Exercises and Solutions: Python Crash Course (2nd Ed.) Chapters 4–6

Chapter 4: Working with Lists

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.

- Modify your for loop to print a sentence using the name of the pizza instead of printing just the name of the pizza. For each pizza you should have one line of output containing a simple statement like *I like pepperoni pizza.*
- Add a line at the end of your program, outside the for loop, that states how much you like pizza. The output should consist of three or more lines about the kinds of pizza you like and then an additional sentence, such as *I really love pizza!*

```
favorite_pizzas = ["pepperoni", "margherita", "hawaiian"]

# Print each pizza name

for pizza in favorite_pizzas:

    print(pizza)

print()


# Print a sentence about each pizza

for pizza in favorite_pizzas:

    print(f"I like {pizza} pizza.")

print("I really love pizza!")
```

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.

- Modify your program to print a statement about each animal, such as *A dog would make a great pet.*
- Add a line at the end of your program stating what these animals have in common. You could print a sentence such as *Any of these animals would make a great pet!*

```
animals = ["dog", "cat", "rabbit"]
```

```
# Print each animal name
for animal in animals:
    print(animal)
print()


# Print a statement about each animal
for animal in animals:
    print(f"A {animal} would make a great pet.")
print("Any of these animals would make a great pet!")
```

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

```
for number in range(1, 21):
    print(number)
```

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. (If the output is taking too long, stop it by pressing Ctrl-C or by closing the output window.)

```
numbers = list(range(1, 1_000_001))
for number in numbers:
    print(number)
```

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. Also, use the sum() function to see how quickly Python can add a million numbers.

```
numbers = list(range(1, 1_000_001))
print(min(numbers))  # should be 1
print(max(numbers))  # should be 1000000
print(sum(numbers))  # sum of 1 to 1,000,000
```

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.

```
odd_numbers = list(range(1, 21, 2))

for number in odd_numbers:
    print(number)
```

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.

```
threes = [value for value in range(3, 31, 3)]

for value in threes:
    print(value)
```

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.

```
cubes = []

for value in range(1, 11):
    cubes.append(value**3)

for cube in cubes:
    print(cube)
```

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

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

print(cubes)
```

4-10. Slices: Using one of the programs you wrote in this chapter (such as the list of numbers), add several lines to the end of the program that do the following:

- Print the message *The first three items in the list are:* then use a slice to print the first three items from the list.
- Print the message *Three items from the middle of the list are:* then print three items from the middle of the list.
- Print the message *The last three items in the list are:* then print the last three items in the list.

```
numbers = list(range(1, 10))  # example list
```

```
print("The first three items in the list are:")
for num in numbers[:3]:
    print(num)
print("Three items from the middle of the list are:")
for num in numbers[3:6]:
    print(num)
print("The last three items in the list are:")
for num in numbers[-3:]:
    print(num)
```

4-11. My Pizzas, Your Pizzas: Start with your program from Exercise 4-1 (page 56). Make a copy of the list of pizzas, and call it friend_pizzas. Then, do the following:

- Add a new pizza to the original list.
- Add a different pizza to the list friend_pizzas.
- Prove that you have two separate lists. Print the message *My favorite pizzas are:* and then use a for loop to print the first list. Print the message *My friend's favorite pizzas are:* and then use a for loop to print the second list. Make sure each new pizza is stored in the appropriate list.

```
pizzas = ["pepperoni", "margherita", "hawaiian"]
friend_pizzas = pizzas[:]  # copy list

pizzas.append("veggie")
friend_pizzas.append("bbq chicken")

print("My favorite pizzas are:")
for pizza in pizzas:
    print(pizza)
```

```
print("\nMy friend's favorite pizzas are:")
for pizza in friend_pizzas:
    print(pizza)
```

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.

```
my_foods = ["pizza", "falafel", "carrot cake"]
friend_foods = ["ice cream", "cookies", "brownies"]


for food in my_foods:
    print(food)


for food in friend_foods:
    print(food)
```

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

- Use a for loop to print each food the restaurant offers.
- Try to modify one of the items, and make sure that Python rejects the change.
- The restaurant changes its menu, replacing two of the items with different foods. Add a line that rewrites the tuple, and then use a for loop to print each of the items on the revised menu.

```
buffet_foods = ("pizza", "pasta", "salad", "soup", "burger")
# Print each food
for food in buffet_foods:
    print(food)


# This will cause an error because tuples are immutable
try:
```

```
    buffet_foods[0] = "ice cream"
except TypeError as e:
    print(f"Error: {e}")


# Rewrite the tuple with new items
buffet_foods = ("nachos", "tacos", "burrito", "chips", "salsa")
for food in buffet_foods:
    print(food)
```

4-14. PEP 8: Look through the original PEP 8 style guide at https://python.org/dev/peps/pep-0008/. You won't use much of it now, but it might be interesting to skim through it.

```
# This is a simple example following PEP 8 style guide.


# Constants should be written in uppercase
MAX_LENGTH = 79


# Function names use lowercase with underscores
def calculate_area(radius):
    """Calculate the area of a circle given its radius."""
    pi = 3.14159
    return pi * radius ** 2



# Class names use CapitalizedWords
class Circle:
    """Represents a circle with a given radius."""
```

```
    def __init__(self, radius):
        """Initialize the circle with a given radius."""
        self.radius = radius

    def get_area(self):
        """Return the area of the circle."""
        return calculate_area(self.radius)

# Using the class and function
if __name__ == "__main__":
    # Instance of the class
    my_circle = Circle(5)

    # Print the area of the circle
    print("Area of the circle:", my_circle.get_area())
```

4-15. Code Review: Choose three of the programs you've written in this chapter and modify each one to comply with PEP 8:

- Use four spaces for each indentation level. (Set your text editor to insert four spaces every time you press Tab, if you haven't already done so.)
- Use less than 80 characters on each line, and set your editor to show a vertical guideline at the 80th character position.
- Don't use blank lines excessively in your program files.

Solution: Here are three revised examples:

1. **Pizzas (Exercise 4-1):**

```
pizzas = ['margherita', 'pepperoni', 'hawaiian']
for pizza in pizzas:
    print(pizza)
```

2. **Odd Numbers (Exercise 4-6):**

```
odd_numbers = list(range(1, 21, 2))
for odd in odd_numbers:
    print(odd)
```

3. **Cubes (Exercise 4-8):**

```
cubes = [value**3 for value in range(1, 11)]
for cube in cubes:
    print(cube)
```

All examples:

- Use 4-space indentation.
- Keep each line under 80 characters.
- Avoid excessive blank lines.
- Use consistent and readable formatting in line with PEP 8.

Chapter 5: if Statements

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. Your code should look something like this:

```
car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')
```

- Look closely at your results, and make sure you understand why each line evaluates to True or False.
- Create at least ten tests. Have at least five tests evaluate to True and another five tests evaluate to False.

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

```
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')

age = 30
print("\nIs age == 30? I predict True.")
print(age == 30)
print("Is age < 25? I predict False.")
print(age < 25)

fruit = 'apple'
print("\nIs fruit == 'Apple'? I predict False.")
print(fruit == 'Apple')
print("Is fruit.lower() == 'apple'? I predict True.")
print(fruit.lower() == 'apple')

numbers = [1, 2, 3, 4, 5]
print("\nIs 3 in numbers? I predict True.")
print(3 in numbers)
print("Is 10 in numbers? I predict False.")
print(10 in numbers)

print("\nIs 2 not in numbers? I predict False.")
print(2 not in numbers)
print("Is age >= 30 and fruit == 'apple'? I predict True.")
print(age >= 30 and fruit == 'apple')
```

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

to conditional_tests.py. Have at least one True and one False result for each of the following:

- Tests for equality and inequality with strings
- Tests using the lower() method
- Numerical tests involving equality and inequality, greater than and less than, greater than or equal to, and less than or equal to
- Tests using the and keyword and the or keyword
- Tests whether an item is in a list
- Tests whether an item is not in a list

```
print("Tests for equality and inequality with strings:")

text = "Hello"

print(text == "Hello")

print(text != "World")


print("\nTests using the lower() method:")

name = "Python"

print(name.lower() == "python")

print(name.lower() == "java")


print("\nNumerical tests involving equality and inequality:")

number = 10

print(number == 10)

print(number > 20)

print(number < 20)

print(number >= 10)

print(number <= 5)
```

```
print("\nTests using the and keyword and the or keyword:")
x = 5
print(x > 3 and x < 10)
print(x < 3 or x > 10)

print("\nTest whether an item is in a list:")
fruits = ["apple", "banana", "cherry"]
print("apple" in fruits)
print("pear" in fruits)

print("\nTest whether an item is not in a list:")
print("apple" not in fruits)
print("pear" not in fruits)
```

5-3. Alien Colors #1: 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'.

- Write an if statement to test whether the alien's color is green. If it is, print a message that the player just earned 5 points.
- Write one version of this program that passes the if test and another that fails. (The version that fails will have no output.)

```
# Version that passes the test
alien_color = 'green'
if alien_color == 'green':
    print("You just earned 5 points!")

# Version that fails the test
alien_color = 'red'
if alien_color == 'green':
    print("You just earned 5 points!")
```

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

- If the alien's color is green, print a statement that the player just earned 5 points for shooting the alien.
- If the alien's color isn't green, print a statement that the player just earned 10 points.

```
alien_color = 'yellow'

if alien_color == 'green':
    print("You just earned 5 points for shooting the alien.")

else:
    print("You just earned 10 points.")
```

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

- If the alien is green, print a message that the player just earned 5 points.
- If the alien is yellow, print a message that the player just earned 10 points.
- If the alien is red, print a message that the player just earned 15 points.
- Write three versions of this program, making sure each message is printed for the appropriate color alien.

```
# Alien is green

alien_color = 'green'

if alien_color == 'green':
    print("You just earned 5 points.")

elif alien_color == 'yellow':
    print("You just earned 10 points.")

elif alien_color == 'red':
    print("You just earned 15 points.")


# Alien is yellow
```

```
alien_color = 'yellow'
if alien_color == 'green':
    print("You just earned 5 points.")
elif alien_color == 'yellow':
    print("You just earned 10 points.")
elif alien_color == 'red':
    print("You just earned 15 points.")


# Alien is red
alien_color = 'red'
if alien_color == 'green':
    print("You just earned 5 points.")
elif alien_color == 'yellow':
    print("You just earned 10 points.")
elif alien_color == 'red':
    print("You just earned 15 points.")
```

5-6. Stages of Life: Write an if-elif-else chain that determines a person's stage of life. Set a value for the variable age, and then:

- If the person is less than 2 years old, print a message that the person is a baby.
- If the person is at least 2 years old but less than 4, print a message that the person is a toddler.
- If the person is at least 4 years old but less than 13, print a message that the person is a kid.
- If the person is at least 13 years old but less than 20, print a message that the person is a teenager.
- If the person is at least 20 years old but less than 65, print a message that the person is an adult.

- If the person is age 65 or older, print a message that the person is an elder.

```
age = 15

if age < 2:
    print("The person is a baby.")
elif age < 4:
    print("The person is a toddler.")
elif age < 13:
    print("The person is a kid.")
elif age < 20:
    print("The person is a teenager.")
elif age < 65:
    print("The person is an adult.")
else:
    print("The person is an elder.")
```

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.

- Make a list of your three favorite fruits and call it favorite_fruits.
- Write five if statements. Each should check whether a certain kind of fruit is in your list. If the fruit is in your list, the if block should print a statement, such as *You really like bananas!*.

```
favorite_fruits = ["apple", "banana", "cherry"]

if "apple" in favorite_fruits:
    print("You really like apple!")
if "banana" in favorite_fruits:
    print("You really like banana!")
if "cherry" in favorite_fruits:
    print("You really like cherry!")
```

```
if "mango" in favorite_fruits:
    print("You really like mango!")
if "grape" in favorite_fruits:
    print("You really like grape!")
```

5-8. Hello Admin: Make a list of five or more usernames, including the name 'admin'. Imagine you are writing code that will print a greeting to each user after they log in to a website. Loop through the list, and print a greeting to each user:

- If the username is 'admin', print a special greeting, such as *Hello admin, would you like to see a status report?*
- Otherwise, print a generic greeting, such as *Hello Jaden, thank you for logging in again.*

```
usernames = ["admin", "john", "alice", "bob", "jane"]
for user in usernames:
    if user == "admin":
        print("Hello admin, would you like to see a status report?")
    else:
        print(f"Hello {user}, thank you for logging in again.")
```

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

- If the list is empty, print the message *We need to find some users!*
- Otherwise, print a greeting to each user as before.

```
usernames = []  # empty list
if not usernames:
    print("We need to find some users!")
else:
    for user in usernames:
        print(f"Hello {user}, thank you for logging in again.")
```

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

- Make a list of five or more usernames called current_users.
- Make another list of five usernames called new_users. Make sure one or two of the new usernames are also in the current_users list.
- Loop through the new_users list to see if each new username has already been used.
    - If it has, print a message that the person will need to enter a new username.
    - If a username has not been used, print a message saying that the username is available.
- Make sure your comparison is case insensitive. If 'John' has been used, 'JOHN' should not be accepted. (To do this, you'll need to make a copy of current_users containing the lowercase versions of all existing users.)

```
current_users = ['alice', 'Bob', 'carol', 'dave', 'Eve']

new_users = ['Frank', 'ALICE', 'eve', 'Grace', 'Heidi']


current_users_lower = [user.lower() for user in current_users]

for new_user in new_users:
    if new_user.lower() in current_users_lower:
        print(f"{new_user}: You will need to enter a new username.")
    else:
        print(f"{new_user}: The username is available.")
```

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.

- Store the numbers 1 through 9 in a list.
- Loop through the list.
- Use an if-elif-else chain inside the loop to print the proper ordinal ending for each number. Your output should read *1st 2nd 3rd 4th 5th 6th 7th 8th 9th*, and each result should be on a separate line.

```
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")
```

5-12. Styling if statements: Review the programs you wrote in this chapter, and make sure you styled your conditional tests appropriately (following PEP 8 guidelines on indentation, line length, etc.).

Solution: PEP 8 Guidelines Applied:

- **Indentation**: 4 spaces per level.
- **Line Length**: Limited to 79 characters.
- **Spacing**: Use one space on each side of comparison operators (e.g., ==, >=).
- **Blank Lines**: Use blank lines to separate logical sections of code.
- **Function Definitions**: Two blank lines before a function definition at top level.
- **Example:**

```
# Correctly styled conditional
age = 25
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

5-13. Your Ideas: At this point, you're a more capable programmer than you were when you started this book. Now that you have a better sense of how real-world situations are modeled in programs, you might be thinking of some

problems you could solve with your own programs. Record any new ideas you have about problems you might want to solve as your programming skills continue to improve. Consider games you might want to write, data sets you might want to explore, and web applications you'd like to create.

Example: Student Report System A simple program to input student grades and print a summary report.

```
students = {
    'Alice': [85, 90, 78],
    'Bob': [70, 68, 72],
    'Charlie': [95, 92, 88],
    'Diana': [100, 98, 99]
}


for name, grades in students.items():
    average = sum(grades) / len(grades)
    print(f"
Student: {name}")
    print(f"Grades: {grades}")
    print(f"Average: {average:.2f}")
    if average >= 90:
        print("Performance: Excellent")
    elif average >= 75:
        print("Performance: Good")
    else:
        print("Performance: Needs Improvement")
```

Chapter 6: Dictionaries

6-1. Person: Use a dictionary to store information about a person you know. Store their first name, last name, age, and the city in which they live. You should

have keys such as first_name, last_name, age, and city. Print each piece of information stored in your dictionary.

```
person = {
    'first_name': 'John',
    'last_name': 'Doe',
    'age': 30,
    'city': 'New York'
}
print(person['first_name'])
print(person['last_name'])
print(person['age'])
print(person['city'])
```

6-2. Favorite Numbers: Use a dictionary to store people's favorite numbers. Think of five names, and use them as keys in your dictionary. Think of a favorite number for each person, and store each as a value in your dictionary. Print each person's name and their favorite number. For even more fun, poll a few friends and get some actual data for your program.

```
favorite_numbers = {
    'Alice': 7,
    'Bob': 3,
    'Carol': 9,
    'Dave': 1,
    'Eve': 5
}
for name, number in favorite_numbers.items():
    print(f"{name}'s favorite number is {number}.")
```

6-3. Glossary: A Python dictionary can be used to model an actual dictionary. However, to avoid confusion, let's call it a glossary.

- Think of five programming words you've learned about in the previous chapters. Use these words as the keys in your glossary, and store their meanings as values.
- Print each word and its meaning as neatly formatted output. You might print the word followed by a colon and then its meaning, or print the word on one line and then the meaning on the next line.

```
glossary = {
    'variable': 'a named space in memory that holds a value',
    'list': 'a collection of items in a particular order',
    'dictionary': 'a collection of key-value pairs',
    'loop': 'a block of code that repeats',
    'function': 'a block of code that performs a specific task'
}
for word, meaning in glossary.items():
    print(f"{word.title()}: {meaning}")
```

6-4. Glossary 2: Now that you know how to loop through a dictionary, clean up the code from Exercise 6-3 by replacing your series of print() calls with a loop that runs through the dictionary's keys and values. When you're sure that your loop works, add five more Python terms to your glossary. When you run your program again, these new words and meanings should automatically be included in the output.

```
glossary = {
    'variable': 'a named space in memory that holds a value',
    'list': 'a collection of items in a particular order',
    'dictionary': 'a collection of key-value pairs',
    'loop': 'a block of code that repeats',
    'function': 'a block of code that performs a specific task',
    'tuple': 'an ordered collection of items that cannot change',
    'conditional': 'a statement that performs different actions based on conditions',
    'iteration': 'the process of repeating a block of code',
```

```
    'comprehension': 'a compact way to process all items in a list and return a list',
    'string': 'a sequence of characters'
}
for word, meaning in glossary.items():
    print(f"{word}: {meaning}")
```

6-5. Rivers: Make a dictionary containing three major rivers and the country each river runs through. One key-value pair might be 'nile': 'egypt'.

- Use a loop to print a sentence about each river, such as *The Nile runs through Egypt.*
- Use a loop to print the name of each river included in the dictionary.
- Use a loop to print the name of each country included in the dictionary.

```
rivers = {
    'nile': 'egypt',
    'amazon': 'brazil',
    'yangtze': 'china'
}
for river, country in rivers.items():
    print(f"The {river.title()} runs through {country.title()}.")
print("\nNames of rivers:")
for river in rivers.keys():
    print(river.title())
print("\nCountries:")
for country in rivers.values():
    print(country.title())
```

6-6. Polling: Use the code in **favorite_languages.py** (page 97).

- Make a list of people who should take the favorite languages poll. Include some names that are already in the dictionary and some that are not.

- Loop through the list of people who should take the poll. If they have already taken the poll, print a message thanking them for responding. If they have not yet taken the poll, print a message inviting them to take the poll.

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python'
}
people = ['jen', 'maria', 'edward', 'alex']


for person in people:
    if person in favorite_languages:
        print(f"Thank you, {person.title()}, for responding.")
    else:
        print(f"{person.title()}, please take the poll!")
```

6-7. People: Start with the program you wrote for Exercise 6-1 (page 99). Make two new dictionaries representing different people, and store all three dictionaries in a list called people. Loop through your list of people. As you loop through the list, print everything you know about each person.

```
person1 = {
    'first_name': 'John',
    'last_name': 'Doe',
    'age': 30,
    'city': 'New York'
}
person2 = {
    'first_name': 'Jane',
```

```
    'last_name': 'Smith',
    'age': 25,
    'city': 'Los Angeles'
}
person3 = {
    'first_name': 'Mike',
    'last_name': 'Brown',
    'age': 40,
    'city': 'Chicago'
}
people = [person1, person2, person3]


for person in people:
    full_name = person['first_name'] + " " + person['last_name']
    age = person['age']
    city = person['city']
    print(f"{full_name}, age {age}, lives in {city}.")
```

6-8. Pets: Make several dictionaries, where each dictionary represents a different pet. In each dictionary, include the kind of animal and the owner's name. Store these dictionaries in a list called pets. Next, loop through your list and as you do, print everything you know about each pet.

```
pet1 = {'animal': 'dog', 'owner': 'Alice'}
pet2 = {'animal': 'cat', 'owner': 'Bob'}
pet3 = {'animal': 'fish', 'owner': 'Carol'}
pets = [pet1, pet2, pet3]


for pet in pets:
    print(f"{pet['owner']} has a {pet['animal']}.")
```

6-9. Favorite Places: Make a dictionary called favorite_places. Think of three names to use as keys in the dictionary, and store one to three favorite places for each person. To make this exercise a bit more interesting, ask some friends to name a few of their favorite places. Loop through the dictionary, and print each person's name and their favorite places.

```
favorite_places = {
    'alice': ['paris', 'new york'],
    'bob': ['tokyo', 'melbourne'],
    'carol': ['rome', 'sydney']
}
for name, places in favorite_places.items():
    print(f"{name.title()}'s favorite places are:")
    for place in places:
        print(f"- {place.title()}")
    print()
```

6-10. Favorite Numbers: Modify your program from Exercise 6-2 (page 99) so each person can have more than one favorite number. Then print each person's name along with their favorite numbers.

```
favorite_numbers = {
    'Alice': [7, 42],
    'Bob': [3, 9],
    'Carol': [5, 10],
    'Dave': [1, 4],
    'Eve': [2, 8]
}
for name, numbers in favorite_numbers.items():
    numbers_str = ", ".join(str(num) for num in numbers)
    print(f"{name}'s favorite numbers are: {numbers_str}.")
```

6-11. Cities: Make a dictionary called cities. Use the names of three cities as keys in your dictionary. Create a dictionary of information about each city and include the country that the city is in, its approximate population, and one fact about that city. The keys for each city's dictionary should be something like country, population, and fact. Print the name of each city and all of the information you have stored about it.

```
cities = {
    'new york': {'country': 'usa', 'population': 8_000_000, 'fact': 'the Big Apple'},
    'paris': {'country': 'france', 'population': 2_148_000, 'fact': 'the City of Light'},
    'tokyo': {'country': 'japan', 'population': 13_960_000, 'fact': 'the eastern world capital of convenience'}
}
for city, info in cities.items():
    country = info['country']
    population = info['population']
    fact = info['fact']
    print(f"{city.title()} is in {country.title()}. It has a population of {population} and is {fact}.")
```

6-12. Extensions: We're now working with examples that are complex enough that they can be extended in any number of ways. Use one of the example programs from this chapter, and extend it by adding new keys and values, changing the context of the program or improving the formatting of the output.

```
favorite_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
    'mike': ['javascript', 'python'],
    'anna': ['c++']
}
```

```
for name, languages in favorite_languages.items():
    print(f"\n{name.title()}'s favorite language(s):")
    for language in languages:
        print(f"  - {language.title()}")
```