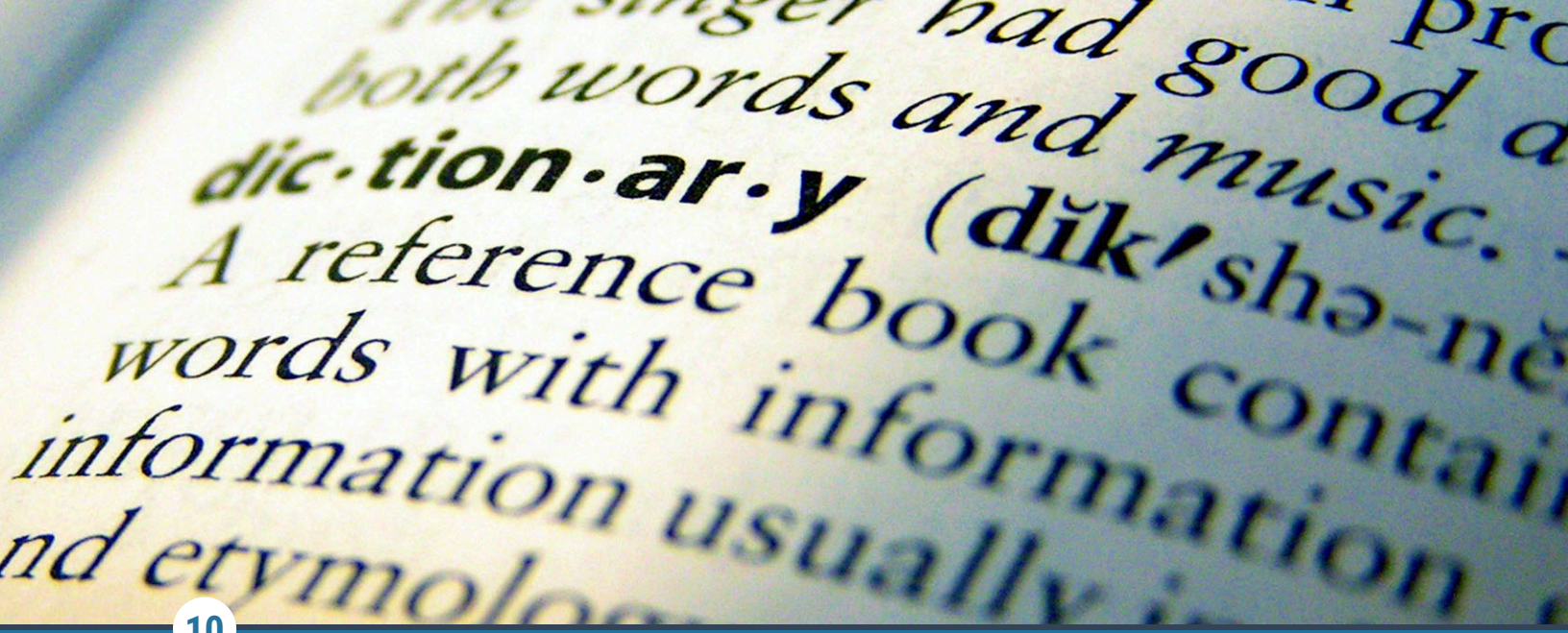

10

Dictionaries

Figure 10.1 credit: modification of work "Dictionary", by Caleb Roenigk/Flickr, CC BY 2.0

Chapter Outline

Introduction

A Python **dictionary** is a data type for storing the data in a key-value pair format. In this chapter, the dictionary data type is introduced. This chapter also introduces ways to use dictionaries, including looping over dictionary items and performing conditional statements on a dictionary.

10.1 Dictionary basics

Learning objectives

By the end of this section you should be able to

- Describe the structure of a dictionary object.
- Use `type()` to identify the type of a dictionary object.
- Explain why a dictionary object does not have duplicate key values.

Dictionaries

A **dictionary** in Python is a container object including key-value pairs. An example of a **key-value** item is a word in an English dictionary with its corresponding definition.

CHECKPOINT

Dictionaries

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-1-dictionary-basics)

CONCEPTS IN PRACTICE

Dictionary basics

Given the dictionary object `days = {"Sunday": 1, "Monday": 2, "Tuesday": 3}`, answer the following questions.

1. What are the keys in the `days` dictionary object?
   a. `"Sunday", "Monday", "Tuesday"`
   b. `1, 2, 3`
   c. `0, 1, 2`

2. What are the values in the `days` dictionary object?
   a. `0, 1, 2`
   b. `1, 2, 3`
   c. `"Sunday", "Monday", "Tuesday"`

3. What is the value associated with key `"Sunday"`?
   a. `"Sunday"`
   b. `1`
   c. `0`

Dictionary type and properties

The **dict** type implements a dictionary in Python. Since a dictionary object is used to look up values using keys, a dictionary object cannot hold duplicate key values.

CHECKPOINT

Dictionary properties

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-1-dictionary-basics)

CONCEPTS IN PRACTICE

Dictionary type and properties

Given the dictionary object `encoding = {"a": 97, "b": 98, "A": 65, "B": 66}`, answer the following questions.

4. What does `type(encoding)` return?

a. `dictionary`
b. `dict`
c. `str`

5. The `encoding` dictionary contains duplicate keys since it has both `"a"` and `"A"` as keys.
   a. true
   b. false

6. A dictionary can have duplicate values.
   a. true
   b. false

TRY IT

Investigate variable types

Given the variables in the code below, examine the variables' types.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-1-dictionary-basics)

10.2 Dictionary creation

Learning objectives

By the end of this section you should be able to

- Create a dictionary object with given key/value pairs.

Dictionary creation

Two methods exist for creating an empty dictionary:

- Using curly braces `{}`. Ex: `dict_1 = {}`.
- Using the `dict()` function. Ex: `dict_2 = dict()`.

A dictionary object can also be created with initial key-value pairs enclosed in curly braces. Ex: `my_dict = {"pizza": 2, "pasta": 3, "drink": 4}` creates a dictionary object `my_dict`. A key and associated value are separated by a colon, and key-value pairs are separated by commas.

CHECKPOINT

Creating a dictionary object

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-2-dictionary-creation)

CONCEPTS IN PRACTICE

Creating dictionary exercises

1. What is the correct syntax to create an empty dictionary in Python?
   a. `my_dict = ()`
   b. `my_dict = {}`
   c. `my_dict = []`

2. What is the correct syntax to create a dictionary with one item `"one": 1`?
   a. `my_dict = ("one": 1)`
   b. `my_dict = ("one", 1)`
   c. `my_dict = {"one": 1}`

3. How many items does the `my_dict` object contain?

```
my_dict = {"a": 1, "b": 2}
```

   a. 1
   b. 2
   c. 4

dict() for dictionary creation

A dictionary object can be created with initial key-value pairs using the `dict()` function.

- Creating a dictionary from a list of tuples.

```
my_list = [("apple", 2), ("banana", 3), ("orange", 4)]
my_dict = dict(my_list)
```

- Creating a dictionary using keyword arguments.

```
my_dict = dict(apple=2, banana=3, orange=4)
```

- Creating a dictionary from another dictionary.

```
old_dict = {"apple": 2, "banana": 3, "orange": 4}
new_dict = dict(old_dict)
```

CHECKPOINT

dict() for dictionary initialization

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-2-dictionary-creation)

CONCEPTS IN PRACTICE

dict() function for dictionary creation

4. What is the correct syntax to create a dictionary with initial values in Python?
   a. `my_dict = [key1:value1, key2:value2]`
   b. `my_dict = dict(key1=value1, key2=value2)`
   c. `my_dict = {key1=value1, key2=value2}`

5. What is the output of the following code?

```
my_dict = dict({"a": 1})
print(my_dict)
```

   a. `{"a": 1}`
   b. `<class, 'dict'>`
   c. `"a"`
      `1`

6. Which option creates a dictionary with two key-value pairs, `"a": "A"` and `"b": "B"`?
   a. `dict(a: A, b: B)`
   b. `dict([("a", "A"), ("b", "B")])`
   c. `dict({"a" = "A", "b" = "B"})`

TRY IT

Personal information dictionary

Create a dictionary, `my_info`, with three key-value pairs. The keys should be `"first name"`, `"last name"`, and `"age"` with the values being corresponding information about yourself. Then, print `my_info`.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-2-dictionary-creation)

10.3 Dictionary operations

Learning objectives

By the end of this section you should be able to

- Recognize that a dictionary object is mutable.
- Evaluate dictionary items, keys, and values.
- Demonstrate the ability to access, evaluate, and modify dictionary items.
- Modify a dictionary by adding items.
- Modify a dictionary by removing items.

Accessing dictionary items

In Python, values associated with keys in a dictionary can be accessed using the keys as indexes. Here are two ways to access dictionary items in Python:

- Square bracket notation: Square brackets `[]` with the key inside access the value associated with that key. If the key is not found, an exception will be thrown.
- `get()` method: The `get()` method is called with the key as an argument to access the value associated with that key. If the key is not found, the method returns `None` by default, or a default value specified as the second argument.

Ex: In the code below, a dictionary object `my_dict` is initialized with items `{"apple": 2, "banana": 3, "orange": 4}`. The square bracket notation and `get()` method are used to access values associated with the keys `"banana"` and `"apple"`, respectively. When accessing the dictionary to obtain the key `"pineapple"`, `-1` is returned since the key does not exist in the dictionary.

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
print(my_dict["banana"]) # Prints: 3
print(my_dict.get("apple")) # Prints: 2
print(my_dict.get("pineapple", -1)) # Prints: -1
```

CHECKPOINT

Accessing dictionary items

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-3-dictionary-operations)

CONCEPTS IN PRACTICE

Dictionary items

Given the dictionary `members = {"Jaya": "Student", "John": "TA", "Ksenia": "Staff"}`, answer the following questions.

1. What is the output of `members["Jaya"]`?
   a. `0`
   b. `None`
   c. `"Student"`

2. What is the output of `members.get("jaya")`?
   a. `0`
   b. `None`
   c. `"Student"`

3. What is the output of `members.get("jaya", "does not exist")`?
   a. `"Student"`
   b. `"does not exist"`
   c. `None`

Obtaining dictionary keys and values

Dictionary keys, values, and both keys and values can be obtained using `keys()`, `values()`, and `items()`

function calls, respectively. The return type of `keys()`, `values()`, and `items()` are `dict_keys`, `dict_values`, and `dict_items`, which can be converted to a `list` object using the list constructor `list()`.

EXAMPLE 10.1

String template formatting for course enrollment requests

A dictionary object with items `{"a": 97, "b": 98, "c": 99}` is created. Functions `keys()`, `values()`, and `items()` are called to obtain keys, values, and items in the dictionary, respectively. `list()` is also used to convert the output to a `list` object.

```
dictionary_object = {"a": 97, "b": 98, "c": 99}

print(dictionary_object.keys())
print(list(dictionary_object.keys()))
print(dictionary_object.values())
print(dictionary_object.items())
```

The above code's output is:

```
dict_keys(["a", "b", "c"])
["a", "b", "c"]
dict_values([97, 98, 99])
dict_items([("a", 97), ("b", 98), ("c", 99)])
```

CONCEPTS IN PRACTICE

Dictionary keys and values

Given the dictionary `numbers = {"one": 1, "two": 2, "three": 3}`, answer the following questions.

4. What is the output type of `numbers.keys()`?
   a. `dict_keys`
   b. `dict_values`
   c. `list`

5. What is the output of `print(numbers.values())`?
   a. `[1, 2, 3]`
   b. `dict_keys([1, 2, 3])`
   c. `dict_values([1, 2, 3])`

6. What is the output of `print(list(numbers.keys()))`?
   a. `["three", "two", "one"]`
   b. `["one", "two", "three"]`
   c. `dict_keys(["one", "two", "three"])`

Dictionary mutability

In Python, a dictionary is a mutable data type, which means that a dictionary's content can be modified after creation. Dictionary items can be added, updated, or deleted from a dictionary after a dictionary object is created.

To add an item to a dictionary, either the square bracket notation or `update()` function can be used.

- Square bracket notation: When using square brackets to create a new key object and assign a value to the key, the new key-value pair will be added to the dictionary.

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
my_dict["pineapple"] = 1
print(my_dict) # Prints: {"apple": 2, "banana": 3, "orange": 4, "pineapple": 1}
```

- `update()` method: the `update()` method can be called with additional key-value pairs to update the dictionary content.

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
my_dict.update({"pineapple": 1, "cherry": 0})
print(my_dict) # Prints: {"apple": 2, "banana": 3, "orange": 4, "pineapple": 1,
"cherry": 0}
```

To modify a dictionary item, the two approaches above can be used on an existing dictionary key along with the updated value. Ex:

- Square bracket notation:

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
my_dict["apple"] = 1
print(my_dict) # Prints: {"apple": 1, "banana": 3, "orange": 4}
```

- `update()` method:

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
my_dict.update({"apple": 1})
print(my_dict) # Prints: {"apple": 1, "banana": 3, "orange": 4}
```

Items can be deleted from a dictionary using the `del` keyword or the `pop()` method.

- `del` keyword:

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
del my_dict["orange"]
print(my_dict) # Prints: {"apple": 2, "banana": 3}
```

- `pop()` method:

```
my_dict = {"apple": 2, "banana": 3, "orange": 4}
deleted_value = my_dict.pop("banana")
print(deleted_value) # Prints: 3
```

```
print(my_dict) # Output: {"apple": 2, "orange": 4}}
```

CHECKPOINT

Modifying dictionary items

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-3-dictionary-operations)

CONCEPTS IN PRACTICE

Modifying a dictionary

Given the dictionary `food = {"Coconut soup": "$15", "Butter Chicken": "$18", "Kabob": "$20"}`, answer the following questions.

7. Which option modifies the value for the key `"Coconut soup"` to `"$11"` while keeping other items the same?
   a. `food = {""Coconut soup": "$15""}`
   b. `food["Coconut soup"] = 11`
   c. `food["Coconut soup"] = "$11"`

8. Which option removes the item `"Butter Chicken": "$18"` from the `food` dictionary?
   a. `food.remove("Butter Chicken")`
   b. `del food["Butter Chicken"]`
   c. `del food.del("Butter Chicken")`

9. What is the content of the `food` dictionary after calling `food.update({"Kabob": "$22", "Sushi": "$16"})`?
   a. `{"Coconut soup": "$15", "Butter Chicken": "$18", "Kabob": "$22", "Sushi": "$16"}`
   b. `{"Coconut soup": "$15", "Butter Chicken": "$18", "Kabob": "$20", "Sushi": "$16"}`
   c. `{ "Sushi": "$16", "Coconut soup": "$15", "Butter Chicken": "$18", "Kabob": "$20"}`

TRY IT

Create a dictionary of cars step-by-step

Follow the steps below to create a dictionary of `cars` and modify it step-by-step.

1. Create an empty dictionary.
2. Add a key-value pair of `"Mustang": 10`.
3. Add another key-value pair of `"Volt": 3`.
4. Print the dictionary.
5. Modify the value associated with key `"Mustang"` to be equal to `2`.

6. Delete key `"Volt"` and the associated value.
7. Print the dictionary content.

```
Prints {"Mustang": 2}
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-3-dictionary-operations)

TRY IT

The number of unique characters

Given a string value, calculate and print the number of unique characters using a dictionary.

```
Input:
string_value = "This is a string"


Prints 10
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-3-dictionary-operations)

10.4 Conditionals and looping in dictionaries

Learning objectives

By the end of this section you should be able to

- Write a conditional statement to check for a key/value.
- Write a `for` loop to iterate over elements of a dictionary.

Conditionals for dictionary

Conditional statements can be used with dictionaries to check if certain keys, values, or dictionary items exist in the dictionary or if a value satisfies a particular condition.

EXAMPLE 10.2

Templates and examples of a conditional statement on a dictionary

dict.items()

```
# (key, value) in dictionary.items()

movies = {"The godfather": 1974, "Interstellar": 2014}

print(("Interstellar", 2014) in movies.items())
```

```
True
```

Table 10.1

Conditionals on values or keys

```
# dictionary[key] operand test_value

movies = {"The godfather": 1974, "Interstellar": 2014}

print(movies["The godfather"] < 2000)
```

```
True
```

Table 10.2

dict.keys()

`# key in dictionary.keys()`
`movies = {"The godfather": 1974, "Interstellar": 2014}`
`print("Interstellar" in movies.keys())`

`True`

Table 10.3

dict.values()

`# value in dictionary.values()`
`movies = {"The godfather": 1974, "Interstellar": 2014}`
`print(2014 in movies.values())`

`True`

Table 10.4

CONCEPTS IN PRACTICE

Conditionals on dictionaries

Given the dictionary `fruit_count = {"apple": 2, "orange": 5, "pomegranate": 1}`, answer the following questions.

1. What is the output of `"apple"` in `fruit_count.keys()`?
   a. `True`
   b. `False`

2. What is the output of `("orange", 5)` in `fruit_count.items()`?
   a. `SyntaxError`
   b. `True`
   c. `False`

3. Which conditional statement checks if the value associated with the key `"pomegranate"` is greater than `0`?
   a. `fruit_count("pomegranate") > 0`
   b. `fruit_count["pomegranate"] > 0`
   c. `fruit_count.get("pomegranate" > 0)`

Looping on a dictionary

Looping over a Python dictionary is a way to iterate through key-value pairs in the dictionary. Looping in a dictionary can be done by iterating over keys or items. When looping using keys, keys are obtained using the `keys()` function and are passed to the loop variable one at a time. When looping over items using the `items()` function, both the key and value for each item are passed to the loop variable.

A FOR LOOP OVER A DICTIONARY RETRIEVES EACH KEY IN THE DICTIONARY

```
for key in dictionary: # Loop expression
  # Statements to execute in the loop

# Statements to execute after the loop
```

EXAMPLE 10.3

Iterating over a dictionary

dict.items()

```
zip_codes = {"Berkeley": 94709, "Santa Cruz": 95064, "Mountain View": 94030}

for key, value in zip_codes.items():
  print(key, value)
```

```
Berkeley 94709
Santa Cruz 95064
Mountain View 94030
```

Table 10.5

dict.keys()

`zip_codes = {"Berkeley": 94709, "Santa Cruz": 95064, "Mountain View": 94030}`
 	
`for key in zip_codes.keys():`
`  print(key)`

`Berkeley`
`Santa Cruz`
`Mountain View`

Table 10.6

dict.values()

`zip_codes = {"Berkeley": 94709, "Santa Cruz": 95064, "Mountain View": 94030}`
 	
`for value in zip_codes.values():`
`  print(value)`

`94709`
`95064`
`94030`

Table 10.7

CONCEPTS IN PRACTICE

Loops on dictionaries

4. Which method is used to loop over the values in a Python dictionary?
   a. `keys()`
   b. `values()`
   c. `items()`

5. What is the output of the following code?

```
fruit_count = {"apple": 2, "orange": 5, "banana": 1}
for key in fruit_count.keys():
  print(key, end = " ")
```

a. `apple orange banana`
b. `2 5 1`
c. `("apple", 2) ("orange", 5) ("banana", 1)`

6. What is the output of the following code?

```
fruit_count = {"apple": 2, "orange": 5, "banana": 1}
for value in fruit_count.values():
  print(value * 2, end = " ")
```

a. `apple orange banana`
b. `2 5 1`
c. `4 10 2`

TRY IT

Character count in a string

Given a string value, calculate and print the number of occurrences of all characters using a dictionary.

```
Input:
string_value = "This is a string"
```

```
Prints {"T": 1, "h": 1, "i": 3, "s": 3, " ": 3, "a": 1, "t": 1, "r": 1, "n": 1,
"g": 1}
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-4-conditionals-and-looping-in-dictionaries)

TRY IT

Calculate the total number of fruits

Given a `fruit_count` dictionary that contains information about fruits and the count of each fruit, calculate the total number of fruits across all fruit types.

```
Input:
fruit_count = {"banana": 2, "orange": 5, "peach": 5}
```

```
Prints 12
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-4-conditionals-and-looping-in-dictionaries)

10.5 Nested dictionaries and dictionary comprehension

Learning objectives

By the end of this section you should be able to

- Explain the structure of nested dictionaries.
- Use dictionary comprehension to create a dictionary object.

Nested dictionaries

As described before, Python dictionaries are a type of data structure that allows for storing data in key-value pairs. **Nested dictionaries** are dictionaries that are stored as values within another dictionary. Ex: An organizational chart with keys being different departments and values being dictionaries of employees in a given department. For storing employee information in a department, a dictionary can be used with keys being employee IDs and values being employee names. The tables below outline the structure of such nested dictionaries and how nested values can be accessed.

EXAMPLE 10.4

Defining nested dictionaries and accessing elements

Defining nested dictionaries

```
company_org_chart = {
  "Marketing": {
    "ID234": "Jane Smith"
  },
  "Sales": {
    "ID123": "Bob Johnson",
    "ID122": "David Lee"
  },
  "Engineering": {
    "ID303": "Radhika Potlapally",
    "ID321": "Maryam Samimi"
  }
}
```

Table 10.8

Accessing nested dictionary items

```
print(company_org_chart["Sales"]["ID122"])
print(company_org_chart["Engineering"]["ID321"])
```

```
David Lee
Maryam Samimi
```

Table 10.9

CONCEPTS IN PRACTICE

Nested dictionary structure

1. What is a nested dictionary in Python?
   a. A dictionary that contains only integers as keys and values.
   b. A dictionary that contains another dictionary as a value.
   c. A dictionary that contains only strings as keys and values.

d. A dictionary that contains only lists as keys and values.

2. How can a value be accessed in a nested dictionary in Python?
   a. by using the key of the outer dictionary
   b. by using the key of the inner dictionary
   c. By using both the keys of the outer and inner dictionaries

3. What is the syntax for creating a nested dictionary in Python?
   a. `{key: value}`
   b. `{key1: {key2: value}}`
   c. `{key: {value: key2}}`

Dictionary comprehension

Dictionary comprehension is a concise and efficient way to create a dictionary in Python. With dictionary comprehension, elements of an iterable object are transformed into key-value pairs. The syntax of dictionary comprehension is similar to list comprehension, but instead of using square brackets, curly braces are used to define a dictionary.

Here is a general syntax for dictionary comprehension:

SYNTAX FOR DICTIONARY COMPREHENSION

```
{key_expression: value_expression for element in iterable}
```

CHECKPOINT

Squares of numbers

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-5-nested-dictionaries-and-dictionary-comprehension)

CONCEPTS IN PRACTICE

Dictionary comprehension

4. What is the output of the following code?

```
numbers = [1, 2, 3, 4, 5]
my_dict = {x: x**2 for x in numbers if x % 2 == 0}
print(my_dict)
```

   a. `{1: 1, 3: 9, 5: 25}`
   b. `{2: 4, 4: 16}`

c. `{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}`

5. What is the output of the following dictionary comprehension?

```
names = ["Alice ", "Bob ", "Charlie "]
name_lengths = {name: len(name) for name in names}
print(name_lengths)
```

a. `{ "Alice ": 5, "Bob ": 3, "Charlie ": 7}`
b. `{5: "Alice ", 3: "Bob ", 7: "Charlie "}`
c. `{ "A ": 5, "B ": 3, "C ": 7}`

6. What is the output of the following dictionary comprehension?

```
my_dict = {i: i**2 for i in range(4)}
print(my_dict)
```

a. `{0: 0, 1: 1, 2: 4, 3: 9}`
b. `{0: 0, 1: 1, 2: 4}`
c. `{1: 1, 2: 4, 3: 9}`

TRY IT

Product prices

Suppose you have a dictionary of product prices, where the keys are product names and the values are their respective prices in dollars. Write a Python program that uses dictionary comprehension to create a new dictionary that has the same keys as the original dictionary, but the values are the prices in euros. Assume that the exchange rate is 1 dollar = 0.85 euros.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-5-nested-dictionaries-and-dictionary-comprehension)

TRY IT

Restructuring the company data

Suppose you have a dictionary that contains information about employees at a company. Each employee is identified by an ID number, and their information includes their name, department, and salary. You want to create a nested dictionary that groups employees by department so that you can easily see the names and salaries of all employees in each department. Write a Python program that when given a dictionary, `employees`, outputs a nested dictionary, `dept_employees`, which groups employees by department.

```
Input:
employees = {
  1001: {"name": "Alice", "department": "Engineering", "salary": 75000},
  1002: {"name": "Bob", "department": "Sales", "salary": 50000},
```

```
    1003: {"name": "Charlie", "department": "Engineering", "salary": 80000},
    1004: {"name": "Dave", "department": "Marketing", "salary": 60000},
    1005: {"name": "Eve", "department": "Sales", "salary": 55000}
}



Resulting dictionary:

{
"Engineering": {1001: {"name": "Alice", "salary": 75000}, 1003: {"name":
"Charlie", "salary": 80000}},
"Sales": {1002: {"name": "Bob", "salary": 50000}, 1005: {"name": "Eve",
"salary": 55000}},
"Marketing": {1004: {"name": "Dave", "salary": 60000}}
}
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-5-nested-dictionaries-and-dictionary-comprehension)

10.6 Chapter summary

Highlights from this chapter include:

- A dictionary in Python is a container object including key-value pairs.
- The `dict` type implements a dictionary in Python.
- A dictionary cannot have duplicate keys.
- A dictionary is a mutable object but keys in the dictionary must be immutable objects.
- A dictionary can be created using curly braces or the `dict()` method.
- Values in the dictionary can be obtained through square bracket notation or the `get()` method.
- Dictionary items, keys, and values can be obtained using `items()`, `keys()`, and `values()` methods, respectively.
- Existing items can be modified or new items can be added to a dictionary using square brackets notation or the `update()` method.
- Items can be removed from a dictionary using the `del` keyword or the `pop()` method.
- Conditional statements can be used with a dictionary to check if the dictionary contains specific keys, values, or key-value pairs.
- Looping on a dictionary can be done by iterating over keys, values, or items.
- Nested dictionaries are dictionaries that are stored as values within another dictionary.
- With dictionary comprehension, elements of an iterable object are transformed into key-value pairs.

At this point, you should be able to use dictionaries in your programs. The programming practice below ties together most topics presented in the chapter.

Concept	Description
Dictionary creation using curly braces	`my_dict = {key1:value1, key2:value2}`

Dictionary creation using the `dict()` method	`# Using a list`
	`my_list = [(key1, value1), (key2, value2)]`
	`my_dict = dict(my_list)`
	
	`# Using keyword arguments`
	`my_dict = dict(key1=value1, key2=value2)`
	
	`# From another dictionary`
	`old_dict = {key1: value1, key2: value2}`
	`new_dict = dict(old_dict)`

Accessing dictionary items	`my_dict = {key1: value1, key2: value2}`
	
	`# Accessing item using square bracket notation`
	`my_dict[key1]`
	
	`# Accessing item through get() method`
	`my_dict.get(key1)`

Accessing all dictionary items	`my_dict.items()`
Accessing all dictionary keys	`my_dict.keys()`
Accessing all dictionary values	`my_dict.values()`

Table 10.10 Chapter 10 reference.

Concept	Description

Adding a new key-value pair or updating an existing key-value pair	`my_dict = {key1: value1, key2: value2}`
	
	`# Updating an item using square bracket notation`
	`my_dict[key1] = new_value`
	`# Adding a new key-value pair using square bracket notation`
	`my_dict[key3] = value3`
	
	`# Updating an item using update() method`
	`my_dict.update({key1: new_value})`
	`# Adding a new key-value pair using update() method`
	`my_dict.update({key3: value3})`

Deleting a key-value pair from a dictionary	`my_dict = {key1: value1, key2: value2}`
	
	`# Using del keyword`
	`del my_dict[key1]`
	
	`# Using pop() method`
	`deleted_value = my_dict.pop(key1)`

Iterating over a dictionary	`for key in dictionary: # Loop expression`
	`  # Statements to execute in the loop`
	
	`#Statements to execute after the loop`

Table 10.10 Chapter 10 reference.

Concept	Description

Nested dictionaries	`{`
	`key_1:{key11:value11, key12:value12},`
	`key_2:{key21:value21, key22:value22}`
	`}`

Dictionary comprehension	`{key_expression: value_expression for element in iterable}`

Table 10.10 Chapter 10 reference.

TRY IT

Even and odd values

Given a list, create a dictionary with two keys, `"even"` and `"odd"`. The values associated with each key must be the list of corresponding even and odd values in the given list.

```
Input:
input_list = [3, 5, 6, 1]


Prints {"even": [6], "odd":[3, 5, 1]}
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/10-6-chapter-summary)