1. Introduction to Python 3

1.1 Programming (general)

program	A computer program consists of instructions executing one at a time.
Input	Input: A program gets data, perhaps from a file, keyboard, touchscreen, network, etc.
Process	Process: A program performs computations on that data, such as adding two values like x + y.
Output	Output: A program puts that data somewhere, such as to a file, screen, or network.
variables	Programs use variables to refer to data, like x.
computational thinking	In the information age, many people believe computational thinking, or creating a sequence of instructions to solve a problem, will become increasingly important for work and everyday life.
algorithm	A sequence of instructions that solves a problem is called an algorithm.
Animation	1.1.1 A basic computer program.
Question set	1.1.2 A basic computer program.
Aside	A program is like a recipe
Learning tool	1.1.3 A first programming activity.
Question set	1.1.4 Instructions.
Learning tool	1.1.5 Computational thinking: Creating algorithms to draw shapes using turtle graphics.

1.2 Programming using Python

Python interpreter	The Python interpreter is a computer program that executes code written in the Python programming language.
interactive interpreter	An interactive interpreter is a program that allows the user to execute one line of code at a time.
Code	Code is a common word for the textual representation of a program (and hence programming is also called *coding*).
line	A line is a row of text.
prompt	The interactive interpreter displays a prompt (">>>") that indicates the interpreter is ready to accept code.
statement	A statement is a program instruction.
Expressions	Expressions are code that return a value when evaluated; for example, the code `wage * hours * weeks` is an expression that computes a number.
variables	The names wage, hours, weeks, and salary are variables, which are named references to values stored by the interpreter.
assignment	A new variable is created by performing an assignment using the = symbol.
print()	Print() function displays variables or expression values.
comments	'#' characters denote comments, which are optional but can be used to explain portions of code to a human reader.
Animation	1.2.1 The Python interpreter.
Question set	1.2.2 Match the Python terms with their definitions.

Animation	1.2.3 Executing a simple Python program.
Question set	1.2.4 Python basics.

1.3 Basic input and output

print()	The primary way to print output is to use the built-in function print().
string literal	Text enclosed in quotes is known as a string literal.
newline	Each print statement will output on a new line. A new output line starts after each print statement, called a newline.
whitespace	Any space, tab, or newline is called whitespace.
newline character	Output can be moved to the next line by printing `\n`, known as a newline character.
input()	Reading input is achieved using the input() function.
string	The input obtained by input() is any text that a user typed, including numbers, letters, or special characters like # or @. Such text in a computer program is called a string and is always surrounded by single or double quotes, for example 'Hello' or "#Goodbye# Amigo!".
type	Strings and integers are each an example of a type; a type determines how a value can behave.
int()	If a string contains only numbers, like '123', then the int() function can be used to convert that string to the integer 123.

Figure	1.3.1 Printing text and new lines.
Question set	1.3.1 Basic text output.
Question set	1.3.2 Basic text output.

Coding challenge	1.3.1 Output simple text.
Coding challenge	1.3.2 Output an eight with asterisks.
Figure	1.3.2 Printing the value of a variable.
Question set	1.3.3 Basic variable output.
Question set	1.3.4 Basic variable output.
Figure	1.3.3 Printing multiple items using a single print statement.
Figure	1.3.4 Printing using newline characters.
Figure	1.3.5 printing without text.
Learning tool	1.3.5 Output simulator.
Question set	1.3.6 Single print statement.
Progression	1.3.3 Enter the output.
Animation	1.3.7 A program can get an input value from the keyboard.
Question set	1.3.8 Reading user input.
Question set	1.3.9 Reading user input.
Figure	1.3.6 Using int() to convert strings to integers.
Figure	1.3.7 Converting user input to integers.
Question set	1.3.10 Converting user input to integers.
Figure	1.3.8 Basic input example.
Coding challenge	1.3.4 Read user input and print to output.

Coding challenge	1.3.5 Read user input numbers and perform a calculation.

1.4 Errors

syntax error	One kind of mistake, known as a syntax error, is to violate a programming language's rules on how symbols can be combined to create a program.
runtime error	The program may have another kind of error called a runtime error, wherein a program's syntax is correct but the program attempts an impossible operation, such as dividing by zero or multiplying strings together (like 'Hello' * 'ABC').
crash	Abrupt and unintended termination of a program is often called a crash of the program.
logic error	Such an error is known as a logic error, because the program is logically flawed.
bug	A logic error is often called a bug.
Figure	1.4.1 A program with a syntax error.
Question set	1.4.1 Syntax errors.
Question set	1.4.2 Common syntax errors.
Animation	1.4.3 Run code frequently to avoid many errors.
Question set	1.4.4 Testing for syntax errors.
Figure	1.4.2 Runtime errors can crash the program.
Table	1.4.1 Common error types.
Question set	1.4.5 Match the lines of code with the error type that they produce.

Figure	1.4.3 The programmer made a mistake that happens to be correct syntax, but has a different meaning.
Figure	1.4.4 The first bug.
Coding challenge	1.4.1 Basic syntax errors.

1.5 Development environment

integrated development environment	Code development is usually done with an integrated development environment, or IDE.
IDE	Code development is usually done with an integrated development environment, or IDE.
Figure	1.5.1 IDLE environment for coding and running Python.
Question set	1.5.1 Development environment basics.

1.6 Computers and programs (general)

bits	0s and 1s are known as bits (*b*inary dig*its*).
processors	To support different calculations, circuits called processors were created to process (aka *execute*) a list of desired calculations, each calculation called an instruction.
instruction	To support different calculations, circuits called processors were created to process (aka *execute*) a list of desired calculations, each calculation called an instruction.
memory	A memory is a circuit that can store 0s and 1s in each of a series of thousands of addressed locations.
program	The programmer-created sequence of instructions is called a program, application, or just app.
application	

	The programmer-created sequence of instructions is called a program, application, or just app.
app	The programmer-created sequence of instructions is called a program, application, or just app.
machine instructions	Instructions represented as 0s and 1s are known as machine instructions.
executable program	A sequence of machine instructions together form an executable program (sometimes just called an *executable*).
assembly	Because 0s and 1s are hard to comprehend, programmers soon created programs called *assemblers* to automatically translate human readable instructions, such as "Mul 97, #9, 98", known as assembly language instructions, into machine instructions.
high-level languages	In the 1960s and 1970s, programmers created high-level languages to support programming using formulas or algorithms.
compilers	To support high-level languages, programmers created compilers, which are programs that automatically translate high-level language programs into executable programs.
Figure	1.6.1 Looking under the hood of a car.
Learning tool	1.6.1 A bit is either 1 or 0, like a light switch is either on or off (click the switch).
Figure	1.6.2 Early computer made from thousands of switches.
Figure	1.6.3 As switches shrunk, so did computers. The computer processor chip on the right has millions of switches.

Figure	1.6.4 Memory.
Animation	1.6.2 Computer processor and memory.
Table	1.6.1 Sample processor instructions.
Animation	1.6.3 Memory stores instructions and data as 0s and 1s.
Animation	1.6.4 Processor executing instructions.
Question set	1.6.5 Computer basics.
Animation	1.6.6 Program compilation and execution.
Question set	1.6.7 Programs.

1.7 Computer tour

Input/output devices

screen	A screen (or monitor) displays items to a user.
keyboard	A keyboard allows a user to provide input to the computer.

Storage

disk	A disk (aka *hard drive*) stores files and other data, such as program files, song/movie files, or office documents.

Memory

RAM	RAM (random-access memory) temporarily holds data read from storage and is designed such that any address can be accessed much faster than disk, in just a few clock ticks (see below) rather than hundreds of ticks.
byte	A byte is 8 bits.

Processor	
processor	The processor runs the computer's programs, reading and executing instructions from memory, performing operations, and reading/writing data from/to memory.
operating system	The operating system allows a user to run other programs and interfaces with the many other peripherals.
cache	A processor may contain a small amount of RAM on its own chip, called cache memory, accessible in one clock tick rather than several, for maintaining a copy of the most-used instructions/data.
Clock	
clock	A processor's instructions execute at a rate governed by the processor's clock, which ticks at a specific frequency.
transistors	Engineers created smaller switches called transistors, which in 1958 were integrated onto a single chip called an integrated circuit or IC.
integrated circuit	Engineers created smaller switches called transistors, which in 1958 were integrated onto a single chip called an integrated circuit or IC.
IC	Engineers created smaller switches called transistors, which in 1958 were integrated onto a single chip called an integrated circuit or IC.
Moore's Law	Moore's Law: The doubling of IC capacity roughly every 18 months, which continues today.
Animation	1.7.1 Some computer components.
Question set	1.7.2 Programs.

1.8 Language history

scripting languages	As computing evolved throughout the 1960s and 1970s, programmers began creating scripting languages to execute programs without the need for compilation.
script	A script is a program whose instructions are executed by another program called an interpreter.
interpreter	A script is a program whose instructions are executed by another program called an interpreter.
Python	In the late 1980s, Guido van Rossum began creating a scripting language called Python and an accompanying interpreter.
backwards compatible	Python 2.7 programs cannot run on Python 3.0 or later interpreters, i.e., Python 3.0 is not backwards compatible.
open-source	Python is an open-source language, meaning the community of users participate in defining the language and creating new interpreters, and is supported by a large community of programmers.

Question set 1.8.1 Python background.

1.9 Why whitespace matters

Animation 1.9.1 Precisely formatting a meeting invite.

Question set 1.9.2 Program correctness includes correctly-formatted output.

Question set 1.9.3 Thinking precisely, and attention to detail.

Aside Programmer attention to details

1.11 Additional practice: Output art

Figure 1.11.1 Output art: Printing a triangle.

1.12 zyLab training: Basics

- **Lab activity** 1.12.1 zyLab training: Basics

1.13 zyLab training: Interleaved input / output

- **Lab activity** 1.13.1 zyLab training: Interleaved input / output

1.14 LAB: Formatted output: Hello World!

- **Lab activity** 1.14.1 LAB: Formatted output: Hello World!

1.15 LAB: Formatted output: No parking sign

- **Lab activity** 1.15.1 LAB: Formatted output: No parking sign

1.16 LAB: Input: Welcome message

- **Lab activity** 1.16.1 LAB: Input: Welcome message

1.17 LAB: Input: Mad Lib

- **Lab activity** 1.17.1 LAB: Input: Mad Lib

1.18 LAB: Warm up: Hello world

- **Lab activity** 1.18.1 LAB: Warm up: Hello world

1.19 LAB: Warm up: Basic output with variables

- **Lab activity** 1.19.1 LAB: Warm up: Basic output with variables

1.20 LAB*: Program: ASCII art

- **Lab activity** 1.20.1 LAB*: Program: ASCII art

1.21 Clone of LAB*: Program: ASCII art

- **Lab activity** 1.21.1 LAB*: Program: ASCII art

1.22 Lab 1 Problem 1

- **Lab activity** 1.22.1 Lab 1 Problem 1

1.23 Lab 1 Problem 2

- **Lab activity** 1.23.1 Lab 1 Problem 2

1.24 Lab 1 Problem 3

Lab activity	1.24.1 Lab 1 Problem 3

2. Variables and Expressions

2.1 Variables and assignments

variable	In a program, a variable is a named item, such as x or num_people, used to hold a value.
assignment statement	An assignment statement assigns a variable with a value, such as x = 5.
incrementing	Increasing a variable's value by 1, as in x = x + 1, is common and known as incrementing the variable.
Progression	2.1.1 People on bus.
Animation	2.1.1 Variables and assignments.
Aside	= is not equals
Question set	2.1.2 Valid assignment statements.
Question set	2.1.3 Variables and assignment statements.
Learning tool	2.1.4 Trace the variable value.
Animation	2.1.5 A variable may appear on the left and right of an assignment statement.
Question set	2.1.6 Variable on both sides.
Coding challenge	2.1.2 Assigning a sum.
Coding challenge	2.1.3 Multiplying the current value of a variable.

2.2 Identifiers

identifier	An identifier, also called a name, is a sequence of letters (a-z, A-Z), underscores (_), and digits (0–9), and must start with a letter or an underscore.

name	An identifier, also called a name, is a sequence of letters (a-z, A-Z), underscores (_), and digits (0–9), and must start with a letter or an underscore.
underscores	An identifier, also called a name, is a sequence of letters (a-z, A-Z), underscores (_), and digits (0–9), and must start with a letter or an underscore.
case sensitive	Python is case sensitive, meaning upper- and lowercase letters differ. Ex: "Cat" and "cat" are different.
Reserved words	Reserved words, or keywords, are words that are part of the language, and thus, cannot be used as a programmer-defined name.
keywords	Reserved words, or keywords, are words that are part of the language, and thus, cannot be used as a programmer-defined name.
PEP 8	PEP 8 (PEP is an acronym for Python Enhancement Proposal) is a document that outlines the basics of how to write Python code neatly and consistently.
Question set	2.2.1 Valid names.
Table	2.2.1 Use meaningful variable names.
Table	2.2.2 Python 3 reserved keywords.
Learning tool	2.2.2 Python 3 name validator.

2.3 Objects

object	An object represents a value and is automatically created by the interpreter when executing a line of code. For example, executing `x = 4` creates a new object to represent the value 4.
garbage collection	

	Deleting unused objects is an automatic process called garbage collection that helps to keep the memory of the computer less utilized.
Name binding	Name binding is the process of associating names with interpreter objects. An object can have more than one name bound to it, and every name is always bound to exactly one object. Name binding occurs whenever an assignment statement is executed, as demonstrated below.
Value	Value: A value such as "20", "abcdef", or 55.
Type	Type: The type of the object, such as integer or string.
Identity	Identity: A unique identifier that describes the object.
type()	The built-in function type() prints the type of an object.
Mutability	Mutability indicates whether the object's value is allowed to be changed.
immutable	Integers and strings are immutable; modifying their values with assignment statements results in new objects being created and the names bound to the new object.
id()	Python provides a built-in function id() that gives the value of an object's identity.
Animation	2.3.1 Creating new objects.
Animation	2.3.2 Manipulating variables.
Figure	2.3.1 Printing displays an object's value.
Figure	2.3.2 Using type() to print an object's type.
Figure	2.3.3 Using id() to print an object's identity.

Question set	2.3.3 Objects basics.

2.4 Numeric types: Floating-point

floating-point number	A floating-point number is a real number, like 98.6, 0.0001, or -666.667.
float	Thus, float is a data type for floating-point numbers.
floating-point literal	A floating-point literal is written with the fractional part even if that fraction is 0, as in 1.0, 0.0, or 99.0.
scientific notation	A floating-point literal using scientific notation is written using an e preceding the power-of-10 exponent, as in 6.02e23 to represent 6.02×10^{23}.
OverflowError	Assigning a floating-point value outside of this range generates an OverflowError.
Overflow	Overflow occurs when a value is too large to be stored in the memory allocated by the interpreter.
Figure	2.4.1 A program using float-type variables.
Question set	2.4.1 Scientific notation.
Figure	2.4.2 Float can overflow.
Question set	2.4.2 Floating-point versus integer.
Coding challenge	2.4.1 Gallons of paint needed to paint walls.

2.5 Arithmetic expressions

expression	An expression is a combination of items, like variables, literals, operators, and parentheses, that evaluates to a value, like 2 * (x + 1).
literal	A literal is a specific value in code like 2.
operator	

	An operator is a symbol that performs a built-in calculation, like +, which performs addition.
addition	The addition operator is +, as in x + y.
+	The addition operator is +, as in x + y.
subtraction	The subtraction operator is -, as in x - y.
-	The subtraction operator is -, as in x - y.
negation	The - operator is for negation, as in -x + y, or x + -y.
multiplication	The multiplication operator is *, as in x * y.
*****	The multiplication operator is *, as in x * y.
division	The division operator is /, as in x / y.
/	The division operator is /, as in x / y.
exponent	The exponent operator is **, as in x ** y (x to the power of y).
******	The exponent operator is **, as in x ** y (x to the power of y).
evaluates	An expression evaluates to a value, which replaces the expression. Ex: If x is 5, then x + 1 evaluates to 6, and y = x + 1 assigns y with 6.
precedence rules	An expression is evaluated using the order of standard mathematics, and such order is known in programming as precedence rules.
Table	2.5.1 Arithmetic operators.
Question set	2.5.1 Expressions.

Question set	2.5.2 Capturing behavior with an expression.
Table	2.5.2 Precedence rules for arithmetic operators.
Animation	2.5.3 Evaluating expressions.
Question set	2.5.4 Evaluating expressions and precedence rules.
Aside	Using parentheses to make the order of evaluation explicit
Question set	2.5.5 Converting a formatted expression to a program expression.

2.6 Python expressions

unary minus	Minus (-) used as negative is known as unary minus.
compound operators	Special operators called compound operators provide a shorthand way to update a variable, such as age += 1 being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.
+=	Special operators called compound operators provide a shorthand way to update a variable, such as age += 1 being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.
-=	Special operators called compound operators provide a shorthand way to update a variable, such as age += 1 being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.
***=**	Special operators called compound operators provide a shorthand way to update a variable, such as age += 1 being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.
/=	Special operators called compound operators provide a shorthand way to update a variable, such as age += 1

being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.

%=

Special operators called compound operators provide a shorthand way to update a variable, such as age += 1 being shorthand for age = age + 1. Other compound operators include -=, *=, /=, and %=.

Figure 2.6.1 Expression example: Leasing cost.

Question set 2.6.1 Simple program with an arithmetic expression.

Question set 2.6.2 Single space around operators.

Question set 2.6.3 Compound operators.

Question set 2.6.4 Assigning an integer literal.

Coding challenge 2.6.1 Computing an average.

Coding challenge 2.6.2 Sphere volume.

Coding challenge 2.6.3 Acceleration of gravity.

2.7 Division and modulo

modulo operator

The modulo operator (%) evaluates the remainder of the division of two integer operands. Ex: 23 % 10 is 3.

%

The modulo operator (%) evaluates the remainder of the division of two integer operands. Ex: 23 % 10 is 3.

Question set 2.7.1 Division and floored division.

Question set 2.7.2 Modulo.

Progression 2.7.1 Enter the output of the integer expressions.

Example 2.7.1 Getting digits.

Example	2.7.2 Get prefix.
Question set	2.7.3 Modulo examples.
Coding challenge	2.7.2 Compute change.

2.8 Module basics

script	Programmers typically write Python program code in a file called a script.
module	A module is a file containing Python code that can be used by other modules or scripts.
import	A module is made available for use via the import statement.
dot notation	Once a module is imported, any object defined in that module can be accessed using dot notation.
__name__	Python programs often use the built-in special name __name__ to determine if the file was executed as a script by the programmer, or if the file was imported by another module.

Animation	2.8.1 Scripts are files executed by the interpreter.
Animation	2.8.2 Importing modules.
Question set	2.8.3 Basic modules.
Figure	2.8.1 Checking if a file was executed as a script.
Question set	2.8.4 Importing modules and executing scripts.

2.9 Math module

math module	Python comes with a standard math module to support such advanced math operations.
module	A module is Python code located in another file.

function	A function is a list of statements that can be executed simply by referring to the function's name.
function call	The process of invoking a function is referred to as a function call.
argument	The item passed to a function is referred to as an argument.
Figure	2.9.1 Importing the math module and calling a math module function.
Table	2.9.1 Functions in the standard math module.
Question set	2.9.1 Variable assignments with math functions.
Coding challenge	2.9.1 Coordinate geometry.
Coding challenge	2.9.2 Tree height.

2.10 Representing text

Unicode	Python uses Unicode to represent every possible character as a unique number, known as a code point.
code point	Python uses Unicode to represent every possible character as a unique number, known as a code point.
newline	A newline character, which indicates the end of a line of text, is encoded as 10.
backslash	The \ is known as a backslash.
escape sequence	The two-item sequence is called an escape sequence.
raw string	Escape sequences can be ignored using a raw string.
ord()	The built-in function ord() returns an encoded integer value for a string of length one.

chr()		The built-in function chr() returns a string of one character for an encoded integer.
	Table	2.10.1 Encoded text values.
	Question set	2.10.1 Unicode.
	Table	2.10.2 Common escape sequences.
	Question set	2.10.2 Escape sequences.
	Figure	2.10.1 Ignoring escape characters with a raw string.
	Learning tool	2.10.3 Using ord() to convert a character to the encoded value.
	Learning tool	2.10.4 Using chr() to convert an encoded value to a character.
	Question set	2.10.5 Text.

2.12 LAB: Divide by x

	Lab activity	2.12.1 LAB: Divide by x

2.13 LAB: Driving costs

	Lab activity	2.13.1 LAB: Driving costs

2.14 LAB: Expression for calories burned during workout

	Lab activity	2.14.1 LAB: Expression for calories burned during workout

2.15 LAB: Using math functions

	Lab activity	2.15.1 LAB: Using math functions

2.16 LAB: Musical note frequencies

	Lab activity	2.16.1 LAB: Musical note frequencies

2.17 LAB: Warm up: Variables, input, and type conversion

Lab activity — 2.17.1 LAB: Warm up: Variables, input, and type conversion

2.18 LAB*: Program: Cooking measurement converter

Lab activity — 2.18.1 LAB*: Program: Cooking measurement converter

2.19 LAB*: Program: Food receipt

Lab activity — 2.19.1 LAB*: Program: Food receipt

3. Types

3.1 String basics

string — A string is a sequence of characters, like the text MARY, that can be stored in a variable.

string literal — A string literal is a string value specified in the source code of a program.

sequence type — Sequence type: A type that specifies a collection of objects ordered from left to right.

len() — The len() built-in function can be used to find the length of a string (and any other sequence type).

brackets — A programmer can access a character at a specific index by appending brackets [ ] containing the index.

string concatenation — A program can add new characters to the end of a string in a process known as string concatenation.

Learning tool — 3.1.1 String indexing.

Question set — 3.1.2 String literals.

Question set — 3.1.3 String basics.

Figure — 3.1.1 Using len() to get the length of a string.

Question set — 3.1.4 Using len() to find the length of a string.

Figure	3.1.2 Accessing individual characters of a string.
Question set	3.1.5 String indexing.
Figure	3.1.3 Strings are immutable and cannot be changed.
Figure	3.1.4 String concatenation.
Question set	3.1.6 String variables.
Coding challenge	3.1.1 Reading multiple data types.
Coding challenge	3.1.2 Concatenating strings.

3.2 List basics

container	A container is a construct used to group related values together and contains references to other objects instead of data.
list	A list is a container created by surrounding a sequence of variables or literals with brackets [ ].
element	A list item is called an element .
index	Elements are ordered by position in the list, known as the element's index, starting with 0.
method	A method instructs an object to perform some action, and is executed by specifying the method name following a "." symbol and an object.
append()	The append() list method is used to add new elements to a list.
pop()	Elements can be removed using the pop() or remove() methods.
remove()	Elements can be removed using the pop() or remove() methods.

Sequence-type functions	Sequence-type functions are built-in functions that operate on sequences like lists and strings.
Sequence-type methods	Sequence-type methods are methods built into the class definitions of sequences like lists and strings.
tuple	A tuple, usually pronounced "tuhple" or "toople", behaves similar to a list but is immutable – once created the tuple's elements cannot be changed.
named tuple	Named tuple allows the programmer to define a new simple data type that consists of named attributes.
namedtuple	The namedtuple package must be imported to create a new named tuple.

Animation	3.2.1 Creating lists.
Question set	3.2.2 Creating lists.
Figure	3.2.1 Access list elements using an indexing expression.
Figure	3.2.2 Modifying list elements.
Question set	3.2.3 Accessing list elements.
Coding challenge	3.2.1 Initialize a list.
Animation	3.2.4 Adding and removing list elements.
Question set	3.2.5 List modification.
Table	3.2.1 Some of the functions and methods useful to lists.
Figure	3.2.3 Using sequence-type functions with lists.
Question set	3.2.6 Using sequence-type functions.
Progression	3.2.2 List functions and methods.

Figure	3.2.4 Using tuples.
Question set	3.2.7 Tuples.
Coding challenge	3.2.3 Initialize a tuple.
Figure	3.2.5 Creating named tuples.
Question set	3.2.8 Named tuples.
Coding challenge	3.2.4 Creating a named tuple

3.3 Set basics

set	A set is an unordered collection of unique elements.
set()	A set can be created using the set() function, which accepts a sequence-type iterable object (list, tuple, string, etc.) whose elements are inserted into the set.
set literal	A set literal can be written using curly braces `{ }` with commas separating set elements.
add()	The add() method places a new element into the set if the set does not contain an element with the provided value.
remove()	The remove() and pop() methods remove an element from the set.
pop()	The remove() and pop() methods remove an element from the set.
Figure	3.3.1 Creating sets.
Question set	3.3.1 Basic sets.
Table	3.3.1 Some of the methods useful to sets.
Animation	3.3.2 Modifying sets.

Question set	3.3.3 Modifying sets.
Coding challenge	3.3.1 Creating and modifying sets.
Table	3.3.2 Common set theory operations.
Animation	3.3.4 Set theory operations.
Question set	3.3.5 Set theory operations.
Coding challenge	3.3.2 Set theory methods.

3.4 Dictionary basics

dictionary	A dictionary is a Python container used to describe associative relationships.
dict	A dictionary is represented by the dict object type.
key	A key is a term that can be located in a dictionary, such as the word "cat" in the English dictionary.
value	A value describes some data associated with a key, such as a definition.
curly braces	A dict object is created using curly braces { } to surround the key:value pairs that comprise the dictionary contents.
key:value pairs	A dict object is created using curly braces { } to surround the key:value pairs that comprise the dictionary contents.
KeyError	If no entry with a matching key exists in the dictionary, then a KeyError runtime error occurs and the program is terminated.
del	The del keyword is used to remove entries from a dictionary: `del prices['papaya']` removes the entry whose key is 'papaya'.

	Figure	3.4.1 Creating a dictionary.
	Question set	3.4.1 Create a dictionary.
	Figure	3.4.2 Accessing dictionary entries.
	Question set	3.4.2 Accessing dictionary entries.
	Figure	3.4.3 Adding and editing dictionary entries.
	Question set	3.4.3 Modifying dictionaries.
	Coding challenge	3.4.1 Modify and add to dictionary.

3.5 Common data types summary

Numeric types	Numeric types int and float represent the most common types used to store data.
Sequence types	Sequence types string, list, and tuple are all containers for collections of objects ordered by position in the sequence, where the first object has an index of 0 and subsequent elements have indices 1, 2, etc.
mapping type	The only mapping type in Python is the dict type. Like a sequence type, a dict serves as a container.

	Table	3.5.1 Common data types.
	Table	3.5.2 Containers: sequence and mapping types.
	Question set	3.5.1 Common data types.
	Question set	3.5.2 Choosing among different container types.
	Question set	3.5.3 Finding errors in container code.

3.7 Type conversions

type conversion

A type conversion is a conversion of one type to another, such as an int to a float.

implicit conversion — An implicit conversion is a type conversion automatically made by the interpreter, usually between numeric types.

- Question set — 3.7.1 Implicit conversions between float and int.
- Table — 3.7.1 Conversion methods for some common types.
- Question set — 3.7.2 Type conversions.
- Coding challenge — 3.7.1 Type casting: Computing average owls per zoo.
- Coding challenge — 3.7.2 Type casting: Reading and adding values.

3.8 Binary numbers

binary number — Because each memory location is composed of bits (0s and 1s), a processor stores a number using base 2, known as a binary number.

decimal number — For a number in the more familiar base 10, known as a decimal number, each digit must be 0-9 and each digit's place is weighed by increasing powers of 10.

base 2 — In base 2, each digit must be 0-1 and each digit's place is weighed by increasing powers of 2.

- Table — 3.8.1 Decimal numbers use weighed powers of 10.
- Table — 3.8.2 Binary numbers use weighed powers of 2.
- Learning tool — 3.8.1 Binary number tool.
- Question set — 3.8.2 Binary numbers.
- Progression — 3.8.1 Create a binary number.

3.9 String formatting

format()	The string format() function allows a programmer to create a string with placeholders that are replaced by values or variable values at execution.
replacement field	A placeholder surrounded by curly braces `{ }` is called a replacement field.
keyword argument	Named replacement allows a programmer to create a keyword argument that defines a name and value in the `format()` parentheses.
format specification	A format specification inside of a replacement field allows a value's formatting in the string to be customized.
presentation type	A common format specification is to provide a presentation type for the value, such as integer (4), floating point (4.0), fixed precision decimal (4.000), percentage (4%), binary (100), etc.
Animation	3.9.1 String formatting.
Table	3.9.1 Three ways to format strings.
Animation	3.9.2 Positional and named replacement in format strings.
Question set	3.9.3 string.format() usage.
Table	3.9.2 Common formatting specification presentation types.
Question set	3.9.4 Format specifications and presentation types.
Table	3.9.3 Referencing the correct format() values in replacement fields.
Question set	3.9.5 Matching code blocks to formatted strings.

Coding challenge	3.9.1 Printing a string.
Progression	3.9.2 String formatting.

3.11 LAB: Input and formatted output: Right-facing arrow

Lab activity	3.11.1 LAB: Input and formatted output: Right-facing arrow

3.12 LAB: Phone number breakdown

Lab activity	3.12.1 LAB: Phone number breakdown

3.13 LAB: Input and formatted output: Caffeine levels

Lab activity	3.13.1 LAB: Input and formatted output: Caffeine levels

3.14 LAB: Input and formatted output: House real estate summary

Lab activity	3.14.1 LAB: Input and formatted output: House real estate summary

3.15 LAB: Simple statistics

Lab activity	3.15.1 LAB: Simple statistics

3.16 LAB: Warm up: Creating passwords

Lab activity	3.16.1 LAB: Warm up: Creating passwords

3.17 LAB*: Program: Painting a wall

Lab activity	3.17.1 LAB*: Program: Painting a wall

4. Branching

4.1 If-else branches (general)

branch	A branch is a program path taken only if an expression's value is true.
If	If branch: A branch taken only if an expression is true.
if-else	An if-else structure has two branches: The first branch is taken if an expression is true, else the other branch is taken.

- **Animation** 4.1.1 Branching concept.
- **Question set** 4.1.2 Branch concept.
- **Animation** 4.1.3 A simple branch: Hotel discount.
- **Question set** 4.1.4 Branches.
- **Animation** 4.1.5 Example if branch: Computing absolute value.
- **Question set** 4.1.6 Example if branch: Absolute value.
- **Animation** 4.1.7 If-else branches.
- **Question set** 4.1.8 If-else branches.
- **Animation** 4.1.9 If-else example: Max.
- **Question set** 4.1.10 If-else example: Max.
- **Animation** 4.1.11 If-elseif-else branch.
- **Question set** 4.1.12 If-elseif-else.

4.2 If-else statement

if

An if statement executes a group of statements if an expression is true.

if-else

An if-else statement executes one group of statements when an expression is true and another group of statements when the expression is false.

equality operator

The equality operator == evaluates to true if the left side and right side are equal.

==

The equality operator == evaluates to true if the left side and right side are equal.

- **Animation** 4.2.1 If statement: Hotel discount.

Question set	4.2.2 If statement.
Construct	4.2.1 If-else statement.
Animation	4.2.3 If-else statement: Car insurance.
Aside	Car insurance prices
Question set	4.2.4 If-else statements.
Question set	4.2.5 Writing an if-else statement.
Progression	4.2.1 Enter the output for the if-else branches.
Coding challenge	4.2.2 Basic if-else expression.
Coding challenge	4.2.3 Basic if-else.
Construct	4.2.2 Multi-branch if-else statement. Only 1 branch will execute.
Figure	4.2.1 Multi-branch if-else example: Anniversaries.
Question set	4.2.6 Multi-branch if-else statements.
Coding challenge	4.2.4 Multi-branch if-else statements: Print century.

4.3 More if-else

nested if-else	A branch's statements can include any valid statements, including another if-else statement, which are known as nested if-else statements.
Learning tool	4.3.1 Nested if-else
Question set	4.3.2 Nested if-else statements.
Learning tool	4.3.3 Multiple distinct if statements.
Question set	4.3.4 If statements.

Progression	4.3.1 Enter the output for the multiple if-else branches.
Coding challenge	4.3.2 Multiple if statements: Printing car features.

4.4 Equality and relational operators

equality operator	An equality operator checks whether two operands' values are the same (==) or different (!=).
Boolean	A Boolean is a type that has just two values: True or False.
==	A == b means a is equal to b.
!=	A != b means a is not equal to b.
relational operator	A relational operator checks how one operand's value relates to another, like being greater than.
<	A < b means a is less than b.
>	A > b means a is greater than b.
<=	A <= b means a is less than or equal to b.
>=	A >= b means a is greater than or equal to b.
operator chaining	

Table	4.4.1 Equality operators.
Question set	4.4.1 Evaluating expressions that have equality operators.
Question set	4.4.2 Creating expressions with equality operators.
Table	4.4.2 Relational operators.
Question set	4.4.3 Evaluating equations having relational operators.

Question set	4.4.4 Creating expressions with relational operators.
Progression	4.4.1 Enter the output for the branches with relational and equality operators.
Coding challenge	4.4.2 Relational operators.
Question set	4.4.5 Chaining relational operators.
Coding challenge	4.4.3 If-else expression: Operator chaining.
Question set	4.4.6 Comparing various types.
Question set	4.4.7 Comparing various types.
Question set	4.4.8 Watch out for assignment in an if-else expression.

4.5 Boolean operators and expressions

Boolean	A Boolean refers to a value that is either True or False.
Boolean operator	A Boolean operator treats operands as True or False and evaluates to a value of True or False.
Boolean expression	A Boolean expression is an expression using Boolean operators.
Boolean AND	Boolean AND: True when both operands are True.
Boolean OR	Boolean OR: True when at least one operand is True.
Boolean NOT	Boolean NOT (opposite): True when the single operand is False (and False when operand is True).
Figure	4.5.1 Creating a Boolean.
Animation	4.5.1 Boolean operators: and, or, and not.
Table	4.5.1 Boolean operators.

	Table	4.5.2 Boolean operators examples.
	Question set	4.5.2 Evaluating expressions with Boolean operators.
	Question set	4.5.3 Boolean operators: Complete the expressions for the given condition.
	Coding challenge	4.5.1 Boolean operators: Detect specific values.
	Coding challenge	4.5.2 Boolean operators: Combining test conditions.
	Coding challenge	4.5.3 Boolean operators: Branching using Boolean variables.

4.6 Membership and identity operators

in	The in and not in operators, known as membership operators, yield True or False if the left operand matches the value of some element in the right operand, which is always a container.
not in	The in and not in operators, known as membership operators, yield True or False if the left operand matches the value of some element in the right operand, which is always a container.
membership operators	The in and not in operators, known as membership operators, yield True or False if the left operand matches the value of some element in the right operand, which is always a container.
substring	A substring, or matching subset of characters, of a larger string.
identity operator	The programmer can use the identity operator, is, to check whether two operands are bound to a single object.
is	

The programmer can use the identity operator, is, to check whether two operands are bound to a single object.

is not

The inverse identity operator, is not, gives the negated value of 'is'.

- **Animation** — 4.6.1 Membership operators: Checking for a value in a list.
- **Figure** — 4.6.1 Membership operators example: Checking for an item in a list.
- **Figure** — 4.6.2 Checking for substrings.
- **Figure** — 4.6.3 Checking for membership in a dict.
- **Question set** — 4.6.2 Membership operators.
- **Figure** — 4.6.4 Identity operators.
- **Question set** — 4.6.3 Membership and identity operators.
- **Coding challenge** — 4.6.1 Boolean operators: Detect specific values.

4.7 Order of evaluation

precedence rules

The order in which operators are evaluated in an expression is known as precedence rules.

- **Table** — 4.7.1 Precedence rules for arithmetic, logical, and relational operators.
- **Animation** — 4.7.1 Applying the precedence rules to an expression can be thought of as a 'tree'.
- **Question set** — 4.7.2 Order of evaluation.
- **Question set** — 4.7.3 Common errors in expressions.

Question set 4.7.4 Order of evaluation.

4.8 Code blocks and indentation

code block A code block is a series of statements that are grouped together.

Figure 4.8.1 Code blocks are indicated with indentation.

Figure 4.8.2 Some indentations are continuations of the previous line.

Figure 4.8.3 List, dict multi-line constructs.

Question set 4.8.1 Indentation.

Coding challenge 4.8.1 Indentation: Fix the program.

4.9 Conditional expressions

conditional expression

ternary operation

Construct 4.9.1 Conditional expression.

Animation 4.9.1 Conditional expression.

Question set 4.9.2 Conditional expressions.

Coding challenge 4.9.1 Conditional expression: Print negative or non-negative.

Coding challenge 4.9.2 Conditional expression: Conditional assignment.

4.11 LAB: Remove gray from RGB

Lab activity 4.11.1 LAB: Remove gray from RGB

4.12 LAB: Smallest number

Lab activity 4.12.1 LAB: Smallest number

4.13 LAB: Interstate highway numbers

Lab activity 4.13.1 LAB: Interstate highway numbers

4.14 LAB: Seasons

Lab activity 4.14.1 LAB: Seasons

4.15 LAB: Exact change

Lab activity 4.15.1 LAB: Exact change

4.16 LAB: Leap year

Lab activity 4.16.1 LAB: Leap year

4.17 LAB: Warm up: Automobile service cost

Lab activity 4.17.1 LAB: Warm up: Automobile service cost

4.18 LAB*: Program: Automobile service invoice

Lab activity 4.18.1 LAB*: Program: Automobile service invoice

4.19 Programming Assignment 2 - Problem 1

Lab activity 4.19.1 Programming Assignment 2 - Problem 1

4.20 Programming Assignment 2 - Problem 2

Lab activity 4.20.1 Programming Assignment 2 - Problem 2

4.21 Programming Assignment 2 - Problem 3

Lab activity 4.21.1 Programming Assignment 2 - Problem 3

5. Loops

5.1 Loops (general)

loop

A loop is a program construct that repeatedly executes the loop's statements (known as the loop body) while the loop's expression is true; when false, execution proceeds past the loop.

loop body

A loop is a program construct that repeatedly executes the loop's statements (known as the loop body) while

	the loop's expression is true; when false, execution proceeds past the loop.
iteration	Each time through a loop's statements is called an iteration.
Animation	5.1.1 Loop concept: Driving a baby around the block.
Question set	5.1.2 Loop concept.
Animation	5.1.3 A simple loop: Summing the input values.
Animation	5.1.4 Loop example: Computing an average.
Question set	5.1.5 Loop example: Average.
Learning tool	5.1.6 Counting negative values in a list of values.
Question set	5.1.7 Counting negative values.
Learning tool	5.1.8 Find the maximum value in the list of values.
Question set	5.1.9 Determining the max value.

5.2 While loops

while loop	A while loop is a construct that repeatedly executes an indented block of code (known as the loop body) as long as the loop's expression is True.
loop body	A while loop is a construct that repeatedly executes an indented block of code (known as the loop body) as long as the loop's expression is True.
iteration	Each execution of the loop body is called an iteration.
sentinel value	The letter 'q' in this case is a sentinel value, a value that when evaluated by the loop expression causes the loop to terminate.

infinite loop An infinite loop is a loop that will always execute because the loop's expression is always True.

- Construct — 5.2.1 While loop.
- Animation — 5.2.1 While loop.
- Question set — 5.2.2 Basic while loops.
- Figure — 5.2.1 While loop example: Face-printing program that ends when user enters 'q'.
- Question set — 5.2.3 Loop expressions.
- Animation — 5.2.4 While loop step-by-step.
- Question set — 5.2.5 While loop iterations.
- Progression — 5.2.1 Enter the output of the while loop.
- Coding challenge — 5.2.2 Basic while loop with user input.
- Coding challenge — 5.2.3 Basic while loop expression.

5.3 More while examples

multi-line comment A multi-line comment, which is delimited at the beginning and end by triple-quotes.

randint() The randint() function provides a new random number each time the function is called.

- Question set — 5.3.1 Loop example: Greatest common divisor.
- Construct — 5.3.1 Multi-line comments.
- Question set — 5.3.2 Conversation program.
- Question set — 5.3.3 Average example with a sentinel.

Progression — 5.3.1 While loop with sentinel.

Coding challenge — 5.3.2 Bidding example.

Coding challenge — 5.3.3 While loop: Insect growth.

5.4 Counting

loop variable

The programmer can use a variable to count the number of iterations, called a loop variable.

Construct — 5.4.1 Counting while loop form.

Question set — 5.4.1 Savings interest program.

Question set — 5.4.2 Basic counting with while loops.

Figure — 5.4.1 While loop with loop variable that counts down.

Figure — 5.4.2 Loop variable increased by 2 per iteration.

Question set — 5.4.3 Forms of counting.

Learning tool — 5.4.4 Counting in a loop simulator.

Construct — 5.4.2 Operators like += are common in loops.

Question set — 5.4.5 Shorthand operators.

Coding challenge — 5.4.1 While loop: Print 1 to N.

Coding challenge — 5.4.2 Printing output using a counter.

5.5 For loops

for loop

A for loop statement loops over each element in a container one at a time, assigning the next element to a variable that can then be used in the loop body.

reversed()

A for loop may also iterate backwards over a sequence, starting at the last element and ending with the first element, by using the reversed() function to reverse the order of the elements.

Construct	5.5.1
Animation	5.5.1 Iterating over a list using a for loop.
Figure	5.5.1 A for loop assigns a dictionary's keys to the loop variable.
Figure	5.5.2 Using a for loop to access each character of a string.
Question set	5.5.2 Creating for loops.
Figure	5.5.3 For loop example: Calculating shop revenue.
Figure	5.5.4 For loop example: Looping over a sequence in reverse.
Question set	5.5.3 For loops.
Coding challenge	5.5.1 For loop: Printing a list
Coding challenge	5.5.2 For loop: Printing a dictionary

5.6 Counting using the range() function

range()

Range() generates a sequence of numbers, starting at zero and ending before a value given inside the parentheses.

Table	5.6.1 Using the range() function.
Question set	5.6.1 The range() function.
Question set	5.6.2 The range() function.

Progression	5.6.1 Enter the for loop's output.

5.7 While vs. for loops

Animation	5.7.1 While/for loop correspondence.
Question set	5.7.2 While loops and for loops.

5.8 Nested loops

nested loop	A nested loop is a loop that appears as part of the body of another loop.
outer loop	The nested loops are commonly referred to as the outer loop and inner loop.
inner loop	The nested loops are commonly referred to as the outer loop and inner loop.
Figure	5.8.1 Nested loops example: Two-letter domain name printing program.
Question set	5.8.1 Nested loops.
Coding challenge	5.8.1 Nested loops: Print rectangle
Coding challenge	5.8.2 Nested loops: Print seats.

5.9 Developing programs incrementally

incremental programming	Experienced programmers practice incremental programming, starting with a simple version of the program, and then growing the program little-by-little into a complete version.
FIXME comment	A FIXME comment attracts attention to code that needs to be fixed in the future.
Figure	5.9.1 First version echoes input phone number string.
Figure	5.9.2 Second version echoes numbers, and has FIXME comment.

enumerate()	The enumerate() function retrieves both the index and corresponding element value at the same time, providing a cleaner and more readable solution.
Unpacking	Unpacking is a process that performs multiple assignments at once, binding comma-separated names on the left to the elements of a sequence on the right.

Figure	5.12.1 Using range() and len() to iterate over a sequence.
Figure	5.12.2 Using list.index() to find the index of each element.
Figure	5.12.3 The enumerate() function.
Question set	5.12.1 enumerate().

5.14 LAB: Convert to binary

Lab activity	5.14.1 LAB: Convert to binary

5.15 LAB: Count input length without spaces, periods, or commas

Lab activity	5.15.1 LAB: Count input length without spaces, periods, or commas

5.16 LAB: Password modifier

Lab activity	5.16.1 LAB: Password modifier

5.17 LAB: Output range with increment of 10

Lab activity	5.17.1 LAB: Output range with increment of 10

5.18 LAB: Print string in reverse

Lab activity	5.18.1 LAB: Print string in reverse

5.19 LAB: Countdown until matching digits

Lab activity	5.19.1 LAB: Countdown until matching digits

5.20 LAB: Brute force equation solver

Lab activity	5.20.1 LAB: Brute force equation solver

5.21 LAB: Smallest and largest numbers in a list

- **Lab activity** — 5.21.1 LAB: Smallest and largest numbers in a list

5.22 LAB: Output values in a list below a user defined amount

- **Lab activity** — 5.22.1 LAB: Output values in a list below a user defined amount

5.23 LAB: Adjust values in a list by normalizing

- **Lab activity** — 5.23.1 LAB: Adjust values in a list by normalizing

5.24 LAB: Warm up: Drawing a right triangle

- **Lab activity** — 5.24.1 LAB: Warm up: Drawing a right triangle

5.25 LAB*: Program: Drawing a half arrow

- **Lab activity** — 5.25.1 LAB*: Program: Drawing a half arrow

6. Functions

6.1 User-defined function basics

function — A function is a named series of statements.

function definition — A function definition consists of the new function's name and a block of statements.

function call — A function call is an invocation of the function's name, causing the function's statements to execute.

def — The def keyword is used to create new functions.

- **Animation** — 6.1.1 Function example: Printing a pizza area.
- **Question set** — 6.1.2 Function basics.
- **Learning tool** — 6.1.3 Calling a function.
- **Coding challenge** — 6.1.1 Basic function call output.
- **Coding challenge** — 6.1.2 Basic function call.

6.2 Function parameters

parameter	A parameter is a function input specified in a function definition.
argument	An argument is a value provided to a function's parameter during a function call.
Learning tool	6.2.1 Function parameters.
Question set	6.2.2 Parameters.
Figure	6.2.1 Function with multiple parameters.
Question set	6.2.3 Multiple parameters.
Question set	6.2.4 Calls with multiple parameters.
Progression	6.2.1 Function parameters.
Coding challenge	6.2.2 Basic function call.
Coding challenge	6.2.3 Function call with parameters: Converting measurements.

6.3 Returning values from functions

return statement	A function may return one value using a return statement.
None	None is a special keyword that indicates no value.
hierarchical function calls	A function's statements may include function calls, known as hierarchical function calls or nested function calls.
nested function calls	A function's statements may include function calls, known as hierarchical function calls or nested function calls.
Animation	6.3.1 Function example: Returning a value.

Question set	6.3.2 Return basics.
Question set	6.3.3 Function calls in an expression.
Learning tool	6.3.4 Program with a function to convert height in feet/inches to centimeters.
Learning tool	6.3.5 Hierarchical function calls
Question set	6.3.6 Hierarchical function calls.
Progression	6.3.1 Enter the output of the returned value.
Coding challenge	6.3.2 Function call in expression.
Coding challenge	6.3.3 Function definition: Volume of a pyramid.

6.4 Dynamic typing

polymorphism	The function's behavior of being able to add together different types is a concept called polymorphism.
dynamic typing	Python uses dynamic typing to determine the type of objects as a program executes.
static typing	In contrast to dynamic typing, many other languages like C, C++, and Java use static typing, which requires the programmer to define the type of every variable and every function parameter in a program's source code.

Learning tool	6.4.1 Polymorphic functions.
Question set	6.4.2 Dynamic and static typing.

6.5 Reasons for defining functions

Modular development	Modular development is the process of dividing a program into separate modules that can be developed and tested separately and then integrated into a single program.

Figure	6.5.1 With user-defined functions, the main program is easy to understand.
Figure	6.5.2 Without user-defined functions, the main program is harder to read and understand.
Question set	6.5.1 Improved readability.
Animation	6.5.2 Redundant code can be replaced by multiple calls to one function.
Question set	6.5.3 Reasons for defining functions.
Coding challenge	6.5.1 Functions: Factoring out a unit-conversion calculation.

6.6 Function stubs

incremental development	Programs are typically written using incremental development, meaning a small amount of code is written and tested, then a small amount more (an incremental amount) is written and tested, and so on.
function stubs	To assist with the incremental development process, programmers commonly introduce function stubs, which are function definitions whose statements haven't been written yet.
pass	One approach is to use the pass keyword, which performs no operation except to act as a placeholder for a required statement.
NotImplementedError	A NotImplementedError can be generated with the statement `raise NotImplementedError`.
Figure	6.6.1 Using the pass statement in a function stub performs no operation.
Figure	6.6.2 A function stub using a print statement.

Figure — 6.6.3 Stopping the program using NotImplementedError in a function stub.

Question set — 6.6.1 Incremental development and function stubs.

Coding challenge — 6.6.1 Function stubs: Statistics.

6.7 Functions with branches/loops

Figure — 6.7.1 Function example: Determining fees given an item selling price for an auction website.

Question set — 6.7.1 Analyzing the ebay fee function.

Question set — 6.7.2 Analyzing the numbers program.

Coding challenge — 6.7.1 Function with branch: Popcorn.

Coding challenge — 6.7.2 Function with loop: Shampoo.

6.8 Functions are objects

bytecode — A part of the value of a function object is compiled bytecode that represents the statements to be executed by the function.

Figure — 6.8.1 Python bytecode.

Animation — 6.8.1 Functions are objects.

Figure — 6.8.2 Functions can be passed as arguments.

Question set — 6.8.2 Function objects.

6.9 Functions: Common errors

Figure — 6.9.1 Copy-paste common error: Pasted code not properly modified. Find error on the right.

Question set — 6.9.1 Copy-pasted sum-of-squares code.

Learning tool

6.9.2 Missing return common error.

Question set — 6.9.3 Common function errors.

Coding challenge — 6.9.1 Function errors: Copying one function to create another.

6.10 Scope of variables and functions

scope — A variable or function object is only visible to part of a program, known as the object's scope.

local variables — Such variables defined inside a function are called local variables.

global variable — A variable defined outside of a function is called a global variable.

global — A global statement must be used to *change* the value of a global variable inside of a function.

Figure — 6.10.1 Variable scope.

Figure — 6.10.2 The global statement (right) allows modifying a global variable.

Figure — 6.10.3 Function definitions must be evaluated before that function is called.

Question set — 6.10.1 Variable/ function scope.

6.11 Namespaces and scope resolution

namespace — A namespace maps names to objects.

Scope — Scope is the area of code where a name is visible.

scope resolution — The process of searching for a name in the available namespaces is called scope resolution.

Animation — 6.11.1 Namespaces.

Figure	6.11.1 Using the globals() to get namespace names.
Animation	6.11.2 Scope resolution.
Question set	6.11.3 Namespaces and scopes.
Question set	6.11.4 Namespaces.
Learning tool	6.11.5 Function scope.
Question set	6.11.6 Namespace and scope.

6.12 Function arguments

pass-by-assignment	Arguments to functions are passed by object reference, a concept known in Python as pass-by-assignment.
Animation	6.12.1 Assignments to parameters have no effect outside the function.
Learning tool	6.12.2 Modification of a list inside a function.
Learning tool	6.12.3 Modification of a list inside a function.
Question set	6.12.4 Arguments and mutability.
Coding challenge	6.12.1 Change order of elements in function list argument.

6.13 Keyword arguments and default parameter values

keyword arguments	Python provides for keyword arguments that allow arguments to map to parameters by name, instead of implicitly by position in the argument list.
default parameter value	A function can have a default parameter value for one or more parameters, meaning that a function call can optionally omit an argument, and the default parameter value will be substituted for the corresponding omitted argument.

default value	A parameter's default value is the value used in the absence of an argument in the function call.
Figure	6.13.1 A function with many arguments.
Figure	6.13.2 Using keyword arguments.
Figure	6.13.3 All keyword arguments must follow positional arguments.
Question set	6.13.1 Keyword arguments.
Figure	6.13.4 Parameter with a default value.
Figure	6.13.5 Valid function calls with default parameter values.
Figure	6.13.6 Mutable default objects remain changed over multiple function calls.
Question set	6.13.2 Default parameter values.
Figure	6.13.7 Mixing keyword arguments and default parameter values allows omitting arbitrary arguments.
Question set	6.13.3 Mixing keyword and default arguments.
Coding challenge	6.13.1 Return number of pennies in total.
Coding challenge	6.13.2 Default parameters: Calculate splitting a check between diners.

6.14 Arbitrary argument lists

***args**	A function definition can include a *args parameter that collects optional positional parameters into an arbitrary argument list tuple.
arbitrary argument list	A function definition can include a *args parameter that collects optional positional parameters into an arbitrary argument list tuple.

****kwargs**	Adding a final function parameter of **kwargs creates a dictionary containing "extra" arguments not defined in the function definition; kwargs is short for keyword arguments.
keyword arguments	Adding a final function parameter of **kwargs creates a dictionary containing "extra" arguments not defined in the function definition; kwargs is short for keyword arguments.
Figure	6.14.1 Arbitrary numbers of position arguments using *args.
Figure	6.14.2 Arbitrary numbers of keyword arguments using **kwargs.
Learning tool	6.14.1 Arbitrary numbers of arguments using *args and **kwargs.
Question set	6.14.2 Arbitrary arguments.

6.15 Multiple function outputs

unpacking	The statement `average, standard_deviation = get_grade_stats(s` utilizes unpacking to perform multiple assignments at once and `standard_deviation` are assigned the first and secor the returned tuple.
Figure	6.15.1 Multiple outputs can be returned in a container.
Question set	6.15.1 Multiple function outputs.

6.16 Help! Using docstrings to document functions

docstring	A docstring is a string literal placed in the first line of a function body.
help()	The help() function can aid a programmer by providing them with all the documentation associated with an object.

6.27 LAB: Even/odd values in a list

Lab activity	6.27.1 LAB: Even/odd values in a list

6.28 LAB: Output values in a list below a user defined amount - functions

Lab activity	6.28.1 LAB: Output values in a list below a user defined amount - functions

6.29 LAB: Warm up: Text analyzer & modifier

Lab activity	6.29.1 LAB: Warm up: Text analyzer & modifier

6.30 LAB*: Program: Authoring assistant

Lab activity	6.30.1 LAB*: Program: Authoring assistant

7. Strings

7.1 String slicing

index	An index is an integer matching to a specific position in a string's sequence of characters.
Slice notation	Slice notation has the form `my_str[start:end]`, which creates a new string whose value mirrors the characters of `my_str` from indices start to end - 1.
stride	The stride determines how much to increment the index after reading each element.
Figure	7.1.1 String slicing.
Animation	7.1.1 Slicing.
Question set	7.1.2 Slicing basics.
Figure	7.1.2 A slice creates a new object.
Learning tool	7.1.3 String slicing tool.
Table	7.1.1 Common slicing operations.

Question set	7.1.4 Slicing.
Figure	7.1.3 Slice stride.
Question set	7.1.5 Slice stride.
Coding challenge	7.1.1 Slice a rhyme.

7.2 Advanced string formatting

field width	A format specification may include a field width that defines the minimum number of characters that must be inserted into the string.
alignment character	A format specification can include an alignment character that determines how a value should be aligned within the width of the field.
fill character	The fill character is used to pad a replacement field when the string being inserted is smaller than the field width.
precision	The optional precision component of a format specification indicates how many digits to the right of the decimal should be included in the output of floating types.

Figure	7.2.1 A formatted table of soccer statistics.
Animation	7.2.1 Field width.
Question set	7.2.2 Format specification field widths.
Progression	7.2.1 Field widths.
Figure	7.2.2 Aligning strings within a field.
Question set	7.2.3 Aligning text in fields.
Table	7.2.1 Using fill characters to pad tables.

Learning tool	7.2.4 Fill characters in strings.
Question set	7.2.5 Fill characters.
Figure	7.2.3 String formatting example: Setting precision of floating-point values.
Question set	7.2.6 Floating-point precision in formatted strings.
Coding challenge	7.2.2 Format temperature output.

7.3 String methods

replace(old, new)	Replace(old, new) -- Returns a copy of the string with all occurrences of the substring old replaced by the string new.
replace(old, new, count)	Replace(old, new, count) -- Same as above, except only replaces the first count occurrences of old.
find(x)	Find(x) -- Returns the position of the first occurrence of item x in the string, else returns -1.
find(x, start)	Find(x, start) -- Same as find(x), but begins the search at position start.
find(x, start, end)	Find(x, start, end) -- Same as find(x, start), but stops the search at position end.
rfind(x)	Rfind(x) -- Same as find(x) but searches the string in reverse, returning the last occurrence in the string.
count(x)	Count(x) -- Returns the number of times x occurs in the string.
ASCII table	An ASCII table provides a quick lookup of ASCII values.
isalnum()	Isalnum() -- Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.

isdigit()	Isdigit() -- Returns True if all characters are the numbers 0-9.
islower()	Islower() -- Returns True if all cased characters are lowercase letters.
isupper()	Isupper() -- Return True if all cased characters are uppercase letters.
isspace()	Isspace() -- Return True if all characters are whitespace.
startswith(x)	Startswith(x) -- Return True if the string starts with x.
endswith(x)	Endswith(x) -- Return True if the string ends with x.
capitalize()	Capitalize() -- Returns a copy of the string with the first character capitalized and the rest lowercased.
lower()	Lower() -- Returns a copy of the string with all characters lowercased.
upper()	Upper() -- Returns a copy of the string with all characters uppercased.
strip()	Strip() -- Returns a copy of the string with leading and trailing whitespace removed.
title()	Title() -- Returns a copy of the string as a title, with first letters of words capitalized.
Learning tool	7.3.1 replace() string method.
Figure	7.3.1 Use 'in' to check if a character or substring is contained by another string.
Table	7.3.1 String comparisons.
Animation	7.3.2 String comparison.

Figure	7.3.2 Identity vs. equality operators.
Question set	7.3.3 String methods: Boolean string comparisons.
Coding challenge	7.3.1 Find abbreviation.
Coding challenge	7.3.2 Replace abbreviation.

7.4 Splitting and joining strings

split()	The string method split() splits a string into a list of tokens.
token	Each token is a substring that forms a part of a larger string.
separator	A separator is a character or sequence of characters that indicates where to split the string into tokens.
join()	The join() string method performs the inverse operation of split() by joining a list of strings together to create a single string.

Animation	7.4.1 Splitting a string into tokens.
Figure	7.4.1 String split example.
Question set	7.4.2 String split() method.
Animation	7.4.3 String join() method.
Figure	7.4.2 String join() example: Comparing join vs. loops.
Question set	7.4.4 String join() method.
Figure	7.4.3 Splitting and joining: Replacing separators.
Figure	7.4.4 Splitting and joining: Editing tokens.
Question set	7.4.5 Splitting and joining strings.

- **Coding challenge** 7.4.1 Extract area code.

7.5 LAB: Checker for integer string

- **Lab activity** 7.5.1 LAB: Checker for integer string

7.6 LAB: Name format

- **Lab activity** 7.6.1 LAB: Name format

7.7 LAB: Count characters

- **Lab activity** 7.7.1 LAB: Count characters

7.8 LAB: Mad Lib - loops

- **Lab activity** 7.8.1 LAB: Mad Lib - loops

7.9 LAB: Palindrome

- **Lab activity** 7.9.1 LAB: Palindrome

7.10 LAB: Acronyms

- **Lab activity** 7.10.1 LAB: Acronyms

7.11 LAB: Contains the character

- **Lab activity** 7.11.1 LAB: Contains the character

7.12 LAB: Warm up: Parsing strings

- **Lab activity** 7.12.1 LAB: Warm up: Parsing strings

7.13 LAB*: Program: Data visualization

- **Lab activity** 7.13.1 LAB*: Program: Data visualization

8. Lists and Dictionaries

8.1 Lists

list	The list object type is one of the most important and often used types in a Python program.
container	A list is a container, which is an object that groups related objects together.

list()	The list() function accepts a single iterable object argument, such as a string, list, or tuple, and returns a new list object.
index	An index is a zero-based integer matching to a specific position in the list's sequence of elements.
mutable	Unlike the string sequence type, a list is mutable and is thus able to grow and shrink without the program having to replace the entire list with an updated copy.
in-place modification	Such growing and shrinking capability is called in-place modification.

Animation	8.1.1 Lists contain references to other objects.
Question set	8.1.2 List indices.
Table	8.1.1 Some common list operations.
Animation	8.1.3 In-place modification of a list.
Learning tool	8.1.4 In-place modification of a list.
Learning tool	8.1.5 In-place modification of a copy of a list.
Animation	8.1.6 List indexing.
Question set	8.1.7 List basics.
Coding challenge	8.1.1 Modify a list.

8.2 List methods

list method	A list method can perform a useful operation on a list such as adding or removing elements, sorting, reversing, etc.

Table	8.2.1 Available list methods.

Animation	8.2.1 In-place modification using list methods.
Question set	8.2.2 List methods.
Coding challenge	8.2.1 Reverse sort of list.

8.3 Iterating over a list

for loop	Looping through a sequence such as a list is so common that Python supports a construct called a for loop, specifically for iteration purposes.
IndexError	Accessing an index that is out of range causes the program to automatically abort execution and generate an IndexError.
enumerate()	The built-in enumerate() function iterates over a list and provides an iteration counter.
Figure	8.3.1 Iterating through a list.
Figure	8.3.2 Iterating through a list example: Finding the maximum even number.
Animation	8.3.1 Using a variable to keep track of a value while iterating over a list.
Question set	8.3.2 List iteration.
Table	8.3.1 Built-in functions supporting list objects.
Question set	8.3.3 Lists and built-in functions.
Coding challenge	8.3.1 Get user guesses.
Coding challenge	8.3.2 Sum extra credit.
Coding challenge	8.3.3 Hourly temperature reporting.

8.4 List games

Learning tool	8.4.1 Find the maximum value in the list.
Learning tool	8.4.2 Negative value counting in list.
Learning tool	8.4.3 Manually sorting largest value.

8.5 List nesting

list nesting	Such embedding of a list inside another list is known as list nesting.
multi-dimensional data structure	List nesting allows for a programmer to also create a multi-dimensional data structure, the simplest being a two-dimensional table, like a spreadsheet or tic-tac-toe board.
nested for loops	A programmer can access all of the elements of nested lists by using nested for loops.

Figure	8.5.1 Multi-dimensional lists.
Animation	8.5.1 List nesting.
Question set	8.5.2 List nesting.
Figure	8.5.2 Representing a tic-tac-toe board using nested lists.
Animation	8.5.3 Two-dimensional list.
Figure	8.5.3 The level of nested lists is arbitrary.
Question set	8.5.4 Multi-dimensional lists.
Animation	8.5.5 Iterating over multi-dimensional lists.
Figure	8.5.4 Iterating through multi-dimensional lists using enumerate().
Question set	8.5.6 Find the error.

Coding challenge 8.5.1 Print multiplication table.

8.6 List slicing

slice notation A programmer can use slice notation to read multiple elements from a list, creating a new list that contains only the desired elements.

stride An optional component of slice notation is the stride, which indicates how many elements are skipped between extracted items in the source list.

Figure 8.6.1 List slice notation.

Animation 8.6.1 List slicing.

Figure 8.6.2 List slicing: Using negative indices.

Question set 8.6.2 List slicing.

Question set 8.6.3 List slicing.

Table 8.6.1 Some common list slicing operations.

Question set 8.6.4 Match the expressions to the list.

8.7 Loops modifying lists

Figure 8.7.1 Modifying a list during iteration example.

Figure 8.7.2 Modifying a list during iteration example: Converting negative values to 0.

Learning tool 8.7.1 Incorrect list modification example.

Learning tool 8.7.2 Corrected list modification example.

Question set 8.7.3 List modification.

Figure 8.7.3 Modifying lists while iterating: Incorrect program.

8.8 List comprehensions

list comprehension

The Python language provides a convenient construct, known as list comprehension, that iterates over a list, modifies each element, and returns a new list consisting of the modified elements.

8.9 Sorting lists

sort()

One of the most useful list methods is sort(), which performs an in-place rearranging of the list elements, sorting the elements from lowest to highest.

sorted()

The sorted() built-in function provides the same sorting functionality as the list.sort() method, however, sorted()

	creates and returns a new list instead of modifying an existing list.
key	Both the list.sort() method and the built-in sorted() function have an optional key argument. The key specifies a function to be applied to each element prior to being compared.
reverse	Sorting also supports the reverse argument. The reverse argument can be set to a Boolean value, either True or False.
Animation	8.9.1 Sorting a list using list.sort().
Figure	8.9.1 list.sort() method example: Alphabetically sorting book titles.
Figure	8.9.2 Using sorted() to create a new sorted list from an existing list without modifying the existing list.
Question set	8.9.2 list.sort() and sorted().
Figure	8.9.3 Using the key argument.
Figure	8.9.4 The key argument to list.sort() or sorted() can be assigned any function.
Question set	8.9.3 Sorting.

8.10 Command-line arguments

Command-line arguments	Command-line arguments are values entered by a user when running a program from a command line. A command line exists in some program execution environments, wherein a user can run a program by typing at a command prompt.
sys.argv	The contents of this command line are automatically stored in the list sys.argv, which is stored in the standard

	library sys module. sys.argv consists of one string element for each argument typed on the command line.
usage message	If the number of arguments is incorrect, the program prints an error message, referred to as a usage message, that provides the user with an example of the correct command-line argument format.
Animation	8.10.1 Command-line arguments.
Figure	8.10.1 Simple use of command line arguments.
Figure	8.10.2 Checking for proper number of command-line arguments.
Question set	8.10.2 Command-line arguments.

8.12 Dictionaries

dict	The dict type implements a dictionary in Python.
dictionary comprehension	The second approach uses dictionary comprehension, which evaluates a loop to create a new dictionary, similar to how list comprehension creates a new list.
dict()	Other approaches use the dict() built-in function, using either keyword arguments to specify the key-value pairs or by specifying a list of tuple-pairs.
Animation	8.12.1 Dictionaries.
Table	8.12.1 Common dict operations.
Question set	8.12.2 Dictionaries.
Coding challenge	8.12.1 Delete from dictionary.

8.13 Dictionary methods

dictionary method	

	A dictionary method is a function provided by the dictionary type (dict) that operates on a specific dictionary object.
Table	8.13.1 Dictionary methods.
Question set	8.13.1 Dictionary methods.

8.14 Iterating over a dictionary

hash	A hash is a transformation of the key into a unique value that allows the interpreter to perform very fast lookup.
view object	A view object provides read-only access to dictionary keys and values.
Construct	8.14.1 A for loop over a dictionary retrieves each key in the dictionary.
Figure	8.14.1 Iterating over a dictionary.
Figure	8.14.2 Use list() to convert view objects into lists.
Question set	8.14.1 Iterating over dictionaries.
Coding challenge	8.14.1 Report country population.

8.15 Dictionary nesting

nested dictionaries	A dictionary may contain one or more nested dictionaries, in which the dictionary contains another dictionary as a value.
data structure	A data structure is a method of organizing data in a logical and coherent fashion.
Figure	8.15.1 Nested dictionaries.
Figure	8.15.2 Nested dictionaries example: Storing grades.
Question set	8.15.1 Nested dictionaries.

8.16 LAB: Varied amount of input data

- **Lab activity** 8.16.1 LAB: Varied amount of input data

8.17 LAB: Filter and sort a list

- **Lab activity** 8.17.1 LAB: Filter and sort a list

8.18 LAB: Middle item

- **Lab activity** 8.18.1 LAB: Middle item

8.19 LAB: Elements in a range

- **Lab activity** 8.19.1 LAB: Elements in a range

8.20 LAB: Word frequencies

- **Lab activity** 8.20.1 LAB: Word frequencies

8.21 LAB: Contact list

- **Lab activity** 8.21.1 LAB: Contact list

8.22 LAB: Replacement words

- **Lab activity** 8.22.1 LAB: Replacement words

8.23 LAB: Warm up: People's weights (Lists)

- **Lab activity** 8.23.1 LAB: Warm up: People's weights (Lists)

8.24 LAB*: Program: Soccer team roster (Dictionaries)

- **Lab activity** 8.24.1 LAB*: Program: Soccer team roster (Dictionaries)

8.25 Programming Assignment 3 Question 1

- **Lab activity** 8.25.1 Programming Assignment 3 Question 1

8.26 Programming Assignment 3 Question 2

- **Lab activity** 8.26.1 Programming Assignment 3 Question 2

8.27 Programming Assignment 3 Question 3

- **Lab activity** 8.27.1 Programming Assignment 3 Question 3

8.28 Programming Assignment 3- Question 4

Lab activity	8.28.1 Programming Assignment 3- Question 4

9. Classes

9.1 Classes: Introduction

object	In a program, an object consists of some internal data items plus operations that can be performed on that data.
Animation	9.1.1 Grouping variables and functions into objects keeps programs understandable.
Question set	9.1.2 Objects.

9.2 Classes: Grouping data

class	The class keyword can be used to create a user-defined type of object containing groups of related variables and functions.
attributes	The object maintains a set of attributes that determines the data and behavior of the class.
instantiation	An instantiation operation is performed by "calling" the class, using parentheses like a function call as in `my_time = Time()`.
instance	An instantiation operation creates an instance, which is an individual object of the given class.
method	A method is a function defined within a class.
__init__	The __init__ method, commonly known as a constructor, is responsible for setting up the initial state of the new instance.
constructor	The __init__ method, commonly known as a constructor, is responsible for setting up the initial state of the new instance.

attribute reference operator	Attributes can be accessed using the attribute reference operator "." (sometimes called the member operator or dot notation).
member operator	Attributes can be accessed using the attribute reference operator "." (sometimes called the member operator or dot notation).
dot notation	Attributes can be accessed using the attribute reference operator "." (sometimes called the member operator or dot notation).
Construct	9.2.1 The class keyword.
Figure	9.2.1 Defining a new class object with two data attributes.
Figure	9.2.2 Using instantiation to create a variable using the Time class.
Animation	9.2.1 Using classes and attribute reference.
Figure	9.2.3 Multiple instances of a class.
Question set	9.2.2 Class terms.
Question set	9.2.3 Classes.
Question set	9.2.4 Classes.
Coding challenge	9.2.1 Declaring a class.
Coding challenge	9.2.2 Access a class' attributes.

9.3 Instance methods

instance method	A function defined within a class is known as an instance method.
method object	

	The Python object that encapsulates the method's code is thus called a method object (as opposed to a pure function object).
special method name	A special method name, indicating that the method implements some special behavior of the class.
Figure	9.3.1 A class definition may include user-defined functions.
Question set	9.3.1 Methods.
Figure	9.3.2 Accidentally forgetting the self parameter of a method generates an error when calling the method.
Question set	9.3.2 Method definitions.
Coding challenge	9.3.1 Creating a method object.

9.4 Class and instance object types

class object	A class object acts as a *factory* that creates instance objects.
class attribute	A class attribute is shared amongst all of the instances of that class.
instance attribute	An instance attribute can be unique to each instance.
Learning tool	9.4.1 Class Time's init method initializes two new Time instance objects.
Figure	9.4.1 A class attribute is shared between all instances of that class.
Figure	9.4.2 An instance attribute can be different between instances of a class.
Animation	9.4.2 Class and instance namespaces.

Figure — 9.4.3 Changing the gmt_offset class attribute affects behavior of all instances.

Question set — 9.4.3 Class and instance objects.

Question set — 9.4.4 Identifying class and instance attributes.

9.6 Class constructors

Figure — 9.6.1 Adding parameters to a constructor.

Figure — 9.6.2 Additional parameters can be added to a class constructor.

Question set — 9.6.1 Method parameters.

Figure — 9.6.3 Constructor default parameters.

Question set — 9.6.2 Default constructor parameters.

Coding challenge — 9.6.1 Defining a class constructor.

9.7 Class interfaces

class interface — A class interface consists of the methods that a programmer calls to create, modify, or access a class instance.

abstract data type (ADT) — A class can be used to implement the computing concept known as an abstract data type (ADT), which is a data type whose creation and update are constrained to specific, well-defined operations (the class interface).

Figure — 9.7.1 A class interface consists of methods to interact with an instance.

Figure — 9.7.2 Internal instance methods.

Question set — 9.7.1 Class interfaces.

9.8 Class customization

Class customization	Class customization is the process of defining how a class should behave for some common operations. Such operations might include printing, accessing attributes, or how instances of that class are compared to each other.
special method names	To customize a class, a programmer implements instance methods with special method names that the Python interpreter recognizes.
operator overloading	Class customization can redefine the functionality of built-in operators like <, >=, +, -, and * when used with class instances, a technique known as operator overloading.
_lt_	Methods like _lt_ above are known as rich comparison methods.
rich comparison methods	Methods like _lt_ above are known as rich comparison methods.
Figure	9.8.1 Implementing __str__() alters how the class is printed.
Learning tool	9.8.1 Implementing __str__() alters how the class is printed.
Figure	9.8.2 Overloading the less-than operator of the Time class allows for comparison of instances.
Table	9.8.1 Rich comparison methods.
Question set	9.8.2 Rich comparison methods.
Coding challenge	9.8.1 Defining __str__.

9.9 More operator overloading: Classes as numeric types

isinstance()	To handle subtraction of arbitrary object types, the built-in isinstance() function can be used.

Figure	9.9.1 Extending the time class with overloaded subtraction operator.
Figure	9.9.2 The isinstance() built-in function.
Table	9.9.1 Methods for emulating numeric types.
Question set	9.9.1 Emulating numeric types with operating overloading.

9.10 Memory allocation and garbage collection

memory allocation	The process of an application requesting and being granted memory is known as memory allocation.
reference count	A reference count is an integer counter that represents how many variables reference an object.
Animation	9.10.1 Memory allocation in Python.
Question set	9.10.2 Memory allocation in Python.
Animation	9.10.3 Python's garbage collection.
Question set	9.10.4 Reference counts and garbage collection.

9.11 LAB: Car value (classes)

Lab activity	9.11.1 LAB: Car value (classes)

9.12 LAB: Nutritional information (classes/constructors)

Lab activity	9.12.1 LAB: Nutritional information (classes/constructors)

9.13 LAB: Artwork label (classes/constructors)

Lab activity	9.13.1 LAB: Artwork label (classes/constructors)

9.14 LAB: Triangle area comparison (classes)

Lab activity	9.14.1 LAB: Triangle area comparison (classes)

9.15 LAB: Winning team (classes)

Lab activity	9.15.1 LAB: Winning team (classes)

9.16 LAB*: Warm up: Online shopping cart (Part 1)

Lab activity	9.16.1 LAB*: Warm up: Online shopping cart (Part 1)

9.17 LAB*: Program: Online shopping cart (Part 2)

Lab activity	9.17.1 LAB*: Program: Online shopping cart (Part 2)

10. Exceptions

10.1 Handling exceptions using try and except

Error-checking code	Error-checking code is code that a programmer introduces to detect and handle errors that may occur while the program executes.
exception-handling	Python has special constructs known as exception-handling constructs because they handle *exceptional circumstances*, another word for errors during execution.
try	Code that potentially may produce an exception is placed in a try block.
except	If the code in the try block causes an exception, then the code placed in a following except block is executed.
exception handling	The try and except constructs are used together to implement exception handling, meaning handling exceptional conditions (errors during execution).
Figure	10.1.1 BMI example without exception handling.
Figure	10.1.2 BMI example with exception handling using try/except.
Construct	10.1.1 Basic exception handling constructs.
Animation	10.1.1 How try and except blocks handle exceptions.

Question set	10.1.2 Exception basics.

10.2 Multiple exception handlers

exception handlers	Multiple exception handlers can be added to a try block by adding additional except blocks and specifying the specific type of exception that each except block handles.
unhandled exception	If no exception handler exists for an error type, then an unhandled exception may occur.
Construct	10.2.1 Multiple except blocks.
Animation	10.2.1 Multiple exception handlers.
Figure	10.2.1 BMI example with multiple exception types.
Figure	10.2.2 Multiple exception types in a single exception handler.
Question set	10.2.2 Multiple exceptions.

10.3 Raising exceptions

raise	Code that detects an error can execute a raise statement, which causes immediate exit from the try block and the execution of an exception handler.
as	The as keyword binds a name to the exception being handled.
Figure	10.3.1 BMI example with error-checking code but without using exception-handling constructs.
Figure	10.3.2 BMI example with error-checking code that raises exceptions.
Question set	10.3.1 Exceptions.

10.4 Exceptions with functions

module	A solution is to use a module, which is a file containing Python code that can be imported and used by scripts, other modules, or the interactive interpreter.
import	To import a module means to execute the code contained by the module, and make the definitions within that module available for use by the importing program.
dependency	A module being required by another program is often called a dependency.
sys.modules	A dictionary of the loaded modules is stored in sys.modules (available from the sys standard library module).
module object	A module object is simply a namespace that contains definitions from the module.
Animation	11.1.1 A module is a file containing Python statements and definitions that can be used by other Python sources.
Question set	11.1.2 Basic importing of modules.
Animation	11.1.3 Importing a module.
Question set	11.1.4 The importing process.
Figure	11.1.1 Contents of my_funcs.py.
Figure	11.1.2 Using factorial from my_funcs.py.
Question set	11.1.5 Basic usage of imported modules.

11.2 Finding modules

built-in module	A built-in module is a module that comes pre-installed with Python; examples of built-in modules include sys, time, and math.

sys.path	If no matching built-in module is found, then the interpreter searches the list of directories contained by sys.path, located in the sys module.
PYTHONPATH	A programmer might set the environmental variable PYTHONPATH in the operating system.
environmental variable	An operating system environmental variable is much like a variable in a Python script, except that an environmental variable is stored by the computer's operating system and can be accessed by every program running on the computer.
Question set	11.2.1 Finding modules.

11.3 Importing specific names from a module

from	A programmer can specify names to import from a module by using the from keyword in an import statement.
hashlib	The program below imports names from the hashlib module, a Python standard library module that contains a number of algorithms for creating a secure hash of a text message.
hash	The program below imports names from the hashlib module, a Python standard library module that contains a number of algorithms for creating a secure hash of a text message.
Construct	11.3.1 Importing specific names from a module.
Animation	11.3.1 'import x' vs 'from x import y'.
Table	11.3.1 'import module' vs. 'from module import names'.
Figure	11.3.1 Using the from keyword to import specific names.
Question set	11.3.2 Importing specific names.

11.4 Executing modules as scripts

__name__	__name__ is a global string variable automatically added to every module that contains the name of the module.
Figure	11.4.1 web_search.py: Get the 1st page of results for a web search.
Figure	11.4.2 domain_freq.py: Importing web_search causes unintended search to occur.
Figure	11.4.3 Checking if a file is the executing script or an imported module.
Figure	11.4.4 web_search.py modified to run as either script or module.
Question set	11.4.1 Executing modules as scripts.

11.5 Reloading modules

reload()	Instead of restarting the entire Python program, the reload() function can be used to reload and re-execute the changed module.
__file__	The __file__ special name contains the path to a module in the computer file system.
Figure	11.5.1 send_gmail.py: Sends a single email through gmail.
Figure	11.5.2 send_coupons.py: Automates emails to loyal customers.
Figure	11.5.3 Modifying send_gmail.py while the program is running updates the email contents.
Figure	11.5.4 Reloading modules doesn't affect attributes imported using 'from'.
Question set	11.5.1 Reloading modules.

11.6 Packages

package	A package is a directory that, when imported, gives access to all of the modules stored in the directory.
Animation	11.6.1 Packages group related modules together.
Figure	11.6.1 Directory structure.
Figure	11.6.2 Importing the ASCIIArt package.
Figure	11.6.3 Importing the canvas module.
Figure	11.6.4 Import cow module from figures subpackage.
Figure	11.6.5 Import the draw function from the cow module.
Question set	11.6.2 Importing packages.

11.7 Standard library

Python standard library	The Python standard library includes various utilities and tools for performing common program behaviors.
Table	11.7.1 Some commonly used Python standard library modules.
Figure	11.7.1 Using the datetime module.
Figure	11.7.2 Using the random module.
Question set	11.7.1 A few standard library modules.
Aside	Review all of the standard library

11.8 LAB: Artwork label (modules)

Lab activity	11.8.1 LAB: Artwork label (modules)

11.9 LAB: Guess the random number

Lab activity	11.9.1 LAB: Guess the random number

11.10 LAB: Quadratic formula

Lab activity	11.10.1 LAB: Quadratic formula

12. Files

12.1 Reading files

open()	A common programming task is to get input from a file using the built-in open() function rather than from a user typing on a keyboard.
file.close()	The file.close() method closes the file, after which no more reads or writes to the file are allowed.
file.read()	The file.read() method returns the file contents as a string.
file.readlines()	The file.readlines() method returns a list of strings, where the first element is the contents of the first line, the second element is the contents of the second line, and so on.
EOF	Each method stops reading when the end-of-file (EOF) is detected, which indicates no more data is available.
Animation	12.1.1 Reading text from a file.
Figure	12.1.1 Creating a file object and reading text.
Question set	12.1.2 Opening files and reading text.
Figure	12.1.2 Calculating the average of data values stored in a file.
Figure	12.1.3 Iterating over the lines of a file.

12.2 Writing files

file.write()	The file.write() method writes a string argument to a file.
flush()	

The flush() file method can be called to force the interpreter to flush the output buffer to disk.

Figure	12.2.1 Writing to a file.
Figure	12.2.2 Numeric values must be converted to strings.
Table	12.2.1 Modes for opening files.
Question set	12.2.1 File modes.
Animation	12.2.2 Output is buffered.
Figure	12.2.3 Using flush() to force an output buffer to write to disk.
Question set	12.2.3 Writing output.

12.3 Interacting with file systems

os module	The Python standard library's os module provides an interface to operating system function calls and is thus a critical piece of a Python programmer's toolbox.
portability	A programmer should consider the portability of a program across different operating systems to avoid scenarios where the program behaves correctly on the programmer's computer but crashes on another.
path separator	The character between directories, e.g., "\\"or "/", is called the path separator, and using the incorrect path separator may result in that file not being found.
os.path.sep	Os.path.sep stores the path separator for the current operating system.
os.walk()	The os.walk() function 'walks' a directory tree like the one above, visiting each subdirectory in the specified path.

Animation	12.3.1 Using the os module to interact with the file system.
Figure	12.3.1 Using os.path.join() to create a portable file path string.
Question set	12.3.2 Portable file paths.
Question set	12.3.3 Path name manipulation functions from os.path.
Figure	12.3.2 Directory structure organized by date.
Figure	12.3.3 Walking a directory tree.

12.4 Binary data

binary data

Some files consist of data stored as a sequence of bytes, known as binary data, that is not encoded into human-readable text using an encoding like ASCII or UTF-8.

bytes object

A bytes object is used to represent a sequence of single byte values, such as binary data read from a file.

bytes()

A byte object can be created using the bytes() built-in function.

binary file mode

Programs can also access files using a binary file mode by adding a "b" character to the end of the mode string in a call to open(), as in `open('myfile.txt', 'rb')`.

header

This sequence constitutes the header of the binary file, which describes the bitmap's contents.

struct

The struct module is a commonly used Python standard library module for *packing* values into sequences of bytes and *unpacking* sequences of bytes into values (like integers and strings).

struct.pack()

	The struct.pack() function packs values such as strings and integers into sequences of bytes.
byte-order	The "<" character indicates the byte-order, or endianness, of the conversion, which determines whether the most-significant or least-significant byte is placed first in the byte sequence.
struct.unpack()	The struct.unpack() module performs the reverse operation of struct.pack(), unpacking a sequence of bytes into a new object.
Figure	12.4.1 Creating a bytes object using a bytes literal.
Figure	12.4.2 Byte string literals.
Question set	12.4.1 Binary Data.
Figure	12.4.3 Inspecting the binary contents of an image file.
Example	12.4.1 Altering a BMP image file.
Figure	12.4.4 Packing values into byte sequences.
Figure	12.4.5 Unpacking values from byte sequences.
Question set	12.4.2 The struct module.

12.5 Command-line arguments and files

Figure	12.5.1 Using command-line arguments to specify the name of an input file.
Question set	12.5.1 Filename command line arguments.

12.6 The 'with' statement

with statement	A with statement can be used to open a file, execute a block of statements, and automatically close the file when complete.

context manager	The with statement creates a context manager, which manages the usage of a resource, like a file, by performing setup and teardown operations.
Construct	12.6.1 The with statement.
Figure	12.6.1 Using the with statement to open a file.
Question set	12.6.1 The with statement.

12.7 Comma separated values files

comma separated values	A comma separated values (csv) file is a simple text-based file format that uses commas to separate data items, called fields.
fields	A comma separated values (csv) file is a simple text-based file format that uses commas to separate data items, called fields.
csv module	The Python standard library csv module can be used to help read and write files in the csv format.
Figure	12.7.1 Contents of a csv file.
Figure	12.7.2 Reading each row of a csv file.
Figure	12.7.3 Using csv file contents to perform calculations.
Figure	12.7.4 Writing rows to a csv module.
Question set	12.7.1 Comma separated values files.

12.8 LAB: Words in a range (lists)

Lab activity	12.8.1 LAB: Words in a range (lists)

12.9 LAB: Word frequencies (lists)

Lab activity	12.9.1 LAB: Word frequencies (lists)

12.10 LAB: Sorting TV Shows (dictionaries and lists)

Lab activity	12.10.1 LAB: Sorting TV Shows (dictionaries and lists)

13. Inheritance

13.1 Derived classes

derived class	The term derived class refers to a class that inherits the class attributes of another class, known as a base class.
base class	The term derived class refers to a class that inherits the class attributes of another class, known as a base class.
inheritance	The derived class is said to *inherit* the attributes of its base class, a concept commonly called inheritance.
Figure	13.1.1 A derived class example: Class Produce is derived from class Items.
Learning tool	13.1.1 Derived class explicitly calls base class' constructor.
Animation	13.1.2 Derived class example: Produce derived from Item.
Animation	13.1.3 UML derived class example: Produce derived from Item.
Learning tool	13.1.4 Interactive inheritance tree.
Question set	13.1.5 Derived classes basics.
Coding challenge	13.1.1 Basic inheritance.

13.2 Accessing base class attributes

inheritance tree	The search for an attribute continues all the way up the inheritance tree, which is the hierarchy of classes from a derived class to the final base class.
Figure	13.2.1 Searching the inheritance tree for an attribute.

Question set 13.2.1 Searching for attributes in the inheritance tree.

13.3 Overriding class methods

overrides Such a member function overrides the method of the base class.

Figure 13.3.1 Produce's display() function overrides Item's display() function.

Figure 13.3.2 Method calling overridden method of base class.

Question set 13.3.1 Overriding base class methods.

Coding challenge 13.3.1 Basic derived class member override.

13.4 Is-a versus has-a relationships

Figure 13.4.1 Composition.

Figure 13.4.2 Inheritance.

Question set 13.4.1 Is-a vs. has-a relationships.

13.5 Mixin classes and multiple inheritance

multiple inheritance A class can inherit from more than one base class, a concept known as multiple inheritance.

Mixins Mixins are classes that provide some additional behavior, by "mixin in" new methods, but are not themselves meant to be instantiated.

Animation 13.5.1 Multiple inheritance.

Figure 13.5.1 Inheriting from multiple base classes.

Figure 13.5.2 Using mixins to extend a class' functionality with new methods.

Question set 13.5.2 Mixin classes and multiple inheritance.

13.6 Testing your code: The unittest module

test suite	Maintaining a test suite, or a set of repeatable tests, that can be run after changing the source code of a program is critical.
unit testing	A programmer commonly performs unit testing, testing the individual components of a program, such as specific methods, class interfaces, data structures, and so on.
unittest	The Python standard library unittest module implements unit testing functionality.
assertions	A unit test performs assertions to check if a computed value meets certain requirements.

Figure	13.6.1 Unit testing with the unittest module.
Table	13.6.1 Assertion methods.
Question set	13.6.1 Unit testing.

13.7 LAB: Pet information (derived classes)

Lab activity	13.7.1 LAB: Pet information (derived classes)

13.8 LAB: Instrument information (derived classes)

Lab activity	13.8.1 LAB: Instrument information (derived classes)

13.9 LAB: Course information (derived classes)

Lab activity	13.9.1 LAB: Course information (derived classes)

13.10 LAB: Book information (overriding member methods)

Lab activity	13.10.1 LAB: Book information (overriding member methods)

13.11 Programming Assignment 4- Build a Monster

Lab activity	13.11.1 Programming Assignment 4- Build a Monster

13.12 Programming Assignment 4: Monster Fight

Lab activity	13.12.1 Programming Assignment 4: Monster Fight

13.13 Programming Assignment 4: Ghosts and Dragons

Lab activity	13.13.1 Programming Assignment 4: Ghosts and Dragons

14. Recursion

14.1 Recursive functions

recursive function	A function that calls itself is known as a recursive function.
Animation	14.1.1 A recursive function example.
Question set	14.1.2 Recursive functions.
Coding challenge	14.1.1 Calling a recursive function.

14.2 Recursive algorithm: Search

base case	The recursive function has an if-else statement, where the if branch is the end of the recursion, known as the base case. The else part has the recursive calls.
Animation	14.2.1 Binary search: A well-known recursive algorithm.
Figure	14.2.1 A recursive function find() carrying out a binary search algorithm.
Figure	14.2.2 Recursively searching a sorted list.
Question set	14.2.2 Recursive search algorithm.

14.3 Adding output statements for debugging

Figure	14.3.1 Output statements can help debug recursive functions, especially if indented based on recursion depth.
Question set	

14.3.1 Recursive function debug statements.

14.4 Creating a recursive function

base case — Every recursive function must have a case that returns a value without performing a recursive call. That case is called the base case.

- Animation — 14.4.1 Writing a recursive function for factorial: First writing the base case, then adding the recursive case.
- Question set — 14.4.2 Creating a recursive function.
- Coding challenge — 14.4.1 Recursive function: Writing the base case.
- Coding challenge — 14.4.2 Recursive function: Writing the recursive case.

14.5 Recursive math functions

depth — The depth of recursion is a measure of how many recursive calls of a function have been made, but have not yet returned.

- Figure — 14.5.1 Fibonacci sequence step-by-step.
- Figure — 14.5.2 Calculate greatest common divisor of two numbers.
- Figure — 14.5.3 Limit on recursion depth.
- Question set — 14.5.1 Recursive GCD.
- Coding challenge — 14.5.1 Writing a recursive math function.

14.6 Recursive exploration of all possibilities

- Animation — 14.6.1 Exploring all possibilities viewed as a tree of choices.
- Figure — 14.6.1 Scramble a word's letters in every possible way.

Figure

	14.6.2 Shopping spree in which you can fit 3 items in your shopping bag.
Figure	14.6.3 Find distance of traveling to 3 cities.
Question set	14.6.2 Recursive exploration.

14.7 LAB: All permutations of names

Lab activity	14.7.1 LAB: All permutations of names

14.8 LAB: Number pattern

Lab activity	14.8.1 LAB: Number pattern

15. Plotting

15.1 Introduction to plotting and visualizing data

matplotlib	The matplotlib package can be used for plotting in Python.
as	The as keyword renames an imported module or package.
show()	The plt.show() function displays the graph.
plot()	The plt.plot() function plots data onto the graph.
Figure	15.1.1 Plot of ocean temperature from 1850 to 2011.
Question set	15.1.1 Introduction to plotting using matplotlib.
Figure	15.1.2 A program to plot ocean temperatures read from a file.
Figure	15.1.3 Plotting multiple lines in the same graph.
Question set	15.1.2 Plotting data using matplotlib.

15.2 Styling plots

format string	The plot() function takes an optional format string argument that specifies the color and style of the plotted line.
line property	A line property is an attribute of the line object created by matplotlib when plot() is called.
Figure	15.2.1 Format string 'r--' sets line color to red and line style to dashed.
Table	15.2.1 Characters to specify the line color, line style, or marker style.
Question set	15.2.1 Line style format strings.
Table	15.2.2 Line properties.
Figure	15.2.2 Use keyword args to change line properties.
Figure	15.2.3 Adding a legend to a plot.
Question set	15.2.2 Line properties and legends.

15.3 Text and annotations

annotation	The annotate() function creates an annotation that links a text label with a specific data point.
Figure	15.3.1 Adding text to a plot.
Figure	15.3.2 Annotating a specific data point.
Question set	15.3.1 Text and annotations.

15.4 Numpy

numpy	The numpy package provides tools for scientific and mathematical computations in Python. For example, numpy includes functions that can be used to perform common linear algebra operations, fast fourier transforms, and statistics.

array	Numpy uses an array data type that is conceptually similar to a list, consisting of an ordered set of elements of the same type.
linspace	The linspace numpy function creates a sequence by segmenting a given range with a specified number of points.
Figure	15.4.1 Creating arrays.
Figure	15.4.2 Pre-initialized arrays.
Question set	15.4.1 Creating arrays.
Figure	15.4.3 Creating sequences using linspace().
Question set	15.4.2 Creating sequences.
Figure	15.4.4 Array operations program.

15.5 Multiple plots

figure()	The figure() function can be used to create multiple figures.
plt.axis()	The plt.axis() function is used to set the range of the x and y axes.
subplot()	The subplot() function allows multiple plots to be created in a single figure, with each subplot having its own set of axes.
twinx()	The twinx() function creates a second axis on a plot.
Figure	15.5.1 Two types of data on the same plot.
Figure	15.5.2 Using multiple figures.
Question set	15.5.1 Multiple figures.

Figure 15.5.3 Using subplots.

Question set 15.5.2 Subplots.

Figure 15.5.4 Adding a second y-axis on the right side of a plot.

Question set 15.5.3 Using multiple axes, subplots, and figures.

15.6 LAB: Descending selection sort with output during execution

Lab activity 15.6.1 LAB: Descending selection sort with output during execution

15.7 LAB: Sorting user IDs

Lab activity 15.7.1 LAB: Sorting user IDs

16. Searching and Sorting Algorithms

16.1 Searching and algorithms

algorithm An algorithm is a sequence of steps for accomplishing a task.

Linear search Linear search is a search algorithm that starts from the beginning of a list, and checks each element until the search key is found or the end of the list is reached.

runtime An algorithm's runtime is the time the algorithm takes to execute.

Animation 16.1.1 Linear search algorithm checks each element until key is found.

Figure 16.1.1 Linear search algorithm.

Question set 16.1.2 Linear search algorithm execution.

Question set 16.1.3 Linear search runtime.

16.2 Binary search

Binary search

Binary search is a faster algorithm for searching a list if the list's elements are sorted and directly accessible (such as a list).

Animation	16.2.1 Using binary search to search contacts on your phone.
Question set	16.2.2 Using binary search to search a contact list.
Animation	16.2.3 Binary search efficiently searches sorted list by reducing the search space by half each iteration.
Figure	16.2.1 Binary search algorithm.
Question set	16.2.4 Binary search algorithm execution.
Animation	16.2.5 Speed of linear search versus binary search to find a number within a sorted list.
Question set	16.2.6 Linear and binary search runtime.

16.3 O notation

Big O notation

Big O notation is a mathematical way of describing how a function (running time of an algorithm) generally behaves in relation to the input size.

Animation	16.3.1 Determining Big O notation of a function.
Question set	16.3.2 Big O notation.
Figure	16.3.1 Rules for determining Big O notation of composite functions.
Question set	16.3.3 Big O notation for composite functions.
Table	16.3.1 Growth rates for different input sizes.
Learning tool	16.3.4 Computational complexity graphing tool.

Figure — 16.3.2 Runtime complexities for various code examples.

Question set — 16.3.5 Big O notation and growth rates.

16.4 Algorithm analysis

worst-case runtime — The worst-case runtime of an algorithm is the runtime complexity for an input that results in the longest execution.

Animation — 16.4.1 Runtime analysis: Finding the max value.

Question set — 16.4.2 Worst-case runtime analysis.

Animation — 16.4.3 Simplified runtime analysis: A constant number of constant time operations is O(1).

Question set — 16.4.4 Constant time operations.

Animation — 16.4.5 Runtime analysis of nested loop: Selection sort algorithm.

Figure — 16.4.1 Common summation: Summation of consecutive numbers.

Question set — 16.4.6 Nested loops.

16.5 Sorting: Introduction

Sorting — Sorting is the process of converting a list of elements into ascending (or descending) order.

Learning tool — 16.5.1 Sort by swapping tool.

Question set — 16.5.2 Sorted elements.

16.6 Selection sort

Selection sort — Selection sort is a sorting algorithm that treats the input as two parts, a sorted part and an unsorted part, and repeatedly selects the proper next value to move from the unsorted part to the end of the sorted part.

Animation	16.6.1 Selection sort.
Question set	16.6.2 Selection sort algorithm execution.
Figure	16.6.1 Selection sort algorithm.
Question set	16.6.3 Selection sort runtime.

16.7 Insertion sort

Insertion sort

Insertion sort is a sorting algorithm that treats the input as two parts, a sorted part and an unsorted part, and repeatedly inserts the next value from the unsorted part into the correct location in the sorted part.

nearly sorted

A nearly sorted list only contains a few elements not in sorted order.

Animation	16.7.1 Insertion sort.
Figure	16.7.1 Insertion sort algorithm.
Question set	16.7.2 Insertion sort algorithm execution.
Question set	16.7.3 Insertion sort runtime.
Question set	16.7.4 Nearly sorted lists.
Animation	16.7.5 Using insertion sort for nearly sorted list.
Question set	16.7.6 Insertion sort algorithm execution for nearly sorted input.

16.8 Quicksort

Quicksort

Quicksort is a sorting algorithm that repeatedly partitions the input into low and high parts (each part unsorted), and then recursively sorts each of those parts.

pivot

The pivot can be any value within the array being sorted, commonly the value of the middle array element.

Animation — 16.8.1 Quicksort partitions data into a low part with data less than/equal to a pivot value and a high part with data greater than/equal to a pivot value.

Question set — 16.8.2 Quicksort pivot location and value.

Question set — 16.8.3 Low and high partitions.

Animation — 16.8.4 Quicksort.

Figure — 16.8.1 Quicksort algorithm.

Learning tool — 16.8.5 Quicksort tool.

Question set — 16.8.6 Quicksort runtime.

Question set — 16.8.7 Worst case quicksort runtime.

16.9 Merge sort

Merge sort

Merge sort is a sorting algorithm that divides a list into two halves, recursively sorts each half, and then merges the sorted halves to produce a sorted list.

Animation — 16.9.1 Merge sort recursively divides the input into two halves, sorts each half, and merges the lists together.

Question set — 16.9.2 Merge sort partitioning.

Animation — 16.9.3 Merging partitions: Smallest element from the left or right partition is added one at a time to a temporary merged list. Once merged, the temporary list is copied back to the original list.

Question set — 16.9.4 Tracing merge operation.

Figure — 16.9.1 Merge sort algorithm.

Question set 16.9.5 Merge sort runtime and memory complexity.

17. Additional Material

17.1 String formatting (old)

string formatting expression A string formatting expression allows a programmer to create a string with placeholders that are replaced by the values of variables.

conversion specifier Such a placeholder is called a conversion specifier, and different conversion specifiers are used to perform a conversion of the given variable value to a different type when creating the string.

Animation 17.1.1 Using string formatting expressions.

Table 17.1.1 Common conversion specifiers.

Question set 17.1.2 String formatting.

Figure 17.1.1 Multiple conversion specifiers.

Question set 17.1.3 Multiple conversion specifiers.

Coding challenge 17.1.1 Printing a string.

Progression 17.1.2 String formatting.

17.2 String formatting using dictionaries

mapping key If a dictionary is used, then all conversion specifiers must include a mapping key component. A mapping key is specified by indicating the key of the relevant value in the dictionary within parentheses.

Animation 17.2.1 Using a dictionary and conversion specifiers with mapping keys.

Figure 17.2.1 Comparing conversion operations using tuples and dicts.

Question set 17.2.2 Mapping keys.