From The Amazon #1
Best-Selling Author

WORK BOOK

INTRO TO PYTHON PROGRAMMING BEGINNERS GUIDE SERIES

JOHN ELDER

Intro To Python Programming Beginners Guide Series Workbook

John Elder

Codemy.com
Las Vegas

Intro To Python Programming Beginners Guide Series Workbook

By: John Elder

Published By Codemy.com
Las Vegas, NV USA

ISBN
979-8-9859654-3-8

First Edition

http://www.Codemy.com

FOR APRIL

TABLE OF CONTENTS

INTRO TO
PYTHON
PROGRAMMING

BEGINNERS GUIDE SERIES
WORKBOOK

ABOUT THE AUTHOR

John Elder is a Web Developer, Entrepreneur, and Author living in Las Vegas, NV. He created one of the earliest online advertising networks in the late nineties and sold it to publicly traded WebQuest International Inc. at the height of the first dot-com boom.

He went on to develop one of the Internet's first Search Engine Optimization tools, the Submission-Spider that was used by over three million individuals, small businesses, and governments in over forty-two countries.

These days John writes about Coding and wealth creation; produces a popular coding Youtube Channel, and runs **Codemy.com** the online learning platform where he’s taught Coding to over 20 million students.

John graduated with honors from Washington University in St. Louis with a degree in Economics. He can be reached at john@codemy.com

INTRODUCTION

Welcome to the Python Programming Workbook!

Hey there, John Elder here—founder of Codemy.com and your friendly neighborhood Python guide. First of all, high five for picking this up!

Whether you're working through this with my *Intro to Python Programming* book…

(**https://www.amazon.com/Intro-Python-Programming-Beginners-Guide/dp/B09VDRSKB7/**)

...or you've just stumbled in here looking for Python practice that doesn't make you fall asleep at your keyboard, you're in the right place.

This workbook is the companion to the original *Intro to Python Programming* book I wrote—a guide I designed to help absolute beginners learn to code in Python without the overwhelm, the jargon, or the snarky attitude you sometimes get from other programmers.

And while that book lays down the knowledge, **this workbook is where you get your reps in**.

Think of the original book like the driving manual, and this workbook like your time behind the wheel. You need both to become a confident, skilled Python programmer.

Reading about loops is one thing—writing a loop that counts backwards from 100 while complimenting you at each step? Totally different beast.

What This Workbook Is (and Isn't)

This isn't a quiz book. There are no multiple choice questions here. I'm not here to test you. I'm here to help you **practice**, play, build muscle memory, and gain real fluency with Python.

Each chapter of the workbook lines up directly with a chapter from the original book. That means **you should absolutely read the chapter first**, get the lay of the land, and then come here to apply what you just learned.

You'll go from reading about concepts to actually using them in fun, silly, practical, and sometimes ridiculous exercises.

Here's how it's structured:

Chapter 2: Writing Our First Python Program
You'll get your hands dirty printing, commenting, debugging, and even creating ASCII art. If you thought "Hello, World!" was all there was to printing in Python… oh buddy.

Chapter 3: Math, Variables, and Inputs
Practice turning boring numbers into interactive calculators, story generators, and grocery totalers. Yes, that's a word now.

Chapter 4: IF/ELSE Statements
This is where logic comes in. You'll decide who gets on the rollercoaster, how to grade exams, and whether someone can hack it in your text adventure.

Chapter 5: Lists
Organize everything from snacks to birthday gifts. Practice modifying, removing, and repeating list items like a pro.

Chapter 6: Loops
You'll count things, break things, skip things, and build things—all while looping until Python tells you to stop. You might even meet a zoo animal or two.

Chapter 7: Functions
Say goodbye to copy-paste and hello to code reusability. Build your own code tools like insult generators, emoji printers, and dramatic battle announcers.

Chapter 8: Dictionaries
Learn to map data like a proper Pythonista. Use key-value pairs to build inventories, translate emoji, simulate vote counts, and even manage a fictional zoo.

Chapter 9: Is This Thing On?!
Time to bring it all together—functions, loops, dictionaries—all working as one in a fully interactive quiz app. Then remix it into your own trivia game, vocabulary test, or emoji decoder.

Conclusion
A friendly nudge to keep going, build your own stuff, and maybe even take over the world (with Python, obviously).

What Makes This Workbook Different

Most programming workbooks are, well… boring. A sea of fill-in-the-blank definitions and "What does this code do?" questions that feel more like a test than practice.

Not here.

This workbook was designed to make you **want** to practice. Every exercise is here to stretch your skills without being soul-crushing.

Some are practical. Some are absurd. Some are mini-challenges. All of them are original. None of them reuse the examples from the book—so you're always applying the concepts in fresh ways.

And every chapter includes **10 exercises**, each with clearly labeled solutions at the end of each chapter so you can check your work, troubleshoot, and learn from your mistakes.

You don't have to get everything perfect the first time. The point is to learn by doing.

How to Use This Workbook

1. **Read the chapter in the original book first.** Don't skip this. The exercises assume you understand the basics presented in the chapter.

2. **Come to the workbook and do the exercises.** Try them on your own before peeking at the answers.

3. **Run the code. Break it. Fix it.** You'll learn 100x more by seeing what happens than just reading a correct answer.

4. **Make it your own.** Every exercise can be remixed. Add your name, your favorite foods, weird facts—whatever makes it stick.

Final Thoughts Before You Dive In

Programming isn't about memorizing syntax—it's about thinking in patterns and solving problems. And the only way to get better at solving problems is to solve a bunch of them.

That's what this workbook is: a problem-solving playground that stays in sync with the book you're already reading.

If you use them together, you're not just going to understand Python—you're going to be able to use it, comfortably and confidently.

So fire up your text editor, warm up your brain, and get ready to code your way through the rest of the book. I'm glad you're here.

Now let's go build something awesome.

- John Elder
Codemy.com

Want To Keep Going After the Workbook?

Build 20 Real Python Programs With Me…

From Practice to Production—One Project at a Time

If you make it through this workbook, you'll have already done more than most people who say they want to learn Python. But here's the truth: knowing *about* Python isn't the same as being able to *use* it. If you really want to get good, you need to build stuff.

That's exactly why I created the **Python Projects** course over at Codemy.com.

In this course, we build **20 real-world projects** together—each one designed to take your skills up a level. No fluff, no filler, just actual applications that teach you how to *think like a programmer* and use Python to solve real problems.

Here's what we'll build together:

1. Number Guessing Game – Learn loops and conditionals with a classic terminal game.

2. Number Guessing with Tkinter – Upgrade the game with a GUI and get started with app development.

3. Rock Paper Scissors – Use functions and basic logic to outsmart the computer.

4. Ticking Clock – Build a real-time clock using Python's time module.

5. Palindrome Checker – Practice string slicing and logical thinking.
6. Using Arrow Keys – Make your apps interactive with keyboard event handling.
7. Password Validator – Learn how to validate strong passwords with Python.
8. Password Generator – Automatically create secure, randomized passwords.
9. Hangman Game – Build a fully functional word game with logic and looping.
10. To-Do List App – Create your own productivity app using basic GUI structure.
11. Word Count Tool – Work with text files and count words with file I/O.
12. Tic Tac Toe Game – Master game flow, input handling, and win detection.
13. Draw with Turtle – Use the Turtle graphics module to code creative designs.
14. Image Resizer – Resize and process images for personal or web use.
15. Anagram Solver – Solve and detect anagrams using clever logic and lists.

16. Typing Speed Test – Create a timed typing tool to test speed and accuracy.

17. Flashcard App – Build an interactive study aid with a dictionary backend.

18. Choose Your Own Adventure – Design a branching story with complex logic.

19. File Manager – Organize and manage files with custom-built scripts.

20. File Subprocessor – Automate file reading and processing for real-world tasks.

It's everything you need to go from "I know Python syntax" to "I built 20 real things with Python."

The full course normally sells for $129, but as a thank-you for working through this workbook, you can grab it now for just **$64** here: **https://order.codemy.com/python-projects-64**

Or learn more about what's inside here:
https://codemy.com/python-projects/

If you liked this workbook, you'll love the course. Hope to see you inside.

CHAPTER TWO

WRITING OUR FIRST PYTHON PROGRAM

Alright! This is where the rubber meets the road. In Chapter Two of the book, we learned how to write and run our very first Python program.

We talked about how to open a text editor (like Sublime Text), create a new .py file, and run it from the terminal. You typed out the classic "Hello World!" — a rite of passage for every new programmer.

We also took a peek at the print() function, how it works, and how Python doesn't need all the weird semicolons and parentheses that other languages demand.

You got a crash course in syntax errors (yay typos!), saw what happens when things go wrong, and learned how to fix them. And let's not forget comments — those handy little notes that explain your code so future-you doesn't scream into the void.

So now that you know how to write, run, and debug a Python program, let's solidify those skills with some fun hands-on practice!

Yes, these are gonna be a little lame because – let's face it – we haven't learned all that much yet. But give them a try anyway!

Exercise 2.1

Hello World – First Impressions Count!

Write a Python program that outputs "Hello, Python World!". Then, personalize it by adding your name or a funny nickname.

Exercise 2.2

Print-It Party – Making Python Talk

Create a Python program that prints a short story about your favorite animal, complete with fun details and excitement.

Exercise 2.3

Whoops-a-Daisy – Learning Through Errors

Write Python code with intentional errors (like missing parentheses, typos, etc.). Try running it and then correct the errors one by one, noting what you fixed and why.

Exercise 2.4

Comment Craze – What's the Big Idea?

Write a small Python program, then add comments explaining each line. Make your comments playful and humorous.

Exercise 2.5
Coding Chaos – Debugging Fun
Take a given Python snippet full of mistakes (create your own or use the one in the solution) and debug it. List each error and explain how you fixed it.

Exercise 2.6
Print the Unexpected – Creative Strings
Use Python's print function to create a program that outputs a humorous dialogue between two fictional characters of your choice.

Exercise 2.7
Hidden Messages – Comment Coding
Write a Python program with hidden secret messages in comments. This would be really cool if we were all twelve years old! :-p

Exercise 2.8
Guess My Error – Spot and Correct
Partner with a fictional friend to write Python programs with intentional errors. Then fix the errors.

Exercise 2.9
Terminal Treasure Hunt
Create a list of instructions using Python's print statements. When executed, these instructions should guide the user through a terminal "treasure hunt."

Exercise 2.10
ASCII Art Adventure
Use Python's print statements to create your own ASCII art masterpiece. It could be your favorite animal, object, or even a funny self-portrait!

Answers to Chapter Two Exercises

Answer 2.1

```
print("Hello, Python World! My name is The Code Whisperer!")
```

Answer 2.2

```
print("Once upon a time, a sloth named Flash raced a cheetah... and won. Sort of. The cheetah stopped for snacks. Thank you, I’m here all week!")
```

Answer 2.3

```
print "This won't work!"
```

Fix:

```
print("This works now!")  # Added parentheses
```

Answer 2.4

```
# This program tells you the meaning of life!
print("The meaning of life is...")  # Drumroll please
print(42)  # Douglas Adams would be proud
```

Answer 2.5 Before:

```
print("Hello World!"
pritn("Oops")
```

After:

```
print("Hello World!")  # Fixed missing parenthesis
print("Oops")          # Fixed typo in function name
```

Answer 2.6

```
print("Batman: Where's the Joker?")
print("Joker: Probably debugging his Python code!")
```

Answer 2.7

```
# X1: Meet at the old bridge
# X2: Bring the USB stick
# X3: The password is "snakes123"
print("Nothing to see here, just a normal program.")
```

Answer 2.8

```
# Friend's broken code:
# prnt("What's up?")
# Fixed version:
print("What's up?")  # Fixed function name typo
```

Answer 2.9

```
print("Welcome to the Python Treasure Hunt!")
print("Clue 1: Look under the keyboard.")
print("Clue 2: Behind the monitor, you'll find your next lead.")
print("Clue 3: Check inside the README file on your Desktop.")
```

Answer 2.10

```
print("""
 /\_/\
( o.o )
 > ^ <
""")
```

CHAPTER THREE

FUN WITH MATH, VARIABLES, AND INPUTS

In Chapter Three of the main book, we got our first real taste of Python's power. You learned how to make your programs do math, store data in variables, and interact with users through input.

You saw how Python handles integers and floats (yes, decimal points matter!), and how you can use variables to keep track of data as your program runs.

We also experimented with mixing variables and strings to build dynamic output.

This is the chapter where Python starts feeling more like a tool and less like a toy.

So let’s play around with what we’ve learned and solidify those skills with some exercises!

Exercise 3.1

Calculator Fun – Math Time!

Write a program that adds, subtracts, multiplies, and divides two numbers. Display the results using print statements.

Exercise 3.2

Floaty Numbers – Decimals Matter

Create a program that divides 7 by 3 and prints the result. Then, try a few other combinations (like 10 divided by 4 or 5 divided by 2) to see how Python handles decimal values.

Exercise 3.3

Variable Swap – Switcheroo!

Assign values to two variables, then swap them and print the results. Do this without hardcoding new values.

Exercise 3.4

Favorite Animal – Input Practice

Ask the user what their favorite animal is, then print a sentence that includes their answer in a funny or creative way.

Exercise 3.5
Birth Year Guess – The Age Calculator
Ask the user what year they were born in. Subtract it from the current year and print their age.

Exercise 3.6
Favorite Number – String + Variable
Ask the user for their favorite number. Then print a fun sentence that includes it.

Exercise 3.7
Story Time – Input & Concatenation
Ask the user for a noun, a verb, and an adjective. Then build a silly one-line story using their inputs.

Exercise 3.8
Python Mad Libs
Create a simple Mad Lib using at least three input() prompts and some variables. Then print the full story.

Exercise 3.9
Grocery Total – Math with Input
Ask the user the cost of two grocery items. Add them together and print the total.

Exercise 3.10
Math Story Problem
Write a program that tells a little story involving math (e.g., “If John has 3 apples and buys 5 more…”). Calculate the result and print it.

Answers to Chapter Three Exercises

Answer 3.1

```
a = 10
b = 5
print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
```

Answer 3.2

```
print(7 / 3)
print(10 / 4)
print(5 / 2)
```

Answer 3.3

```
a = 1
b = 2
a, b = b, a
print("a:", a)
print("b:", b)
```

Answer 3.4

```
animal = input("What is your favorite animal? ")
print("Wow! A " + animal + " would make a great coding buddy!")
```

Answer 3.5

```
birth_year = int(input("What year were you born? "))
current_year = 2025
age = current_year - birth_year
print("You are", age, "years old.")
```

Answer 3.6

```
fav_num = input("What’s your favorite number? ")
print("Wow! " + fav_num + " is a great number!")
```

Answer 3.7

```
noun = input("Give me a noun: ")
verb = input("Give me a verb: ")
adjective = input("Give me an adjective: ")
print("The", adjective, noun, verb, "down the hill!")
```

Answer 3.8

```
animal = input("Animal: ")
color = input("Color: ")
action = input("Action verb: ")
print("One day, a", color, animal, action, "into the sunset.")
```

Answer 3.9

```
item1 = float(input("Enter the cost of item 1: "))
item2 = float(input("Enter the cost of item 2: "))
total = item1 + item2
print("Your total grocery bill is:", total)
```

Answer 3.10

```
apples = 3
bought = 5
total = apples + bought
print("John had", apples, "apples and bought", bought, "more. Now he
has", total, "apples.")
```

CHAPTER FOUR
IF/ELSE STATEMENTS

In Chapter Four of the main book, you learned how to make decisions in your code using if, elif, and else.

This is Python's way of letting your program take different paths depending on the data it sees. We walked through how Python checks conditions in order, how comparison operators work (==, !=, <, >, etc.), and how you can chain conditions using and and or.

You also saw how indentation matters — Python won't tolerate sloppy code formatting!

Now let's reinforce those ideas with some fun and practical exercises.

The exercises will start to get a little harder and a little more interesting from here on out!

Exercise 4.1
Roller Coaster Ride – Height Check
Write a program that asks for a user's height in inches. If they are 48 inches or taller, print that they can ride the roller coaster. Otherwise, print a message that says they need to grow a bit more.

Exercise 4.2
Even or Odd?
Ask the user for a number. Use an if/else statement to print whether the number is even or odd.

Exercise 4.3
Password Checker
Ask the user to enter a password. If the password is "swordfish", print "Access granted". Otherwise, print "Access denied".

Exercise 4.4
Grading Machine
Ask the user to enter a number between 0 and 100. Print the letter grade using if/elif/else:
- 90 or higher: A
- 80–89: B
- 70–79: C
- 60–69: D
- Below 60: F

Exercise 4.5
Double Trouble
Ask the user for two numbers. Use if/else to print which one is larger, or print a message if they are the same.

Exercise 4.6
Animal Match Game
Ask the user to input their favorite animal. If it's "dog", "cat", or "rabbit", print "Great choice!". Otherwise, print "That's an interesting pick!"

Exercise 4.7
Dinner Plans – Logical Operators
Ask the user if they are hungry (yes or no) and if they have food at home (yes or no).
- If hungry and no food at home, print "Time to order takeout!"
- If hungry and food at home, print "Time to cook!"
- If not hungry, print "Maybe later."

Exercise 4.8
Number Range Checker
Ask the user for a number. Use if and and to check if the number is between 10 and 20, inclusive.

Exercise 4.9
Weather Wizard
Ask the user if it's raining and if it's cold. Print different suggestions depending on the answers (e.g., "Bring an umbrella", "Wear a coat", etc.)

Exercise 4.10
Choose Your Path – Text Adventure
Write a simple text-based game where the user chooses between two paths. Use if/else to give different story outcomes.

Answers to Chapter Four Exercises

Answer 4.1

```
height = int(input("What is your height in inches? "))
if height >= 48:
   print("You can ride the roller coaster!")
else:
   print("Sorry, you need to grow a bit more.")
```

Answer 4.2

```
num = int(input("Enter a number: "))
if num % 2 == 0:
   print("Even")
else:
   print("Odd")
```

Answer 4.3

```
password = input("Enter the password: ")
if password == "swordfish":
   print("Access granted")
else:
   print("Access denied")
```

Answer 4.4

```
grade = int(input("Enter a number between 0 and 100: "))
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
else:
    print("F")
```

Answer 4.5

```
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
if num1 > num2:
    print("The first number is larger.")
elif num2 > num1:
    print("The second number is larger.")
else:
    print("Both numbers are the same.")
```

Answer 4.6

```
animal = input("What is your favorite animal? ")
if animal == "dog" or animal == "cat" or animal == "rabbit":
    print("Great choice!")
else:
    print("That's an interesting pick!")
```

Answer 4.7

```
hungry = input("Are you hungry? (yes/no): ")
have_food = input("Do you have food at home? (yes/no): ")
if hungry == "yes" and have_food == "no":
    print("Time to order takeout!")
elif hungry == "yes" and have_food == "yes":
    print("Time to cook!")
else:
    print("Maybe later.")
```

Answer 4.8

```
num = int(input("Enter a number: "))
if num >= 10 and num <= 20:
    print("That number is within range!")
else:
    print("Out of range.")
```

Answer 4.9

```
raining = input("Is it raining? (yes/no): ")
cold = input("Is it cold? (yes/no): ")
if raining == "yes" and cold == "yes":
    print("Bring an umbrella and wear a coat!")
elif raining == "yes":
    print("Bring an umbrella!")
elif cold == "yes":
    print("Wear a coat!")
else:
    print("You're good to go!")
```

Answer 4.10

```
path = input("Do you choose the left path or the right path? ")
if path == "left":
    print("You find a treasure chest!")
else:
    print("You fall into a pit of snakes!")
```

CHAPTER FIVE
LISTS

In Chapter Five of the main book, we learned how to create lists, access items using indexes, change individual elements, and use methods like .append(), .insert(), .remove(), and .pop().

We also experimented with lists that contain other lists — also known as multidimensional lists.

Let's reinforce those concepts with exercises that match what you've seen in the book so far.

Exercise 5.1
Favorite Foods
Create a list of your three favorite foods and print the whole list.

Exercise 5.2
Add to the List
Start with a list of two school supplies. Add two more using .append() and print the final list.

Exercise 5.3
Replacing the Middle
Create a list with three silly objects. Change the second one to something even sillier, then print the result.

Exercise 5.4
Pop Goes the List
Make a list of three sounds. Use .pop() to remove the last one and print what was removed and the updated list.

Exercise 5.5
Insert It In
Create a list with three movies. Use .insert() to add a new movie at the beginning. Then print the updated list.

Exercise 5.6
Remove That!
Create a list with five chores. Use .remove() to delete your least favorite chore. Print the final list.

Exercise 5.7
First and Last
Make a list of four emojis. Print the first and last using indexing.

Exercise 5.8
Double Trouble
Create a list of your three favorite animals. Then make a second list of your three favorite colors. Combine them into one list that contains both lists. Print the result.

Exercise 5.9
Multidimensional Peek
Make a list called outer that contains two lists: one of fruits and one of vegetables. Print one item from each inner list using double indexing.

Exercise 5.10

Build a Nested List from Scratch

Ask the user to enter two colors and two shapes. Store colors in one list and shapes in another. Combine both lists into one nested list and print it.

Answers to Chapter Five Exercises

Answer 5.1

```
foods = ["pizza", "sushi", "tacos"]
print(foods)
```

Answer 5.2

```
supplies = ["pencil", "notebook"]
supplies.append("eraser")
supplies.append("highlighter")
print(supplies)
```

Answer 5.3

```
objects = ["banana", "shoe", "lamp"]
objects[1] = "rubber chicken"
print(objects)
```

Answer 5.4

```
sounds = ["bang", "boing", "pop"]
removed = sounds.pop()
print("Removed:", removed)
print("Updated list:", sounds)
```

Answer 5.5

```
movies = ["Titanic", "Avatar", "Up"]
movies.insert(0, "Inception")
print(movies)
```

Answer 5.6

```
chores = ["dishes", "laundry", "vacuum", "dust", "mop"]
chores.remove("laundry")
print(chores)
```

Answer 5.7

```
emojis = ["😃", "😂", "😎", "🤖"]
print(emojis[0])
print(emojis[3])
```

Answer 5.8

```
animals = ["cat", "dog", "turtle"]
colors = ["red", "blue", "green"]
combined = [animals, colors]
print(combined)
```

Answer 5.9

```
outer = [["apple", "banana"], ["carrot", "broccoli"]]
print(outer[0][1])  # banana
print(outer[1][0])  # carrot
```

Answer 5.10

```
colors = []
shapes = []
colors.append(input("Enter a color: "))
colors.append(input("Enter another color: "))
shapes.append(input("Enter a shape: "))
shapes.append(input("Enter another shape: "))
nested = [colors, shapes]
print(nested)
```

CHAPTER SIX
LOOPS

In Chapter Six of the main book, we learned about two powerful tools for repeating actions in code: the while loop and the for loop.

A while loop keeps running as long as a condition is true, and a for loop lets us step through each item in a list.

We also explored break, continue, and pass, as well as the use of else blocks with loops.

These exercises will help you practice what you learned in this chapter — and only what you learned so far.

Exercise 6.1
Even Countdown
Use a while loop to count down from 10 to 2 by twos. Print each number.

Exercise 6.2
Zoo Animals
Create a list of four zoo animals. Use a for loop to print each one followed by the phrase " is cool!"

Exercise 6.3
Count and Compliment
Use a while loop to count from 1 to 5. When the number is 3, print "Three is the best number!" instead of the number.

Exercise 6.4
Build a Word
Make a list of letters that spell out a short word (like ['c', 'a', 't']). Use a for loop to print the word one letter at a time.

Exercise 6.5
No Fours Allowed
Write a for loop that prints numbers 1 through 5 but skips 4 using continue.

Exercise 6.6
Stop at Banana
Make a list of fruit: ['apple', 'orange', 'banana', 'grape']. Use a for loop to print each fruit. Use break to stop the loop when you reach "banana".

Exercise 6.7
Empty Loop Else
Create a for loop that runs through an empty list. Add an else block that prints "Nothing to see here!"

Exercise 6.8
Just Passing Through
Loop through the word "class". If the letter is "s", use pass. Print the rest of the letters.

Exercise 6.9
Password Practice
Use a while loop to keep asking the user to enter the word "code". Only stop once they type it exactly.

Exercise 6.10
Buzz Until Stop
Use a while loop that runs forever. Ask the user to type a word. If the word is "stop", break the loop. Otherwise, print "buzz!"

Answers to Chapter Six Exercises

Answer 6.1

```
i = 10
while i >= 2:
    print(i)
    i -= 2
```

Answer 6.2

```
animals = ["lion", "zebra", "elephant", "giraffe"]
for animal in animals:
    print(animal + " is cool!")
```

Answer 6.3

```
i = 1
while i <= 5:
    if i == 3:
        print("Three is the best number!")
    else:
        print(i)
    i += 1
```

Answer 6.4

```
letters = ['c', 'a', 't']
for letter in letters:
    print(letter)
```

Answer 6.5

```
for i in [1, 2, 3, 4, 5]:
    if i == 4:
        continue
    print(i)
```

Answer 6.6

```
fruits = ['apple', 'orange', 'banana', 'grape']
for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)
```

Answer 6.7

```
items = []
```

```
for item in items:
   print(item)
else:
   print("Nothing to see here!")
```

Answer 6.8

```
for letter in "class":
   if letter == "s":
      pass
   else:
      print(letter)
```

Answer 6.9

```
entry = ""
while entry != "code":
   entry = input("Enter the password: ")
```

Answer 6.10

```
while True:
    word = input("Type a word: ")
    if word == "stop":
        break
    print("buzz!")
```

CHAPTER SEVEN
FUNCTIONS

Let’s Put Some “Fun” in Functions

In this chapter we rolled up our sleeves and built our own little Python machines—functions!

You learned how to define a function using the def keyword, how to call it by name to make it run, and how to pass in information using parameters and arguments.

You also saw how to give your function parameters default values, which is super handy when you want your function to work even if no info is passed in.

This is where coding starts to feel powerful.

Instead of repeating code over and over, you can name it, pack it into a function, and reuse it any time you like.

Clean, tidy, and just plain cool.

Let’s keep the momentum going with some offbeat, hands-on exercises to drive this home.

Exercise 7.1
The Spam Factory
Write a function called spam_repeater that takes two parameters: a word and a number. It should print the word that number of times, once per line. Call it with "SPAM" and 7.

Exercise 7.2
Mad Libs: Python Edition
Write a function called mad_lib that takes three arguments: name, color, and animal. It should print a silly sentence using all three, like “Once upon a time, <name> met a <color> <animal>.” Call it at least three times with different inputs.

Exercise 7.3
Default Insult Generator
Create a function called insult that takes two parameters: name and adjective. Both should have default values: "you" and "smelly". The function should print “Hey <name>, you’re so <adjective>!” Call it with no arguments, then with one, then with two.

Exercise 7.4
Fighter Intro Announcer
Write a function called intro that takes three arguments: name, power, and weapon. It should print “Prepare yourselves! <name> arrives, wielding <weapon>, powered by <power>!” Call it three times with different combinations.

Exercise 7.5
Emoji Printer
Write a function called print_emojis that takes two parameters: an emoji and a number. It should print that emoji that number of times, all on one line. Use it to print 20 fire emojis, 10 thumbs-up emojis, and 3 alien emojis.

Exercise 7.6
Custom Liner
Write a function called draw_line that takes two arguments: a character (default is "*") and a length (default is 10). It should print a line using the character repeated the given number of times. Call it with no arguments, then with "-", 20.

Exercise 7.7
Build-A-Band
Create a function called band_setup that takes three arguments: singer, guitarist, and drummer. Make drummer a default parameter set to "Drum Machine". It should print “Now performing: <singer> on vocals, <guitarist> on guitar, and <drummer> on drums!” Try calling it with and without the drummer.

Exercise 7.8
Code Red Alert
Define a function called alert that takes two parameters: level and message. It should print "[<level>] ALERT: <message>!" Call it three times using levels like "INFO", "WARNING", and "PANIC".

Exercise 7.9
Commentator Combo
Write a function called battle_log that takes three parameters: hero, villain, and location. Give location a default value like "on a volcano". It should print "Today, <hero> will face off against <villain> <location>." Call it once with and once without a location.

Exercise 7.10
Function Multipack
Create three functions:
- open_box prints "You opened the box!"
- surprise prints "A snake jumps out!"
- open_surprise_box calls the first two in order

Call open_surprise_box once to test.

Answers to Chapter Seven Exercises

Answer 7.1

```
def spam_repeater(word, times):
    for i in range(times):
        print(word)

spam_repeater("SPAM", 7)
```

Answer 7.2

```
def mad_lib(name, color, animal):
    print("Once upon a time, " + name + " met a " + color + " " + animal +
".")

mad_lib("Lisa", "purple", "penguin")
mad_lib("Mike", "green", "zebra")
mad_lib("Ava", "blue", "narwhal")
```

Answer 7.3

```
def insult(name="you", adjective="smelly"):
    print("Hey " + name + ", you’re so " + adjective + "!")

insult()
insult("Bob")
insult("Karen", "loud")
```

Answer 7.4

```
def intro(name, power, weapon):
    print("Prepare yourselves! " + name + " arrives, wielding " + weapon +
", powered by " + power + "!")

intro("Zephyr", "wind magic", "boomerang")
intro("Blazefang", "lava breath", "flaming axe")
intro("Cuddles", "puppy eyes", "plush hammer")
```

Answer 7.5

```
def print_emojis(emoji, count):
    print(emoji * count)

print_emojis("🔥", 20)
print_emojis("👍", 10)
print_emojis("👽", 3)
```

Answer 7.6

```
def draw_line(char="*", length=10):
    print(char * length)

draw_line()
draw_line("-", 20)
```

Answer 7.7

```
def band_setup(singer, guitarist, drummer="Drum Machine"):
    print("Now performing: " + singer + " on vocals, " + guitarist + " on guitar, and " + drummer + " on drums!")

band_setup("Liz", "Tom", "Danny")
band_setup("Echo", "Vox")
```

Answer 7.8

```
def alert(level, message):
    print("[" + level + "] ALERT: " + message + "!")

alert("INFO", "All systems normal.")
alert("WARNING", "Disk space running low.")
alert("PANIC", "Aliens are in the server room!")
```

Answer 7.9

```
def battle_log(hero, villain, location="on a volcano"):
    print("Today, " + hero + " will face off against " + villain + " " + location + ".")

battle_log("Captain Code", "Bugzilla")
battle_log("Syntax Sorcerer", "NullPointer", "inside a debugger")
```

Answer 7.10

```
def open_box():
   print("You opened the box!")

def surprise():
   print("A snake jumps out!")

def open_surprise_box():
   open_box()
   surprise()

open_surprise_box()
```

CHAPTER EIGHT
DICTIONARIES

Keys to the Kingdom

Dictionaries are Python's way of storing data in pairs—like labels and values.

You've got keys, and each key is connected to a value. It's like a super-powered list where you don't have to remember what position something is in—you just use its label instead.

In this chapter, you learned how to make dictionaries with curly braces, how to add key-value pairs, how to change existing values, how to loop through all the items, and how to get the value associated with a key.

Dictionaries are everywhere in real Python code, so the more comfortable you get with them, the better.

Let's reinforce what you've learned with some practical and (slightly ridiculous) exercises.

Exercise 8.1
Escape Room Profile Loader
Create a dictionary called player that stores a game player's name, inventory count, access level, and location. After defining it, simulate the player picking up an item by increasing the inventory count by 1, then changing their location to "Vault". Finally, print a sentence that uses both updated values.

Exercise 8.2
Build-a-Planet
Create a dictionary called planet that includes keys like name, number_of_moons, and breathable (True or False). Then add a new key called danger_level with a value of your choice. Print a full sentence for each key-value pair that describes the planet (example: This planet has 2 moons).

Exercise 8.3
Quiz Score Evaluator
Create a dictionary called quiz_scores with at least four students and their numeric scores. Then loop through the dictionary and print a custom message for each student: if their score is 70 or higher, say they passed; otherwise, say they need to study more.

Exercise 8.4

Monster Weakness Map

Create a dictionary that maps three fictional monster names to their weaknesses. Then write a loop that prints out a warning for each one, like: Warning: To defeat Slarg, use fire.

Exercise 8.5

Shopping Cart Updater

Start with an empty dictionary called shopping_cart. Then add at least three items to it where the keys are product names and the values are prices. Then increase the price of one item by 10 and print the full cart.

Exercise 8.6

Menu Calorie Counter

Create a dictionary called menu with three dishes and their calorie counts. Write a loop that calculates and prints the total number of calories for all dishes combined.

Exercise 8.7

Currency Converter

Write a dictionary that maps three countries to their currency (like USA to dollar). Then simulate changing the currency of one of them (for example, Italy from lira to euro) and print before and after.

Exercise 8.8
Vote Tally Simulator
Create a dictionary called votes with at least five candidate names as keys and zero as the starting vote count. Then simulate 10 random votes by manually incrementing different candidates' counts. Print the final tally.

Exercise 8.9
Winter Wardrobe Filter
Make a dictionary called wardrobe that has clothing items as keys and seasons as values. Write a loop that filters and prints only the clothing items appropriate for winter.

Exercise 8.10
Zoo Inventory Tracker
Build a nested dictionary called zoo where each key is an animal species and the value is another dictionary with keys like count and habitat. Add three animals. Then loop through and print out a summary line for each one.

Answers To Chapter Eight Exercises

Answer 8.1

```
player = {
    "name": "Eli",
    "inventory_count": 3,
    "access_level": "basic",
    "location": "Lobby"
}
player["inventory_count"] += 1
player["location"] = "Vault"
print("Player now has", player["inventory_count"], "items and is in",
player["location"])
```

Answer 8.2

```
planet = {
    "name": "Xenor",
    "number_of_moons": 2,
    "breathable": False
}
planet["danger_level"] = "high"

print("Planet name:", planet["name"])
print("This planet has", planet["number_of_moons"], "moons.")
print("Breathable atmosphere:", planet["breathable"])
print("Danger level:", planet["danger_level"])
```

Answer 8.3

```
quiz_scores = {
    "Alice": 85,
    "Bob": 67,
    "Charlie": 73,
    "Daisy": 59
}
for student in quiz_scores:
    score = quiz_scores[student]
    if score >= 70:
        print(student + " passed!")
    else:
        print(student + " needs to study more.")
```

Answer 8.4

```
monsters = {
    "Slarg": "fire",
    "Grumble": "ice",
    "Flarp": "sunlight"
}
for name in monsters:
    print("Warning: To defeat " + name + ", use " + monsters[name] + ".")
```

Answer 8.5

```
shopping_cart = {}
shopping_cart["tofu"] = 3.50
shopping_cart["bread"] = 2.00
shopping_cart["salsa"] = 4.25
shopping_cart["salsa"] = shopping_cart["salsa"] + 1.00
print(shopping_cart)
```

Answer 8.6

```
menu = {
    "burrito": 550,
    "salad": 320,
    "smoothie": 410
}
total = 0
for dish in menu:
    total += menu[dish]
print("Total calories:", total)
```

Answer 8.7

```
currencies = {
    "USA": "dollar",
    "Japan": "yen",
    "Italy": "lira"
}
print("Before:", currencies)
currencies["Italy"] = "euro"
print("After:", currencies)
```

Answer 8.8

```
votes = {
    "Ava": 0,
    "Ben": 0,
    "Cara": 0,
    "Dan": 0,
    "Elle": 0
}
votes["Ava"] += 3
votes["Ben"] += 1
votes["Cara"] += 2
votes["Dan"] += 2
votes["Elle"] += 2
print(votes)
```

Answer 8.9

```
wardrobe = {
   "scarf": "winter",
   "tank top": "summer",
   "boots": "winter",
   "shorts": "summer",
   "jacket": "winter"
}
for item in wardrobe:
   if wardrobe[item] == "winter":
      print(item)
```

Answer 8.10

```
zoo = {
    "giraffe": {
        "count": 5,
        "habitat": "savannah"
    },
    "penguin": {
        "count": 12,
        "habitat": "ice cave"
    },
    "koala": {
        "count": 3,
        "habitat": "eucalyptus grove"
    }
}
for animal in zoo:
    info = zoo[animal]
    print(animal + ": " + str(info["count"]) + " in " + info["habitat"])
```

CHAPTER NINE
IS THIS THING ON?!

Time to Put It All Together

This chapter brought everything together—dictionaries, loops, and functions—to build something useful: a flashcard game.

You learned how to store questions and answers in a dictionary, how to loop through those questions, ask the user for input, and compare their response to the correct answer.

This is where Python starts to feel powerful. You're not just printing stuff anymore—you're interacting.

In this workbook chapter, you'll rework and remix those same ideas: using dictionaries to store prompts, looping through them, calling functions to organize behavior, and adding your own twists to the format.

These problems are designed to help you take full ownership of this pattern.

Exercise 9.1
Riddle Dictionary
Create a dictionary called riddles with at least three riddles as keys and their answers as values. Loop through the dictionary and print each riddle followed by the correct answer (you're not asking the user—just displaying the Q&A).

Exercise 9.2
run_quiz Function
Write a function called run_quiz that takes a dictionary of questions and correct answers. Inside the function, loop through the questions, ask the user for input, and compare it to the correct answer. At the end, print how many they got right.

Exercise 9.3
Math Fact Checker
Make a dictionary called math_facts where the keys are simple math problems as strings (like 5 + 3) and the values are correct answers as integers. Write a loop that asks the user each question and compares the user's input (converted to int) to the answer.

Exercise 9.4
trivia_round Function
Create a function called trivia_round that accepts a player name and then runs a quiz round using a fixed dictionary of three trivia questions. Print the player's name and final score at the end.

Exercise 9.5
Simulated Speed Quiz
Write a function called speed_quiz that loops through a question dictionary but only gives the user 5 seconds to answer each question. (Note: Just simulate the time limit with a message, no timing code required.)

Exercise 9.6
Vocabulary Reflection Quiz
Create a dictionary of vocabulary words where the key is a fancy word and the value is a plain definition. Write a loop that prints each word and asks the user to guess what it means. Don't score it—just for reflection.

Exercise 9.7
Case-Insensitive Quiz
Write a function called all_caps_quiz that takes a question-answer dictionary and accepts user input normally, but converts both the input and correct answer to uppercase before comparing. This makes it case-insensitive.

Exercise 9.8
Error Type Quiz
Create a dictionary of common Python errors (like NameError, TypeError, etc.) and their descriptions. Loop through and ask the user to type in the correct error type based on the description. Score 1 point for each correct match.

Exercise 9.9
Emoji Meaning Quiz
Write a quiz game where the dictionary stores emoji as keys and words like smile or alien as the correct answers. Ask the user what each emoji means. At the end, print how many they got right.

Exercise 9.10
Custom Flashcard Builder
Create a function called custom_flashcards that asks the user to enter three new question-answer pairs (via input) and adds them to a dictionary. Then loop through that dictionary and quiz the user using what they just created.

Answers To Chapter Nine Exercises

Answer 9.1

```
riddles = {
    "What has hands but can’t clap?": "A clock",
    "What has a head and a tail but no body?": "A coin",
    "What gets wetter as it dries?": "A towel"
}

for riddle in riddles:
    print("Riddle:", riddle)
    print("Answer:", riddles[riddle])
```

Answer 9.2

```
def run_quiz(questions):
    score = 0
    for q in questions:
        answer = input(q + " ")
        if answer == questions[q]:
            score += 1
    print("You got", score, "out of", len(questions), "correct.")

quiz = {
    "What’s the capital of France?": "Paris",
    "2 + 2?": "4",
    "What color is the sky?": "blue"
}

run_quiz(quiz)
```

Answer 9.3

```
math_facts = {
    "5 + 3": 8,
    "10 - 6": 4,
    "7 * 2": 14
}

for problem in math_facts:
    user_answer = int(input(problem + " = "))
    if user_answer == math_facts[problem]:
        print("Correct!")
    else:
        print("Wrong.")
```

Answer 9.4

```
def trivia_round(name):
    score = 0
    trivia = {
        "What planet is known as the Red Planet?": "Mars",
        "Who wrote Hamlet?": "Shakespeare",
        "What's the largest ocean?": "Pacific"
    }
    for q in trivia:
        answer = input(q + " ")
        if answer == trivia[q]:
            score += 1
    print(name + ", your score is", score)

trivia_round("Ava")
```

Answer 9.5

```
def speed_quiz():
    questions = {
        "What’s 9 + 10?": "19",
        "What comes after Tuesday?": "Wednesday"
    }
    for q in questions:
        print(q + " (You have 5 seconds to answer!)")
        answer = input("> ")
        if answer == questions[q]:
            print("Correct!")
        else:
            print("Wrong!")

speed_quiz()
```

Answer 9.6

```
vocab = {
    "gregarious": "sociable",
    "obfuscate": "to confuse",
    "benevolent": "kind"
}

for word in vocab:
    input("What does '" + word + "' mean? ")
    print("Definition:", vocab[word])
```

Answer 9.7

```
def all_caps_quiz(qset):
    score = 0
    for q in qset:
        answer = input(q + " ")
        if answer.upper() == qset[q].upper():
            score += 1
    print("Score:", score)

questions = {
    "What’s the opposite of hot?": "cold",
    "What color are bananas?": "yellow"
}

all_caps_quiz(questions)
```

Answer 9.8

```
errors = {
    "Raised when a variable doesn’t exist": "NameError",
    "Raised when you add a number to a string": "TypeError",
    "Raised when you divide by zero": "ZeroDivisionError"
}

score = 0
for desc in errors:
    guess = input(desc + ": ")
    if guess == errors[desc]:
        score += 1

print("Final score:", score)
```

Answer 9.9

```
emoji_quiz = {
    "😊": "smile",
    "🔥": "fire",
    "👽": "alien"
}

score = 0
for e in emoji_quiz:
    answer = input("What does " + e + " mean? ")
    if answer == emoji_quiz[e]:
        score += 1

print("You got", score, "right.")
```

Answer 9.10

```
def custom_flashcards():
    custom = {}
    for i in range(3):
        question = input("Enter a question: ")
        answer = input("Enter the answer: ")
        custom[question] = answer

    for q in custom:
        user = input(q + " ")
        if user == custom[q]:
            print("Correct!")
        else:
            print("Incorrect.")

custom_flashcards()
```

CONCLUSION

You Did It—Seriously

If you’ve made it this far, congratulations! You now know more about Python than most people who say they want to learn to code.

You've worked your way through variables, data types, strings, numbers, and lists.

You learned how to store things, retrieve them, manipulate them, and loop over them.

You built decision logic with if statements, learned the difference between for and while loops, and saw how to let your code repeat itself like a robot with a purpose.

Then you leveled up and learned how to write your own functions—mini-programs that let you reuse logic, clean up your code, and organize your thinking.

You added parameters, experimented with default values, and started building smarter, more flexible code.

That’s not beginner stuff. That’s real development.

And you didn’t stop there.

You picked up dictionaries, one of Python’s most important data structures. You learned how to map keys to values, loop through them, and even use them to build a fully working flashcard game.

That wasn't just a theoretical exercise. That was real software, made by you.

The important thing now isn't just that you finished the book—it's what that means.

You've seen the core building blocks of Python, and more importantly, you've used them. You've written real code and made it do what you want.

That's the whole point.

So don't stop here. Keep building. Make small projects. Break things. Fix them.

This is where programming gets fun, and you've already done the hard part.

You're not just someone who wants to learn Python. You're someone who knows how to write Python.

Now go build something cool!

APPENDIX ONE

Want To Keep Going After the Workbook?

Build 20 Real Python Programs With Me…

From Practice to Production—One Project at a Time

If you make it through this workbook, you'll have already done more than most people who say they want to learn Python.

But here's the truth: knowing *about* Python isn't the same as being able to *use* it. If you really want to get good, you need to build stuff.

That's exactly why I created the **Python Projects** course over at Codemy.com.

In this course, we build **20 real-world projects** together—each one designed to take your skills up a level.

No fluff, no filler, just actual applications that teach you how to *think like a programmer* and use Python to solve real problems.

Here's what we'll build together:

1. Number Guessing Game – Learn loops and conditionals with a classic terminal game.

2. Number Guessing with Tkinter – Upgrade the game with a GUI and get started with app development.

3. Rock Paper Scissors – Use functions and basic logic to outsmart the computer.

4. Ticking Clock – Build a real-time clock using Python’s time module.

5. Palindrome Checker – Practice string slicing and logical thinking.

6. Using Arrow Keys – Make your apps interactive with keyboard event handling.

7. Password Validator – Learn how to validate strong passwords with Python.

8. Password Generator – Automatically create secure, randomized passwords.

9. Hangman Game – Build a fully functional word game with logic and looping.

10. To-Do List App – Create your own productivity app using basic GUI structure.

11. Word Count Tool – Work with text files and count words with file I/O.

12. Tic Tac Toe Game – Master game flow, input handling, and win detection.

13. Draw with Turtle – Use the Turtle graphics module to code creative designs.

14. Image Resizer – Resize and process images for personal or web use.

15. Anagram Solver – Solve and detect anagrams using clever logic and lists.

16. Typing Speed Test – Create a timed typing tool to test speed and accuracy.

17. Flashcard App – Build an interactive study aid with a dictionary backend.

18. Choose Your Own Adventure – Design a branching story with complex logic.

19. File Manager – Organize and manage files with custom-built scripts.

20. File Subprocessor – Automate file reading and processing for real-world tasks.

It's everything you need to go from "I know Python syntax" to "I built 20 real things with Python."

The full course normally sells for $129, but as a thank-you for working through this workbook, you can grab it now for just **$64** here: **https://order.codemy.com/python-projects-64**

Or learn more about what's inside here: **https://codemy.com/python-projects/**

If you liked this workbook, you'll love the course. Hope to see you inside.

—John Elder
Founder, Codemy.com

THE END

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES