Python Crash Course

Complete Study Notes

Eric Matthes

July 14, 2025

Contents

Chapter 1: Getting Started 4
Chapter 2: Variables and Simple Data Types 6
Chapter 3: Introducing Lists 14
Chapter 4: Working with Lists 25
Chapter 5: if Statements 37
Chapter 6: Dictionaries 47
Chapter 7: User Input and while Loops 58
Chapter 8: Functions 64
Chapter 9: Classes 72
Chapter 10: Files and Exceptions 80
Chapter 11: Testing Your Code 88
Chapters 12-14: Alien Invasion Project 94
Chapters 15-17: Data Visualization and APIs Project 105
Chapter End Exercises and Practice Problems 115

Chapter 1: Getting Started

1. Setting Up Your Programming Environment

Definition: Installing Python and a text editor to create your first Python program.
The chapter covers:

- Installing Python on different operating systems (Windows, macOS, Linux)
- Installing Sublime Text editor
- Configuring the development environment
- Understanding Python versions

2. Running Your First Python Program

Definition: Creating and running a simple ”Hello World” program to verify your setup.

Listing 1: Hello World program

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

print("Hello Python world!")
```

3. Understanding What Happens When You Run a Program

Definition: How the Python interpreter processes your code and displays output.
When you run the program:

- The .py extension tells your editor it’s a Python program
- The Python interpreter reads through the program
- It determines what each word means (print is a function)
- It executes the code and displays output
- Your editor uses syntax highlighting to show different parts of code

4. Running Programs from Terminal

Definition: Alternative way to run Python programs using command line.
On Windows:

```
C:\> cd Desktop\python_work
C:\Desktop\python_work> python hello_world.py
Hello Python world!
```

On macOS and Linux:

```
~$ cd Desktop/python_work/
~/Desktop/python_work$ python hello_world.py
Hello Python world!
```

Key Takeaways

- Python is installed on most systems, but you may need to install it
- A text editor like Sublime Text makes programming easier
- The .py extension tells your system it's a Python program
- You can run programs from your editor or from the terminal
- The Python interpreter reads and executes your code
- Syntax highlighting helps you understand your code
- Troubleshooting is a normal part of programming

Chapter 2: Variables and Simple Data Types

1. What Really Happens When You Run hello_world.py

Definition: Understanding how the Python interpreter processes your code and what happens behind the scenes.

When you run a Python program:

- The .py extension indicates it's a Python program
- The Python interpreter reads through the program
- It determines what each word means (print is a function)
- It executes the code and displays output
- Your editor uses syntax highlighting to show different parts of code

2. Variables

Definition: A name that represents a value stored in memory. Variables are used to store and reference data.

Listing 2: Variables and strings

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

message = "hello python world!"
print(message)

message = "hello python Crash Course world!"
print(message)
```

3. Naming and Using Variables

Definition: Rules and guidelines for creating meaningful and valid variable names in Python.

Rules:

- **Use letters, digits, and underscores only**
- **Start with a letter or underscore (not a digit)**
- **Spaces are not allowed, but underscores can separate words**
- **Avoid Python keywords and function names**
- **Variable names should be short but descriptive**
- **Be careful with lowercase l and uppercase O (confused with 1 and 0)**

Examples of Good Variable Names:

```python
message = "Hello Python world!"
message_1 = "Hello Python Crash Course world!"
greeting_message = "Hello!"
```

Examples of Bad Variable Names:

```python
# Don't start with a digit
1_message = "Hello"  # Error!

# Don't use spaces
greeting message = "Hello"  # Error!

# Don't use Python keywords
print = "Hello"  # Avoid this!
```

4. Avoiding Name Errors When Using Variables

Definition: Common mistakes and how to fix them when working with variables.
Common Errors:

- Misspelling variable names
- Forgetting to set a variable's value before using it
- Using inconsistent spelling

Example of a Name Error:

```python
message = "Hello Python Crash Course reader!"
print(mesage)  # NameError: name 'mesage' is not defined
```

5. Variables Are Labels

Definition: Variables are better thought of as labels that you can assign to values, not boxes that store values.

This distinction becomes important as you write more complex programs.

Exercise 2-1: Simple Message Assign a message to a variable, and then print that message.

Listing 3: Exercise 2-1: Simple Message

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# simple_message.py -- print out one message

message = "I love Jung EunBi."

print(message)
```

Exercise 2-2: Simple Messages Assign a message to a variable, and print that message. Then change the value of the variable to a new message, and print the new message.

Listing 4: Exercise 2-2: Simple Messages

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# simple_messages.py -- print out some messages

message = "I love Jung EunBi."

print(message)

message = "Jung EunBi loves me."

print(message)
```

6. Strings

Definition: A series of characters. Anything inside quotes is considered a string in Python.

```python
"This is a string."
'This is also a string.'
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
```

7. Changing Case in a String with Methods

Definition: String methods that modify the case of strings.

Listing 5: String methods

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

name = "ada lovelace"
print(name.title())
# title() is method of the string 'name'
# it changes each word to title case, where each word begins with a
   capital letter

name = "Ada Lovelace"
print(name.upper())
# .upper change lowercase letters into capital letters
print(name.lower())
# .lower change capital letters into lowercase ones
```

8. Using Variables in Strings

Definition: Different ways to include variables in strings.

f-strings (Python 3.6+):

```python
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
```

.format() method:

```python
first_name = "ada"
last_name = "lovelace"
full_name = "{} {}".format(first_name, last_name)
```

9. Adding Whitespace to Strings

Definition: Using tabs and newlines to format strings.

```python
print("Python")
print("\tPython")  # Tab
print("Languages:\nPython\nC\nJavaScript")  # Newlines
```

10. Stripping Whitespace

Definition: Removing extra whitespace from strings.

```python
favorite_language = ' python '
favorite_language.rstrip()  # Remove right whitespace
favorite_language.lstrip()  # Remove left whitespace
favorite_language.strip()   # Remove both sides
```

Exercise 2-3: Personal Message Use a variable to represent a person's name, and print a message to that person.

Listing 6: Exercise 2-3: Personal Message

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# personal_message.py -- print out personal message

name = "Eunbi"
message = "would you marry me?"

print (f"{name}, {message}")
```

Exercise 2-4: Name Cases Use a variable to represent a person's name, and print that person's name in lowercase, uppercase, and title case.

Listing 7: Exercise 2-4: Name Cases

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# name_cases.py -- print out names in lowercase, uppercase and title
    case

name = "jung eunbi"

print(f"Lowercase: {name.lower()}")
print(f"Uppercase: {name.upper()}")
print(f"Title Case: {name.title()}")
```

Exercise 2-6: Famous Quote Find a quote from a famous person you admire. Print the quote and the name of its author.

Listing 8: Exercise 2-6: Famous Quote

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# quote.py -- print out some great persons the his / her quote

person = "Jung Eun Bi"
quote = "As an idol, one hamburger per day is maximum."

print(f"{person} once said, \"{quote}\"")
```

Exercise 2-7: Stripping Names Use a variable to represent a person's name, and include some whitespace characters. Print the name using each of the three stripping functions.

Listing 9: Exercise 2-7: Stripping Names

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# strip.py -- manipulating string with strip functions.

name = " Jung Eun Bi "

name2 = " Jung \n Eun \t Bi "

print("For no \\n and \\t characters:")
print(f"No strip: {name}")
print(f"With lstrip(): {name.lstrip()}")
print(f"With rstrip(): {name.rstrip()}")
print(f"WIth strip(): {name.strip()}")

print("When \\n and \\t characters are included:")
print(f"No strip: {name2}")
print(f"With lstrip(): {name2.lstrip()}")
print(f"With rstrip(): {name2.rstrip()}")
print(f"WIth strip(): {name2.strip()}")
```

f-strings and Formatting:

Listing 10: f-strings and string formatting

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
# this is f-strings (f = format)
# concatenate variables into a string
print(full_name)
print(f"Hello, {full_name.title()}")
message = f"Hello, {full_name.title()}"
print(message)
```

```
full_name = "{} von {}".format(first_name, last_name)
print(full_name)
```

String Formatting with Newlines and Tabs:

Listing 11: String Formatting

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

print("Python")
print("\tPython")
print("Languages:\n\tPython\n\tC\n\tJavascript")
```

String Stripping Methods:

Listing 12: String Stripping

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_language = ' python aa '

print(favourite_language)

print(favourite_language.rstrip())
# rstrip : remove extra whitespacve on the right of a string

print("-----")

print(favourite_language)

favourite_language = favourite_language.rstrip()

print(favourite_language)
# now the string is being modified and assigned back to the value

print("-----")

favourite_language = ' python aa '

print(favourite_language)

print(favourite_language.lstrip())
# lstrip : remove extra whitespacve on the left of a string

print(favourite_language.strip())
# strip : remove extar whitespace on the left and right of a string
```

String Concatenation and Apostrophes:

Listing 13: String Concatenation

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

message = "One of Python\'s strengths is its diverse community."

```

```python
# apostrophe is represented by \'

print(message)
```

11. Numbers

Definition: Working with integers and floats in Python.

Integers:

```python
2 + 3
3 - 2
2 * 3
3 / 2
3 ** 2  # Exponentiation
```

Floats:

```python
0.1 + 0.1
0.2 + 0.1
3 * 0.1
```

Integers and Floats:

```python
3 / 2  # Results in 1.5 (float)
3 // 2 # Results in 1 (integer division)
```

12. Underscores in Numbers

Definition: Using underscores to make large numbers more readable.

```python
universe_age = 14_000_000_000
print(universe_age)  # Prints 14000000000
```

13. Multiple Assignment

Definition: Assigning multiple variables at once.

```python
x, y, z = 0, 0, 0
```

14. Constants

Definition: Variables that are meant to stay the same throughout a program (written in ALL_CAPS).

```python
MAX_CONNECTIONS = 5000
```

Exercise 2-8: Number Eight Write addition, subtraction, multiplication, and division operations that each result in the number 8.

Listing 14: Exercise 2-8: Number Eight

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# eight.py -- print results of four calculations that can result in
  eight

print(7+1)  # integer mix integer gererates integer
print(100/12.5) # integer mix float generates float
print(17.8-9.8)
print(2*4)
```

15. Comments

Definition: Text in code that is ignored by Python but provides information to programmers.

```python
# This is a comment explaining the code
name = "ada"  # This comment is on the same line
```

Exercise 2-11: Zen of Python Enter `import this` into a Python terminal session and skim through the additional principles.

Listing 15: Exercise 2-11: Zen of Python

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# zenofpython.py -- show "Zen of Python"

import this
```

Key Takeaways

- Variables store data that can be reused throughout a program
- Follow Python naming conventions: use snake_case for variables
- Avoid Python keywords and start variable names with letters or underscores
- Strings are the primary way to work with text in Python
- f-strings provide a convenient way to embed variables in text
- String methods like title(), upper(), and lower() modify text case
- Comments help make code readable and maintainable
- Numbers include integers and floats
- Constants are written in ALL_CAPS
- The Python interpreter is strict about syntax and variable names

Chapter 3: Introducing Lists

1. List - Collection of Items

Definition: A collection of items in a particular order, enclosed in square brackets and separated by commas.

Listing 16: Basic list operations

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

bicycles = ['trek', 'cannondale', 'redline',  'specialized']
print(bicycles)

print("---------")
print(bicycles[0])

print("---------")
print(bicycles[0].title())
message = f"My firs bicycle was a {bicycles[0].title()}"
print(message)

print("---------")
print(bicycles[1])
print(bicycles[3])

# -1 item becomes the last item
# the last item can be known without counting the total number of
   items
print(bicycles[-1])
```

Exercise 3-1: Names Store the names of a few of your friends in a list called names. Print each person's name by accessing each element in the list, one at a time.

Listing 17: Exercise 3-1: Names

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# gfriend.py -- list out the name of your friends

print(gfriend[0])
gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

print(gfriend[0])
print(gfriend[1])
print(gfriend[2])
print(gfriend[3])
print(gfriend[4])
print(gfriend[5])
```

Exercise 3-2: Greetings Start with the list you used in Exercise 3-1, but instead of just printing each person's name, print a message to them. The text of each message should be the same, but each message should be personalized with the person's name.

Listing 18: Exercise 3-2: Greetings

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# greetings.py -- say greetings to each of the members

greeting = ", guten Tag!"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(gfriend[0] + greeting)
print(gfriend[1] + greeting)
print(gfriend[2] + greeting)
print(gfriend[3] + greeting)
print(gfriend[4] + greeting)
print(gfriend[5] + greeting)
```

2. Index - Position in List

Definition: The position of an item in a list, starting from 0 for the first item.

```python
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])  # trek
print(bicycles[1])  # cannondale
```

3. Negative Index - Accessing from End

Definition: Using negative numbers to access items from the end of a list (-1 is the last item).

```python
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])  # specialized
print(bicycles[-2])  # redline
```

4. Modifying List Elements

Definition: Changing the value of an item in a list by using its index.

Listing 19: Modifying and manipulating lists

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

motorcycles[0] = "ducati"
print(motorcycles)

print("---------")
motorcycles = ['honda', 'yamaha', 'suzuki']
# appending elements to list, preset to the last position
motorcycles.append('ducati')
```

```python
print(motorcycles)

print("---------")
motorcycles = []
# appending elements to list, one by one
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
# insert can insert to the specific location
motorcycles.insert(0, 'ducati')
print(motorcycles)
print("---------")
# del can delete one of the elements
del motorcycles[0]
print(motorcycles)

print("---------")
# pop : chop out the last item and store that into the last value
poped_motorcycles = motorcycles.pop()
print(motorcycles)
print(poped_motorcycles)

print("---------")
motorcycles = ['honda', 'yamaha', 'suzuki']
last_owned = motorcycles.pop()
print(f"The last motorcycle I owned was a {last_owned.title()}.")

print("---------")
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")

print("---------")
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
# removing item by value
motorcycles.remove('ducati')
print(motorcycles)

print("---------")
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
# removing item by value, same result as pop
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
print(motorcycles)
print(f"\nA {too_expensive.title()} is too expensive for me.")
```

Exercise 3-3: Your Own List Think of your favorite mode of transportation, such as a motorcycle or a car, and make a list that stores several examples. Use your list to print a series of statements about these items, such as ”I would like to own a Honda motorcycle.”

Listing 20: Exercise 3-3: Transportation

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

transportation = ["bus", "bike", "motorcycle", "foot", "van", "train
   "]
brandName = ["Honda", "BMW", "Toyota"]

message = "I go to school by"

print(message + " " + brandName[0] + " " + transportation[0] + ".")
```

5. append() Method - Adding to End

Definition: A method that adds an item to the end of a list.

```python
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.append('ducati')
print(motorcycles)  # ['honda', 'yamaha', 'suzuki', 'ducati']
```

6. insert() Method - Adding at Position

Definition: A method that adds an item at a specific position in a list.

```python
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)  # ['ducati', 'honda', 'yamaha', 'suzuki']
```

7. del Statement - Removing by Index

Definition: A statement that removes an item from a list using its index.

```python
motorcycles = ['honda', 'yamaha', 'suzuki']
del motorcycles[0]
print(motorcycles)  # ['yamaha', 'suzuki']
```

8. pop() Method - Removing and Returning

Definition: A method that removes the last item from a list and returns it.

```python
motorcycles = ['honda', 'yamaha', 'suzuki']
popped_motorcycle = motorcycles.pop()
print(popped_motorcycle)  # suzuki
print(motorcycles)  # ['honda', 'yamaha']
```

9. remove() Method - Removing by Value

Definition: A method that removes an item from a list by its value.

```python
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
motorcycles.remove('ducati')
print(motorcycles)  # ['honda', 'yamaha', 'suzuki']
```

Exercise 3-4: Guest List If you could invite anyone, living or deceased, to dinner, who would you invite? Make a list that includes at least three people you'd like to invite to dinner. Then use your list to print a message to each person, inviting them to dinner.

Listing 21: Exercise 3-4: Guest List

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# dinner.py -- invite members to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")
```

Exercise 3-5: Changing Guest List You just heard that one of your guests can't make the dinner, so you need to send out a new set of invitations. You'll have to think of someone else to invite.

Listing 22: Exercise 3-5: Changing Guest List

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'
print(f"Current list: {gfriend}")
```

Exercise 3-6: More Guests You just found a bigger dinner table, so now more space is available. Think of three more guests to invite to dinner.

Listing 23: Exercise 3-6: More Guests

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'

print("\n---")
print("and Sinb will bring WJSN come.")
gfriend.append("WJSN")
print(f"Current list: {gfriend}")

print("also, Eunha will bring another SinB to the dinner.\nThe two
    SinBs need to sit together.")
gfriend.insert(4, "Sinb")
print(f"Current list: {gfriend}")
```

Exercise 3-7: Shrinking Guest List You just found out that your new dinner table won't arrive in time for the dinner, and you have space for only two guests.

Listing 24: Exercise 3-7: Shrinking Guest List

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'

print("\n---")
print("and Sinb will bring WJSN come.")
gfriend.append("WJSN")
print(f"Current list: {gfriend}")

print("also, Eunha will bring another SinB to the dinner.\nThe two
    SinBs need to sit together.")
gfriend.insert(4, "Sinb")
print(f"Current list: {gfriend}")

print("\n---")
print("Now one SinB kicks another out.")
del gfriend[4]
print(f"Current list{gfriend}")

print("\n---")
print("Eunha is being dissed. She is sad and she left for crying.")
gfriend.remove("eunha")
print(f"Current list:{gfriend}")

print("\n---")
print(f"{gfriend.pop(0)} goes to comfort Eunha.")
print(f"Current list: {gfriend}")
```

10. Empty List - Starting Fresh

Definition: A list with no items, created using empty square brackets.

```python
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
print(motorcycles)  # ['honda', 'yamaha']
```

Exercise 3-8: Seeing the World Think of at least five places in the world you'd like to visit.

Listing 25: Exercise 3-8: Seeing the World

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

countries = ["Deutsch", "Japan", "Great Britain", "Taiwan"]

print(f"Countries I want to go: {countries}.")

countries_sorted = countries
countries_sorted.sort()
print(f"Countries I want to go: {countries_sorted}.")
countries_sorted_reverse = countries
countries_sorted_reverse.sort(reverse=True)
```

```python
print(f"Countries I want to go: {countries_sorted_reverse}.")
print(f"Countries I want to go: {sorted(countries)}.")
countries_reversed = countries
countries_reversed.reverse()
print(f"Countries I want to go: {countries_reversed}.")
```

Exercise 3-9: Dinner Guests Working with one of the programs from Exercises 3-4 through 3-7 (pages 46-47), use len() to print a message indicating the number of people you are inviting to dinner.

Listing 26: Exercise 3-9: Dinner Guests

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
   invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}, {len(gfriend)} dinner mates.")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'

print("\n---")
print("and Sinb will bring WJSN come.")
gfriend.append("WJSN")
print(f"Current list: {gfriend}, {len(gfriend)} dinner mates.")

print("also, Eunha will bring another SinB to the dinner.\nThe two
   SinBs need to sit together.")
gfriend.insert(4, "Sinb")
print(f"Current list: {gfriend}, {len(gfriend)} dinner mates.")

print("\n---")
print("Now one SinB kicks another out.")
del gfriend[4]
print(f"Current list{gfriend}, {len(gfriend)} dinner mates.")

print("\n---")
print("Eunha is being dissed. She is sad and she left for crying.")
gfriend.remove("eunha")
print(f"Current list:{gfriend}, {len(gfriend)} dinner mates.")

print("\n---")
print(f"{gfriend.pop(0)} goes to comfort Eunha.")
print(f"Current list: {gfriend}, {len(gfriend)} dinner mates.")
```

Key Takeaways

- **Lists are ordered collections** - Items maintain their position in the list
- **Indexing starts at 0** - First item is at index 0, second at index 1, etc.
- **Negative indices** - Use -1 for last item, -2 for second-to-last, etc.
- **Lists are mutable** - You can change, add, and remove items after creation
- **append() vs insert()** - append() adds to end, insert() adds at specific position
- **del vs pop() vs remove()** - Three different ways to remove items:
  - **del** - Removes by index, doesn't return value
  - **pop()** - Removes by index, returns the removed value
  - **remove()** - Removes by value (first occurrence only)
- **Empty lists** - Start with empty brackets [] and build up
- **len() function** - Counts items in a list, useful for loops and conditionals
- **String formatting with lists** - Use f-strings with list items: f"list[0]"
- **String methods on list items** - Apply string methods to list elements: list[0].title()
- **Variable assignment with pop()** - Store returned value: item = list.pop()
- **remove() with variables** - Remove items stored in variables: list.remove(variable)
- **Common errors to avoid**:
  - Accessing index that doesn't exist (IndexError)
  - Removing item that doesn't exist (ValueError)
  - Forgetting that indexing starts at 0
  - Using remove() on item not in list
- **List methods modify the original list** - They don't create a new list
- **You can store any data type** - Strings, numbers, other lists, etc.
- **List operations in practice**:
  - Building lists dynamically with append()
  - Modifying lists based on user input
  - Using len() to check list size
  - Combining string formatting with list access

Chapter 4: Working with Lists

1. for Loop - Iterating Through Lists

Definition: A loop that runs once for each item in a list or other collection.

Listing 27: Basic for loop

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(magician)

for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()
       }.\n")

print("Thank you, everyone. That was a great magic show!")
```

Exercise 4-1: Pizzas Think of at least three kinds of your favorite pizza. Store these pizza names in a list, and then use a for loop to print the name of each pizza.

Listing 28: Exercise 4-1: Pizzas

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pizzas = ['peccato', 'diavola', 'capricciosa']

for pizza in pizzas:
    print(f"I like {pizza.title()}")

print("The above statements are fake.")
```

Exercise 4-2: Animals Think of at least three different animals that have a common characteristic. Store the names of these animals in a list, and then use a for loop to print out the name of each animal.

Listing 29: Exercise 4-2: Animals

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

animals = ['cats', 'dogs', 'lions']

for animal in animals:
    print(f"{animal.title()} have four legs.")

print("Any of them can be a great pet.")
```

2. Loop Variable - Current Item

Definition: The variable that holds the current item being processed in a loop.

```python
for magician in magicians:
    print(magician)  # magician is the loop variable
```

3. Indentation - Code Blocking

Definition: The use of spaces or tabs to indicate which lines of code belong together in a block.

```python
for magician in magicians:
    print(magician)  # This line is indented
    print("Great trick!")  # This line is also indented
print("Thank you!")  # This line is not indented
```

4. range() Function - Number Sequences

Definition: A function that generates a sequence of numbers for use in loops.

Listing 30: Using range()

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

for value in range(1, 5):
    print(value)


numbers = list(range(1,6))
print(numbers)
```

Even Numbers:

Listing 31: Chapter04/403_even_numbers.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

even_numbers = list(range(2, 11, 2))
print(even_numbers)
# range: from 2 to 11, adds 2 repeatedly
```

Exercise 4-3: Counting to Twenty Use a for loop to print the numbers from 1 to 20, inclusive.

Listing 32: Exercise 4-3: Counting to Twenty

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,21))

for number in numbers:
    print(f"{number}")
```

Exercise 4-4: One Million Make a list of the numbers from one to one million, and then use a for loop to print the numbers.

Listing 33: Exercise 4-4: One Million

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,1000001))

for number in numbers:
    print(f"{number}")
```

Exercise 4-5: Summing a Million Make a list of the numbers from one to one million, and then use min() and max() to make sure your list actually starts at one and ends at one million.

Listing 34: Exercise 4-5: Summing a Million

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,1000001))

print(f"Max : {max(numbers)}")
print(f"Min : {min(numbers)}")
print(f"Sum : {sum(numbers)}")
```

Exercise 4-6: Odd Numbers Use the third argument of the range() function to make a list of the odd numbers from 1 to 20. Use a for loop to print each number.

Listing 35: Exercise 4-6: Odd Numbers

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

even_numbers = list(range(2,21,2))

for number in even_numbers:
    print(f"{number}")
```

Exercise 4-7: Threes Make a list of the multiples of 3 from 3 to 30. Use a for loop to print the numbers in your list.

Listing 36: Exercise 4-7: Threes

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

threes_numbers = list(range(3, 31 ,3))

for number in threes_numbers:
    print(f"{number}")
```

5. List Comprehension - Compact Lists

Definition: A way to create lists using a compact syntax with loops and conditions.

Listing 37: List comprehensions

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

squares = []
```

```python
for value in range(1, 11):
    square = value ** 2
    squares.append(square)
    # or squares.append(value**2)

print(squares)

squares = [value ** 2 for value in range(1, 22)]
print(squares)
```

Exercise 4-8: Cubes A number raised to the third power is called a cube. For example, the cube of 2 is written as 2**3 in Python. Make a list of the first 10 cubes (that is, the cube of each integer from 1 through 10), and use a for loop to print out the value of each cube.

Listing 38: Exercise 4-8: Cubes

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cube = []

for number in list(range(1, 11)):
    cube.append(number**3)

for member in cube:
    print(f"{member}")
```

Exercise 4-9: Cube Comprehension Use a list comprehension to generate a list of the first 10 cubes.

Listing 39: Exercise 4-9: Cube Comprehension

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cube = [value ** 3 for value in range(1,11)]

for member in cube:
    print(f"{member}")
```

6. Slicing - List Portions

Definition: A way to work with a portion of a list by specifying start and end indices.

```python
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])  # ['charles', 'martina', 'michael']
print(players[1:4])  # ['martina', 'michael', 'florence']
print(players[:4])   # ['charles', 'martina', 'michael', 'florence']
print(players[2:])   # ['michael', 'florence', 'eli']
```

Listing 40: Working with players list

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

players = ['charles', 'martina', 'michael', 'florence', 'eli']
```

```python
print(players[0:3])

print(players[1:4])

# from starr to 4
print(players[:4])

# start from 2
print(players[2:])

# last three players
print(players[-3:])

print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())
```

Exercise 4-10: Slices Using one of the programs you wrote in this chapter, add several lines to the end of the program that do the following:

Listing 41: Exercise 4-10: Slices

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

animals = ['cats', 'dogs', 'lions']

for animal in animals:
    print(f"{animal.title()} have four legs.")

print("Any of them can be a great pet.")

print("\n----")
animals.append('elephant')
print(f"Adding {animals[-1]}.\nNow we have the following animals:")
for animal in animals:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the first three animals:")
for animal in animals[:3]:
    print(f"{animal.title()}")

print("\n----")
animals.append('sharks')
print(f"Adding {animals[-1]}.\nNow we have the following animals:")
for animal in animals:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the middle three animals:")
for animal in animals[(int)(len(animals)/2-1):(int)(len(animals)
    /2+2)]:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the last three animals:")
for animal in animals[-3:]:
    print(f"{animal.title()}")
```

7. Copying Lists - Creating Duplicates

Definition: Creating a copy of a list to avoid modifying the original.

```python
my\_foods = ['pizza', 'falafel', 'carrot cake']
friend\_foods = my\_foods[:]  # Create a copy
```

Listing 42: Working with food lists

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

my_foods = ['pizza', 'falafel', 'carrpt cake']
friend_foods = my_foods[:]
# all of the items in the list
# i.e. the list can then be copied

print("My favouried foods are :")
print(my_foods)

print("My friends's favouried foods are :")
print(friend_foods)

print("---------")

my_foods.append('cannoli')
friend_foods.append('ice cream')

print("My favouried foods are :")
print(my_foods)

print("My friends's favouried foods are :")
print(friend_foods)

print("---------")
```

Exercise 4-11: My Pizzas, Your Pizzas Start with your program from Exercise 4-1. Make a copy of the list of pizzas, and call it friend_pizzas. Then, do the following:

Listing 43: Exercise 4-11: My Pizzas, Your Pizzas

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pizzas = ['peccato', 'diavola', 'capricciosa']

for pizza in pizzas:
    print(f"I like {pizza.title()}.")
print("The above statements are fake.")
```

```python

friend_pizzas = pizzas[:]

print("\n----")
print("Another pizza list as per below:")
for pizza in friend_pizzas:
    print(f"{pizza.title()}")

friend_pizzas.append('Clam pie')
print("\n----")
print(f"Adding {friend_pizzas[-1]}\nThe pizza list:")
for pizza in friend_pizzas:
    print(f"{pizza.title()}")
```

Exercise 4-12: More Loops All versions of foods.py in this section have avoided using for loops when printing to save space. Choose a version of foods.py, and write two for loops to print each list of foods.

Listing 44: Exercise 4-12: More Loops

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

my_foods = ['pizza', 'falafel', 'carrpt cake']
friend_foods = my_foods[:]

print("My favouried foods are :")
for food in my_foods:
    print(f"{food.title()}")

print("My friends's favouried foods are :")
for food in friend_foods:
    print(f"{food.title()}")

print("\n---------")
print("Adding one food for each of mine and friend's list:\n")
my_foods.append('cannoli')
friend_foods.append('ice cream')

print("Now, my favouried foods are :")
for food in my_foods:
    print(f"{food.title()}")

print("My friends's favouried foods are :")
for food in friend_foods:
    print(f"{food.title()}")
```

8. Tuples - Immutable Lists

Definition: An immutable list that cannot be modified after creation, using parentheses instead of square brackets.

```python
dimensions = (200, 50)
```

```python
print(dimensions[0])  # 200
print(dimensions[1])  # 50
# dimensions[0] = 250  # This would cause an error
```

Listing 45: Working with dimensions

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

dimensions = (300, 50)
print(dimensions[0])
print(dimensions[1])

# nothing can modify Tuple, i.e. assign the value of it

for dimension in dimensions :
    print(dimension)

# but we can re-define the Tuple

dimensions = (1000,20)

for dimension in dimensions :
    print(dimension)
```

Exercise 4-13: Buffet A restaurant offers a simple buffet with five basic foods. Think of five simple foods, and store them in a tuple.

Listing 46: Exercise 4-13: Buffet

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

foods = ('pizza', 'falafel', 'carrpt cake', 'sushi', 'ice cream')

print("Food in a buffet:")
for food in foods:
    print(f"{food.title()}")

print("\n---")
print("Now there is a new menu:")
foods = ('fried rice', 'onigiri', 'carrpt cake', 'sushi', 'ice cream
    ')

print("Food in a buffet:")
for food in foods:
    print(f"{food.title()}")
```

Key Takeaways

- **for loops** iterate through each item in a list automatically
- **Proper indentation** is crucial for defining loop blocks

- **range()** generates sequences of numbers for loops
- **list()** function converts range() objects to lists
- **min() and max()** functions find minimum and maximum values in lists
- **List comprehensions** create lists efficiently in one line
- **Slicing** allows you to work with portions of lists using [start:end]
- **Copying lists** with [:] prevents modifying the original list
- **Loop variables** hold the current item being processed
- **String formatting in loops** - Use f-strings with loop variables
- **Exponentiation** - Use ** operator for powers (e.g., 2**3 for cubes)
- **Tuples** - Immutable sequences using parentheses
  - **Creation**: Use parentheses instead of brackets - foods = ('pizza', 'falafel')
  - **Immutability**: Cannot modify individual elements after creation
  - **Iteration**: Can use for loops with tuples just like lists
  - **Redefinition**: Can reassign entire tuple to new values
  - **Use cases**: Store data that shouldn't change (like buffet menus)
- **Common patterns**:
  - Using range() with for loops for counting
  - List comprehensions for creating number sequences
  - Slicing for accessing portions of lists
  - Copying lists to avoid side effects
  - Using min() and max() to verify list contents
  - Using tuples for immutable data collections
- **Important concepts**:
  - Indentation defines code blocks in Python
  - Loop variables can be used in string formatting
  - range() can take start, stop, and step arguments
  - List comprehensions are more efficient than building lists in loops
  - Tuples are immutable and use parentheses
  - Slicing syntax: [start:end:step]
  - Tuples can be iterated with for loops
  - Entire tuples can be reassigned but individual elements cannot
- **Methods and functions covered**:

- **range(start, stop, step)** - Generate number sequences
- **list(range())** - Convert range to list
- **min(list)** - Find minimum value
- **max(list)** - Find maximum value
- **list**[**start:end**] - Slice lists
- **list**[**:**] - Copy entire list
- [**expression for item in iterable**] - List comprehension
- **tuple()** - Create immutable sequences
- **tuple iteration** - for loops with tuples

Chapter 5: if Statements

1. if Statement - Conditional Execution

Definition: A statement that allows you to examine the current state of a program and respond appropriately.

Listing 47: Basic if statements

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cars = ['audi', 'bmw', 'subaru', 'toyota']

for car in cars:
    if car == 'bmw':
        print(car.upper())
    elif car == 'audi':
        print(car.lower())
    else:
        print(car.title())
```

Exercise 5-1: Conditional Tests Write a series of conditional tests. Print a statement describing each test and your prediction for the results of each test.

Listing 48: Exercise 5-1: Conditional Tests

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

car = 'subaru'
print("Is car == 'subaru'? I predict it is True.")
print(car == 'subaru')

print("Is car == 'audi'? I predict it is False.")
print(car == 'audi')
```

Exercise 5-2: More Conditional Tests You don't have to limit the number of tests you create to 10. If you want to try more comparisons, write more tests and add them to conditional_tests.py.

Listing 49: Exercise 5-2: More Conditional Tests

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# guess_number.py
# system randomally define a number between 1 and 100
# let user guess the number
# if user's guess is out of range, warning will be issued (only
    three out-of-range guess allowed)
# if user's guess is in the range but not matching the answer, user
    need to guess again
# if the guess is correct, exit the program

import random

```

```python
number = random.randint(1, 100)
out_of_range_chance = 3
guessRange = list(range(1, 101))

while True:
    guess = int(input(f"Input an integer between {guessRange[0]} and
        {guessRange[-1]}: "))
    if (guess > guessRange[-1]) or (guess < guessRange[0]):
        out_of_range_chance = out_of_range_chance - 1
        if out_of_range_chance > 0:
            print("Your guess is out of the range of available
               gueese. Try again!")
            continue
        else:
            print("There are too many out of range guesses. Get out
               of the game!")
            break
    elif guess == number:
        print("Congradulations! You have got a correct guess.")
        break
    else:
        if guess > number:
            print("Your guess is too large. Please try again.")
            guessRange = guessRange[:guessRange.index(guess)]
            continue
        else:
            print("Your guess is too small. Please try again.")
            guessRange = guessRange[guessRange.index(guess+1):]
            continue
```

2. Conditional Test - True/False Check

Definition: An expression that can be evaluated as True or False, used to decide whether code should be executed.

```python
car = 'bmw'
car == 'bmw'   # True
car == 'audi'   # False
```

3. Equality Operator - ==

Definition: An operator that checks if two values are equal, returning True or False.

```python
answer = 42
if answer == 42:
    print("Correct!")
```

4. Inequality Operator - !=

Definition: An operator that checks if two values are not equal, returning True or False.

Listing 50: Inequality testing

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

answer = 172

if answer != 42:
    print("That is not the correct answer. Please try again!")
```

Exercise 5-3: Alien Colors Imagine an alien was just shot down in a game. Create a variable called alien_color and assign it a value of 'green', 'yellow', or 'red'.

Listing 51: Exercise 5-3: Alien Colors

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']

points = 0

print(f"Now you have {points} points.\n")

guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct. You get five
       points!\n")
    points += 5
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

Exercise 5-4: Alien Colors 2 Choose a color for an alien as you did in Exercise 5-3, and write an if-else chain.

Listing 52: Exercise 5-4: Alien Colors 2

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']
awards = [5, 10, 10]

points = 0

print(f"Now you have {points} points.\n")

guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct.\n")
    award = awards[alien_car.index(guess)]
    print(f"You get {award} points!\n")
    points += award
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

Exercise 5-5: Alien Colors 3 Turn your if-else chain from Exercise 5-4 into an if-elif-else chain.

Listing 53: Exercise 5-5: Alien Colors 3

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']
awards = [5, 10, 15]

points = 0

print(f"Now you have {points} points.\n")

guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct.\n")
    award = awards[alien_car.index(guess)]
    print(f"You get {award} points!\n")
    points += award
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

5. elif Statement - Multiple Conditions

Definition: A statement that allows you to check multiple conditions when the first if statement is False.

```python
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 5
else:
    price = 10
```

6. else Statement - Default Action

Definition: A statement that provides a default action when all previous conditions are False.

```python
if age < 4:
    price = 0
else:
    price = 10
```

Exercise 5-6: Stages of Life Write an if-elif-else chain that determines a person's stage of life.

Listing 54: Exercise 5-6: Stages of Life

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

age = input("Input your age: ")

if age < 2:
    print("Baby")
elif age < 4:
    print("Toddler")
elif age < 13:
    print("Kid")
elif age < 20:
    print("Teenager")
elif age < 65:
    print("Adult")
else:
    print("Elder")
```

Exercise 5-7: Favorite Fruit Make a list of your favorite fruits, and then write a series of independent if statements that check for certain fruits in your list.

Listing 55: Exercise 5-7: Favorite Fruit

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

fruits = ['banana', 'orange', 'fdf', 'apple']

fruit = 'banana'

if fruit in fruits:
    print("I like bananas")
else:
    print(f"{fruit.title()}")
```

7. in Operator - Membership Test

Definition: An operator that checks if a value exists in a list or other collection.

Listing 56: Membership testing

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

requested_toppings = ['mushrooms', 'extra cheese']

if requested_toppings != 'anchovies':
    print("Hold the anchovies!")

print("---------------")

if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")

print("---------------")

requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
    print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")

print("---------------")

requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")

print("---------------")

requested_toppings = []

if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")
else:
    print("Are you sure you want a plain pizza?")

print("---------------")

available_toppings = ['mushrooms', 'olives', 'green peppers', 'pepperoni', 'pineapple', 'extra cheese']

requested_toppings = ['mushroom', 'french fries', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")

print("\nFinished making your pizza!")
```

Exercise 5-8: Hello Admin Make a list of five or more usernames, including the name 'admin'.

Listing 57: Exercise 5-8: Hello Admin

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

for user in users:
    if user in admin:
        print(f"Hello admin {user}, would you like to see a status
           report?")
    else:
        print(f"Hello {user}, thank you for logging in again.")
```

Exercise 5-9: No Users Add an if test to hello_admin.py to make sure the list of users is not empty.

Listing 58: Exercise 5-9: No Users

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

if users:
    for user in users:
        if user in admin:
            print(f"Hello admin {user}, would you like to see a
               status report?")
        else:
            print(f"Hello {user}, thank you for logging in again.")
else:
    print("There is no user.")
```

Exercise 5-10: Checking Usernames Do the following to create a program that simulates how websites ensure that everyone has a unique username.

Listing 59: Exercise 5-10: Checking Usernames

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
walk_in_users = ['eunso', 'yeorum', 'IU', 'sana', 'eunha', 'sinb']
walk_in_users_updated = []

for walk_in_user in walk_in_users:
    walk_in_users_updated.append(walk_in_user.lower())
```

```python

if users:
    for walk_in_user in walk_in_users_updated:
        if walk_in_user in users:
            print(f"{walk_in_user.title()} name is already in user
               list. Please use another name.")
        else:
            users.append(walk_in_user)
            print(f"{walk_in_user.title()} has been newly registered
               .")
else:
    print("There is no registered user.")
```

8. Boolean Values - True/False

Definition: Values that represent the truth or falsity of a condition.

```python
game_active = True
can_edit = False
```

9. and Operator - Multiple Conditions

Definition: An operator that returns True only if all conditions are True.

```python
age_0 = 22
age_1 = 18
age_0 >= 21 and age_1 >= 21  # False
```

10. or Operator - Alternative Conditions

Definition: An operator that returns True if any condition is True.

```python
age_0 = 22
age_1 = 18
age_0 >= 21 or age_1 >= 21  # True
```

11. not Operator - Negation

Definition: An operator that negates a condition, returning the opposite boolean value.

```python
banned_users = ['andrew', 'carolina', 'david']
user = 'marie'
if user not in banned_users:
    print(f"{user.title()}, you can post a response if you wish.")
```

Exercise 5-11: Ordinal Numbers Ordinal numbers indicate their position in a list, such as 1st or 2nd. Most ordinal numbers end in th, except 1, 2, and 3.

Listing 60: Exercise 5-11: Ordinal Numbers

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes
```

```python

numbers = list(range(1,10))

for number in numbers:
    if number == 1:
        print(f"{number}st")
    elif number == 2:
        print(f"{number}nd")
    elif number == 3:
        print(f"{number}rd")
    else:
        print(f"{number}th")
```

Key Takeaways

- **if statements** allow programs to make decisions based on conditions
- **Conditional tests** evaluate to True or False and control program flow
- **Equality operators**: == (equal), != (not equal)
- **Comparison operators**: <, >, <=, >= for numerical comparisons
- **elif statements** provide additional conditions when if is False
- **else statements** provide default actions when all conditions are False
- **in operator** tests membership in lists, strings, and other collections
- **not in operator** tests for absence in collections
- **Boolean operators**:
  - **and** - True only if ALL conditions are True
  - **or** - True if ANY condition is True
  - **not** - Negates a condition (True becomes False, vice versa)
- **Boolean values** are True and False (capitalized in Python)
- **Indentation** is crucial for defining code blocks in if statements
- **Complex conditions** can combine multiple operators and parentheses
- **String comparisons** are case-sensitive by default
- **Empty lists** evaluate to False in boolean contexts
- **Common patterns**:
  - Checking for specific values with ==
  - Testing ranges with <, >, <=, >=

  - Validating membership with in/not in
  - Combining conditions with and/or/not
  - Using if-elif-else for multiple exclusive conditions

- **Best practices**:
  - Use descriptive variable names for clarity
  - Keep conditions simple and readable
  - Use parentheses to clarify operator precedence
  - Test edge cases and boundary conditions
  - Use meaningful boolean variable names (e.g., is_active, can_vote)

Chapter 6: Dictionaries

1. Dictionary - Key-Value Pairs

Definition: A collection of key-value pairs that allows you to connect pieces of related information.

Listing 61: Basic dictionary operations

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("-----------")

alien_0 = {
        'color': 'green',
        'points' : 5
        }
print(alien_0['color'])
print(alien_0['points'])

print_H()

new_pionts = alien_0['points']

print(f"You just earned {new_pionts} points!")

print_H()

print(alien_0)

alien_0['x_position'] = 0
alien_0['y_position'] = 25

print(alien_0)

print_H()

alien_0 = {}

alien_0['color'] = 'green'
alien_0['points'] = 5

print(alien_0)

print_H()

print(f"The alien is {alien_0['color']}")
alien_0['color'] = 'Yellow'

print(f"The alien is now {alien_0['color']}")

```

```
print_H()

alien_0 = {
        'x_position' : 0,
        'y_position' : 23,
        'speed' : 'medium'
        }
print(f"Original positon: {alien_0['x_position']}")

if alien_0['speed'] == 'slow':
    x_increment = 1
elif alien_0['speed'] == 'medium':
    x_increment = 2
else:
    x_increment = 3

alien_0['x_position'] = alien_0['x_position'] + x_increment

print(f"New position : {alien_0['x_position']}")

print_H()

alien_0 = {
        'color' : 'green',
        'points' : 5
        }
print(alien_0)

del alien_0['points']
print(alien_0)
```

Exercise 6.1 - Person Information:

Listing 62: Exercise 6.1: Creating a person dictionary

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

person = {
    'first_name' : 'eunbi',
    'last_name' : 'jung',
    'age' : 26,
    'city' : 'seoul'
}

print(f"I am going to talk about my wife:\nHer name is {person['
   last_name'].title() + ' ' + person['first_name'].title()}.\nShe
   is {person['age']} years old.\nShe is living in {person['city'].
   title()}.")
```

2. Key-Value Pair - Dictionary Element

Definition: A set of values associated with each other, where a key is used to access its associated value.

```python
alien_0 = {'color': 'green', 'points': 5}
```

Exercise 6.2 - Favorite Numbers:

Listing 63: Exercise 6.2: Storing favorite numbers

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_number = {
    'sowon' : 1,
    'yerin' : 2,
    'eunha' : 3,
    'yuju' : 4,
    'sinb' : 5,
    'umji' : 6
}

print(f"Sowon's favourite number is {favourite_number['sowon']}.")
print(f"Yerin's favourite number is {favourite_number['yerin']}.")
print(f"Eunha's favourite number is {favourite_number['eunha']}.")
print(f"Yuju's favourite number is {favourite_number['yuju']}.")
print(f"Sinb's favourite number is {favourite_number['sinb']}.")
print(f"Umji's favourite number is {favourite_number['umji']}.")
```

3. Accessing Values - Dictionary Lookup

Definition: The process of retrieving a value from a dictionary using its key.

```python
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])  # 'green'
```

Exercise 6.3 - Glossary:

Listing 64: Exercise 6.3: Accessing dictionary values

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

glossary = {
    'die Adresse' : "address",
    'die Webseite' : "website"
}

print(f"'die Adresse' means {glossary['die Adresse'].title()}.")
print(f"'die Webseite' means {glossary['die Webseite'].title()}.")
```

4. Adding Key-Value Pairs - Dictionary Modification

Definition: The process of adding new key-value pairs to an existing dictionary.

```python
alien_0 = {'color': 'green', 'points': 5}
alien_0['x_position'] = 0
alien_0['y_position'] = 25
```

5. Starting with Empty Dictionary - Dynamic Creation

Definition: Creating a dictionary with no key-value pairs and adding them as needed.

```python
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
```

6. Modifying Values - Dictionary Updates

Definition: Changing the value associated with a key in a dictionary.

```python
alien_0 = {'color': 'green', 'points': 5}
alien_0['color'] = 'yellow'
```

7. Removing Key-Value Pairs - del Statement

Definition: Permanently removing a key-value pair from a dictionary using the del statement.

```python
alien_0 = {'color': 'green', 'points': 5}
del alien_0['points']
```

8. Looping Through Dictionary - items() Method

Definition: Iterating through all key-value pairs in a dictionary.

Listing 65: Looping through dictionaries

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("-------------------")

favourite_languages = {
        'jen' : 'python',
        'sarah' : 'c',
        'edward' : 'ruby',
        'phil' : 'python'
        }

language = favourite_languages['sarah'].title()
print(f"Sarah's favourite language is {language}.")

print_H()

```

```python
for name, language in favourite_languages.items():
    print(f"{name.title()}'s favourite language is {language.title()
       }")

print_H()

for name in favourite_languages.keys():
    print(name.title())
# loopint through the keys is actually the default bahaviour when
   looping through a dictionary
# the .keys() can be omitted

print_H()

friends = ['phil', 'sarah']

for name in favourite_languages.keys():
    print(f"Hi {name.title()}.")

    if name in friends:
        language = favourite_languages[name].title()
        print(f"\t{name.title()}, I see you love {language}!")

print_H()

if 'erin' not in favourite_languages.keys():
    print("Erin, please take our poll!")

print_H()

for name in sorted(favourite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")

print_H()

print("The folloing langauges have been mentiond:")
for language in set(favourite_languages.values()):
    print(language.title())

print_H()

favourite_languages = {
        'jen' : ['python', 'ruby'],
        'sarah' : ['c'],
        'edward' : ['ruby', 'go'],
        'phil' : ['python', 'haskell']
        }

for name, languages in favourite_languages.items():
    print(f"\n{name.title()}'s favourite languages are:")
    for language in languages:
        print(f"\t{language.title()}")
```

Exercise 6.4 - Glossary with Looping:

Listing 66: Exercise 6.4: Looping through dictionary items

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

glossary = {
    'die Adresse' : "address",
    'die Webseite' : "website",
        'können' :  "can",
    'Velen Dank' : "Very thnks"
}

for word, meaning in glossary.items():
    print(f"{word.title()} means {meaning.title()}.")
```

Exercise 6.5 - Rivers:

Listing 67: Exercise 6.5: Looping through rivers and countries

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

rivers = {
    'nile' : "egypt",
        'eunha' : "Jung Eun Bi",
    'komogawa' : "japan"
}
countRiver = 0
countCountry = 0

for river, country in rivers.items():
    print(f"The {river.title()} runs through {country.title()}.")

for river in rivers.keys():
    countRiver += 1
    print(f"River {countRiver}: {river.title()}")

for country in rivers.values():
    countCountry += 1
    print(f"Country {countCountry}: {country.title()}")
```

9. Looping Through Keys - keys() Method

Definition: Iterating through all keys in a dictionary.

```python
favourite_languages = {'jen': 'python', 'sarah': 'c'}
for name in favourite_languages.keys():
    print(name.title())
```

10. Looping Through Values - values() Method

Definition: Iterating through all values in a dictionary.

```python
favourite_languages = {'jen': 'python', 'sarah': 'c'}
for language in favourite_languages.values():
    print(language.title())
```

Exercise 6.6 - Favorite Languages with Conditional Logic:

Listing 68: Exercise 6.6: Checking if keys exist in dictionary

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_languages = {
    'jen' : 'python',
        'sarah' : 'c',
    'edward' : 'ruby',
    'phil' : 'python'
}

for name, language in favourite_languages.items():
    print(f"{name.title()}s favourite language is {language.title()
       }.")

print("\n-----\n")

people = ['david', 'stefan', 'modric', 'sarah', 'phil']

for person in people:
    if person in favourite_languages:
        print(f"{person.title()}, thank you for the poll.\nYour
           favourite language is {favourite_languages.get(person).
           title()}.")
    else:
        print(f"{person.title()}, please take the poll.")
```

11. Nesting - Dictionaries in Dictionaries

Definition: Storing multiple dictionaries in a list, or a list of items as a value in a dictionary.

```python
aliens = []
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
```

Exercise 6.7 - Multiple People:

Listing 69: Exercise 6.7: List of dictionaries

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

members = []
```

```python

countMember = 0

for count in range(6):
    person = {
        'nick_name' : 'eunha',
        'first_name' : 'eunbi',
        'last_name' : 'jung',
        'age' : 26,
        'city' : 'seoul'
    }
    members.append(person)

members[0]['first_name'] = 'sojung'
members[0]['last_name'] = 'kim'
members[0]['age'] = 27
members[0]['nick_name'] = 'sowon'

members[1]['first_name'] = 'yerin'
members[1]['last_name'] = 'jung'
members[1]['nick_name'] = 'yerin'

members[3]['first_name'] = 'yuna'
members[3]['last_name'] = 'choi'
members[3]['age'] = 25
members[3]['nick_name'] = 'yuju'

members[4]['last_name'] = 'hwang'
members[4]['age'] = 24
members[4]['nick_name'] = 'sinb'

members[5]['first_name'] = 'yewon'
members[5]['last_name'] = 'kim'
members[5]['age'] = 24
members[5]['nick_name'] = 'umji'

for member in members:
    countMember += 1
    print(f"I am going to talk about my wife no. {countMember}:\nHer name is {member['last_name'].title() + ' ' + member['first_name'].title()}.\nShe is {member['age']} years old.\nShe is living in {member['city'].title()}.")
    print("...\n")
```

Exercise 6.8 - Pets:

Listing 70: Exercise 6.8: List of pet dictionaries

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pets = []

pet= {
    'type' : 'cat',
    'name' : 'david',
    'owner' : 'lawrence',
    'weight' : 44,
    'food' : "meat"
}
pets.append(pet)

pet= {
    'type' : 'dog',
    'name' : 'alan',
    'owner' : 'steve',
    'weight' : 29,
    'food' : "sausage"
}
pets.append(pet)

pet= {
    'type' : 'parrot',
    'name' : 'baga',
    'owner' : 'sarah',
    'weight' : 3,
    'food' : "peanuts"
}
pets.append(pet)
for pet in pets:
    print(f"{pet['type'].title()}'s names is {pet['name']}, owner is
        {pet['owner'].title()}.")
    print(f"Weight is {pet['weight']}, and it eats {pet['food']}.")
```

12. List in Dictionary - Complex Data

Definition: Using a list as a value in a dictionary to store multiple items.

```
favourite_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go']
}
```

Exercise 6.9 - Favorite Places:

Listing 71: Exercise 6.9: Lists as dictionary values

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_places = {
    "steven" : ['tokyo', 'pusan', 'yokohama'],
    "apple" : ['new york', 'london'],
    "baka" : ['rome', 'frankfurt', 'seoul','taipei']
}

```

```python
for name, places in favourite_places.items():
    print(f"{name.title()}\'s favourite place are:")
    for place in places:
        print(f"{place.title()}")
```

Exercise 6.10 - Favorite Numbers:

Listing 72: Exercise 6.10: Lists of numbers in dictionary

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_numbers = {
    'sowon' : [1, 3, 4, 8],
    'yerin' : [2, 6, 9],
    'eunha' : [3, 7, 11],
    'yuju' : [4, 112, 1100],
    'sinb' : [5, 6, 7],
    'umji' : [1, 6]
}

for person, numbers in favourite_numbers.items():
    print(f"{person.title()}'s favourite numbers are:")
    for number in numbers:
        print(number)
```

13. Dictionary in Dictionary - Nested Structures

Definition: Storing a dictionary as a value in another dictionary.

```python
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton'
    }
}
```

Exercise 6.11 - Cities:

Listing 73: Exercise 6.11: Nested dictionaries

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cities = {
    'tokyo' : {
        'country' : 'japan',
    'population' : 1_000_000,
        'food' : 'sushi'},
    'new york' : {
        'country' : 'the unided states',
        'population' : 2_000_000,
        'food' : 'hamburger'
    },
    'hongkong' : {
        'country' : 'hongkong',
        'population' : 6_000_000,
        'food' : 'noodles'
    }
}

for city, information in cities.items():
    print(f"Information of {city.title()}:")
#    for country, population, food in information.items():
    print(f"Country: {information['country'].title()}\nPopulation: {
       information['population']}\nFamous food :{information['food
       '].title()}\n")
```

Key Takeaways

- Dictionaries store key-value pairs using curly braces {}
- Keys must be immutable (strings, numbers, tuples)
- Values can be any data type (strings, numbers, lists, dictionaries)
- Use square brackets to access values by key: dict['key']
- Add new key-value pairs by assigning to a new key
- Modify values by assigning to an existing key
- Use del statement to remove key-value pairs permanently
- Loop through all key-value pairs with .items() method
- Loop through keys with .keys() method (default behavior when looping)
- Loop through values with .values() method
- Use set() to get unique values from .values()
- Check if key exists with 'in' operator: 'key' in dict
- Use .get() method for safe key access with default value
- Dictionaries can store lists and other dictionaries as values
- Nesting allows complex data structures (lists of dicts, dicts of dicts)
- Dictionary keys are case-sensitive
- Dictionary order is preserved (Python 3.7+)
- Use .copy() to create a shallow copy of a dictionary
- Use dict() constructor to create dictionaries from other sequences

Chapter 7: User Input and while Loops

1. input() Function - User Input

Definition: A function that pauses your program and waits for the user to enter some text, which is then stored as a string.

Listing 74: Basic user input

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

name = input("Please enter your name: ")
print(f"\nHello, {name}!")


prompt = "If you tell us whp you are, we can personalize the message
    you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print(f"\nHello, {name}!")
```

2. while Loop - Conditional Repetition

Definition: A loop that runs as long as, or while, a certain condition is true.

Listing 75: while loop with user input

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("-----------------")

message = input("Tell me something, and I will repeat it back to you
   . ")
print(message)

print_H()

prompt = "\nTell me something, and I will repeat it back to you: "
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
        print(message)

print_H()

prompt = "\nTell me something, and I will repeat it back to you: "
prompt += "\nEnter 'quit' to end the program. "
```

```python

message = ""
active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

3. int() Function - String to Integer

Definition: A function that converts a string containing a number to an integer.

Listing 76: Converting string input to integer

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

height = input("How tall are you, in inches? ")
height = int(height)

if height >= 48:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little older.")
```

4. Flag - Loop Control Variable

Definition: A variable that acts as a signal to the program, often used to control while loops.

```python
active = True
while active:
    message = input("Enter 'quit' to end: ")
    if message == 'quit':
        active = False
    else:
        print(message)
```

5. break Statement - Immediate Exit

Definition: A statement that immediately exits a loop without running any remaining code in the loop.

```python
while True:
    city = input("Enter a city name (or 'quit' to exit): ")
    if city == 'quit':
        break
    print(f"I'd love to go to {city.title()}!")
```

6. continue Statement - Skip Iteration

Definition: A statement that skips the rest of the current iteration and returns to the beginning of the loop.

```python
current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue
    print(current_number)
```

7. Modulo Operator - %

Definition: An operator that divides one number by another and returns the remainder.

```python
number = 4 % 2  # 0 (even)
number = 5 % 2  # 1 (odd)
```

8. Moving Items Between Lists - List Operations

Definition: The process of removing items from one list and adding them to another list.

```python
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []

while unconfirmed_users:
    current_user = unconfirmed_users.pop()
    confirmed_users.append(current_user)
```

9. Removing All Instances - List Cleanup

Definition: Removing all occurrences of a specific value from a list.

```python
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

10. Filling Dictionary with User Input - Dynamic Data

Definition: Building a dictionary by collecting user input in a loop.

```python
responses = {}
polling_active = True

while polling_active:
    name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb someday
        ? ")

    responses[name] = response

    repeat = input("Would you like to let another person respond? (
        yes/ no) ")
    if repeat == 'no':
        polling_active = False
```

Practical Examples from Chapter 7

Working with User Input and Loops

Chapter 7 introduces user input and while loops. Here are the key files:

Basic User Input:

Listing 77: Chapter07/702_greeter.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

name = input("Please enter your name: ")
print(f"\nHello, {name}!")


prompt = "If you tell us whp you are, we can personalize the message
    you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print(f"\nHello, {name}!")
```

while Loops with User Input:

Listing 78: Chapter07/701_parrot.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("-----------------")

message = input("Tell me something, and I will repeat it back to you
    . ")
print(message)

print_H()

prompt = "\nTell me something, and I will repeat it back to you: "
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
        print(message)

print_H()

prompt = "\nTell me something, and I will repeat it back to you: "
prompt += "\nEnter 'quit' to end the program. "

message = ""
active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Numerical Input and Type Conversion:

Listing 79: Chapter07/703_rollercoster.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

height = input("How tall are you, in inches? ")
height = int(height)

if height >= 48:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little older.")
```

To run these programs:

```
python Chapter07/702_greeter.py
python Chapter07/701_parrot.py
python Chapter07/703_rollercoster.py
```

Key Takeaways

- input() gets user input as a string
- int() converts string to integer
- while loops run while condition is True
- Use flags to control while loops
- break exits a loop immediately
- continue skips to next iteration

- % operator gives remainder
- Use while loops to move items between lists
- remove() removes first occurrence of value
- Build dictionaries with user input
- Always provide clear prompts to users

Chapter 8: Functions

1. Function - Reusable Code Block

Definition: A named block of code that performs a specific task and can be called from other parts of your program.

Listing 80: Basic function definition

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("\n--------------------\n")

def greet_user():
    """Display a simple greeting."""
    print("Hello")

def greet_user_a(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user()
greet_user_a('jesse')

print_H()

def get_formatted_name(first_name, last_name):
    """ Return a full name, neatly formatted. """
    full_name = f"{first_name} {last_name}"
    return full_name.title()

def greet_formatted_user(first_name, last_name):
    """ Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

while True:
    print("Please tell me your name: ")
    print("enter 'q' at any time to quit")

    f_name = input("First name: ")
    if f_name == 'q':
        break
    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"Hello, {formatted_name}!")
```

2. def Statement - Function Definition

Definition: A statement that defines a function, specifying its name and parameters.

```python
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
```

3. Parameter - Function Input

Definition: A piece of information that a function needs to do its job, specified in the function definition.

```python
def greet_user(username):
    print(f"Hello, {username.title()}!")
```

4. Argument - Function Call Value

Definition: A piece of information that's passed from a function call to a function.

```python
greet_user('jesse')  # 'jesse' is the argument
```

5. Return Value - Function Output

Definition: The value that a function returns to the calling line of code.

Listing 81: Function with return value

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("=--------------=")

def get_formatted_name(first_name, last_name):
    """ Return a full name, neatly formatted. """
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

print_H()

def get_formatted_name_A(first_name, middle_name, last_name):
    """ Return a full name, neatly formatted."""
    full_name = f"{first_name} {middle_name} {last_name}"
    return full_name.title()

musician = get_formatted_name_A('john', 'lee', 'hooker')
print(musician)

print_H()
```

```python

def get_formatted_name_B(first_name, last_name, middle_name = ''):
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name_B('Jimi', 'hendrix')
print(musician)
musician = get_formatted_name_B('john', 'hooker', 'lee')
print(musician)
```

6. Default Parameter Value - Optional Arguments

Definition: A parameter that has a default value, making it optional when calling the function.

```python
def get_formatted_name(first_name, last_name, middle_name=''):
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()
```

7. Positional Arguments - Order-Based

Definition: Arguments that must be passed to a function in the same order as the parameters are defined.

```python
def describe_pet(animal_type, pet_name):
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
```

8. Keyword Arguments - Name-Based

Definition: Arguments that are passed to a function by parameter name, allowing any order.

```python
describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
```

9. Arbitrary Arguments - *args

Definition: A parameter that allows a function to accept any number of arguments.

Listing 82: Arbitrary arguments

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def make_pizza(*toppings):
    """Print the list of toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')

def make_pizza_a(*toppings):
    """Summarise the pizza with the following toppings."""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
        print(f" - {topping}")

make_pizza_a('pepperoni')
make_pizza_a('mushrooms', 'green peppers', 'extra cheese')

def make_pizza_b(size, *toppings):
    """Summarise the pizza we are about to make."""
    print(f"\nMaking a {size} - inch pizza with the following
       toppings:")
    for topping in toppings:
        print(f" - {topping}")

make_pizza_b(16, 'pepperoni')
make_pizza_b(12, 'mushrooms', 'green peppers', 'extra cheese')
```

10. Arbitrary Keyword Arguments - **kwargs

Definition: A parameter that allows a function to accept any number of keyword arguments.

```python
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user
       ."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
```

11. Docstring - Function Documentation

Definition: A string that describes what a function does, enclosed in triple quotes.

```python
def greet_user(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")
```

12. Module - Code Organization

Definition: A file containing functions and variables that can be imported into other programs.

```python
# pizza.py
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings
       :")
    for topping in toppings:
        print(f"- {topping}")
```

13. import Statement - Module Usage

Definition: A statement that makes functions and variables from a module available in your program.

```python
import pizza
pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

14. from...import Statement - Selective Import

Definition: A statement that imports specific functions from a module.

```python
from pizza import make_pizza
make_pizza(16, 'pepperoni')
```

Practical Examples from Chapter 8

Working with Functions

Chapter 8 introduces functions and their various forms. Here are the key files:

Basic Function Definition:

Listing 83: Chapter08/801_greeter.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("\n-------------------\n")

def greet_user():
    """Display a simple greeting."""
    print("Hello")
```

```python

def greet_user_a(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user()
greet_user_a('jesse')

print_H()

def get_formatted_name(first_name, last_name):
    """ Return a full name, neatly formatted. """
    full_name = f"{first_name} {last_name}"
    return full_name.title()

def greet_formatted_user(first_name, last_name):
    """ Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

while True:
    print("Please tell me your name: ")
    print("enter 'q' at any time to quit")

    f_name = input("First name: ")
    if f_name == 'q':
        break
    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"Hello, {formatted_name}!")
```

Functions with Return Values:

Listing 84: Chapter08/803_formatted_name.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("=-------------=")

def get_formatted_name(first_name, last_name):
    """ Return a full name, neatly formatted. """
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

print_H()

```

```python
def get_formatted_name_A(first_name, middle_name, last_name):
    """ Return a full name, neatly formatted."""
    full_name = f"{first_name} {middle_name} {last_name}"
    return full_name.title()

musician = get_formatted_name_A('john', 'lee', 'hooker')
print(musician)

print_H()

def get_formatted_name_B(first_name, last_name, middle_name = ''):
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name_B('Jimi', 'hendrix')
print(musician)
musician = get_formatted_name_B('john', 'hooker', 'lee')
print(musician)
```

Functions with Arbitrary Arguments:

Listing 85: Chapter08/807_pizza.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def make_pizza(*toppings):
    """Print the list of toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')

def make_pizza_a(*toppings):
    """Summarise the pizza with the following toppings."""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
        print(f" - {topping}")

make_pizza_a('pepperoni')
make_pizza_a('mushrooms', 'green peppers', 'extra cheese')

def make_pizza_b(size, *toppings):
    """Summarise the pizza we are about to make."""
    print(f"\nMaking a {size} - inch pizza with the following
       toppings:")
    for topping in toppings:
        print(f" - {topping}")

make_pizza_b(16, 'pepperoni')
make_pizza_b(12, 'mushrooms', 'green peppers', 'extra cheese')
```

To run these programs:

```
python Chapter08/801_greeter.py
python Chapter08/803_formatted_name.py
python Chapter08/807_pizza.py
```

Key Takeaways

- Functions are reusable blocks of code
- Use def to define a function
- Parameters receive information in functions
- Arguments provide information to functions
- return sends a value back to the calling line
- Default parameters make arguments optional
- Positional arguments must be in correct order
- Keyword arguments can be in any order
- *args accepts any number of arguments
- **kwargs accepts any number of keyword arguments
- Docstrings document what functions do
- Modules organize code into files
- import makes modules available
- from...import brings specific functions

Chapter 9: Classes

1. Class - Object Blueprint

Definition: A blueprint for creating objects, defining what attributes and methods the objects will have.

Listing 86: Basic class definition

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Dog:
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Intiate name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")

my_dog = Dog('William', 6)
your_dog = Dog('Lucy', 3)

print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()

print(f"My dog's name is {your_dog.name}.")
print(f"My dog is {your_dog.age} years old.")
your_dog.sit()
```

2. Object - Class Instance

Definition: An instance of a class that contains data and behavior defined by the class.

```python
my_dog = Dog('Willie', 6)
your_dog = Dog('Lucy', 3)
```

3. Attribute - Object Data

Definition: A variable that belongs to an object, accessed using dot notation.

```python
print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
```

4. Method - Object Behavior

Definition: A function that belongs to a class, defining what an object can do.

```
my_dog.sit()
my_dog.roll_over()
```

5. __init__() Method - Constructor

Definition: A special method that Python runs automatically whenever you create a new instance of a class.

```
def __init__(self, name, age):
    """Initialize name and age attributes."""
    self.name = name
    self.age = age
```

6. self Parameter - Object Reference

Definition: A reference to the instance of the class, allowing you to access attributes and methods.

```
def sit(self):
    """Simulate a dog sitting in response to a command."""
    print(f"{self.name} is now sitting.")
```

7. Instance - Class Object

Definition: An individual object created from a class, with its own set of attributes.

```
my_dog = Dog('Willie', 6)   # my_dog is an instance
your_dog = Dog('Lucy', 3)   # your_dog is another instance
```

8. Inheritance - Class Relationship

Definition: A feature that allows you to model relationships between classes, where a child class inherits attributes and methods from a parent class.

Listing 87: Inheritance example

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("------------------------")

"""A set of classes used to represent gas and electric car."""

from car import Car

class Battery:

```

```python
    """A simple attempt to model a battery for an electic car."""

    def __init__(self, battery_size = 75):
        """Initialise the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides.
           """
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315
        print(f"This car can go about {range} miles on a full charge
           .")

class ElectricCar(Car):

    """ Represents aspects of a car, specific to electric vehicles.
       """

    def __init__(self, make, model, year):
        """
        Initiate attributes of the parent class.
        Then initiaise attributes specific to an eletric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def fill_gas_tank(self):
        """Electric cars don't have gas tanks."""
        print("This car doesm't need a gas tank!")
```

9. Parent Class - Base Class

Definition: A class that is inherited from, also called a base class or superclass.

```python
class Car:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
```

10. Child Class - Derived Class

Definition: A class that inherits from another class, also called a derived class or sub-class.

```python
class ElectricCar(Car):
    """Represents aspects of a car, specific to electric vehicles.
       """
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
```

11. super() Function - Parent Access

Definition: A function that helps you make connections between parent and child classes.

```python
def __init__(self, make, model, year):
    super().__init__(make, model, year)
    self.battery = Battery()
```

12. Method Overriding - Custom Behavior

Definition: The ability to define a method in a child class that has the same name as a method in the parent class.

```python
def fill_gas_tank(self):
    """Electric cars don't have gas tanks."""
    print("This car doesn't need a gas tank!")
```

13. Instance as Attribute - Object Composition

Definition: Using an instance of one class as an attribute in another class.

```python
class Battery:
    def __init__(self, battery_size=75):
        self.battery_size = battery_size

class ElectricCar(Car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery = Battery()  # Instance as attribute
```

14. Importing Classes - Module Usage

Definition: Bringing classes from one module into another module for use.

```python
from car import Car
from electric_car import ElectricCar

my_tesla = ElectricCar('tesla', 'model s', 2019)
```

Practical Examples from Chapter 9

Working with Classes

Chapter 9 introduces object-oriented programming with classes. Here are the key files:

Basic Class Definition:

Listing 88: Chapter09/901_dog.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Dog:
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Intiate name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")

my_dog = Dog('William', 6)
your_dog = Dog('Lucy', 3)

print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()

print(f"My dog's name is {your_dog.name}.")
print(f"My dog is {your_dog.age} years old.")
your_dog.sit()
```

Advanced Class with Methods:

Listing 89: Chapter09/902_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

"""A class that can be used to represent a car."""

class Car:

    """A simple attempt to represent a car."""

    def __init__ (self, make, model, year):
        """Initiate attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        """Set the odometer reading to the given value."""
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
```

Inheritance and Class Relationships:

Listing 90: Chapter09/electric_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("------------------------")

"""A set of classes used to represent gas and electric car."""

from car import Car

class Battery:

    """A simple attempt to model a battery for an electic car."""

    def __init__(self, battery_size = 75):
        """Initialise the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides.
        """
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315
        print(f"This car can go about {range} miles on a full charge.")

class ElectricCar(Car):

    """ Represents aspects of a car, specific to electric vehicles. """

    def __init__(self, make, model, year):
        """
        Initiate attributes of the parent class.
        Then initiaise attributes specific to an eletric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def fill_gas_tank(self):
        """Electric cars don't have gas tanks."""
        print("This car doesm't need a gas tank!")
```

To run these programs:

```
python Chapter09/901_dog.py
python Chapter09/902_car.py
python Chapter09/electric_car.py
```

Key Takeaways

- Classes are blueprints for creating objects
- Objects are instances of classes
- Attributes store data in objects
- Methods define behavior of objects
- __init__() initializes new instances
- self refers to the current instance
- Inheritance creates class relationships
- Child classes inherit from parent classes
- super() calls parent class methods
- Method overriding customizes behavior
- Objects can contain other objects

- Import classes to use them in other modules
- Classes help organize and structure code

Chapter 10: Files and Exceptions

1. File - Data Storage

Definition: A collection of information stored as a unit on a computer, accessible by programs.

```python
filename = 'pi_digits.txt'
with open(filename) as file_object:
    contents = file_object.read()
```

2. open() Function - File Access

Definition: A function that opens a file and returns a file object, which contains methods and attributes for working with the file.

```python
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
```

3. with Statement - File Context

Definition: A statement that ensures a file is properly closed after the block of code using it is finished.

```python
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
# File is automatically closed here
```

4. read() Method - File Content

Definition: A method that reads the entire contents of a file as a string.

Listing 91: Reading file contents

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("=================")

with open('pi_digits.txt') as file_object:
    contents = file_object.read()
print(contents)
print(contents.rstrip())

print_H()

file_path = 'd:/CSS.md'
with open(file_path) as file_object:
    contents = file_object.read()
print(contents)
```

```python

print_H()

file_path = 'd:/Pandoc.md'
with open(file_path) as file_object:
    for line in file_object:
        print(line.rstrip())

print_H()

filename = 'd:/Markdown.md'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line.rstrip())

print_H()

filename = 'pi_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.strip()

print(pi_string)
print(len(pi_string))
```

5. readlines() Method - Line List

Definition: A method that reads each line from a file and stores them in a list.

```python
with open('pi_digits.txt') as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line.rstrip())
```

6. write() Method - File Writing

Definition: A method that writes a string to a file, overwriting the file's contents.

```python
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

7. append Mode - 'a' Parameter

Definition: A file mode that adds content to the end of a file instead of overwriting it.

```python
filename = 'programming.txt'
with open(filename, 'a') as file_object:
    file_object.write("I also love finding meaning in large datasets
        .\n")
```

8. Exception - Error Handling

Definition: An error that occurs during program execution, which can be caught and handled.

Listing 92: Exception handling

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("========================")

try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")

print_H()

print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\n First number: ")
    if first_number == 'q':
        break
    second_number = input("\n Second number: ")
    if second_number == 'q':
        break
    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
        print("You can't divide bu 0!")
    else:
        print(answer)
```

9. try-except Block - Error Catching

Definition: A block of code that tries to run some code and catches any exceptions that occur.

```python
try:
    answer = int(first_number) / int(second_number)
```

```python
except ZeroDivisionError:
    print("You can't divide by 0!")
```

10. else Block - Success Handling

Definition: A block of code that runs only if the try block succeeds (no exceptions occur).

```python
try:
    answer = int(first_number) / int(second_number)
except ZeroDivisionError:
    print("You can't divide by 0!")
else:
    print(answer)
```

11. FileNotFoundError - Missing File

Definition: An exception that occurs when trying to open a file that doesn't exist.

Listing 93: Handling missing files

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def count_words(filename):
    """Count the approximate number of words in a file."""
    try:
        with open(filename, encoding='utf-8') as f:
            contents = f.read()
    except FileNotFoundError:
        #print(f"Sorry, the file {filename} does not exist.")
        pass
    else:
        words =  contents.split()
        num_words = len(words)
        print(f"The file {filename} has about {num_words} words.")

filename = '1005_alice/alice.txt'
count_words(filename)

print("===============")

filenames = ['1005_alice/alice.txt', '1003_programming/programming.
    txt', '1001_pi/pi_digits.txt']
for filename in filenames:
    count_words(filename)
```

12. ZeroDivisionError - Division by Zero

Definition: An exception that occurs when trying to divide by zero.

```python
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

13. ValueError - Invalid Conversion

Definition: An exception that occurs when trying to convert a string to a number when the string doesn't contain a valid number.

```python
try:
    age = int(input("Enter your age: "))
except ValueError:
    print("Please enter a valid number.")
```

14. pass Statement - Silent Failure

Definition: A statement that tells Python to do nothing in a block, often used in exception handling.

```python
try:
    with open(filename) as f:
        contents = f.read()
except FileNotFoundError:
    pass  # Do nothing if file not found
```

15. JSON - Data Format

Definition: A lightweight data format that's easy for programs to parse and generate.

```python
import json

numbers = [2, 3, 5, 7, 11, 13]
filename = 'numbers.json'
with open(filename, 'w') as f:
    json.dump(numbers, f)
```

Practical Examples from Chapter 10

Working with Files and Exceptions

Chapter 10 introduces file handling and exception handling. Here are the key files:

File Reading Operations:

Listing 94: Chapter10/1001_pi/file_reader.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("==================")

with open('pi_digits.txt') as file_object:
    contents = file_object.read()
print(contents)
print(contents.rstrip())

print_H()

file_path = 'd:/CSS.md'
with open(file_path) as file_object:
    contents = file_object.read()
print(contents)

print_H()

file_path = 'd:/Pandoc.md'
with open(file_path) as file_object:
    for line in file_object:
        print(line.rstrip())

print_H()

filename = 'd:/Markdown.md'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line.rstrip())

print_H()

filename = 'pi_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.strip()

print(pi_string)
print(len(pi_string))
```

Exception Handling:

Listing 95: Chapter10/1004_division_calculator.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def print_H():
    print("========================")

```

```python
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")

print_H()

print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\n First number: ")
    if first_number == 'q':
        break
    second_number = input("\n Second number: ")
    if second_number == 'q':
        break
    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
        print("You can't divide bu 0!")
    else:
        print(answer)
```

File Error Handling:

Listing 96: Chapter10/1006_word_count.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def count_words(filename):
    """Count the approximate number of words in a file."""
    try:
        with open(filename, encoding='utf-8') as f:
            contents = f.read()
    except FileNotFoundError:
        #print(f"Sorry, the file {filename} does not exist.")
        pass
    else:
        words =  contents.split()
        num_words = len(words)
        print(f"The file {filename} has about {num_words} words.")

filename = '1005_alice/alice.txt'
count_words(filename)

print("===============")

filenames = ['1005_alice/alice.txt', '1003_programming/programming.
   txt', '1001_pi/pi_digits.txt']
for filename in filenames:
    count_words(filename)
```

To run these programs:

```
python Chapter10/1001_pi/file_reader.py
python Chapter10/1004_division_calculator.py
python Chapter10/1006_word_count.py
```

Key Takeaways

- Files store data persistently
- open() creates file objects
- with ensures proper file closing
- read() gets entire file content
- readlines() gets list of lines
- write() overwrites file content
- 'a' mode appends to files
- Exceptions handle errors gracefully
- try-except catches exceptions
- else runs on successful try
- FileNotFoundError for missing files
- ZeroDivisionError for division by zero
- ValueError for invalid conversions
- pass does nothing in a block
- JSON stores structured data

Chapter 11: Testing Your Code

1. Test - Code Verification

Definition: A piece of code that verifies that another piece of code works correctly.

```python
def test_first_last_name():
    """Do names like 'Janis Joplin' work?"""
    formatted_name = get_formatted_name('janis', 'joplin')
    assert formatted_name == 'Janis Joplin'
```

2. Unit Test - Function Testing

Definition: A test that verifies that one aspect of a function works correctly.

Listing 97: Unit testing with unittest

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import unittest
from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Test for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')

    def test_first_last_middle_name(self):
        """Do names like 'Wolfgang Amadeus Mozart' work?"""
        formatted_name = get_formatted_name('wolfgang', 'mozart', '
           amadeus')
        self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')

if __name__ == '__main__':
    unittest.main()
```

3. Test Case - Test Class

Definition: A class that contains a series of unit tests that can be run together.

```python
class NamesTestCase(unittest.TestCase):
    """Tests for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')
```

4. assertEqual() Method - Value Comparison

Definition: A method that verifies that a value you expect matches the value the function returns.

```
formatted_name = get_formatted_name('janis', 'joplin')
self.assertEqual(formatted_name, 'Janis Joplin')
```

5. unittest Module - Testing Framework

Definition: A Python module that provides tools for testing your code.

```
import unittest
from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    def test_first_last_name(self):
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')
```

6. setUp() Method - Test Preparation

Definition: A method that runs before each test method, allowing you to create objects once and use them in all your test methods.

```
def setUp(self):
    """Create a survey and a set of responses for use in all test
       methods."""
    question = "What language did you first learn to speak?"
    self.my_survey = AnonymousSurvey(question)
    self.responses = ['English', 'Spanish', 'Mandarin']
```

7. Test Function - Individual Test

Definition: A function that tests a specific aspect of your code.

Listing 98: Function to be tested

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

def get_formatted_name(first, last, middle=''):
    """Generate a neatly formatter full name"""
    if middle:
        full_name = f"{first} {middle} {last}"
    else:
        full_name = f"{first} {last}"
    return full_name.title()
```

8. assertIn() Method - Membership Test

Definition: A method that verifies that an item is in a list.

```python
def test_store_single_response(self):
    """Test that a single response is stored properly."""
    self.my_survey.store_response(self.responses[0])
    self.assertIn(self.responses[0], self.my_survey.responses)
```

9. assertNotIn() Method - Non-Membership Test

Definition: A method that verifies that an item is not in a list.

```python
def test_duplicate_responses(self):
    """Test that duplicate responses are not stored."""
    self.my_survey.store_response(self.responses[0])
    self.my_survey.store_response(self.responses[0])
    self.assertEqual(len(self.my_survey.responses), 1)
```

10. Test Runner - Test Execution

Definition: A tool that runs your tests and reports the results.

```python
if __name__ == '__main__':
    unittest.main()
```

11. Failing Test - Bug Detection

Definition: A test that fails, indicating that there's a problem with the code being tested.

```python
def test_first_last_middle_name(self):
    """Do names like 'Wolfgang Amadeus Mozart' work?"""
    formatted_name = get_formatted_name('wolfgang', 'mozart', '
       amadeus')
    self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')
```

12. Passing Test - Success Verification

Definition: A test that passes, indicating that the code being tested works correctly.

```python
def test_first_last_name(self):
    """Do names like 'Janis Joplin' work?"""
    formatted_name = get_formatted_name('janis', 'joplin')
    self.assertEqual(formatted_name, 'Janis Joplin')
```

13. Test Coverage - Code Verification

Definition: The percentage of your code that's covered by tests.

```python
# Test different scenarios
def test_empty_string(self):
    """Test with empty strings."""
    result = get_formatted_name('', '')
    self.assertEqual(result, ' ')

def test_single_name(self):
    """Test with single name."""
    result = get_formatted_name('john', '')
    self.assertEqual(result, 'John ')
```

14. Integration Test - System Testing

Definition: A test that verifies that multiple parts of your system work together correctly.

Listing 99: Integration testing with user input

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from survey import AnonymousSurvey

# Define a question, and make a survey.
question = "What language did you first learn to speak?"
my_survey = AnonymousSurvey(question)

# Show the question, and store responses to the question.
my_survey.show_question()
print("Enter 'q' at any time to quit.\n")
while True:
    response = input("Language: ")
    if response == 'q':
        break
    my_survey.store_response(response)

# Show the survey results.
print("\nThank you o everyone who participated in the survey!")
my_survey.show_results()
```

Practical Examples from Chapter 11

Working with Testing

Chapter 11 introduces testing and test-driven development. Here are the key files:

Function to be Tested:

Listing 100: Chapter11/name_function.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def get_formatted_name(first, last, middle=''):
    """Generate a neatly formatted full name"""
    if middle:
        full_name = f"{first} {middle} {last}"
    else:
        full_name = f"{first} {last}"
    return full_name.title()
```

Unit Tests:

Listing 101: Chapter11/test_name_function.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import unittest
from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Test for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')

    def test_first_last_middle_name(self):
        """Do names like 'Wolfgang Amadeus Mozart' work?"""
        formatted_name = get_formatted_name('wolfgang', 'mozart', '
           amadeus')
        self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')

if __name__ == '__main__':
    unittest.main()
```

Integration Testing:

Listing 102: Chapter11/language_survey.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from survey import AnonymousSurvey

# Define a question, and make a survey.
question = "What language did you first learn to speak?"
my_survey = AnonymousSurvey(question)

# Show the question, and store responses to the question.
my_survey.show_question()
print("Enter 'q' at any time to quit.\n")
while True:
    response = input("Language: ")
    if response == 'q':
        break
    my_survey.store_response(response)

# Show the survey results.
print("\nThank you o everyone who participated in the survey!")
my_survey.show_results()
```

To run these tests:

```
python Chapter11/test_name_function.py
python Chapter11/language_survey.py
```

Key Takeaways

- Tests verify code works correctly
- Unit tests check individual functions
- Test cases group related tests
- assertEqual() compares expected and actual values
- unittest provides testing framework
- setUp() prepares test data
- assertIn() checks list membership
- assertNotIn() checks non-membership
- Test runners execute tests
- Failing tests indicate bugs
- Passing tests verify correctness
- Test coverage measures completeness
- Integration tests check system parts
- Write tests before fixing bugs
- Tests help prevent regressions

Chapters 12-14: Alien Invasion Project

Project Overview: Alien Invasion Game

Chapters 12-14 focus on building a complete 2D game using Pygame. The project progresses from basic game setup to a fully functional space shooter with scoring, levels, and user interface elements.

Chapter 12: A Ship that Fires Bullets

1. Pygame - Game Development Library

Definition: A set of Python modules designed for writing video games, providing tools for graphics, sound, and input handling.

```python
import pygame
pygame.init()
```

2. Game Loop - Core Game Logic

Definition: The main loop that runs continuously during gameplay, handling events, updating game state, and rendering graphics.

```python
def run_game(self):
    """Start the main loop for the game."""
    while True:
        # Watch for keyboard and mouse events.
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        # Update game objects
        self.ship.update()

        # Redraw the screen
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()
        pygame.display.flip()
```

3. Surface - Drawing Canvas

Definition: A Pygame object that represents a rectangular area where you can draw graphics.

```python
self.screen = pygame.display.set_mode((1200, 800))
self.screen.fill((230, 230, 230))  # Fill with background color
```

4. Rect - Rectangle Object

Definition: A Pygame object that represents a rectangle, used for positioning and collision detection.

```python
self.rect = self.image.get_rect()
self.rect.midbottom = self.screen_rect.midbottom
```

5. Sprite - Game Object

Definition: A 2D object that can be drawn on the screen, typically representing game characters or elements.

```python
class Ship:
    """A class to manage the ship."""
    def __init__(self, ai_game):
        self.screen = ai_game.screen
        self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()
```

6. Event Handling - User Input

Definition: The process of detecting and responding to user actions like key presses and mouse movements.

```python
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        sys.exit()
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = True
    elif event.type == pygame.KEYUP:
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
```

7. Movement Flags - State Management

Definition: Boolean variables that track whether an object should be moving in a particular direction.

```python
# Movement flags
self.moving_right = False
self.moving_left = False

def update(self):
    """Update the ship's position based on movement flags."""
    if self.moving_right and self.rect.right < self.screen_rect.
       right:
        self.x += self.settings.ship_speed
    if self.moving_left and self.rect.left > 0:
        self.x -= self.settings.ship_speed
```

8. Bullet Class - Projectile System

Definition: A class that manages bullets fired by the ship, including their movement and collision detection.

```python
class Bullet(Sprite):
    """A class to manage bullets fired from the ship."""

    def __init__(self, ai_game):
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # Create a bullet rect at (0, 0) and then set correct position.
        self.rect = pygame.Rect(0, 0, self.settings.bullet_width,
            self.settings.bullet_height)
        self.rect.midtop = ai_game.ship.rect.midtop

        # Store the bullet's position as a decimal value.
        self.y = float(self.rect.y)

    def update(self):
        """Move the bullet up the screen."""
        # Update the decimal position of the bullet.
        self.y -= self.settings.bullet_speed
        # Update the rect position.
        self.rect.y = self.y

    def draw_bullet(self):
        """Draw the bullet to the screen."""
        pygame.draw.rect(self.screen, self.color, self.rect)
```

Chapter 13: Aliens

9. Alien Fleet - Enemy Management

Definition: A group of alien sprites that move together and represent the enemies in the game.

```python
class Alien(Sprite):
    """A class to represent a single alien in the fleet."""

    def __init__(self, ai_game):
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings

        # Load the alien image and set its rect attribute.
        self.image = pygame.image.load('images/alien.bmp')
        self.rect = self.image.get_rect()

        # Start each new alien near the top left of the screen.
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        # Store the alien's exact horizontal position.
        self.x = float(self.rect.x)
```

10. Fleet Movement - Coordinated Motion

Definition: The synchronized movement of all aliens in the fleet, including direction changes and dropping down.

```
def _check_fleet_edges(self):
    """Respond appropriately if any aliens have reached an edge."""
    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    """Drop the entire fleet and change the fleet's direction."""
    for alien in self.aliens.sprites():
        alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1
```

11. Collision Detection - Hit Testing

Definition: The process of determining when game objects touch or overlap, used for bullet-alien collisions.

```
def _check_bullet_alien_collisions(self):
    """Respond to bullet-alien collisions."""
    # Remove any bullets and aliens that have collided.
    collisions = pygame.sprite.groupcollide(
        self.bullets, self.aliens, True, True)

    if collisions:
        for aliens in collisions.values():
            self.stats.score += self.settings.alien_points * len(
               aliens)
        self.sb.prep_score()
```

12. Sprite Groups - Object Collections

Definition: Pygame containers that hold multiple sprites, making it easier to update and draw them together.

```python
from pygame.sprite import Group

class AlienInvasion:
    def __init__(self):
        # Create groups to store bullets and aliens
        self.bullets = Group()
        self.aliens = Group()

        self._create_fleet()

    def _create_fleet(self):
        """Create the fleet of aliens."""
        # Create an alien and find the number of aliens in a row.
        alien = Alien(self)
        alien_width, alien_height = alien.rect.size

        available_space_x = self.settings.screen_width - (2 *
           alien_width)
        number_aliens_x = available_space_x // (2 * alien_width)

        # Determine the number of rows of aliens that fit on the
           screen.
        ship_height = self.ship.rect.height
        available_space_y = (self.settings.screen_height -
                            (3 * alien_height) - ship_height)
        number_rows = available_space_y // (2 * alien_height)

        # Create the full fleet of aliens.
        for row_number in range(number_rows):
            for alien_number in range(number_aliens_x):
                self._create_alien(alien_number, row_number)
```

13. Game Stats - State Tracking

Definition: A class that tracks game statistics like score, level, and lives remaining.

```python
class GameStats:
    """Track statistics for Alien Invasion."""

    def __init__(self, ai_game):
        """Initialize statistics."""
        self.settings = ai_game.settings
        self.reset_stats()

        # Start game in an inactive state.
        self.game_active = False

        # High score should never be reset.
        self.high_score = 0

    def reset_stats(self):
        """Initialize statistics that can change during the game."""
        self.ships_left = self.settings.ship_limit
        self.score = 0
        self.level = 1
```

Chapter 14: Scoring

14. Score Display - UI Elements

Definition: Visual elements that show the player's current score, high score, and level.

```python
class Scoreboard:
    """A class to report scoring information."""

    def __init__(self, ai_game):
        """Initialize scorekeeping attributes."""
        self.ai_game = ai_game
        self.screen = ai_game.screen
        self.screen_rect = self.screen.get_rect()
        self.settings = ai_game.settings
        self.stats = ai_game.stats

        # Font settings for scoring information.
        self.text_color = (30, 30, 30)
        self.font = pygame.font.SysFont(None, 48)

        # Prepare the initial score images.
        self.prep_score()
        self.prep_high_score()
        self.prep_level()
        self.prep_ships()

    def prep_score(self):
        """Turn the score into a rendered image."""
        rounded_score = round(self.stats.score, -1)
        score_str = "{:,}".format(rounded_score)
        self.score_image = self.font.render(score_str, True,
                self.text_color, self.settings.bg_color)

        # Display the score at the top right of the screen.
        self.score_rect = self.score_image.get_rect()
        self.score_rect.right = self.screen_rect.right - 20
        self.score_rect.top = 20
```

15. Play Button - User Interface

Definition: A clickable button that allows players to start or restart the game.

```python
class Button:
    def __init__(self, ai_game, msg):
        """Initialize button attributes."""
        self.screen = ai_game.screen
        self.screen_rect = self.screen.get_rect()

        # Set the dimensions and properties of the button.
        self.width, self.height = 200, 50
        self.button_color = (0, 255, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        # Build the button's rect object and center it.
        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        # The button message needs to be prepped only once.
        self._prep_msg(msg)

    def _prep_msg(self, msg):
        """Turn msg into a rendered image and center text on the
           button."""
        self.msg_image = self.font.render(msg, True, self.text_color
           ,
                self.button_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center

    def draw_button(self):
        """Draw blank button and then draw message."""
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

16. Mouse Events - Click Detection

Definition: Events that occur when the user moves or clicks the mouse, used for button interactions.

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.stats.game_active:
        # Reset the game settings.
        self.settings.initialize_dynamic_settings()

        # Reset the game statistics.
        self.stats.reset_stats()
        self.stats.game_active = True
        self.sb.prep_score()
        self.sb.prep_level()
        self.sb.prep_ships()

        # Get rid of any remaining aliens and bullets.
        self.aliens.empty()
        self.bullets.empty()

        # Create a new fleet and center the ship.
        self._create_fleet()
        self.ship.center_ship()

        # Hide the mouse cursor.
        pygame.mouse.set_visible(False)
```

17. Level Progression - Difficulty Scaling

Definition: The system that increases game difficulty as the player advances through levels.

```
def _check_aliens_bottom(self):
    """Check if any aliens have reached the bottom of the screen."""
    screen_rect = self.screen.get_rect()
    for alien in self.aliens.sprites():
        if alien.rect.bottom >= screen_rect.bottom:
            # Treat this the same as if the ship got hit.
            self._ship_hit()
            break

def _ship_hit(self):
    """Respond to the ship being hit by an alien."""
    if self.stats.ships_left > 0:
        # Decrement ships_left, and update scoreboard.
        self.stats.ships_left -= 1
        self.sb.prep_ships()

        # Get rid of any remaining aliens and bullets.
        self.aliens.empty()
        self.bullets.empty()

        # Create a new fleet and center the ship.
        self._create_fleet()
        self.ship.center_ship()

        # Pause.
        sleep(0.5)
    else:
        self.stats.game_active = False
        pygame.mouse.set_visible(True)
```

Practical Examples from the Alien Invasion Project

Complete Game Structure

The Alien Invasion project demonstrates a complete game development workflow:

Main Game File:

Listing 103: Project1/adding_ship_image/alien_invasion.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import sys

import pygame

from settings import Settings
from ship import Ship

class AlienInvasion:
    """Overall class to manage game assets and behavior."""

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height
               ))
        pygame.display.set_caption("Alien Invasion")

        self.ship = Ship(self)


    def run_game(self):
        """Start the main loop for the game."""
        while True:
            # Watch for keyboard and mouse events.
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()

            # Redraw the screen during each pass through the loop.
            self.screen.fill(self.settings.bg_color)
            self.ship.blitme()

            # Make the most recently drawn screen visible.
            pygame.display.flip()

if __name__ == '__main__':
    # Make a game instance, and run the game.
    ai = AlienInvasion()
    ai.run_game()
```

Game Settings:

Listing 104: Project1/adding_ship_image/settings.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Settings:
    """A class to store all settings for Alien Invasion."""

    def __init__(self):
        """Initialize the game's settings."""
        # Screen settings
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)
```

Ship Class:

Listing 105: Project1/adding_ship_image/ship.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import pygame

class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        self.screen = ai_game.screen
        self.screen_rect = ai_game.screen.get_rect()

        # Load the ship image and get its rect.
        self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()

        # Start each new ship at the bottom center of the screen.
        self.rect.midbottom = self.screen_rect.midbottom

    def blitme(self):
        """Draw the ship at its current location."""
        self.screen.blit(self.image, self.rect)
```

To run the game:

```
python Project1/adding_ship_image/alien_invasion.py
```

Key Takeaways

- Pygame provides tools for 2D game development
- Game loops handle events, updates, and rendering

- Sprites represent game objects with position and graphics
- Event handling captures user input
- Collision detection determines object interactions
- Sprite groups manage collections of game objects
- Game stats track score, lives, and level
- UI elements like buttons enhance user experience
- Mouse events enable interactive elements
- Level progression increases game difficulty
- Proper game state management is crucial
- Code organization improves maintainability
- Real-time graphics require efficient rendering
- User feedback through scoring and visual elements

Project Progression

1. **Chapter 12**: Basic game setup, ship movement, and bullet firing
2. **Chapter 13**: Alien fleet creation, movement, and collision detection
3. **Chapter 14**: Scoring system, UI elements, and game completion

This three-chapter project demonstrates complete game development from concept to finished product, covering all essential aspects of 2D game programming with Python and Pygame.

Chapters 15-17: Data Visualization and APIs Project

Project Overview: Data Visualization and APIs

Chapters 15-17 cover the complete development of data visualization skills and API integration using Python. This project focuses on creating meaningful visualizations and working with real-world data through APIs.

Chapter 15: Generating Data

Project Focus: Creating and visualizing data using Python libraries
Key Concepts:

- **Matplotlib**: Primary plotting library for Python
- **Plotly**: Interactive plotting library for web-based visualizations
- **Random Data Generation**: Creating synthetic data for testing
- **Data Visualization**: Converting data into meaningful charts
- **Statistical Analysis**: Understanding data distributions

Main Projects:

Rolling Dice Simulation

Concept: Simulating dice rolls and analyzing probability distributions
Key Components:

- **Die Class**: Object-oriented approach to dice simulation
- **Probability Analysis**: Understanding random distributions
- **Data Collection**: Gathering results from multiple trials
- **Visualization**: Creating bar charts of results

Implementation:

Listing 106: Project2/Dice/die.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from random import randint

class Die:
    # A class representing a single die. #

    def __init__(self, num_sides = 6):
        # Assume a six-sided die. #
        self.num_sides = num_sides

    def roll(self):
        # Return a random value between 1 and number of sides. #
        return randint(1, self.num_sides)
```

Visualization Code:

Listing 107: Project2/Dice/die_visual.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

from die import Die
from plotly import offline
from plotly.graph_objs import Bar, Layout

# Create a D6.
die = Die()

# Make some rolls, and store results in a list.
results = []
for roll_num in range(1000):
    result = die.roll()
    results.append(result)

# Analyse the results
frequencies = []
for value in range(1, die.num_sides+1):
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualize the results
x_value = list(range(1, die.num_sides+1))
data = [Bar(x=x_value, y=frequencies)]

x_axis_config = {'title': 'Result'}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(title='Results of rolling one D6 1000 times',
   xaxis = x_axis_config, yaxis = y_axis_config)
offline.plot({'data': data, 'layout' : my_layout}, filename = 'd6.
   html')
```

Random Walks

Concept: Creating random movement patterns and visualizing paths

Key Components:

- **RandomWalk Class**: Generating random movement patterns
- **Coordinate Systems**: Working with x,y coordinates
- **Path Visualization**: Plotting movement trails

- **Statistical Patterns**: Understanding random behavior

Implementation:

Listing 108: Project2/RandomWalk/random_walk.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from random import choice

class Randomwalk:
# A class to generate random walks. #

    def __init__(self, num_points = 5000):
        # Initialize attricutes of a walk. #
        self.num_points = num_points

        # All walks start at (0,0)
        self.x_values = [0]
        self.y_values = [0]

        # determine the range possible for each steps
        self.step = [value for value in range(0,9)]


    def get_step(self):
        # Decide which direction to go and how far to go in that direction.
        direction = choice([1, -1])
        distance = choice(self.step)
        step = direction * distance
        return step

    def fill_walk(self):
        # Calculate all the points in the walk #
        # Keep talking steps until the walk reaches the desired length. #
        while len(self.x_values) < self.num_points:

            x_step = self.get_step()
            y_step = self.get_step()

            # Reject moves that go nowhere. #
            if x_step == 0 and y_step == 0:
                continue

            # Calculate the new position. #
            x = self.x_values[-1] + x_step
            y = self.y_values[-1] + y_step

            self.x_values.append(x)
            self.y_values.append(y)
```

Chapter 16: Downloading Data

Project Focus: Working with real-world data from various sources
Key Concepts:

- **CSV Files**: Reading and processing comma-separated values
- **Data Cleaning**: Handling missing or invalid data
- **Date/Time Processing**: Working with temporal data
- **Error Handling**: Managing data inconsistencies
- **Data Analysis**: Extracting meaningful insights

Main Projects:

Weather Data Visualization

Concept: Analyzing and visualizing weather patterns from real data
Key Components:

- **CSV Processing**: Reading weather data files
- **Date Handling**: Converting string dates to datetime objects
- **Data Filtering**: Handling missing or invalid values
- **Comparative Analysis**: Comparing multiple datasets
- **Advanced Plotting**: Creating complex visualizations

Implementation:

Listing 109: Project2/Weather/highs_lows_2018.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import csv
import matplotlib.pyplot as plt
from datetime import datetime

filename = 'data/sitka_weather_2018_simple.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    # Get dates, high and low temperatures from this file.
    dates, highs1, lows1 = [], [], []
    for row in reader:
        dates.append(datetime.strptime(row[header_row.index('DATE')
           ], '%Y-%m-%d'))
        highs1.append(int(row[header_row.index('TMAX')]))
        lows1.append(int(row[header_row.index('TMIN')]))
```

```python

filename = 'data/death_valley_2018_simple.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    # Get dates, high and low temperatures from this file.
    dates, highs2, lows2 = [], [], []
    for row in reader:
        current_date = datetime.strptime(row[header_row.index('DATE'
            )], '%Y-%m-%d')
        try:
            high = int(row[header_row.index('TMAX')])
            low = int(row[header_row.index('TMIN')])
        except ValueError:
            print(f"Missing data for {current_date}.")
        else:
            dates.append(current_date)
            highs2.append(high)
            lows2.append(low)

# Plot the high and low temperatures
# Shade the temperature range
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, highs1, c='red', alpha=0.5, label='Sitka High')
ax.plot(dates, lows1, c='blue', alpha=0.5, label='Stika Low')
ax.fill_between(dates, highs1, lows1, facecolor='blue', alpha=0.1)
ax.legend()
ax.plot(dates, highs2, c='brown', alpha=0.5, label="Death Valley
    High")
ax.plot(dates, lows2, c='green', alpha=0.5, label="Death Valley Low"
    )
ax.fill_between(dates, highs2, lows2, facecolor='yellow', alpha=0.1)
ax.legend()

# Format plot
ax.set_title("Daily high and low temperatures, 2018", fontsize = 24)
ax.set_xlabel("", fontsize = 16)
fig.autofmt_xdate()
ax.set_ylabel("Temperature (F)", fontsize = 16)
ax.tick_params(axis='both', which='major', labelsize = 16)

plt.show()
```

Global Data Mapping

Concept: Working with global datasets and creating maps

Key Components:

- **JSON Data**: Processing structured data formats

- **Geographic Data**: Working with location-based information
- **Data Aggregation**: Combining multiple data sources
- **Interactive Maps**: Creating web-based visualizations

Chapter 17: Working with APIs

Project Focus: Integrating with web services and external data sources
Key Concepts:

- **API (Application Programming Interface)**: Interface for accessing external services
- **HTTP Requests**: Making web requests to APIs
- **JSON Processing**: Working with API response data
- **Authentication**: Securing API access
- **Rate Limiting**: Managing API usage limits

Main Projects:

GitHub API Integration

Concept: Accessing GitHub's API to analyze repository data
Key Components:

- **API Authentication**: Using tokens for secure access
- **Data Extraction**: Parsing API responses
- **Error Handling**: Managing API failures
- **Interactive Visualizations**: Creating web-based charts

Implementation:

Listing 110: Project2/Working_API/python_repos_visual.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import requests
from plotly.graph_objs import Bar
from plotly import offline

# Make an API call and store the response.

url = 'https://api.github.com/search/repositories?q=language:python&
    sort=stars'
headers = {'Accept': 'applicaiton/vnd.github.v3+json'}
r = requests.get(url, headers = headers)
print(f"Status code: {r.status_code}")
```

```python

# Process results.
response_dict = r.json()
repo_dicts = response_dict['items']
repo_links, stars, labels = [], [], []
for repo_dict in repo_dicts:
    repo_name = repo_dict['name']
    repo_url = repo_dict['html_url']
    repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
    repo_links.append(repo_link)

    stars.append(repo_dict['stargazers_count'])

    owner = repo_dict['owner']['login']
    description = repo_dict['description']
    label = f"{owner}<br />{description}"
    labels.append(label)

# Make visualization.
data = [{
    'type' : 'bar',
    'x' : repo_links,
    'y' : stars,
    'hovertext' : labels,
    'marker' : {
        'color' : 'rgb(60, 100, 150)',
        'line' : {
            'width' : 1.5,
            'color' : 'rgb(25, 25, 25)'
        }
    },
    'opacity' : 0.6,
}]
my_layout = {
    'title' : 'Most-Starred Python Projects on Github',
    'titlefont' : {
        'size' : 28
    },
    'xaxis' : {
        'title' : 'Repository',
        'titlefont' : {
            'size' : 24
        },
        'tickfont' : {
            'size' : 14
        },
    },
    'yaxis' : {'title' : 'stars'},
}
fig = {'data' : data, 'layout' : my_layout}
offline.plot(fig, filename = 'python_repos.html')
```

Hacker News API

Concept: Working with news data and creating visualizations
Key Components:

- **Real-time Data**: Accessing current information
- **Data Filtering**: Selecting relevant information
- **User Interaction**: Creating clickable elements
- **Data Storytelling**: Presenting insights effectively

Technical Skills Developed

Data Visualization Libraries

- **Matplotlib**: Static plotting and basic charts
- **Plotly**: Interactive web-based visualizations
- **Seaborn**: Statistical data visualization
- **Pygal**: SVG-based charts for web

Data Processing

- **Pandas**: Data manipulation and analysis
- **NumPy**: Numerical computing
- **CSV Module**: File I/O operations
- **JSON Module**: Structured data processing

Web Integration

- **Requests Library**: HTTP client for APIs
- **URL Handling**: Managing web addresses
- **Response Processing**: Handling API data
- **Error Management**: Robust API interactions

Project Outcomes

Data Analysis Skills

- **Statistical Understanding**: Probability and distributions
- **Data Cleaning**: Handling real-world data issues
- **Pattern Recognition**: Identifying trends in data
- **Insight Generation**: Drawing conclusions from data

Visualization Techniques

- **Chart Selection**: Choosing appropriate visualizations
- **Design Principles**: Creating effective charts
- **Interactive Elements**: Engaging user experiences
- **Storytelling**: Communicating data insights

API Integration

- **Service Integration**: Connecting to external data
- **Authentication**: Secure API access
- **Data Transformation**: Converting API responses
- **Error Handling**: Robust application design

Real-World Applications

Business Intelligence

- **Sales Analysis**: Tracking performance metrics
- **Market Research**: Understanding customer behavior
- **Financial Modeling**: Analyzing economic data
- **Operational Metrics**: Monitoring business processes

Scientific Research

- **Climate Analysis**: Weather pattern studies
- **Statistical Modeling**: Probability distributions
- **Data Mining**: Discovering patterns in large datasets
- **Research Visualization**: Presenting findings effectively

Web Development

- **Dashboard Creation**: Real-time data displays
- **API Development**: Building data services
- **Interactive Applications**: User-driven visualizations
- **Data-Driven Websites**: Dynamic content generation

Advanced Concepts

Data Ethics

- **Privacy Protection**: Handling sensitive information
- **Data Accuracy**: Ensuring reliable information
- **Transparency**: Clear data presentation
- **Responsible Use**: Ethical data practices

Performance Optimization

- **Memory Management**: Efficient data handling
- **Processing Speed**: Optimizing calculations
- **API Efficiency**: Minimizing request overhead
- **Scalability**: Handling large datasets

Project Summary

Chapters 15-17 provide comprehensive training in:

1. **Data Generation**: Creating and simulating data
2. **Data Acquisition**: Accessing real-world information
3. **Data Processing**: Cleaning and preparing data
4. **Data Visualization**: Creating meaningful charts
5. **API Integration**: Working with external services
6. **Interactive Applications**: Building engaging experiences

This project develops essential skills for data science, business intelligence, and modern web development, providing a solid foundation for working with real-world data and creating impactful visualizations.

Chapter End Exercises and Practice Problems

This document contains the chapter end exercises and practice problems from Chapters 1-11 of ”Python Crash Course” to reinforce learning and provide hands-on practice.

Chapter End Exercises Overview

Each chapter includes practical exercises that reinforce the concepts learned. These exercises provide hands-on practice with real Python code and help solidify understanding of programming concepts.

Chapter 1: Getting Started

Exercise Focus: Basic Python setup and first programs

Key Concepts Practiced:

- Writing your first Python program
- Using the print() function
- Understanding Python syntax
- Running Python programs

Sample Exercise:

```python
# Exercise: Write a simple message
message = "Hello, Python world!"
print(message)
```

Chapter 2: Variables and Simple Data Types

Exercise Focus: Variables, strings, and basic data types

Key Concepts Practiced:

- Creating and using variables
- String manipulation and formatting
- Working with different data types
- Using f-strings for formatting

Exercise Examples:
Exercise 2.1 - Simple Message:

Listing 111: Chapter02/ex2.1.simple_message.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# simple_message.py -- print out one message

message = "I love Jung EunBi."

print(message)
```

Exercise 2.2 - Simple Messages:

Listing 112: Chapter02/ex2.2.simple_messages.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# simple_messages.py -- print out some messages

message = "I love Jung EunBi."

print(message)

message = "Jung EunBi loves me."

print(message)
```

Exercise 2.3 - Personal Message:

Listing 113: Chapter02/ex2.3.personal_message.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# personal_message.py -- print out personal message

name = "Eunbi"
message = "would you marry me?"

print (f"{name}, {message}")
```

Exercise 2.4 - Name Cases:

Listing 114: Chapter02/ex2.4.name_cases.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# name_cases.py -- print out names in lowercase, uppercase and title
    case

name = "jung eunbi"

print(f"Lowercase: {name.lower()}")
print(f"Uppercase: {name.upper()}")
print(f"Title Case: {name.title()}")
```

Exercise 2.6 - Famous Quote:

Listing 115: Chapter02/ex2.6.quote.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# quote.py -- print out some great persons the his / her quote

person = "Jung Eun Bi"
quote = "As an idol, one hamburger per day is maximum."

print(f"{person} once said, \"{quote}\"")
```

Exercise 2.7 - Stripping Names:

Listing 116: Chapter02/ex2.7.strip.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# strip.py -- manipulating string with strip functions.

name = " Jung Eun Bi "

name2 = " Jung \n Eun \t Bi "

print("For no \\n and \\t characters:")
print(f"No strip: {name}")
print(f"With lstrip(): {name.lstrip()}")
print(f"With rstrip(): {name.rstrip()}")
print(f"WIth strip(): {name.strip()}")

print("When \\n and \\t characters are included:")
print(f"No strip: {name2}")
print(f"With lstrip(): {name2.lstrip()}")
print(f"With rstrip(): {name2.rstrip()}")
print(f"WIth strip(): {name2.strip()}")
```

Chapter 3: Introducing Lists

Exercise Focus: Working with lists and list operations
Key Concepts Practiced:

- Creating and accessing lists
- List indexing and slicing
- Modifying list elements
- List methods and operations

Exercise Examples:
Exercise 3.1 - Names:

Listing 117: Chapter03/ex3.1.gfriend.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# gfriend.py -- list out the name of your friends

print(gfriend[0])
gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

print(gfriend[0])
print(gfriend[1])
print(gfriend[2])
print(gfriend[3])
print(gfriend[4])
print(gfriend[5])
```

Exercise 3.2 - Greetings:

Listing 118: Chapter03/ex3.2.greetings.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

# greetings.py -- say greetings to each of the members

greeting = ", guten Tag!"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(gfriend[0] + greeting)
print(gfriend[1] + greeting)
print(gfriend[2] + greeting)
print(gfriend[3] + greeting)
print(gfriend[4] + greeting)
print(gfriend[5] + greeting)
```

Exercise 3.3 - Your Own List:

Listing 119: Chapter03/ex3.3.transportation.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

transportation = ["bus", "bike", "motorcycle", "foot", "van", "train
    "]
brandName = ["Honda", "BMW", "Toyota"]

message = "I go to school by"

print(message + " " + brandName[0] + " " + transportation[0] + ".")
```

Exercise 3.4 - Guest List:

Listing 120: Chapter03/ex3.4.dinner.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

# dinner.py -- invite members to the my dinner

invitation = ", would you join my dinner tonight?"

```

```python
gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")
```

Exercise 3.5 - Changing Guest List:

Listing 121: Chapter03/ex3.5.update_dinner.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'
print(f"Current list: {gfriend}")
```

Exercise 3.6 - More Guests:

Listing 122: Chapter03/ex3.6.update_dinner.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'

print("\n---")
print("and Sinb will bring WJSN come.")
gfriend.append("WJSN")
print(f"Current list: {gfriend}")

print("also, Eunha will bring another SinB to the dinner.\nThe two
    SinBs need to sit together.")
gfriend.insert(4, "Sinb")
print(f"Current list: {gfriend}")
```

Exercise 3.7 - Shrinking Guest List:

Listing 123: Chapter03/ex3.7.update_dinner.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# update_dinner.py -- some of the members cannot come to dinner, so
    invite again them to the my dinner

invitation = ", would you join my dinner tonight?"

gfriend = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
print(f"Current list: {gfriend}")
print(f"{gfriend[0]}{invitation}")
print(f"{gfriend[1]}{invitation}")
print(f"{gfriend[2]}{invitation}")
print(f"{gfriend[3]}{invitation}")
print(f"{gfriend[4]}{invitation}")
print(f"{gfriend[5]}{invitation}")

print("\n---")
print(f"{gfriend[1]} cannot come to my dinner. But IU can.")
gfriend[1] = 'IU'

print("\n---")
print("and Sinb will bring WJSN come.")
gfriend.append("WJSN")
print(f"Current list: {gfriend}")

print("also, Eunha will bring another SinB to the dinner.\nThe two
    SinBs need to sit together.")
gfriend.insert(4, "Sinb")
print(f"Current list: {gfriend}")

print("\n---")
print("Now one SinB kicks another out.")
del gfriend[4]
print(f"Current list{gfriend}")

print("\n---")
print("Eunha is being dissed. She is sad and she left for crying.")
gfriend.remove("eunha")
print(f"Current list:{gfriend}")

print("\n---")
print(f"{gfriend.pop(0)} goes to comfort Eunha.")
print(f"Current list: {gfriend}")
```

Chapter 4: Working with Lists

Exercise Focus: Loops, list operations, and numerical ranges

Key Concepts Practiced:

- Using for loops with lists
- Working with numerical ranges
- List comprehensions
- Slicing and copying lists

Exercise Examples:

Exercise 4.1 - Pizzas:

Listing 124: Chapter04/ex4.1.pizza.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pizzas = ['peccato', 'diavola', 'capricciosa']

for pizza in pizzas:
    print(f"I like {pizza.title()}")

print("The above statements are fake.")
```

Exercise 4.2 - Animals:

Listing 125: Chapter04/ex4.2.animal.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

animals = ['cats', 'dogs', 'lions']

for animal in animals:
    print(f"{animal.title()} have four legs.")

print("Any of them can be a great pet.")
```

Exercise 4.3 - Counting to Twenty:

Listing 126: Chapter04/ex4.3.count.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

```

```python
numbers = list(range(1,21))

for number in numbers:
    print(f"{number}")
```

Exercise 4.4 - One Million:

Listing 127: Chapter04/ex4.4.count.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,1000001))

for number in numbers:
    print(f"{number}")
```

Exercise 4.5 - Summing a Million:

Listing 128: Chapter04/ex4.5.million.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,1000001))

print(f"Max : {max(numbers)}")
print(f"Min : {min(numbers)}")
print(f"Sum : {sum(numbers)}")
```

Exercise 4.6 - Odd Numbers:

Listing 129: Chapter04/ex4.6.count.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

even_numbers = list(range(2,21,2))

for number in even_numbers:
    print(f"{number}")
```

Exercise 4.7 - Threes:

Listing 130: Chapter04/ex4.7.count.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

threes_numbers = list(range(3, 31 ,3))

for number in threes_numbers:
    print(f"{number}")
```

Exercise 4.8 - Cubes:

Listing 131: Chapter04/ex4.8.cubic.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cube = []

```

```python
for number in list(range(1, 11)):
    cube.append(number**3)

for member in cube:
    print(f"{member}")
```

Exercise 4.9 - Cube Comprehension:

Listing 132: Chapter04/ex4.9.cubic.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cube = [value ** 3 for value in range(1,11)]

for member in cube:
    print(f"{member}")
```

Exercise 4.10 - Slices:

Listing 133: Chapter04/ex4.10.animal.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

animals = ['cats', 'dogs', 'lions']

for animal in animals:
    print(f"{animal.title()} have four legs.")

print("Any of them can be a great pet.")

print("\n----")
animals.append('elephant')
print(f"Adding {animals[-1]}.\nNow we have the following animals:")
for animal in animals:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the first three animals:")
for animal in animals[:3]:
    print(f"{animal.title()}")

print("\n----")
animals.append('sharks')
print(f"Adding {animals[-1]}.\nNow we have the following animals:")
for animal in animals:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the middle three animals:")
for animal in animals[(int)(len(animals)/2-1):(int)(len(animals)
   /2+2)]:
    print(f"{animal.title()}")

print("\n----")
print("Picking up the last three animals:")
for animal in animals[-3:]:
    print(f"{animal.title()}")
```

Exercise 4.11 - My Pizzas, Your Pizzas:

Listing 134: Chapter04/ex4.11.pizza.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pizzas = ['peccato', 'diavola', 'capricciosa']

for pizza in pizzas:
    print(f"I like {pizza.title()}.")
print("The above statements are fake.")

friend_pizzas = pizzas[:]

print("\n----")
print("Another pizza list as per below:")
for pizza in friend_pizzas:
    print(f"{pizza.title()}")

friend_pizzas.append('Clam pie')
print("\n----")
print(f"Adding {friend_pizzas[-1]}\nThe pizza list:")
for pizza in friend_pizzas:
    print(f"{pizza.title()}")
```

Exercise 4.12 - More Loops:

Listing 135: Chapter04/ex4.12.foods.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

my_foods = ['pizza', 'falafel', 'carrpt cake']
friend_foods = my_foods[:]

print("My favouried foods are :")
for food in my_foods:
    print(f"{food.title()}")

print("My friends's favouried foods are :")
for food in friend_foods:
    print(f"{food.title()}")

print("\n---------")
print("Adding one food for each of mine and friend's list:\n")
my_foods.append('cannoli')
friend_foods.append('ice cream')

print("Now, my favouried foods are :")
for food in my_foods:
    print(f"{food.title()}")

print("My friends's favouried foods are :")
for food in friend_foods:
    print(f"{food.title()}")
```

Exercise 4.13 - Buffet:

Listing 136: Chapter04/ex4.13.buffet.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

foods = ('pizza', 'falafel', 'carrpt cake', 'sushi', 'ice cream')

print("Food in a buffet:")
for food in foods:
    print(f"{food.title()}")

print("\n---")
print("Now there is a new menu:")
foods = ('fried rice', 'onigiri', 'carrpt cake', 'sushi', 'ice cream
   ')

print("Food in a buffet:")
for food in foods:
    print(f"{food.title()}")
```

Chapter 5: if Statements

Exercise Focus: Conditional logic and decision making
Key Concepts Practiced:

- Writing if statements
- Using conditional tests
- Boolean logic and operators
- Complex conditional logic

Exercise Examples:
Exercise 5.1 - Conditional Tests:

Listing 137: Chapter05/ex5.1.cars.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

car = 'subaru'
print("Is car == 'subaru'? I predict it is True.")
print(car == 'subaru')

print("Is car == 'audi'? I predict it is False.")
print(car == 'audi')
```

Exercise 5.2 - More Conditional Tests:

Listing 138: Chapter05/ex5.2.guessing_number.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

# guess_number.py
# system randomally define a number between 1 and 100
# let user guess the number
# if user's guess is out of range, warning will be issued (only
    three out-of-range guess allowed)
# if user's guess is in the range but not matching the answer, user
    need to guess again
# if the guess is correct, exit the program

import random

number = random.randint(1, 100)
out_of_range_chance = 3
guessRange = list(range(1, 101))

while True:
    guess = int(input(f"Input an integer between {guessRange[0]} and
        {guessRange[-1]}: "))
    if (guess > guessRange[-1]) or (guess < guessRange[0]):
        out_of_range_chance = out_of_range_chance - 1
        if out_of_range_chance > 0:
            print("Your guess is out of the range of available
               gueese. Try again!")
            continue
        else:
            print("There are too many out of range guesses. Get out
               of the game!")
            break
    elif guess == number:
        print("Congradulations! You have got a correct guess.")
        break
    else:
        if guess > number:
            print("Your guess is too large. Please try again.")
            guessRange = guessRange[:guessRange.index(guess)]
            continue
        else:
            print("Your guess is too small. Please try again.")
            guessRange = guessRange[guessRange.index(guess+1):]
            continue
```

Exercise 5.3 - Alien Colors:

Listing 139: Chapter05/ex5.3.alien_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']
```

```python

points = 0

print(f"Now you have {points} points.\n")

guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct. You get five
       points!\n")
    points += 5
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

Exercise 5.4 - Alien Colors 2:

Listing 140: Chapter05/ex5.4.alien_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']
awards = [5, 10, 10]

points = 0

print(f"Now you have {points} points.\n")

guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct.\n")
    award = awards[alien_car.index(guess)]
    print(f"You get {award} points!\n")
    points += award
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

Exercise 5.5 - Alien Colors 3:

Listing 141: Chapter05/ex5.5.alien_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

alien_car = ['green', 'yellow', 'red']
awards = [5, 10, 15]

points = 0

print(f"Now you have {points} points.\n")

```

```
guess = input("Guess the car's color (green / yellow / red): ").
    lower()

if guess in alien_car:
    print("Congratulations: Your guess is correct.\n")
    award = awards[alien_car.index(guess)]
    print(f"You get {award} points!\n")
    points += award
    print(f"Now you have {points} points.")
else:
    print("Wrong guess.")
```

Exercise 5.6 - Stages of Life:

Listing 142: Chapter05/ex5.6.stages_of_life.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

age = input("Input your age: ")

if age < 2:
    print("Baby")
elif age < 4:
    print("Toddler")
elif age < 13:
    print("Kid")
elif age < 20:
    print("Teenager")
elif age < 65:
    print("Adult")
else:
    print("Elder")
```

Exercise 5.7 - Favorite Fruit:

Listing 143: Chapter05/ex5.7.fruits.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

fruits = ['banana', 'orange', 'fdf', 'apple']

fruit = 'banana'

if fruit in fruits:
    print("I like bananas")
else:
    print(f"{fruit.title()}")
```

Exercise 5.8 - Hello Admin:

Listing 144: Chapter05/ex5.8.users.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
```

```python
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

for user in users:
    if user in admin:
        print(f"Hello admin {user}, would you like to see a status
            report?")
    else:
        print(f"Hello {user}, thank you for logging in again.")
```

Exercise 5.9 - No Users:

Listing 145: Chapter05/ex5.9.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']

if users:
    for user in users:
        if user in admin:
            print(f"Hello admin {user}, would you like to see a
                status report?")
        else:
            print(f"Hello {user}, thank you for logging in again.")
else:
    print("There is no user.")
```

Exercise 5.10 - Checking Usernames:

Listing 146: Chapter05/ex5.10.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

admin = ['sowon', 'yerin', 'eunha']
users = ['sowon', 'yerin', 'eunha', 'yuju', 'sinb', 'umji']
walk_in_users = ['eunso', 'yeorum', 'IU', 'sana', 'eunha', 'sinb']
walk_in_users_updated = []

for walk_in_user in walk_in_users:
    walk_in_users_updated.append(walk_in_user.lower())

if users:
    for walk_in_user in walk_in_users_updated:
        if walk_in_user in users:
            print(f"{walk_in_user.title()} name is already in user
                list. Please use another name.")
        else:
            users.append(walk_in_user)
            print(f"{walk_in_user.title()} has been newly registered
                .")
else:
    print("There is no registered user.")
```

Exercise 5.11 - Ordinal Numbers:

Listing 147: Chapter05/ex5.11.ordinary.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

numbers = list(range(1,10))

for number in numbers:
    if number == 1:
        print(f"{number}st")
    elif number == 2:
        print(f"{number}nd")
    elif number == 3:
        print(f"{number}rd")
    else:
        print(f"{number}th")
```

Chapter 6: Dictionaries

Exercise Focus: Working with key-value pairs and dictionary operations

Key Concepts Practiced:

- Creating and accessing dictionaries
- Modifying dictionary contents
- Looping through dictionaries
- Nesting data structures

Exercise Examples:

Exercise 6.1 - Person:

Listing 148: Chapter06/ex6.1.person.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

person = {
    'first_name' : 'eunbi',
    'last_name' : 'jung',
    'age' : 26,
    'city' : 'seoul'
}

print(f"I am going to talk about my wife:\nHer name is {person['
    last_name'].title() + ' ' + person['first_name'].title()}.\nShe
    is {person['age']} years old.\nShe is living in {person['city'].
    title()}.")
```

Exercise 6.2 - Favorite Numbers:

Listing 149: Chapter06/ex6.2.favourite_number.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

```

```python
favourite_number = {
    'sowon' : 1,
    'yerin' : 2,
    'eunha' : 3,
    'yuju' : 4,
    'sinb' : 5,
    'umji' : 6
}

print(f"Sowon's favourite number is {favourite_number['sowon']}.")
print(f"Yerin's favourite number is {favourite_number['yerin']}.")
print(f"Eunha's favourite number is {favourite_number['eunha']}.")
print(f"Yuju's favourite number is {favourite_number['yuju']}.")
print(f"Sinb's favourite number is {favourite_number['sinb']}.")
print(f"Umji's favourite number is {favourite_number['umji']}.")
```

Exercise 6.3 - Glossary:

Listing 150: Chapter06/ex6.3.glossary.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

glossary = {
    'die Adresse' : "address",
    'die Webseite' : "website"
}

print(f"'die Adresse' means {glossary['die Adresse'].title()}.")
print(f"'die Webseite' means {glossary['die Webseite'].title()}.")
```

Exercise 6.4 - Glossary 2:

Listing 151: Chapter06/ex6.4.glossary.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

glossary = {
    'die Adresse' : "address",
    'die Webseite' : "website",
        'können' :  "can",
    'Velen Dank' : "Very thnks"
}

for word, meaning in glossary.items():
    print(f"{word.title()} means {meaning.title()}.")
```

Exercise 6.5 - Rivers:

Listing 152: Chapter06/ex6.5.river.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

rivers = {
    'nile' : "egypt",
        'eunha' : "Jung Eun Bi",
    'komogawa' : "japan"
}
countRiver = 0
countCountry = 0

for river, country in rivers.items():
    print(f"The {river.title()} runs through {country.title()}.")

for river in rivers.keys():
    countRiver += 1
    print(f"River {countRiver}: {river.title()}")

for country in rivers.values():
    countCountry += 1
    print(f"Country {countCountry}: {country.title()}")
```

Exercise 6.6 - Polling:

Listing 153: Chapter06/ex6.6.favourite_languages.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_languages = {
    'jen' : 'python',
        'sarah' : 'c',
    'edward' : 'ruby',
    'phil' : 'python'
}

for name, language in favourite_languages.items():
    print(f"{name.title()}s favourite language is {language.title()
      }.")

print("\n-----\n")

people = ['david', 'stefan', 'modric', 'sarah', 'phil']

for person in people:
    if person in favourite_languages:
        print(f"{person.title()}, thank you for the poll.\nYour
          favourite language is {favourite_languages.get(person).
          title()}.")
    else:
        print(f"{person.title()}, please take the poll.")
```

Exercise 6.7 - People:

Listing 154: Chapter06/ex6.7.person.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

members = []

countMember = 0
```

```

for count in range(6):
    person = {
        'nick_name' : 'eunha',
        'first_name' : 'eunbi',
        'last_name' : 'jung',
        'age' : 26,
        'city' : 'seoul'
    }
    members.append(person)

members[0]['first_name'] = 'sojung'
members[0]['last_name'] = 'kim'
members[0]['age'] = 27
members[0]['nick_name'] = 'sowon'

members[1]['first_name'] = 'yerin'
members[1]['last_name'] = 'jung'
members[1]['nick_name'] = 'yerin'

members[3]['first_name'] = 'yuna'
members[3]['last_name'] = 'choi'
members[3]['age'] = 25
members[3]['nick_name'] = 'yuju'

members[4]['last_name'] = 'hwang'
members[4]['age'] = 24
members[4]['nick_name'] = 'sinb'

members[5]['first_name'] = 'yewon'
members[5]['last_name'] = 'kim'
members[5]['age'] = 24
members[5]['nick_name'] = 'umji'

for member in members:
    countMember += 1
    print(f"I am going to talk about my wife no. {countMember}:\nHer
        name is {member['last_name'].title() + ' ' + member['
       first_name'].title()}.\nShe is {member['age']} years old.\
       nShe is living in {member['city'].title()}.")
    print("...\n")
```

Exercise 6.8 - Pets:

Listing 155: Chapter06/ex6.8.pets.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

pets = []

pet= {
    'type' : 'cat',
    'name' : 'david',
```

```python
    'owner' : 'lawrence',
    'weight' : 44,
    'food' : "meat"
}
pets.append(pet)

pet= {
    'type' : 'dog',
    'name' : 'alan',
    'owner' : 'steve',
    'weight' : 29,
    'food' : "sausage"
}
pets.append(pet)

pet= {
    'type' : 'parrot',
    'name' : 'baga',
    'owner' : 'sarah',
    'weight' : 3,
    'food' : "peanuts"
}
pets.append(pet)
for pet in pets:
    print(f"{pet['type'].title()}'s names is {pet['name']}, owner is
        {pet['owner'].title()}.")
    print(f"Weight is {pet['weight']}, and it eats {pet['food']}.")
```

Exercise 6.9 - Favorite Places:

Listing 156: Chapter06/ex6.9.favourite_places.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_places = {
    "steven" : ['tokyo', 'pusan', 'yokohama'],
    "apple" : ['new york', 'london'],
    "baka" : ['rome', 'frankfurt', 'seoul','taipei']
}

for name, places in favourite_places.items():
    print(f"{name.title()}\'s favourite place are:")
    for place in places:
        print(f"{place.title()}")
```

Exercise 6.10 - Favorite Numbers:

Listing 157: Chapter06/ex6.10.favourite_numbers.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

favourite_numbers = {
    'sowon' : [1, 3, 4, 8],
    'yerin' : [2, 6, 9],
    'eunha' : [3, 7, 11],
    'yuju' : [4, 112, 1100],
    'sinb' : [5, 6, 7],
    'umji' : [1, 6]
}

for person, numbers in favourite_numbers.items():
    print(f"{person.title()}'s favourite numbers are:")
    for number in numbers:
        print(number)
```

Exercise 6.11 - Cities:

Listing 158: Chapter06/ex6.11.cities.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

cities = {
    'tokyo' : {
        'country' : 'japan',
    'population' : 1_000_000,
        'food' : 'sushi'},
    'new york' : {
        'country' : 'the unided states',
        'population' : 2_000_000,
        'food' : 'hamburger'
    },
    'hongkong' : {
        'country' : 'hongkong',
        'population' : 6_000_000,
        'food' : 'noodles'
    }
}

for city, information in cities.items():
    print(f"Information of {city.title()}:")
#     for country, population, food in information.items():
    print(f"Country: {information['country'].title()}\nPopulation: {
       information['population']}\nFamous food :{information['food
       '].title()}\n")
```

Chapter 7: User Input and while Loops

Exercise Focus: Getting user input and controlling program flow
Key Concepts Practiced:

- Getting user input with input()
- Using while loops
- Controlling loop execution

- Data type conversion

Exercise Examples:
Exercise 7.1 - Rental Car:

Listing 159: Chapter07/ex7.1.rental_car.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

car = input("Which tell me what kind of rental car you would like to
    have: ")
print(f"Let me see if I can find you a {car.title()}.\n")
```

Exercise 7.2 - Restaurant Seating:

Listing 160: Chapter07/ex7.2.restaurant.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

no_of_ppl = int(input("Please tell me how many of people in your
   dinner group: "))

if no_of_ppl > 8:
    print("Sorry, please wait for a while.\n")
else:
    print("Your table is ready.\n")
```

Exercise 7.3 - 10s:

Listing 161: Chapter07/ex7.3.multiple.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

number = int(input("Enter a number: "))

if number % 10 == 0:
    print(f"{number} is divisible by 10.\n")
else:
    print(f"{number} is not divisible by 10.\n")
```

Exercise 7.4 - Pizza Toppings:

Listing 162: Chapter07/ex7.4.pizza_toppings.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pizza_toppings = []

active = True
while active == True:
    pizza_topping = input("Enter one pizza topping (or type \'quit\'
        to exit): ").lower()
    if pizza_topping == 'quit':
        active = False
    else :
        pizza_toppings.append(pizza_topping

if len(pizza_toppings) == 0:
    print("You will get a bare pizza.\n")
else:
    print(f"There are {len(pizza_toppings)} of pizza toppings:")
    for pizza_topping in pizza_toppings:
       print(pizza_topping.title())
```

Exercise 7.5 - Movie Tickets:

Listing 163: Chapter07/ex7.5.movie_tickets.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

customers = []
group = {
    'baby' : 0,
    'child' : 0,
    'adult' : 0
}
unit_price = {
    'baby' : 0,
    'child' : 10,
    'adult' : 15
}
total_customers = 0
total_cost = 0
active = True

# input the ages

while active:
    customer = int(input("Please enter customer's age (input \'0\'
       to quit): "))
    if customer == 0:
        active = False
    else:
        customers.append(customer)

# divide the customers into groups

if len(customers) == 0:
    print("There is no one watching the movie.")
else:
    for customer in customers:
        if customer < 3:
            group['baby'] += 1
        elif (customer >= 3) and (customer < 12):
            group['child'] += 1
        else:
            group['adult'] += 1

    # calculate the cost
    print("\nNumber of customers: ")
    for item, value in group.items():
        total_customers += value
        cost = value * unit_price[item]
        total_cost += cost
        print(f"{item.title()}\t:\t{value} customers\t\tSubtotal: ${
           cost}")
    print(f"------------\nTotal\t:\t{total_customers} customers\t\
       tGrand Total: ${total_cost}")
```

Exercise 7.8 - Deli:

Listing 164: Chapter07/ex7.8.deli.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

sandwich_orders = ['hamburger', 'club sandwich', 'doner sandwich', '
   chicken breast sandwich', 'porilainen']
finished_orders = []

def make_sandwich(sandwich):
    print(f"I make your {sandwich.title()}.")

def finished_sandwich(sandwich):
    print(f"{sandwich.title()} has been finished.")

print("Sandwich orders:")
for sandwich in sandwich_orders:
    print(f"{sandwich.title()}")
print("\n------\n")

while len(sandwich_orders) != 0:
    processing = sandwich_orders.pop(0)
    make_sandwich(processing)
    finished_orders.append(processing)
    finished_sandwich(processing)

print("\n-------\nFinished sandwich orders:")
for sandwich in finished_orders:
    print(f"{sandwich.title()}")
```

Exercise 7.9 - No Pastrami:

Listing 165: Chapter07/ex7.9.deli.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

sandwich_orders = ['hamburger', 'club sandwich', 'doner sandwich', '
   chicken breast sandwich', 'porilainen', 'pastrami a', 'pastrami b
   ', 'pastrami c']
finished_orders = []

def make_sandwich(sandwich):
    print(f"I make your {sandwich.title()}.")
```

```python

def finished_sandwich(sandwich):
    print(f"{sandwich.title()} has been finished.")

def no_pastrami(sandwich):
    print(f"Sorry, there is no pastrami, so {sandwich.title()} will
       be skipped.")

print("Sorry, there is no pastrami available now.\n")
print("Sandwich orders:")
for sandwich in sandwich_orders:
    print(f"{sandwich.title()}")
print("\n------\n")

while len(sandwich_orders) != 0:
    processing = sandwich_orders.pop(0)
    make_sandwich(processing)
    if 'pastrami' in processing:
        no_pastrami(processing)
        continue
    finished_orders.append(processing)
    finished_sandwich(processing)

print("\n-------\nFinished sandwich orders:")
for sandwich in finished_orders:
    print(f"{sandwich.title()}")
```

Chapter 8: Functions

Exercise Focus: Creating and using functions

Key Concepts Practiced:

- Defining functions with def
- Passing arguments to functions
- Returning values from functions
- Using default parameters

Exercise Examples:

Exercise 8.1 - Message:

Listing 166: Chapter08/ex8.1.message.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def display_message():
    print("I am going to learn this chapter.")


display_message()
```

Exercise 8.2 - Favorite Book:

Listing 167: Chapter08/ex8.2.favourite_book.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def favourite_book(title):
    print(f"One of my favourite books is {title.title()}.")

books = ['Alice in the Wonderland', 'The Wealth of Nations', '1984']

for book in books:
    favourite_book(book)
```

Exercise 8.3 - T-Shirt:

Listing 168: Chapter08/ex8.3.t-shirt.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def make_shirt(size = 'M', message = 'Wie heißen Sie?'):
    print(f"We will make {size} T-shirt with a slogen of \"{message
       }.\"\n")

make_shirt()

make_shirt('S')

# make_shirt(, 'Ich heiße hihi.')
# cannot empty the first argument.

make_shirt(message = 'Und Sie?', size = 'L')

make_shirt('XL', 'Ich heiße Stefan.')
```

Exercise 8.5 - Cities:

Listing 169: Chapter08/ex8.5.cities.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cities = []

def describe_city(city, country):
    print(f"{city.title()} is in {country.title()}.\n")

cityInsert = {
    'city_name' : 'osaka',
    'country' : 'japan'
}

cities.append(cityInsert)

cityInsert = {
    'city_name' : 'munich',
    'country' : 'germany'
}

cities.append(cityInsert)

cityInsert = {
    'city_name' : 'london',
    'country' : 'britain'
}

cities.append(cityInsert)

for city in cities:
    describe_city(city['city_name'], city['country'])
```

Exercise 8.6 - City Names:

Listing 170: Chapter08/ex8.6.cities.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

cities = []

def describe_city(city, country):
    print(f"{city.title()}, {country.title()}")

def city_country(city, country):
    city_insert = {}
    city_insert['city_name'] = city.lower()
    city_insert['country'] = country.lower()
    cities.append(city_insert)

city_country('OsAka', 'japan')
city_country('berlin', 'germAny')
city_country('paris', 'FrancE')

for city in cities:
    describe_city(city['city_name'], city['country'])
```

Exercise 8.7 - Album:

Listing 171: Chapter08/ex8.7.albums.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

albums = []

def make_album (album_name, artist):
    album = {}
    album['album_name'] = album_name
    album['artist'] = artist
    album['no_of_songs'] = None
    return album

```

```python
def print_album(albums):
    for album in albums:
        if album['no_of_songs'] != None:
            print(f"{album['album_name']} of {album['artist']} has
               number of songs: {album['no_of_songs']}")
        else:
            print(f"{album['album_name']} of {album['artist']} has
               no songs.")

albums.append(make_album('Beam of Prism', 'VIVIZ'))
albums.append(make_album('Summer Vibe', 'VIVIZ'))
albums.append(make_album('VarioUS', 'VVIZ'))

albums[0]['no_of_songs'] = 7

print_album(albums)
```

Exercise 8.8 - User Albums:

Listing 172: Chapter08/ex8.8.albums.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

albums = []

def make_album (album_name, artist):
    album = {}
    album['album_name'] = album_name
    album['artist'] = artist
    album['no_of_songs'] = None
    return album

def print_album(albums):
    for album in albums:
        if album['no_of_songs'] != None:
            print(f"\"{album['album_name']}\" of {album['artist']}
               has number of songs: {album['no_of_songs']}")
        else:
            print(f"\"{album['album_name']}\" of {album['artist']}
               has no songs.")

albums.append(make_album('Beam of Prism', 'VIVIZ'))
albums.append(make_album('Summer Vibe', 'VIVIZ'))
albums.append(make_album('VarioUS', 'VVIZ'))
albums[0]['no_of_songs'] = 7

while True:
    print("Enter detail of an album.")
    print("(enter \'q\' at any time to quit)")

    album_name = input("Album Name: ")
    if album_name == 'q':
        break

    artist = input("Artist Name: ")
    if artist == 'q':
        break

    no_of_songs = input("No. of Albums: ")
    if no_of_songs == 'q':
        break
    elif no_of_songs == '0':
        no_of_songs = None

    albums.append(make_album(album_name, artist))
    if no_of_songs != None:
        albums[-1]['no_of_songs'] = int(no_of_songs)

print("\n")
print_album(albums)
```

Exercise 8.9 - Messages:

Listing 173: Chapter08/ex8.9.messages.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

messages = [
"A: Puh. Wie es hier aussieht!\nWo ist das Telefon?\nVielleicht in der Küche?",
"B: Nein. Hier ist kein Telefon!\nAber hier ist eine Uhr!",
"A: Oh, das ist die Uhr von Stefan, oder?",
"B: Stimmt! Ich schreibe Stefan.\nEr sucht die Uhr bestimmt...",
"A: Hmm, wo sind die Schlüssel?",
"B: Vielleicht im Wohnzimmer?",
"A: Nein, hier sind keine Schlüssel.",
"B: Ah, hier.",
"A: Super, danke.",
"A: Ah, es ist Stefans Uhr.\nAhm, Julia: Ist hier auch ein Rucksack?",
"B: Stefans Rucksack?\nNein. Tut mir leid.\nHier ist kein Rucksack."
]

def print_message(messages):
    for message in messages:
        print(f"{message}\n")

print_message(messages)
```

Exercise 8.10 - Sending Messages:

Listing 174: Chapter08/ex8.10.messages.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from time import sleep

```

```
messages = [
"A: Puh. Wie es hier aussieht!\nWo ist das Telefon?\nVielleicht in
    der Küche?",
"B: Nein. Hier ist kein Telefon!\nAber hier ist eine Uhr!",
"A: Oh, das ist die Uhr von Stefan, oder?",
"B: Stimmt! Ich schreibe Stefan.\nEr sucht die Uhr bestimmt...",
"A: Hmm, wo sind die Schlüssel?",
"B: Vielleicht im Wohnzimmer?",
"A: Nein, hier sind keine Schlüssel.",
"B: Ah, hier.",
"A: Super, danke.",
"A: Ah, es ist Stefans Uhr.\nAhm, Julia: Ist hier auch ein Rucksack?
    ",
"B: Stefans Rucksack?\nNein. Tut mir leid.\nHier ist kein Rucksack."
]

def print_message(messages):
    for message in messages:
        print(f"{message}\n")

sent_messages = []

def send_message(messages, sent_messages):
    while messages:
        current_message = messages.pop(0)
        print(f"Sending below message:\n...\n{current_message}\n..."
            )
        sent_messages.append(current_message)
        sleep(1)
        print("Message sent!\n")

print("Current mesages are:\n------\n")
print_message(messages)
send_message(messages[:], sent_messages)
print("------\nNow the messages are:\n------\n")
print_message(sent_messages)
```

Exercise 8.12 - Sandwiches:

Listing 175: Chapter08/ex8.12.sandwiches.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

def order_sandwich(*ingridents):
    print("Your order is:")
    for ingrident in ingridents:
        print(f"{ingrident.title()}")

order_sandwich('subway series', 'classic sandwiches')
order_sandwich('wraps', 'fresh melts')
order_sandwich('breakfast')
```

Exercise 8.13 - User Profile:

Listing 176: Chapter08/ex8.13.user_profile.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user
       ."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein', location='
   princeton', field='physics')
print(user_profile)

my_profile = build_profile('baga', 'shit', location='shit', food='
   rabbits')
print(my_profile)
```

Exercise 8.14 - Cars:

Listing 177: Chapter08/ex8.14.cars.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

cars = []

def make_car(manufacturer, model, **car):
    car['manufacturer'] = manufacturer
    car['model'] = model
    return car

car = make_car('subaru', 'outback', color = 'blue', tow_package =
   True)
cars.append(car)

for car in cars:
    print(car)
```

Exercise 8.15 - Printing Models:

Listing 178: Chapter08/ex8.15.printing_models.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from ex8_15_printing_functions import *

unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []

print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Chapter 9: Classes

Exercise Focus: Object-oriented programming with classes

Key Concepts Practiced:

- Creating classes and objects
- Defining methods and attributes
- Using inheritance
- Working with instances

Exercise Examples:

Exercise 9.1 - Restaurant:

Listing 179: Chapter09/ex9.1.restaurant.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Restaurant:
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"Restaurant Name: {self.restaurant_name.title()}")
        print(f"Cuisine Type: {self.cuisine_type.title()}")

    def open_restaurant(self):
        print(f"{self.restaurant_name.title()} is open now.")

sukiya = Restaurant('sukiya', 'japanese beef rice')
sukiya.describe_restaurant()
sukiya.open_restaurant()
```

Exercise 9.2 - Three Restaurants:

Listing 180: Chapter09/ex9.2.restaurants.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Restaurant:
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"Restaurant Name: {self.restaurant_name.title()}")
        print(f"Cuisine Type: {self.cuisine_type.title()}")

    def open_restaurant(self):
        print(f"{self.restaurant_name.title()} is open now.")

```

```python
sukiya = Restaurant('sukiya', 'japanese beef rice')
sukiya.describe_restaurant()
sukiya.open_restaurant()

hardees =  Restaurant('hardees', 'hamburger')
hardees.describe_restaurant()

abc = Restaurant('abc', 'western food')
abc.describe_restaurant()
```

Exercise 9.3 - Users:

Listing 181: Chapter09/ex9.3.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class users:
    def __init__(self, first_name, last_name, gender, staff_no):
        self.first_name = first_name
        self.last_name = last_name
        self.title = 'staff'
        self.gender = gender
        self.staff_no = staff_no

    def describe_user(self):
        print("Staff Profile:\n-------")
        print(f"Name: {self.first_name.title()} {self.last_name.
           title()}")
        print(f"Staff ID: {self.staff_no}")
        if self.gender == 'M':
            print("Gender: Male")
        elif self.gender == 'F':
            print("Gender: Female")
        elif self.gender == 'O':
            print("Gender: Other")
        print(f"Title: {self.title.title()}")

    def greet_user(self):
        print(f"Hello, {self.first_name.title()} {self.last_name.
           title()} !!!")

sowon = users('sowon', 'kim', 'F', '1234567')
sowon.describe_user()
sowon.greet_user()

pyo = users('pyo', 'pyo', 'O', '23456')
pyo.describe_user()
pyo.greet_user()

daniel = users('daniel', 'kang', 'M', '2134123')
daniel.describe_user()
daniel.greet_user()
```

Exercise 9.4 - Number Served:

Listing 182: Chapter09/ex9.4.restaurants.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Restaurant:
### Description of the class ###
# attributes
# restaurant_name : name of the restaurant
# cuisine_type : what sort of food can be eaten fron that restuarant
# number_served : number of customers that the restaurant has served
    ; default 0
### End of description ###

### Methods ###

    # __init__ : initialize the class
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type
        self.number_served = 0

    # describe_restaurant : print the information of restaurant
    def describe_restaurant(self):
        print(f"Restaurant Name: {self.restaurant_name.title()}")
        print(f"Cuisine Type: {self.cuisine_type.title()}")
        print(f"Number of Customers Served: {self.number_served}")

    # open_restaurant : print an message for siumating the opeing of
        that restaurant
    def open_restaurant(self):
        print(f"{self.restaurant_name.title()} is open now.")

    # set_number_served : set the number of customers that have been
        served
    def set_number_served(self, numbers):
        self.number_served = numbers
        print(f"The new number of customers served becomes {self.
           number_served}.")

    # increment_numbers_served : increase the number of customers
       who've been served
    def increment_numbers_served(self, increment):
        self.number_served += increment
        print(f"Addind {increment} customers, the number of
           customers served is {self.number_served}.")

### End of Methods ###

sukiya = Restaurant('sukiya', 'japanese beef rice')
sukiya.describe_restaurant()
```

```python

print('\n')
hardees =  Restaurant('hardees', 'hamburger')
hardees.describe_restaurant()

print('\n')
abc = Restaurant('abc', 'western food')
abc.open_restaurant()
abc.describe_restaurant()

print("------")

print(f"\nSet the number of customers of {sukiya.restaurant_name.
    title()} -")
sukiya.set_number_served(100)

print(f"\nSet the number of customers of {hardees.restaurant_name.
    title()} -")
hardees.set_number_served(10000)

new_customer = 1
print(f"\nThere are {new_customer} customers coming in {abc.
    restaurant_name.title()} -")
abc.increment_numbers_served(new_customer)

print("------")

print("\nShow restaurants' information again:")
sukiya.describe_restaurant()
print('\n')
hardees.describe_restaurant()
print('\n')
abc.describe_restaurant()
```

Exercise 9.5 - Login Attempts:

Listing 183: Chapter09/ex9.5.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class users:
### Description of the class ###
# attributes
# first_name : first name of the user
# last_name : last name of the user
# gender : gender of the user ; 'M' is male, 'F' is female, 'O' is
    others / transgender
# staff_no : staff ID
# login_attempts : number of trials for login
### End of description ###

### Methods ###

```

```python
    # __init__ : initialize the class
    def __init__(self, first_name, last_name, gender, staff_no):
        self.first_name = first_name
        self.last_name = last_name
        self.title = 'staff'
        self.gender = gender
        self.staff_no = staff_no
        self.login_attempts = 0

    # describe_user : show descriptions of the user
    def describe_user(self):
        print("Staff Profile:\n-------")
        print(f"Name: {self.first_name.title()} {self.last_name.title()}")
        print(f"Staff ID: {self.staff_no}")
        if self.gender == 'M':
            print("Gender: Male")
        elif self.gender == 'F':
            print("Gender: Female")
        elif self.gender == 'O':
            print("Gender: Other")
        print(f"Title: {self.title.title()}")

    # greet_user : greet to user after login success
    def greet_user(self):
        print(f"Hello, {self.first_name.title()} {self.last_name.title()} !!!")

    # increment_login_attempts : increase number of attempts by 1 for each failed login trials
    def increment_login_attempts(self):
        self.login_attempts += 1
        print(f"Now {self.staff_no}'s login attempt number is {self.login_attempts}.")

    # reset_login_attempts : set the login attempts to zero
    def reset_login_attempts(self):
        self.login_attempts = 0
        print(f"Now {self.staff_no}'s login attempt number is {self.login_attempts}.")

### End of Methods ###

sowon = users('sowon', 'kim', 'F', '1234567')
sowon.describe_user()
sowon.greet_user()

print('\n')

pyo = users('pyo', 'pyo', 'O', '23456')
pyo.describe_user()
pyo.greet_user()
for value in range(0,3):
    print(f"{pyo.staff_no} login failed:")
    pyo.increment_login_attempts()
print("Finally login successed:")
pyo.reset_login_attempts()

print('\n')
daniel = users('daniel', 'kang', 'M', '2134123')
daniel.describe_user()
daniel.greet_user()
```

Exercise 9.6 - Ice Cream Stand:

Listing 184: Chapter09/ex9.6.restaurants.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

class Restaurant:
### Description of the class ###
# attributes
# restaurant_name : name of the restaurant
# cuisine_type : what sort of food can be eaten fron that restuarant
# number_served : number of customers that the restaurant has served
    ; default 0
### End of description ###

### Methods ###

    # __init__ : initialize the class
    def __init__(self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type
        self.number_served = 0

    # describe_restaurant : print the information of restaurant
    def describe_restaurant(self):
        print(f"Restaurant Name: {self.restaurant_name.title()}")
        print(f"Cuisine Type: {self.cuisine_type.title()}")
        print(f"Number of Customers Served: {self.number_served}")

    # open_restaurant : print an message for siumating the opeing of
        that restaurant
    def open_restaurant(self):
        print(f"{self.restaurant_name.title()} is open now.")

    # set_number_served : set the number of customers that have been
        served
    def set_number_served(self, numbers):
        self.number_served = numbers
        print(f"The new number of customers served becomes {self.
           number_served}.")

    # increment_numbers_served : increase the number of customers who've been served
    def increment_numbers_served(self, increment):
        self.number_served += increment
        print(f"Addind {increment} customers, the number of customers served is {self.number_served}.")

### End of Methods ###

class IceCreamStand(Restaurant):
### Description of the class ###
# child class of Restaurant
# flavors : a List of ice-crean flavors
### End of description ###

### Methods ###

    # __init__ : initialize the clsss
    def __init__(self, restaurant_name, cuisine_type, flavors):
        super().__init__(restaurant_name, cuisine_type)
        self.flavors = flavors[:]

    # describe_restaurant : add ice-cream flavors available
    def describe_restaurant(self):
        super().describe_restaurant()
        print("Ice-Cream flavors available:")
        for flavor in self.flavors:
            print(f"{flavor.title()}")

### End of Methods ###

sukiya = Restaurant('sukiya', 'japanese beef rice')
sukiya.describe_restaurant()

print('\n')
hardees =  Restaurant('hardees', 'hamburger')
hardees.describe_restaurant()

print('\n')
abc = Restaurant('abc', 'western food')
abc.open_restaurant()
abc.describe_restaurant()

print("------")

print(f"\nSet the number of customers of {sukiya.restaurant_name.title()} -")
sukiya.set_number_served(100)

print(f"\nSet the number of customers of {hardees.restaurant_name.title()} -")
```

```python
hardees.set_number_served(10000)

new_customer = 1
print(f"\nThere are {new_customer} customers coming in {abc.
    restaurant_name.title()} -")
abc.increment_numbers_served(new_customer)

print("------")

print("\nShow restaurants' information again:")
sukiya.describe_restaurant()
print('\n')
hardees.describe_restaurant()
print('\n')
abc.describe_restaurant()

print("------")

appolo = IceCreamStand('appolo', 'ice cream stand', ['chocolate', '
    vanilla'])
appolo.describe_restaurant()
```

Exercise 9.7 - Admin:

Listing 185: Chapter09/ex9.7.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

#### CLASS SETUP ####

### User Class :
# attributes
# first_name : first name of the user
# last_name : last name of the user
# gender : gender of the user ; 'M' is male, 'F' is female, 'O' is
    others / transgender
# staff_no : staff ID
# login_attempts : number of trials for login

class users:

    # __init__ : initialize the class
    def __init__(self, first_name, last_name, gender, staff_no):
        self.first_name = first_name
        self.last_name = last_name
        self.title = 'staff'
        self.gender = gender
        self.staff_no = staff_no
        self.login_attempts = 0

    # describe_user : show descriptions of the user
    def describe_user(self):
        print("Staff Profile:\n-------")
        print(f"Name: {self.first_name.title()} {self.last_name.
           title()}")
        print(f"Staff ID: {self.staff_no}")
        if self.gender == 'M':
            print("Gender: Male")
        elif self.gender == 'F':
            print("Gender: Female")
        elif self.gender == 'O':
            print("Gender: Other")
        print(f"Title: {self.title.title()}\n------")

    # greet_user : greet to user after login success
    def greet_user(self):
        print(f"Hello, {self.first_name.title()} {self.last_name.
           title()} !!!")

    # increment_login_attempts : increase number of attempts by 1
       for each failed login trials
    def increment_login_attempts(self):
        self.login_attempts += 1
        print(f"Now {self.staff_no}'s login attempt number is {self.
           login_attempts}.")

    # reset_login_attempts : set the login attempts to zero
    def reset_login_attempts(self):
        self.login_attempts = 0
        print(f"Now {self.staff_no}'s login attempt number is {self.
           login_attempts}.")

### Admin class

### Admin Class :
# inheritance of user class
# privileges : the abilities of an admin

class admin(users):

    # __init__ : initialize the class
    def __init__(self, first_name, last_name, gender, staff_no):
        super().__init__(first_name, last_name, gender, staff_no)
        self.privileges = ['can add post', 'can delete post', 'can
           ban user']

    # show_privileges : show admin's privileges
    def show_privileges(self):
        for privilege in self.privileges:
            print(f"{privilege.title()}")

### END OF CLASS SETUP ####

sowon = users('sowon', 'kim', 'F', '1234567')
```

```python
sowon.describe_user()
sowon.greet_user()

print('\n')

pyo = users('pyo', 'pyo', 'O', '23456')
pyo.describe_user()
pyo.greet_user()
for value in range(0,3):
    print(f"\n{pyo.staff_no} login failed:")
    pyo.increment_login_attempts()
print("Finally login successed:")
pyo.reset_login_attempts()

print('\n')
daniel = users('daniel', 'kang', 'M', '2134123')
daniel.describe_user()
daniel.greet_user()

yerin = admin('yerin', 'jung', 'F', '23141234')
yerin.describe_user()
yerin.greet_user()
yerin.show_privileges()
```

Exercise 9.8 - Privileges:

Listing 186: Chapter09/ex9.8.users.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

#### CLASS SETUP ####

### User Class :
# attributes
# first_name : first name of the user
# last_name : last name of the user
# gender : gender of the user ; 'M' is male, 'F' is female, 'O' is
   others / transgender
# staff_no : staff ID
# login_attempts : number of trials for login

class users:

    # __init__ : initialize the class
    def __init__(self, first_name, last_name, gender, staff_no):
        self.first_name = first_name
        self.last_name = last_name
        self.title = 'staff'
        self.gender = gender
        self.staff_no = staff_no
        self.login_attempts = 0

    # describe_user : show descriptions of the user
    def describe_user(self):
        print("Staff Profile:\n-------")
        print(f"Name: {self.first_name.title()} {self.last_name.
           title()}")
        print(f"Staff ID: {self.staff_no}")
        if self.gender == 'M':
            print("Gender: Male")
        elif self.gender == 'F':
            print("Gender: Female")
        elif self.gender == 'O':
            print("Gender: Other")
        print(f"Title: {self.title.title()}\n------")

    # greet_user : greet to user after login success
    def greet_user(self):
        print(f"Hello, {self.first_name.title()} {self.last_name.
           title()} !!!")

    # increment_login_attempts : increase number of attempts by 1
       for each failed login trials
    def increment_login_attempts(self):
        self.login_attempts += 1
        print(f"Now {self.staff_no}'s login attempt number is {self.
           login_attempts}.")

    # reset_login_attempts : set the login attempts to zero
    def reset_login_attempts(self):
        self.login_attempts = 0
        print(f"Now {self.staff_no}'s login attempt number is {self.
           login_attempts}.")

### Admin Class :
# inheritance of user class
# privileges : the abilities of an admin

class admin(users):

    # __init__ : initialize the class
    def __init__(self, first_name, last_name, gender, staff_no):
        super().__init__(first_name, last_name, gender, staff_no)
        self.privileges = privileges()

    # show_privileges : show admin's privileges
    def show_privileges(self):
        self.privileges.show_privileges()

### Privileges Class :
# privileges : stor the abilities

class privileges():

    # __init__ : initialize the class
    def __init__(self):
        self.privileges = ['can add post', 'can delete post', 'can
           ban user']

    # show_privileges : show admin's privileges
    def show_privileges(self):
        for privilege in self.privileges:
            print(f"{privilege.title()}")

### END OF CLASS SETUP ####

sowon = users('sowon', 'kim', 'F', '1234567')
sowon.describe_user()
sowon.greet_user()

print('\n')

pyo = users('pyo', 'pyo', 'O', '23456')
pyo.describe_user()
pyo.greet_user()
for value in range(0,3):
    print(f"\n{pyo.staff_no} login failed:")
    pyo.increment_login_attempts()
print("Finally login successed:")
pyo.reset_login_attempts()

print('\n')
daniel = users('daniel', 'kang', 'M', '2134123')
daniel.describe_user()
daniel.greet_user()

yerin = admin('yerin', 'jung', 'F', '23141234')
yerin.describe_user()
yerin.greet_user()
yerin.show_privileges()
```

Exercise 9.13 - Dice:

Listing 187: Chapter09/ex9.13.dice.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from random import randint

class Dice():
### attributes ###
# sides : no. of sides of the dice

    # _init_ : initialize the dice class
    def __init__(self, sides = 6):
        self.sides = sides

    # draw_dice : draw a dice, and then give out an result (integer)
    def roll_dice(self):
        return randint(1, self.sides)

    # show_dice : tell user how many sides this dice has
    def show_dice(self):
        print(f"This dice has {self.sides} sides.\n")

### End of class ###

dice1 = Dice();
dice1.show_dice();
for i in range(1,11):
    print(f"Draw # {i} : {dice1.roll_dice()}")

print("\n---\n")
dice2 = Dice(10)
dice2.show_dice();
for i in range(1,11):
    print(f"Draw # {i} : {dice2.roll_dice()}")

print("\n---\n")
dice3 = Dice(20)
dice3.show_dice();
for i in range(1,11):
    print(f"Draw # {i} : {dice3.roll_dice()}")
```

Chapter 10: Files and Exceptions

Exercise Focus: File handling and error management

Key Concepts Practiced:

- Reading and writing files
- Handling exceptions
- Working with different file formats
- Error handling strategies

Exercise Examples:

Exercise 10.1 - Learning Python:

Listing 188: Chapter10/ex10.1.learning_python/learning_python.py

```
# Python Crash Course, 2Ed, writtern by Eric Matthes

filename = 'learning_python.txt'

# first time : print the contents once by reading in the entire file
    .

```

```python
with open(filename) as file_object:
    contents = file_object.read()

print(contents)

# second time : print the contents by looping over the file object

with open(filename) as file_object:
    for line in file_object:
        print(line.rstrip())

# third time : print the contents by storing the lines in a list and
    the working with them outside the with block

with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line.rstrip())
```

Exercise 10.3 - Guest:

Listing 189: Chapter10/ex10.3.guest/guest.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

filename = 'guest.txt'

with open(filename, 'w') as file_object:
    name = input("Input your name >> ")
    file_object.write(name)
```

Exercise 10.4 - Guest Book:

Listing 190: Chapter10/ex10.4.guest_book/guest_book.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

filename = 'guest_book.txt'

with open(filename, 'w') as file_object:
    while True:
        name = input("Input your name >> ")
        if (name[:].lower() == 'q') :
            break
        else :
            file_object.write(f"{name}\n")
```

Exercise 10.5 - Programming Poll:

Listing 191: Chapter10/ex10.5.programming_poll/programming_poll.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

filename = 'programming_poll.txt'
```

```python

with open(filename, 'w') as file_object:
    while True:
        name = input("Input your name >> ")
        if (name[:].lower() == 'q') :
            break
        else :
            reason = input(f"{name}, why do you like programming? >>
                ")
            file_object.write(f"{name} : {reason}\n")
```

Exercise 10.6 - Addition:

Listing 192: Chapter10/ex10.6.addition.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def addition (num1 , num2):
    return num1 + num2

def get_input():
    num = input("Enter the number >> ")
    try:
        int(num)
    except ValueError:
        print("The input is not digit.\nPlease try again.")
        return None
    else:
        return int(num)

print("The first number:")
x = get_input()
print("The second number:")
y = get_input()

if (x and y) != False :
    print(f"{x} + {y} = {x + y}")
```

Exercise 10.7 - Addition Calculator:

Listing 193: Chapter10/ex10.7.addition.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

while True:
    x = input("Enter the first number (or enter \'q\' to quit) >> ")
    if x.lower() == 'q':
        break
    y = input("Enter the second number (or enter\'q\' to quit) >> ")
    if y.lower() == 'q':
        break

    try:
        int(x)
        int(y)
    except ValueError:
        print("One of the numbers are not integers.\nTry again.")
    else:
        print(f"{int(x)} + {int(y)} = {int(x) + int(y)}")
```

Exercise 10.8 - Cats and Dogs:

Listing 194: Chapter10/ex10.8.pets/pets.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

pet_files = ['cats.txt', 'dogs.txt', 'mice.txt']

def print_pet(filename):
    try:
        with open(filename) as f:
            pet_names = f.read()
    except:
        print(f"There is no {filename}.")
        return None
    else:
        return pet_names

for pet_file in pet_files:
    message = print_pet(pet_file)
    if message != None:
        print(message)
```

Exercise 10.11 - Favorite Number:

Listing 195: Chapter10/ex10.11.favourite_number/input.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import json

filename = 'data.json'

while True:
    str = input("Input your favourite number >> ")
    try:
        fav_num = int(str)
    except ValueError:
        print("You have not input integer.\nPlease enter again.")
        continue;
    else:
        print("Your favourite number has been recorded.")
        break

with open(filename, 'w') as f:
    json.dump(fav_num, f)
```

Exercise 10.12 - Favorite Number Remembered:

Listing 196: Chapter10/ex10.12.favourite_number/favourite_number.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from read_data import get_fav_num
from write_data import record_fav_num

fav_num = get_fav_num()
if fav_num != None:
    print(f"I know your favourite number! it's {fav_num}.")
else:
    record_fav_num()
```

Exercise 10.13 - Verify User:

Listing 197: Chapter10/ex10.13.remember_me/remember_me.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import json
import os

def get_stored_username():

    """Get stored username if available."""

    filename = 'username.json'
    if os.path.getsize(filename) > 0:
        try:
            with open(filename) as f:
                username = json.load(f)
        except FileNotFoundError:
            return None
        else:
            return username
    else:
        return None

def greet_user():

    """ Greet the user by name. """

    username = get_stored_username()
    if username:
        confirmation = input(f"Are you {username}?\n(Input \'Y\' or
           \'y\' if yes, other input will be cosidered as no.)\n>> "
           ).lower()
        if confirmation == 'y':
            print(f"Welcome back, {username}!")
        else:
            username = get_new_username()
            print(f"We'll remember you whe you come back, {username
               }!")
    else:
        username = get_new_username()
        print(f"We'll remember you whe you come back, {username}!")

def get_new_username():

    """Pronpt for a new username."""

    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f:
        json.dump(username, f)
    return username

greet_user()
```

Chapter 11: Testing Your Code

Exercise Focus: Writing tests and test-driven development

Key Concepts Practiced:

- Writing unit tests
- Using the unittest framework
- Testing different scenarios
- Test-driven development

Exercise Examples:

Exercise 11.1 - City, Country:

Listing 198: Chapter11/name_function.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

def get_formatted_name(first, last, middle=''):
    """Generate a neatly formatter full name"""
    if middle:
        full_name = f"{first} {middle} {last}"
    else:
        full_name = f"{first} {last}"
    return full_name.title()
```

Exercise 11.2 - Population:

Listing 199: Chapter11/test_name_function.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

import unittest
from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Test for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')

    def test_first_last_middle_name(self):
        """Do names like 'Wolfgang Amadeus Mozart' work?"""
        formatted_name = get_formatted_name('wolfgang', 'mozart', '
           amadeus')
        self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')

if __name__ == '__main__':
    unittest.main()
```

Exercise 11.3 - Employee:

Listing 200: Chapter11/language_survey.py

```python
# Python Crash Course, 2Ed, writtern by Eric Matthes

from survey import AnonymousSurvey

# Define a question, and make a survey.
question = "What language did you first learn to speak?"
my_survey = AnonymousSurvey(question)

# Show the question, and store responses to the question.
my_survey.show_question()
print("Enter 'q' at any time to quit.\n")
while True:
    response = input("Language: ")
    if response == 'q':
        break
    my_survey.store_response(response)

# Show the survey results.
print("\nThank you o everyone who participated in the survey!")
my_survey.show_results()
```

Summary of Exercises

The exercises provide comprehensive practice covering:

- **85+ exercise files** across all chapters
- **Progressive difficulty** from basic to advanced concepts
- **Real-world applications** and practical examples
- **Hands-on coding practice** with immediate feedback

- **Concept reinforcement** through varied problem types

Exercise Categories

1. **Basic Syntax**: Variables, print statements, data types
2. **Data Structures**: Lists, dictionaries, tuples
3. **Control Flow**: if statements, loops, functions
4. **Object-Oriented Programming**: Classes, inheritance, methods
5. **File Operations**: Reading, writing, exception handling
6. **Testing**: Unit tests, test cases, test-driven development

How to Use These Exercises

1. **Complete exercises sequentially** within each chapter
2. **Modify and experiment** with the code examples
3. **Create your own variations** of the exercises
4. **Test your understanding** by explaining the code
5. **Build upon concepts** from previous chapters

These exercises provide essential practice for mastering Python programming concepts and building confidence in writing real Python code.