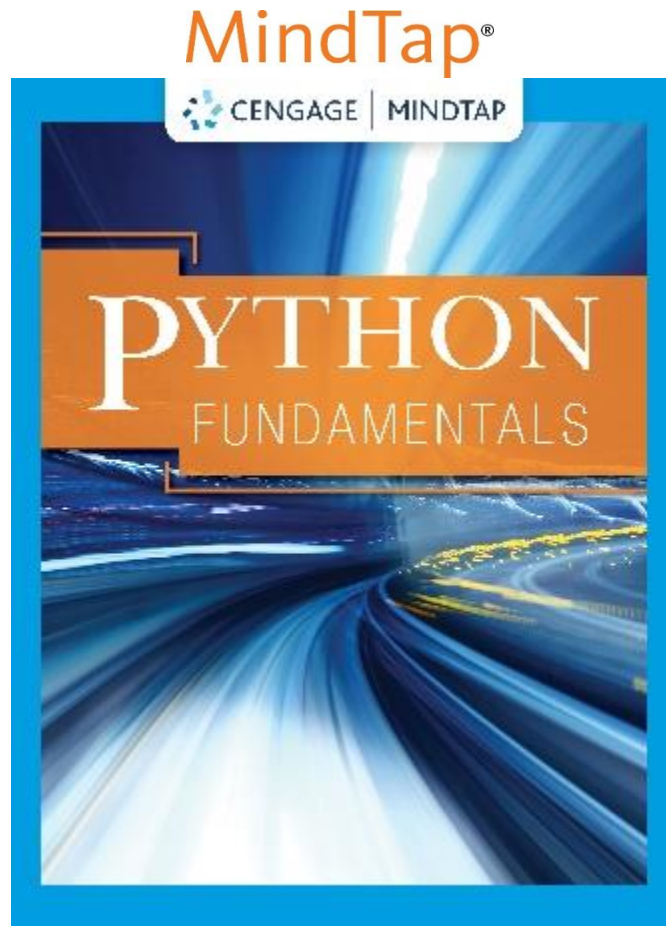


**Python Fundamentals**  
**ISBN MindTap:**



Welcome to *Python Fundamentals*. This Instructor's Manual will help you navigate the unique activities that are included in the MindTap, which will better enable you to include the exercises in your curriculum. While the content included in this MindTap is specific to the discipline and course, the functionality will act the same as you move from product to product.

For additional resources on our MindTap platform, please click [HERE](#). At this site, you will find User Guides, Self-Training Videos, Training Webinars, and Podcasts. We also include Resources that are specific to your campus's LMS, should additional information be needed. Student versions of the same resources are located [HERE](#). This link can be shared with your students directly, should they have any questions about the product.

## **At a Glance**

### **Instructor's Manual Table of Contents**

- Course Learning Design
- Lab Details
- Module Objectives
- Solutions to Reflection

## Course Learning Design

In creating the digital learning path, we aimed to provide your students with a coherently structured experience that:

- supports and aligns the learning objectives with the course content, instructional strategies, and assessments;
- addresses individual learner differences and preferences;
- welcomes learners of all abilities and backgrounds; and
- enhances learner motivation by providing them with relevant, applicable learning experiences consistent with their own learning and professional goals.

We're excited to present you with the digital course experience and want to draw your attention to some of the design decisions we made as part of ensuring your confidence in our ability to create an effective, quality learning experience.

Course Learning Design	
Course Description	As you work with the language, you'll learn about control statements, delve into controlling program flow, and gradually work on more structured programs via functions. <i>MindTap for Python Fundamentals</i> teaches problem-solving skills for building efficient applications. As you settle into the Python ecosystem, you'll learn about data structures and study ways to correctly store and represent information. By working through specific examples, you'll learn how Python implements object-oriented programming (OOP) concepts of abstraction, encapsulation of data, inheritance, and polymorphism. Coverage also includes an overview of how imports, modules, and packages work in Python, how you can handle errors to prevent apps from crashing, as well as file manipulation.
Course Approach (9 modules in course)	This course teaches students how to write systematic code in Python and improve application efficiency with hands-on practice, step-by-step instruction, and provides immediate feedback and troubleshooting support on their code. Students will develop skills that are in-demand by employers by completing authentic, real-world coding projects that can be added to their GitHub portfolios.
Module Approach	<p>Each module is broken into 2–6 lessons—within each lesson are activities that align to meet specific learning objectives that are concrete and actionable.</p> <p>Within each lesson, the student will read some narrative and follow up with hands-on learning. There are four types of online labs in this course:</p> <ol style="list-style-type: none"><li>1. <b>Practice Exercises</b> (Ungraded) provide an opportunity to practice a new concept in a short coding activity. Students are provided with guided instructional materials alongside a live computing environment. There will typically be 1–3 practice labs in each lesson and there are on average around 5 lessons per module (around 5 practice/module).</li><li>2. <b>Lab Activities</b> (Auto-Graded) are coding activities that are completed by a student and contain auto-grading that feeds directly to the gradebook. Learners demonstrate an understanding of numerous concepts by completing tasks. Tasks are verified using unit tests, I/O tests, image and webpage comparison, debugging tests, and many other checks. There will be a lab assessment for every lesson and there are on average around 5 lessons per module (around 5 labs per module).</li><li>3. <b>Module Lab Assessments</b> (Auto- and Manual-Graded) encompass all the learning objectives in the module. Students are asked to complete a larger, authentic assignment with many tasks. Some tasks will be verified using unit tests, I/O tests, image and webpage comparison, debugging tests but other tasks will be unique to each student's project and will require manual grading. The goal of these assignments is to prove that students have mastered the learning objectives in the module and in doing so have also created a program for their GitHub portfolio (1 Module Lab Assessment per module).</li><li>4. <b>Capstone Lab Assessment</b> (Auto- and Manual-Graded) is a final project that is the summative assessment. The goal of this assignment is to prove that students have mastered the course objectives and in doing so have also created a program for their GitHub portfolio (1 Capstone Lab Assessment per course).</li></ol>

## Python Fundamentals, First Edition

Learning Path Activities	How many in course	What is it?	Why it matters?	Seat time
Welcome to Your Course	1	This is a brief overview of the course objectives that will be covered in the modules of this MindTap.	Students will gain a clear understanding of the course objectives and will explore how this course offers the opportunity to not only read but watch videos, engage in critical-thinking simulations and hands-on trainings, teach them how to use the technology, and take quizzes to practice and check their understanding.	5 minutes
Getting Started Resources	1	This section includes videos that provide an overview of the MindTap platform and the Coding IDE. There are 3 lab Pre-Requisites, 2 of which count toward the grade.	Students will learn how to use MindTap to its fullest potential, which will help them excel in the course.  They'll also be introduced to the IDE's functionality in 6 brief videos. They'll then complete 3 Lab Pre-Requisites, 1 is practice and 2 count toward their grade.	30 minutes
Pre- and Post-Course Assessments	27 questions each assessment	Brief survey to-assess students' knowledge of the subject matter before and after completing the course.	<b>For students:</b> It creates awareness around what they will learn (pre) and how much they have learned (post). <b>For instructors:</b> It establishes a baseline of what students already know (pre) and demonstrates how much they learned (post). <b>For administrators:</b> Coupling the pre- and post-course assessment provides data on how much the students learned and the overall impact of the course.	40 minutes
<b>Module Content (9 modules total)</b>				
Readings for each module lesson; 2–6 lessons per module	~7 Short readings per module (69 total in course)	Readings reinforce learning objectives.	Students will read succinct, focused excerpts vs long chapters (then move into an interactive activity).	55 minutes
Practice Exercises	~5 per module (48 total in course)	Short coding exercises in an IDE (non-graded)	Students complete step-by-step coding exercises that offer a practical, hands-on approach to acquiring and retaining new concepts and skills.	2-5 minutes
Lab Activity (Graded)	~5 per module (41 total in course)	Scenario-based coding labs in an IDE (auto-graded)	These scenario-based activities bring together skills learned throughout the topics and lessons to solve real-world problems.	30 minutes
Reflection	~6 per module (51 total in course)	Essay question	The reflection prompt challenges students to develop higher-level thinking and promotes problem-solving. This is also an opportunity for you to confirm that tricky topics are understood.	15 minutes
Module Quizzes	~1 per module (9 total in course)	Includes 10 multiple-choice questions at the end of each module.	The student can integrate material across the entire lesson and check their understanding before moving on to the next lesson.	10 minutes
Module Lab Assessment (Auto & Manual Grading)	1 per module (9 total in course)	A larger coding project in our IDE that assesses whether students have mastered the Learning Objectives in the module.	A larger lab with an authentic development project with many tasks. Upon completion, students will have 9 large coding projects for their GitHub portfolios.	1–2 hours
Capstone Lab Assessment	1 per course	Final coding project in our IDE that assesses whether	A larger lab with an authentic development project with many tasks. Upon completion, students will have	2–5 hours

## Python Fundamentals, First Edition

		students have mastered the Course Objectives.	1 additional coding project to add to their GitHub portfolio.	
Instructor Test Bank	1 per module (9 total in course)	An exam of 451 objective-based questions based on each module available in the CNOW app.	The Test Bank evaluates the student on their mastery of that module.	30 minutes

Topic/Chapter	Assignments
Module 1 Introducing Python	Lessons 1.1 – 1.4 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 2 Data Types	Lessons 2.1 – 2.4 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 3 Control Statements	Lessons 3.1 – 3.9 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 4 Functions	Lessons 4.1 – 4.4 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 5 Lists and Tuples	Lessons 5.1 – 5.6 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 6 Dictionaries and Sets	Lessons 6.1 – 6.6 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 7 Object-Oriented Programming	Lessons 7.1 – 7.7 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 8 Modules, Packages, and File Operations	Lessons 8.1 – 8.7 Reading Practice Exercises Lab Activities Reflection Module Quiz
Module 9 Error Handling	Lessons 9.1 – 9/4 Reading Practice Exercises Lab Activities Reflection Module Quiz
Capstone Lab Assessment:	

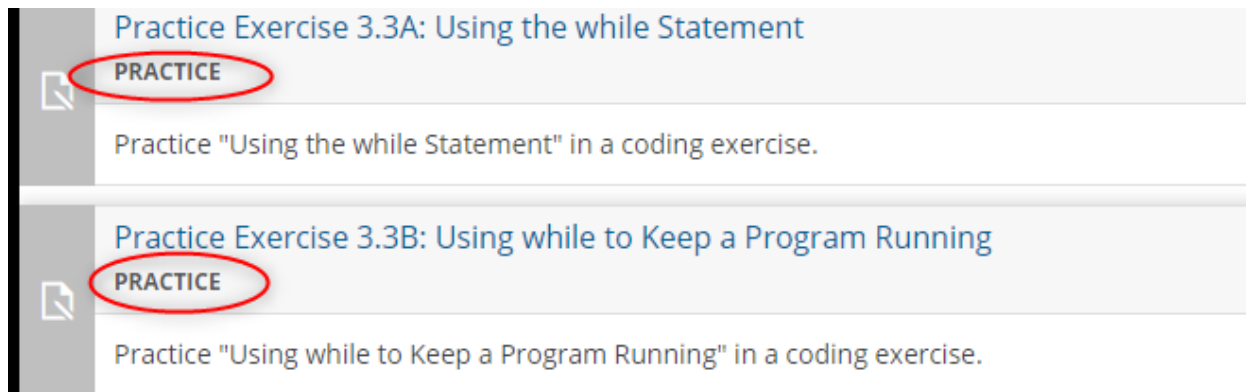
## Lab Details

There are 48 Practice Exercises, 41 Lab Activities, 9 Module Lab Assessments, and 1 Capstone Lab Assessment across 9 modules.

## Lab Types

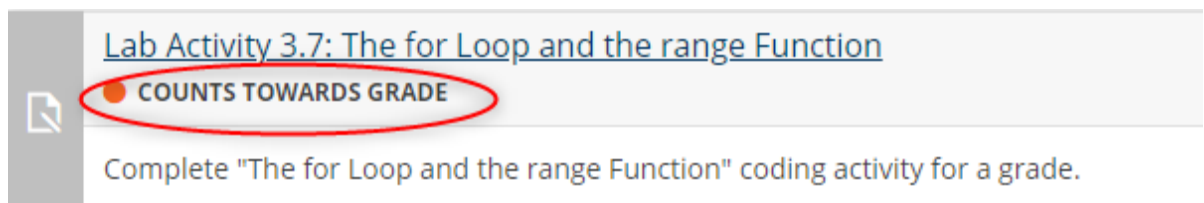
### Practice Exercises:

- Practice Exercises are coding lab assignments within the IDE that allow you to practice writing and running code.
- Practice Exercises are not graded and are not captured in the Progress App. These are designated in the learning path:



### Lab Activities:

- Lab Activities are coding lab assignments within the IDE that run tests against your code to ensure that the objectives in the activity have been satisfied.
- Lab Activities are automatically graded unless otherwise noted in the learning path as “PRACTICE”. All graded labs are designated in the learning path as “COUNTS TOWARDS GRADE”.



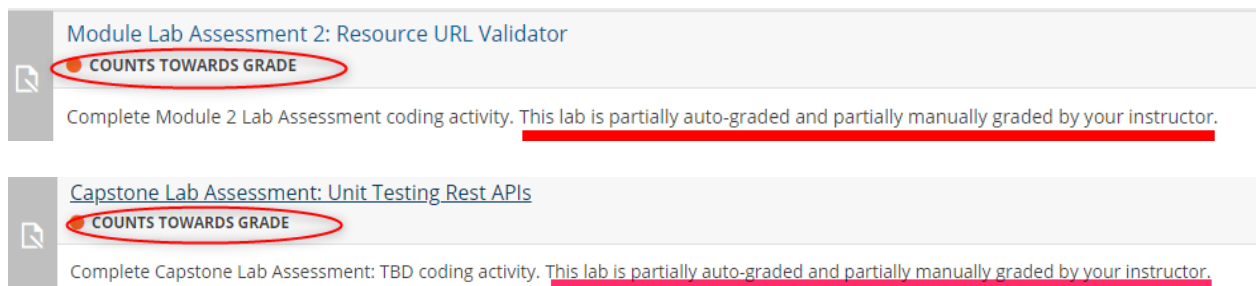
- You will work through the Lab Activities and “Run Checks” as you work through the problems. Once you have completed the assignment, you can “Submit”, which will send your lab to your instructor.



- Note that instructors have the capability to review code submissions and alter grades as they see fit. Grade submissions are not final.

### Module Lab Assessments and Capstone Lab Assessment:

- The Lab Assessments are coding lab assignments within the IDE that provide you with an authentic scenario to test your coding skills.
- There is one Module Lab Assessment per module, and one Capstone Lab Assessment for the entire course.
- Lab Assessments are partially automatically graded and partially manually graded by your instructor. These are designated in the learning path with a “This lab is partially auto-graded and partially manually graded by your instructor” description. All graded labs are designated in the learning path as “COUNTS TOWARDS GRADE”.



- Note that instructors have the capability to review code submissions and alter grades as they see fit. Grade submissions are not final.

### List of Coding Labs

Coding IDE Lab Prerequisite for Practice Exercises	Practice
Coding IDE Lab Prerequisite for Lab Activities	Auto-Grade
Coding IDE Lab Prerequisite for Module and Capstone Lab Assessments	Auto / Manual Grade
<b>Module 1</b>	
Practice Exercise 1.1A: Checking our Python Installation	Practice
Practice Exercise 1.1B: Working with the Python Interpreter	Practice
Lab Activity 1.1: Working with the Python Shell	Auto-Grade

Practice Exercise 1.2: Creating a Script	Practice
Lab Activity 1.2: Running Simple Python Scripts	Auto-Grade
Practice Exercise 1.3A: Checking the Type of a Value	Practice
Practice Exercise 1.3B: Using Variables	Practice
Lab Activity 1.3A: Using Variables and Assign Statements	Auto-Grade
Practice Exercise 1.3C: Python Keywords	Practice
Lab Activity 1.3B: Variable Assignment and Variable Naming Conventions	Auto-Grade
Practice Exercise 1.4A: Fetching and Using User Input	Practice
Practice Exercise 1.4B: The Importance of Proper Indentation	Practice
Lab Activity 1.4A: Fixing Indentations in a Code Block	Auto-Grade
Lab Activity 1.4B: Implementing User Input and Comments in a Script	Auto-Grade
Module Lab Assessment 1: Creating a Unit Converter	Auto / Manual Grade
<b>Module 2</b>	
Practice Exercise 2.1: Converting Between Different Types of Number Systems	Practice
Lab Activity 2.1A: Order of Operations	Auto-Grade
Lab Activity 2.1B: Using Different Arithmetic Operators	Auto-Grade
Lab Activity 2.2A: String Slicing	Practice
Lab Activity 2.2B: Working with Strings	Auto-Grade
Practice Exercise 2.2: Using Escape Sequences	Practice
Lab Activity 2.2C: Manipulating Strings	Auto-Grade
Practice Exercise 2.3: List References	Practice
Lab Activity 2.3: Working with Lists	Auto-Grade
Lab Activity 2.4: Using Boolean Operators	Auto-Grade
Module Lab Assessment 2: Resource URL Validator	Auto / Manual Grade
<b>Module 3</b>	
Practice Exercise 3.2: Using the if Statement	Practice
Lab Activity 3.2: Working with the if Statement	Auto-Grade
Practice Exercise 3.3A: Using the while Statement	Practice
Practice Exercise 3.3B: Using while to Keep a Program Running	Practice
Lab Activity 3.3: Working with the while Statement	Auto-Grade
Practice Exercise 3.6: Using the for Loop	Practice
Lab Activity 3.7: The for Loop and the range Function	Auto-Grade
Practice Exercise 3.8: Using Nested Loops	Practice
Lab Activity 3.8: Nested Loops	Auto-Grade
Lab Activity 3.9: Breaking Out of Loops	Auto-Grade
Module Lab Assessment 3: Abby's Ice Cream Shop	Auto / Manual Grade
<b>Module 4</b>	
Practice Exercise 4.2: Defining Global and Local Variables	Practice
Lab Activity 4.3: Function Arguments	Auto-Grade



Practice Exercise 4.4: Creating a Lambda Function	Practice
Lab Activity 4.4: Using Lambda Functions	Auto-Grade
Module Lab Assessment 4: AppInvest Return on Investment Function	Auto / Manual Grade
<b>Module 5</b>	
Lab Activity 5.2: Using the List Methods	Auto-Grade
Practice Exercise 5.4: Creating a Tuple	Practice
Practice Exercise 5.5A: Accessing Tuple Elements Using Indexing	Practice
Practice Exercise 5.5B: Accessing Tuple Elements Using Slicing	Practice
Lab Activity 5.6: Using Tuple Methods	Auto-Grade
Module Lab Assessment 5: Creating a Blackjack Simulator	Auto / Manual Grade
<b>Module 6</b>	
Lab Activity 6.1A: Creating a Dictionary	Auto-Grade
Practice Exercise 6.1: Adding, Reading, and Iterating through a Dictionary	Practice
Lab Activity 6.1B: Arranging and Presenting Data Using Dictionaries	Auto-Grade
Lab Activity 6.1C: Combining Dictionaries	Auto-Grade
Practice Exercise 6.2: Updating, Editing, and Copying from a Dictionary	Practice
Lab Activity 6.4: Building a Set	Auto-Grade
Practice Exercise 6.4: Adding, Reading, Editing, and Building a Set	Practice
Lab Activity 6.5: Creating Unions of Elements in a Collection	Auto-Grade
Practice Exercise 6.5: Unions, Differences, and Intersection of Sets	Practice
Practice Exercise 6.6: Frozen Sets	Practice
Module Lab Assessment 6: Space Explorer (Dictionaries and Sets)	Auto / Manual Grade
<b>Module 7</b>	
Practice Exercise 7.2: Adding Attributes to a Class	Practice
Lab Activity 7.2: Defining a Class and Objects	Auto-Grade
Lab Activity 7.3: Defining Methods in a Class	Auto-Grade
Practice Exercise 7.4A: Declaring a Class with Instance Attributes	Practice
Practice Exercise 7.4B: Implementing a Counter for Instances of a Class	Practice
Lab Activity 7.4: Creating Class Attributes	Auto-Grade
Practice Exercise 7.5A: Testing our Factory Method	Practice
Practice Exercise 7.5B: Accessing Class Attributes from within Class Methods	Practice
Lab Activity 7.5: Creating Class Methods and Using Information Hiding	Auto-Grade
Practice Exercise 7.6: Implementing Class Inheritance	Practice
Lab Activity 7.6: Overriding Methods	Auto-Grade
Practice Exercise 7.7: Implementing Multiple Inheritance	Practice

Lab Activity 7.7: Practicing Multiple Inheritance	Auto-Grade
Module Lab Assessment 7: Petri Dish Simulation (Object-Oriented Programming)	Auto / Manual Grade
<b>Module 8</b>	
Practice Exercise 8.1: Creating and Importing a User-Defined Module	Practice
Practice Exercise 8.2A: Importing Modules	Practice
Practice Exercise 8.2B: Importing Functions from User-Defined Modules	Practice
Practice Exercise 8.3: Inspecting Modules and Packages	Practice
Lab Activity 8.3A: Inspecting Modules	Auto-Grade
Lab Activity 8.3B: Listing the Resources Defined in a Package or Module	Auto-Grade
Lab Activity 8.3C: Using Resources in a Module	Auto-Grade
Practice Exercise 8.6A: Creating and Writing to a Text File	Practice
Practice Exercise 8.6B: Read Using with Keyword	Practice
Practice Exercise 8.6C: File Operations	Practice
Lab Activity 8.6: Performing File Operations	Auto-Grade
Practice Exercise 8.7A: Reading a CSV file	Practice
Practice Exercise 8.7B: Write a dict to CSV	Practice
Lab Activity 8.7: Working with Files	Auto-Grade
Practice Exercise 8.7C: Working with JSON	Practice
Module Lab Assessment 8: Mailing List Validation File Processing	Auto / Manual Grade
<b>Module 9</b>	
Practice Exercise 9.1A: Raise an Exception	Practice
Practice Exercise 9.1B: Raise an Exception with the raise Keyword	Practice
Lab Activity 9.2: Identifying Error Scenarios	Auto-Grade
Practice Exercise 9.3A: Implement a try...except Block	Practice
Practice Exercise 9.3B: Implementing the try...except...else Block	Practice
Lab Activity 9.3: Handling Errors	Auto-Grade
Practice Exercise 9.4: Catch an Error and Raise an Exception	Practice
Lab Activity 9.4: Creating Your Own Custom Exception Class	Auto-Grade
Module Lab Assessment 9: Error Handling	Auto / Manual Grade
Capstone Lab Assessment: Unit Testing Rest APIs	Auto / Manual Grade

### A Note to Instructors:

**COUNTS TOWARD GRADE/PRACTICE:** Whether a lab COUNTS TOWARD GRADE or is PRACTICE, as indicated in the Learning Path, is preset and cannot be changed. Changing the Gradeable field within MindTap will not change the gradeability of the actual labs. We

recommend not changing these default settings to avoid discrepancies between the MindTap plank description and the actual lab.

- COUNTS TOWARD GRADE = lab is automatically or manually graded and the score is captured in the Progress App
- PRACTICE = lab is not graded and the score is not captured in the Progress App

The screenshot displays two lab activity entries from the MindTap interface. The first entry, 'Practice Exercise 2.1: Converting Between Different Types of Number Systems', features a 'PRACTICE' label circled in red. The second entry, 'Lab Activity 2.1A: Order of Operations', features a 'COUNTS TOWARDS GRADE' label circled in red. Both entries include a description of the activity and a document icon.

Activity Title	Label	Description
Practice Exercise 2.1: Converting Between Different Types of Number Systems	PRACTICE	Practice "Converting Between Different Types of Number Systems" in a coding exercise.
Lab Activity 2.1A: Order of Operations	COUNTS TOWARDS GRADE	Complete "Order of Operations" coding activity for a grade.

# Module 1

## Introducing Python

### Module Objectives

- Use the Python interactive shell to write simple programs
- Write and run simple Python scripts
- Write and run dynamic scripts that take arguments from the command line
- Use variables and describe the different types of values that variables can be assigned
- Get user input from the keyboard for your Python programs
- Explain the importance of comments and write them in Python
- Explain the importance of whitespace and indentation in Python

## Solutions to Reflection

### Lesson 1.1 Reflection

1. Which other operating system shells are you familiar with?

**ANS:** Bourne-again shell (Bash), Z shell, and Korn shell for Unix-based systems such as Linux and the Windows and Windows PowerShell.

2. What are the benefits of the Python interactive shell?

**ANS:** It allows you to quickly execute Python commands without having to compile anything. It also allows you to do quick tests.

### Lesson 1.2 Reflection

1. What are the advantages of two different methods of running Python programs?

**ANS:**

Pros of the interactive shell:

- 1) Running Python programs via the interactive shell has the benefit of being able to run quick tests/experiments as opposed to running a saved module, which has the overheads of creating a file and having to change and save it if you want to run those changes.
- 2) You can run quick calculations using the Python interactive shell.

Pros of running a saved script:

- 1) You can reuse code by either running the same script or importing it into your module.

- 2) You can automate different tasks and have them run on your system on a schedule.

### Lesson 1.3 Reflection

1. Why do we get a syntax error when we try to assign reserved words?

**ANS:** The Python interpreter tries to parse the keyword's syntactical meaning, which doesn't fit into the context of a variable assignment. For example, when you try assigning the keyword `if`, Python looks for a condition next but instead finds an equals sign, which is invalid.

2. Mark the following variable names as valid or invalid and state why.

- 1) TestTOKEN
- 2) new-list
- 3) \_\_
- 4) TOTAL
- 5) if
- 6) array2
- 7) 2\_nd
- 8) void
- 9) five%
- 10) n123456789
- 11) LastLetter
- 12) maximum width
- 13) \$HOME

**ANS:** Valid variable names: 1, 3, 4, 6, 8, 10, 11. Invalid variable names: 2, 5, 7, 9, 12, 13.

### Lesson 1.4 Reflection

1. How does Python indentation help while writing scripts and code?

**ANS:** The indentation in Python can increase the clarity of the code by making it obvious to a human reader which blocks of code belong together.

## Module 2

### Data Types

#### Module Objectives

- Explain the different numerical data types
- Use operators on numerical data types
- Explain strings and implement string operations, such as indexing, slicing, and string formatting
- Describe escape sequences
- Explain lists and perform simple operations on them
- Use Boolean expressions and Boolean operators

### Solutions to Reflection

#### Lesson 2.1 Reflection

1. Which of the following are invalid Python operations and why?

- 1)  $941 + 38$
- 2)  $348 / 29$
- 3)  $51 \backslash \backslash 3$
- 4)  $23 - 17$
- 5)  $9^2$
- 6)  $100++$
- 7)  $-35$
- 8)  $\text{exp}(2, 3)$

**ANS:** 3, 5, 6, and 8. These particular options do not use the correct mathematical operators

#### Lesson 2.2 Reflection

1. What are string indices? What are indices used for?

**ANS:** String indices are the positions of each of the characters of the string. Indices are used when accessing the characters of the string.

#### Lesson 2.3 Reflection

1. How would you describe an array?

**ANS:** An array is a collection of values. They are used to hold multiple values that are related to each other.

2. Given the following change to the list, what will be the new value of the `mystery_box` list?

```
>>> mystery_box = ["sock", 42.5, "book", 650]
>>> mystery_box[2] = "string"
>>> print(mystery_box)
```

Given the following change to the list, what will be the new value of the `corvids` lists?

```
>>> corvids = ["crow", "raven", "magpie"]
>>> corvids[3] = "jackdaw"
>>> print(corvids)
```

**ANS:** `["sock", 42.5, "string", 650]`. It will fail on the second line and raise an `IndexError` since there is no element at index 3 to change.

## Lesson 2.4 Reflection

1. What is the difference between the `is` and `==` operators?

**ANS:** The `is` operator checks the identity of each object passed to it and checks whether they are the same object while the `==` operator simply compares the value of the object regardless of whether or not they are different objects.

## Module 3

### Control Statements

#### Module Objectives

- Describe the different control statements in Python
- Control program execution flow using control statements such as if and while
- Use looping structures in your Python programs
- Implement branching within looping structures such as for and range
- Implement breaking out of loops

### Solutions to Reflection

#### Lesson 3.1 Reflection

1. What are some example execution flow scenarios that you think Python can handle?

**ANS:** Mathematical computations and Boolean decisions (true-or-false scenarios)

#### Lesson 3.2 Reflection

1. How can we check multiple conditions using the `if` block?

**ANS:** We can use the `elif` (else if) block to check multiple conditions. If a condition specified in the if block is not satisfied or if it evaluates to false, it checks for the condition specified in the `elif` block. If all the specified conditions are evaluated to False, the `else` block is executed.

#### Lesson 3.3 Reflection

1. What will happen if a condition in while statement does not become false? Will the program still be useful?

**ANS:** A loop will continue to run infinitely if a condition never becomes False. This will lead to an infinite loop. This loop can only be used where the server needs to run continuously.

#### Lesson 3.4 Reflection

1. When is it best to use a while loop and when is it best to use an if loop?

**ANS:** When it is not possible to predict the number of iterations, you can use **while** loop. 'If' statement can be used when conditions are known, and the decisions are to be made depending upon certain conditions.



### Lesson 3.5 Reflection

1. Why is iteration important? Why is it also called as looping?

**ANS:** Iterations simplify the programs by stating the steps that we are going to repeat, thus helping us to avoid the code redundancy. Since we go back to one of the previous steps to check if the condition is met, we call it looping.

### Lesson 3.6 Reflection

1. What will happen if the condition is not specified in for loop and the while loop?

**ANS:** If the condition of the 'for' loop is not specified, the iteration will continue for infinite times. In contrast to this, if the condition of the 'while' loop is not specified, a compilation error occurs.

### Lesson 3.7 Reflection

1. What are some supported data types for the range function arguments based on the examples we have seen?

**ANS:** Integers are the supported data types for range function arguments.

### Lesson 3.8 Reflection

1. How do nested for loops work?

**ANS:** Nested loops are loops within loops. In case we have a nested loop in the code, first the iteration of the outer loop is executed. Next, the nested loop is triggered by this outer loop. On execution of the inner loop, the control is again passed to the outer loop to complete the next iteration. This repeats until the looping conditions are satisfied.

### Lesson 3.9 Reflection

1. What are the differences between pass, continue, and break?

**ANS:** The break statement pauses or exits an execution flow if a condition is met. The continue statement skips over a condition's result and resumes the execution flow while the pass statement simply tells the program to do nothing.

## Module 4

### Functions

#### Module Objectives

- Describe the various function types in Python
- Define global and local variables
- Define a function that takes in a variable number of arguments

### Solutions to Reflection

#### Lesson 4.1 Reflection

1. What are some of the names and uses of built in functions we have encountered so far?

**ANS:** We used `print()` to print the given object/text to the console. We used string functions such as `len()` to calculate the length of the string. We also covered the `range()` function, which is used to return a sequence of numbers until a specified limit. It starts from 0 and increments by 1 every time.

#### Lesson 4.2 Reflection

1. What is the difference between functions and methods? Give an example of each.

**ANS:** A method always belongs to a given object while function does not necessarily belong to an object. Function is defined as `function_name(parameters)` whereas a method defined as `object.method()`. Functions are applicable for different type of objects. Consider `len()` function. It can be used with both strings and lists while `replace()` is a string method that cannot be used with list.

#### Lesson 4.3 Reflection

1. What are the benefits of using keyword arguments?

**ANS:** Keyword arguments can be rearranged to improve readability. Arguments with default values can be left out. Arguments call can be done by using their names thus denoting what each argument represents.

#### Lesson 4.4 Reflection

1. In what scenarios do we use anonymous functions?

**ANS:** When you have to repeat a single task throughout the program temporary functions can be useful. Here, anonymous function come to the rescue as it is valid between the scope.

## Module 5

### Lists and Tuples

#### Module Objectives

- Create and access lists in Python
- Describe the various methods that are available in lists, and use them in your Python programs
- Create and access tuples in Python
- Identify the differences between tuples and list
- Implement the various built-in methods that are available with tuples

### Solutions to Reflection

#### Lesson 5.1 Reflection

1. What are the similarities between lists and arrays?

**ANS:** The similarities between list and array are that for both the datatypes are mutable and are used for data storage. We can iterate over both the datatypes and both can be sliced and indexed.

#### Lesson 5.2 Reflection

1. Which list method can be used to iterate over a string? How does it work?

**ANS:** list.extend(iterable) method can be used to iterate over the given string or any other iterable datatype. It appends all the items from the string to the list by looping through every character in the string (including the spaces) thus extending the list further.

#### Lesson 5.3 Reflection

1. What are the advantages of using list comprehensions?

**ANS:** We can reduce three lines of code into one, which can be understood by anyone who is aware of list comprehensions. Python allocates the memory to the list before adding any the elements to it. This helps us to avoid resizing the list on runtime, thus making the code faster.

#### Lesson 5.4 Reflection

1. Is tuple mutable or immutable? Explain why.

**ANS:** Tuples are immutable as we cannot make changes to the elements of a tuple once it has been assigned. However, if the assigned element is a list (mutable in nature), we can change its nested item.

#### Lesson 5.5 Reflection

1. When is it preferable to use slicing and indexing? Why?

**ANS:** When the index position of an object is unknown, we can use the slicing instead of indexing. Trying to acquire the value of an invalid index will lead to `IndexError`. In contrast, slicing will simply return an empty object if the given index is not found.

#### Lesson 5.6 Reflection

1. Is it possible to convert a list into tuple? How?

**ANS:** Yes, the `tuple()` method is used to convert a list into tuple. The syntax of the `tuple()` function is `tuple(list)`. Here, the list is the sequence that we want to convert into a tuple. If we do not pass any sequence to the function, it will return an empty tuple.

## Module 6

### Dictionaries and Sets

#### Module Objectives

- Create and use dictionaries
- Use methods and attributes associated with dictionaries
- Describe and use ordered dictionaries to store and retrieve data in a predictable order
- Create sets, as well as add, read, and remove data from them
- Describe the attributes defined on set objects
- Describe frozen sets

### Solutions to Reflection

#### Lesson 6.1 Reflection

1. What is the difference between ordered and unordered dictionary?

**ANS:** Insertion order is not tracked in a regular dictionary. However, the insertion order is tracked and utilized while creating an iterator in an unordered dictionary. When testing for equality, regular dictionary considers its contents whereas ordered dictionary looks at the order in which items were inserted.

#### Lesson 6.2 Reflection

1. What are the differences in adding data to a dictionary on creation when using the dict() function and using curly bracket notation? Do they result in the same dictionary object being created?

**ANS:** Differences are in syntax (dict method takes arguments like a=b whereas curly bracket notation uses : to denote key value pairs). The result is the same dictionary object.

#### Lesson 6.3 Reflection

1. What benefits would you get from using the in keyword instead of iteration?

**ANS:** Faster code execution and more concise syntax.

#### Lesson 6.4 Reflection

1. Can you name a set of items they have encountered in real life?

**ANS:** A set brings to mind a collection of items. Students can name any collection of unique items as sets they have encountered. An example could be a set of names, for example, Peter, Paul, and Mary.

#### Lesson 6.5 Reflection

1. Imagine a situation where you have three sets: A, B, and C. Set A is the names of people who ate food A, set B is of people who ate food B, and so on. A person could have eaten more than one food type so a name could, for example, be in both set A and C. You have another set of names D. This is a set of names of people who got food poisoning. How would you go about finding which food caused the food poisoning?

**ANS:** A decent attempt would be to get the intersection of set D with A, B, and C and the intersection with the highest number of names would reveal the culprit.

#### Lesson 6.6 Reflection

1. Can you name a set of items you have encountered in real life?

**ANS:** A set brings to mind a collection of items. Students can name any collection of unique items as sets they have encountered. An example could be a set of names, for example, Peter, Paul, and Mary.

## Module 7

### Object-Oriented Programming

#### Module Objectives

- Explain different OOP concepts and the importance of OOP
- Instantiate a class
- Describe how to define instance methods and pass arguments to them
- Declare class attributes and class methods
- Describe how to override methods
- Implement multiple inheritance

#### Lesson 7.1 Reflection

1. What is a programming paradigm? Name and explain any two.

**ANS:** A programming paradigm is way of solving problems using programming languages. Some common paradigms are object-oriented and procedural. Functional programming is a way of developing software by creating functions throughout. Object-oriented programs are developed using classes and objects.

#### Lesson 7.2 Reflection

1. What are the benefits of using multiple inheritance?

**ANS:** Multiple Inheritance allows to compose classes using the functionalities of multiple modules at the same time. This also enables code reusability.

2. What are some characteristics of a person you can think of?

**ANS:** Every person has a name, age, weight, occupation, and so on.

#### Lesson 7.3 Reflection

1. How is the *this* keyword used in Python?

**ANS:** An equivalent to self in other languages is the *this* keyword, such as in C++, Java, or JavaScript.

#### Lesson 7.4 Reflection

1. What is the function of the `__dict__` attribute?

**ANS:** All Python objects have a hidden dictionary attribute that holds all the attributes of that object. It helps with stability.



### Lesson 7.5 Reflection

1. What are the behaviors of an object referred to as?

**ANS:** Methods.

2. What is the use of pass?

**ANS:** pass is a placeholder for when we don't have any instructions to place in a block. It essentially tells the interpreter to do nothing.

### Lesson 7.6 Reflection

1. What is the purpose of overriding?

**ANS:** To modify the behavior of a subclass that's already been defined by a parent class.

### Lesson 7.7 Reflection

1. What is one of the risks with attributes in multiple inheritance, and how does python resolve issues?

**ANS:** With multiple inheritance there may be issues with attributes if they have the same name. When implementing a function with the same name Python gives priority to the parent classes in the order of depth first, left to right.

## Module 8

### Modules, Packages, and File Operations

#### Module Objectives

- Describe what Python modules are and create your own
- Describe what Python packages are and create your own
- Work with the built-in Python modules
- Describe the `file` object in Python
- Read and write to Python files
- Work with structured data in Python files

### Solutions to Reflection

#### Lesson 8.1 Reflection

1. Why do we use Python import statement? What happens when the interpreter encounters an import statement?

**ANS:** The primary use of import statement is to gain access to the code present in another module. The compiler then imports the module if the module is present.

#### Lesson 8.2 Reflection

1. In what situation can file operations fail?

**ANS:** File operations can fail in situations wherein the operation is not defined for some reasons like opening a file that does not exist or writing a file opened for reading. It can also fail in situations where usage of modes is improper.

#### Lesson 8.3 Reflection

1. What are some Python best practices when building user modules?

**ANS:** Some best practices include using four spaces for indentation, using snake\_case for variable names, and so on. The following are some examples:

1. Naming variables, functions, modules—lowercase\_with\_underscores.
2. Naming classes—CapitalizeFirstLetters.
3. Avoid variable names like `k`, `c`, and so on, except where their meanings can be derived from the context, for example, looping.
4. Use comments sparingly.
5. Write tests.

#### Lesson 8.4 Reflection

1. What does the PATH variable normally do?

**ANS:** The PATH variable defines where executable items are to be found on a computer's disk. Any commands you run will be resolved by the OS referencing this variable to know where to look for the program you are trying to invoke.

#### Lesson 8.5 Reflection

1. What kind of file types have you come across?

**ANS:** Examples could be .txt, .xlsx, .docx, and so on.

#### Lesson 8.6 Reflection

1. What do you think the w flag does? What are the common errors in file operations?

**ANS:** Using w instead of a will clear the existing file. Examples are FileNotFoundError, opening the file in the wrong mode, and so on.

2. Structured data is data that must be arranged in a particular format. Have you ever encountered any structured data before?

**ANS:** Students could have encountered JSON, XML, or CSV files. Other examples are spreadsheets, for example, Pages files or Excel files.

#### Lesson 8.7 Reflection

1. What is a CSV file and what are the advantages of this format?

**ANS:** A CSV file uses a predetermined separation character, such as a comma, to separate data points. It can be saved as plain text or even copy and pasted, so has fairly small files size and can easily be used by various applications.

2. When might you see JSON data?

**ANS:** JSON is Javascript Object Notation and is often found in data transfer used by third-party APIs.

## Module 9

### Error Handling

#### Module Objectives

- Describe what errors and exceptions are
- Handle errors and exceptions when they occur
- Define and use your own custom exceptions

### Solutions to Reflection

#### Lesson 9.1 Reflection

1. What is the difference between errors and exceptions?

**ANS:** Most of the times we cannot handle the errors. They are unchecked exceptions and we cannot do anything to handle them. They are also referred to as bugs. Exceptions are errors that occur during the execution of the program.

#### Lesson 9.2 Reflection

1. What leads to syntax error in Python?

**ANS:** Syntax errors can be caused by misplacing a keyword in the code. It can also be caused by a missing semicolon or comma or due to extra braces. It can also be caused by misspelling a keyword, incorrect indentation, or empty code block.

#### Lesson 9.3 Reflection

1. Why do you think we should handle errors and exceptions?

**ANS:** To prevent our code from responding unexpectedly to error conditions.

#### Lesson 9.4 Reflection

1. What is the advantage of creating your own custom exception class?

**ANS:** You can catch errors that might be specific to your code or give error reports that specifically tell you what kind of error you have found.

