuda                               0.716981
Argentina                                       503.171192
Armenia                                          40.240260
Australia                                       335.213149
Austria                                         173.193566
```

Figure 6.15 Computing average production per area using the agg method

Chapter 5: Time Series

Pandas has four main time related classes:

- Timestamp - Timestamp represents a single timestamp, a specific point in time, for example, Timestamp('2016-09-01 10:05:00'). Timestamp is interchangeable with Python's datetime in most cases.
- Period - Period represents a single time span, such as a specific day or month, for example, Period('2016-03-05', 'D') or Period('2016-01', 'M').
- DatetimeIndex - The index of Timestamp is DatetimeIndex.
- PeriodIndex - The index of Period is PeriodIndex.

Note: Pandas represents timestamps using instances of Timestamp and sequences of timestamps using instances of DatetimeIndex. Pandas uses Period objects for scalar values and PeriodIndex for sequences of spans.

5.1 Converting to Datetime

We will continue the code from Figure 6.15. The current index set of the DataFrame is the Item column. We will now change the index to be the Year column and revert the Item column to an ordinary column.

1. The reset_index() method in Figure 6.16 resets the index: The old index is added as a column, and a new sequential index is used.

```
>>> df.reset_index(inplace=True)
>>> df.head()
                  Item Area Abbreviation  Area Code         Area  Item Code  \
0  Wheat and products                AFG          2  Afghanistan       2511
1  Wheat and products                AFG          2  Afghanistan       2511
2  Wheat and products                AFG          2  Afghanistan       2511
3  Wheat and products                AFG          2  Afghanistan       2511
4  Wheat and products                AFG          2  Afghanistan       2511

   Element Code Element         Unit  latitude  longitude  Year  Production
0          5142    Food  1000 tonnes     33.94      67.71  1961      1928.0
1          5142    Food  1000 tonnes     33.94      67.71  1962      1904.0
2          5142    Food  1000 tonnes     33.94      67.71  1963      1666.0
3          5142    Food  1000 tonnes     33.94      67.71  1964      1950.0
4          5142    Food  1000 tonnes     33.94      67.71  1965      2001.0
```

Figure 6.16 Resetting the DataFrame index

2. The set_index() method in Figure 6.17 sets the DataFrame index using the Year column.

```
>>> df.set_index('Year',inplace=True)
>>> df.head()
                  Item Area Abbreviation  Area Code         Area  Item Code  \
Year
1961  Wheat and products                AFG          2  Afghanistan       2511
1962  Wheat and products                AFG          2  Afghanistan       2511
1963  Wheat and products                AFG          2  Afghanistan       2511
1964  Wheat and products                AFG          2  Afghanistan       2511
1965  Wheat and products                AFG          2  Afghanistan       2511

      Element Code Element         Unit  latitude  longitude  Production
Year
1961          5142    Food  1000 tonnes     33.94      67.71      1928.0
1962          5142    Food  1000 tonnes     33.94      67.71      1904.0
1963          5142    Food  1000 tonnes     33.94      67.71      1666.0
1964          5142    Food  1000 tonnes     33.94      67.71      1950.0
1965          5142    Food  1000 tonnes     33.94      67.71      2001.0
```

Figure 6.17 Setting the index using the Year column

3. The code in Figure 6.18 first uses the Pandas to_datetime() function to convert the index into DatetimeIndex. The format parameter uses '%Y' for parsing the year. But in this context, PeriodIndex would be more suitable since the frequency is year. Hence, the to_period() method is then chained for casting the DatetimeIndex to PeriodIndex at the frequency of year, denoted by the argument 'A'. Note that a number of string aliases are given to useful common time series frequencies. Refer to http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases for details on offset aliases.

```
>>> df.index=pd.to_datetime(df.index,format='%Y').to_period('A')
>>> df.head()
     Area Abbreviation  Area Code         Area  Item Code                Item  \
Year
1961               AFG          2  Afghanistan       2511  Wheat and products
1962               AFG          2  Afghanistan       2511  Wheat and products
1963               AFG          2  Afghanistan       2511  Wheat and products
1964               AFG          2  Afghanistan       2511  Wheat and products
1965               AFG          2  Afghanistan       2511  Wheat and products

      Element Code Element         Unit  latitude  longitude  Production
Year
1961          5142    Food  1000 tonnes     33.94      67.71      1928.0
1962          5142    Food  1000 tonnes     33.94      67.71      1904.0
1963          5142    Food  1000 tonnes     33.94      67.71      1666.0
1964          5142    Food  1000 tonnes     33.94      67.71      1950.0
1965          5142    Food  1000 tonnes     33.94      67.71      2001.0
>>> type(df.index)
<class 'pandas.core.indexes.period.PeriodIndex'>
```

Figure 6.18 Converting the index to PeriodIndex

4. Now check the type of the DataFrame index. We can see that it's PeriodIndex in Figure 6.18.

Note: In Step 3 above Lists of Timestamp and Period, when being set as the index, are automatically coerced to DatetimeIndex and PeriodIndex respectively.

Read

The Pandas to_datetime() function also has parameters to change the date parse order. For example, we can pass in the argument dayfirst = True to parse the date in European date format. Read the Pandas API reference (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html?highlight=to_datetime#pandas.to_datetime) for more details and examples.

Activity 1

Create a Python file to reproduce the codes with the dataset FAO.csv in this study unit. Make sure that you can understand the code and output.

5.2 Working with Dates

The example in Figure 6.18 uses PeriodIndex. We are going to use another dataset here (GOOGL.csv) to illustrate DatetimeIndex.

Dataset

Historical stock data for GOOGL, 2006-2017 (GOOGL.csv)

The dataset's attributes are:

- Date - in format yyyy-mm-dd
- Open - price of the stock at market open
- High - highest price reached in the day
- Low - lowest price reached in the day
- Close - price of the stock at market close
- Volume - number of shares traded
- Name - the stock's ticker name

The code in Figure 6.19 imports the dataset GOOGL.csv with the index as DatetimeIndex.

1. We first import GOOGL.csv using the read_csv() function. The parameter index_col specifies the Date column to use as the row index labels of the DataFrame. The parameter parse_dates, if True, will parse the index to DatetimeIndex.
2. Now check the type of the DataFrame index. We can see that it's DatetimeIndex in Figure 6.19.

```
>>> import pandas as pd
>>> df = pd.read_csv('GOOGL.csv', index_col='Date', parse_dates=True)
>>> df.head()
              Open    High     Low   Close    Volume   Name
Date
2006-01-03  211.47  218.05  209.32  217.83  13137450  GOOGL
2006-01-04  222.17  224.70  220.09  222.84  15292353  GOOGL
2006-01-05  223.22  226.00  220.97  225.85  10815661  GOOGL
2006-01-06  228.66  235.49  226.85  233.06  17759521  GOOGL
2006-01-09  233.44  236.94  230.70  233.68  12795837  GOOGL
>>> type(df.index)
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

Figure 6.19 Importing data with the index as DatetimeIndex

Read

Read the Pandas API reference (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv) for more details and examples on read_csv() function.

5.3 Plotting Time Series

The code in Figure 6.20 is to plot a line chart for the Close column and its 30-day moving average.

```
>>> import matplotlib.pyplot as plt
>>> df['Close 30 Moving Ave']=df['Close'].rolling(30).mean()
>>> df.head()
              Open    High     Low   Close    Volume   Name  \
Date
2006-01-03  211.47  218.05  209.32  217.83  13137450  GOOGL
2006-01-04  222.17  224.70  220.09  222.84  15292353  GOOGL
2006-01-05  223.22  226.00  220.97  225.85  10815661  GOOGL
2006-01-06  228.66  235.49  226.85  233.06  17759521  GOOGL
2006-01-09  233.44  236.94  230.70  233.68  12795837  GOOGL

            Close 30 Moving Ave
Date
2006-01-03                  NaN
2006-01-04                  NaN
2006-01-05                  NaN
2006-01-06                  NaN
2006-01-09                  NaN
>>> df[['Close','Close 30 Moving Ave']].plot();plt.show()
<matplotlib.axes._subplots.AxesSubplot object at 0x10e400e10>
```

Figure 6.20 Code to plot the close price and its 30-day moving average

1. We first add a new column for the 30-day moving average of the close prices. It is calculated using the two chained methods rolling() and mean(). The method rolling(30) provides rolling window calculations. The argument 30 specifies the size of the moving window. Each window will be a fixed size. This is the number of observations, i.e. 30 days in this context, used for calculating the average.
2. Next is to select the two columns "Close" and "Close 30 Moving Ave" and plot a line chart by default using the plot() method. On DataFrame, plot() is a convenience to plot all of the selected columns with labels. The output is shown in Figure 6.21. You may zoom out a certain area of the plot using the "Zoom to rectangle" button. An example is given in Figure 6.22.

Read

Read the API reference (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.rolling.html) for more details and examples on the rolling method.

Figure 6.21 Plotting the close price and its 30-day moving average

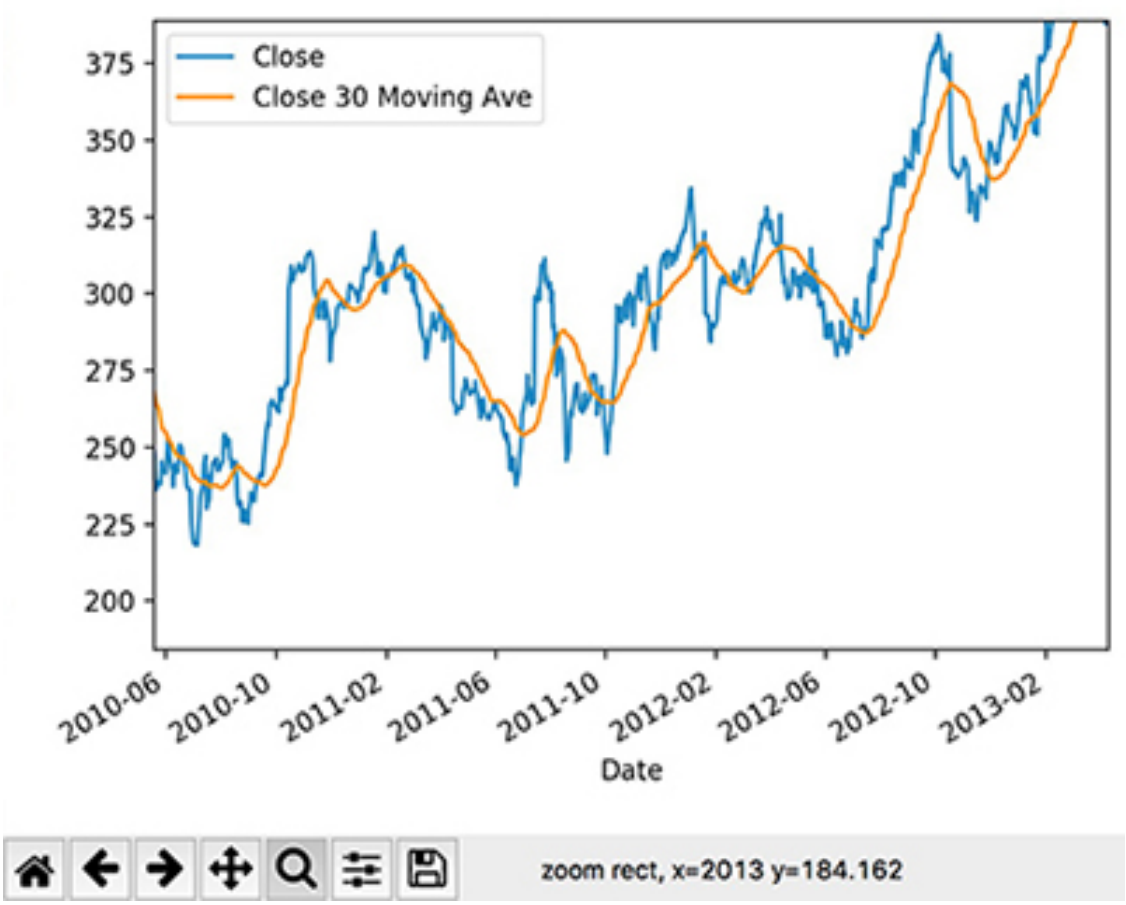

Figure 6.22 Zooming out the plotting in Figure 6.21

Activity 2

Create a Python file to reproduce the codes with the dataset GOOGL.csv in this study unit. Make sure that you can understand the code and output.

Summary

In this unit, we have seen how Python can be used to manipulate, clean, and query data using the Pandas library. Querying the DataFrame structures is done in a few different ways, such as using the iloc or loc attributes for row-based querying, or using the square brackets on the object itself for column-based querying. We also saw that one can query the DataFrame through Boolean masking. We then explored how to reduce and process data using grouping and aggregation. We also talked about some features of Pandas useful for time series data analysis.

Formative Assessment

1. How is a Pandas DataFrame different from a 2D Numpy array?
    a. Basically the same thing.
    b. 2D Numpy arrays have more advanced subsetting capabilities than Pandas DataFrames.
    c. Pandas DataFrames' axes have less semantic meaning than 2D Numpy arrays.
    d. In Pandas DataFrames, different columns can contain different data types.

2. Which option correctly describes Pandas DataFrames?
    a. The rows correspond to observations. The columns correspond to variables.
    b. The rows correspond to variables. The columns correspond to observations.
    c. The rows and columns both correspond to observations, but in different dimensions.
    d. The rows and columns both correspond to variables, but in different dimensions.

3. Which Pandas function do you use to import data from a comma-separated value (CSV) file into a Pandas DataFrame?
    a. read_csv()
    b. read_tsv()
    c. read_datafile()
    d. read_todf()

4. Which technique should you use to select an entire row by its row index label when accessing data in a Pandas DataFrame?
    a. loc
    b. iloc
    c. Square brackets: []

d. Parentheses: ()

5. For the purchase records stored in a DataFrame df as below, how would you get a list of all items which had been purchased?

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

a. df.iloc['Item Purchased']
b. df('Item Purchased')
c. df['Item Purchased']
d. df.loc['Item Purchased']

6. For the purchase records stored in a DataFrame df as below, how would you update the DataFrame, applying a discount of 20% across all the values in the 'Cost' column?

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

a. df.Cost *0.8
b. df.loc['Cost'] *0.8
c. df['Cost'] *0.8

d. df[‘Cost’] *= 0.8

7. Suppose we have a CSV file exercise.csv that looks like this:

Exercise CSV			
Week 1 Exercises			
Activity ID	Activity Type	Activity Duration	Calories
125	Running	65	450
126	Biking	40	280
127	Running	90	850
128	Walking	30	160

Which of the following would return a DataFrame with the columns = [‘Activity Type’, ‘Activity Duration’, ‘Calories’] and index = [125, 126, 127, 128] with the name ‘Activity ID’? Assume we have imported Pandas using “import pandas as pd”.

a. pd.read_excel(‘exercise.csv’, skiprows=2, sep=’\t’)

b. pd.read_csv(‘exercise.csv’, skiprows=2, index_col=0)

c. pd.read_excel(‘exercise.csv’, skiprows=2, index_col=0)

d. pd.read_csv(‘exercise.csv’, skiprows=2, sep=’,’)

8. For purchase records stored in a DataFrame df as shown below, write a query to return all of the names of people who bought products worth more than $3.00.

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

a. df.loc[df['Cost']>3,'Name']

b. df[df['Cost']>3,'Name']

c. df['Name','Cost'>3]

d. df['Cost'>3, 'Name']

9. For purchase records stored in a DataFrame df as shown below, write a query to find the total quantity for each type of item purchased.

	Item Purchased	Name	Quantity	Unit Price
Store 1	Coffee	Chris	1	22.5
Store 1	Bread	Kevyn	3	2.5
Store 2	Milk	Vinod	2	5.0
Store 3	Coffee	Chris	1	22.5
Store 3	Bread	Peter	1	2.5
Store 2	Milk	Peter	3	5.0

   a. df.groupby('Item Purchased'). agg({'Quantity':sum})
   b. df['Item Purchased'].sum()
   c. df.groupby('Item Purchased').sum()
   d. sum(df['Item Purchased'])

10. For purchase records stored in a DataFrame df as shown below, write a query to find the total amount of money received (Unit Price x Quantity) for each type of item purchased.

	Item Purchased	Name	Quantity	Unit Price
Store 1	Coffee	Chris	1	22.5
Store 1	Bread	Kevyn	3	2.5
Store 2	Milk	Vinod	2	5.0
Store 3	Coffee	Chris	1	22.5
Store 3	Bread	Peter	1	2.5
Store 2	Milk	Peter	3	5.0

   a. df['total']=df['Quantity']*df['Unit Price']; df.groupby('Item Purchased'). agg({'total':sum})
   b. df.groupby('Item Purchased'). agg({df['Quantity']*df['Unit Price']:sum})

c. df.groupby('Item Purchased').sum(df['Quantity']*df['Unit Price'])

d. sum(df['Quantity']*df['Unit Price'])['Item Purchased']

Solutions or Suggested Answers

Formative Assessment

1. How is a Pandas DataFrame different from a 2D Numpy array?

    a. Basically the same thing.
    Incorrect. Refer to Chapter 1 on DataFrames.

    b. 2D Numpy arrays have more advanced subsetting capabilities than Pandas DataFrames.
    Incorrect. Refer to Chapter 1 on DataFrames.

    c. Pandas DataFrames' axes have less semantic meaning than 2D Numpy arrays.
    Incorrect. Refer to Chapter 1 on DataFrames.

    d. In Pandas DataFrames, different columns can contain different data types.
    Correct.

2. Which option correctly describes Pandas DataFrames?

    a. The rows correspond to observations. The columns correspond to variables.
    Correct.

    b. The rows correspond to variables. The columns correspond to observations.
    Incorrect. Refer to Section 1.1 on DataFrames.

    c. The rows and columns both correspond to observations, but in different dimensions.
    Incorrect. Refer to Section 1.1 on DataFrames.

    d. The rows and columns both correspond to variables, but in different dimensions.

Incorrect. Refer to Section 1.1 on DataFrames.

3. Which Pandas function do you use to import data from a comma-separated value (CSV) file into a Pandas DataFrame?

   a. read_csv()

      Correct.

   b. read_tsv()

      Incorrect. Refer to Section 1.2.

   c. read_datafile()

      Incorrect. Refer to Section 1.2.

   d. read_todf()

      Incorrect. Refer to Section 1.2.

4. Which technique should you use to select an entire row by its row index label when accessing data in a Pandas DataFrame?

   a. loc

      Correct.

   b. iloc

      Incorrect. Refer to Section 2.2 on row access.

   c. Square brackets: []

      Incorrect. Refer to Section 2.2 on row access.

   d. Parentheses: ()

      Incorrect. Refer to Section 2.2 on row access.

5. For the purchase records stored in a DataFrame df as below, how would you get a list of all items which had been purchased?

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

   a. df.iloc['Item Purchased']
   Incorrect. Refer to Section 2.1 on column access.

   b. df('Item Purchased')
   Incorrect. Refer to Section 2.1 on column access.

   c. df['Item Purchased']
   Correct.

   d. df.loc['Item Purchased']
   Incorrect. Refer to Section 2.1 on column access.

6. For the purchase records stored in a DataFrame df as below, how would you update the DataFrame, applying a discount of 20% across all the values in the 'Cost' column?

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

a. df.Cost *0.8

Incorrect. Refer to Section 2.1 on column access.

b. df.loc['Cost'] *0.8

Incorrect. Refer to Section 2.1 on column access.

c. df['Cost'] *0.8

Incorrect. Refer to Section 2.1 on column access.

d. df['Cost'] *= 0.8

Correct.

7. Suppose we have a CSV file exercise.csv that looks like this:

Exercise CSV			
Week 1 Exercises			
Activity ID	Activity Type	Activity Duration	Calories
125	Running	65	450
126	Biking	40	280
127	Running	90	850
128	Walking	30	160

Which of the following would return a DataFrame with the columns = ['Activity Type', 'Activity Duration', 'Calories'] and index = [125, 126, 127, 128] with the name 'Activity ID'? Assume we have imported Pandas using "import pandas as pd".

a. pd.read_excel('exercise.csv', skiprows=2, sep='\t')

Incorrect. Refer to Chapter 1 on importing data.

b. pd.read_csv('exercise.csv', skiprows=2, index_col=0)

Correct.

c. pd.read_excel('exercise.csv', skiprows=2, index_col=0)

Incorrect. Refer to Chapter 1 on importing data.

d. pd.read_csv('exercise.csv', skiprows=2, sep=',')

Incorrect. Refer to the API reference http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv on read_csv() function.

8. For purchase records stored in a DataFrame df as shown below, write a query to return all of the names of people who bought products worth more than $3.00.

	Cost	Item Purchased	Name
Store 1	22.5	Coffee	Chris
Store 1	2.5	Bread	Kevyn
Store 2	5.0	Milk	Vinod

   a. df.loc[df['Cost']>3,'Name']
   Correct.

   b. df[df['Cost']>3,'Name']
   Incorrect. Refer to Section 2.4 on Boolean masking.

   c. df['Name','Cost'>3]
   Incorrect. Refer to Section 2.4 on Boolean masking.

   d. df['Cost'>3, 'Name']
   Incorrect. Refer to Section 2.4 on Boolean masking.

9. For purchase records stored in a DataFrame df as shown below, write a query to find the total quantity for each type of item purchased.

	Item Purchased	Name	Quantity	Unit Price
Store 1	Coffee	Chris	1	22.5
Store 1	Bread	Kevyn	3	2.5
Store 2	Milk	Vinod	2	5.0
Store 3	Coffee	Chris	1	22.5
Store 3	Bread	Peter	1	2.5
Store 2	Milk	Peter	3	5.0

a. df.groupby('Item Purchased'). agg({'Quantity':sum})
Correct.

b. df[‘Item Purchased’].sum()
Incorrect. Refer to Chapter 4 on grouping.

c. df.groupby('Item Purchased').sum()
Incorrect. Refer to Chapter 4 on grouping.

d. sum(df[‘Item Purchased’])
Incorrect. Refer to Chapter 4 on grouping.

10. For purchase records stored in a DataFrame df as shown below, write a query to find the total amount of money received (Unit Price x Quantity) for each type of item purchased.

	Item Purchased	Name	Quantity	Unit Price
Store 1	Coffee	Chris	1	22.5
Store 1	Bread	Kevyn	3	2.5
Store 2	Milk	Vinod	2	5.0
Store 3	Coffee	Chris	1	22.5
Store 3	Bread	Peter	1	2.5
Store 2	Milk	Peter	3	5.0

a. df['total']=df['Quantity']*df['Unit Price']; df.groupby('Item Purchased'). agg({'total':sum})
Correct.

b. df.groupby('Item Purchased'). agg({df['Quantity']*df['Unit Price']:sum})
Incorrect. Refer to Chapter 4 on grouping.

c. df.groupby('Item Purchased').sum(df['Quantity']*df['Unit Price'])
Incorrect. Refer to Chapter 4 on grouping.

d. sum(df['Quantity']*df['Unit Price'])['Item Purchased']
Incorrect. Refer to Chapter 4 on grouping.

References

Zed, S. (2017). *Learn Python the hard way*. Addison-Wesley Professional.

Numeric types – int, float, complex. (n.d.). Retrieved from https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Operator precedence. (n.d.). Retrieved from https://docs.python.org/3/reference/expressions.html#operator-precedence