BYU IDAHO CSE 111 | Programming with Functions

08 Prepare: Dictionaries

In this lesson, you will learn how to store data in and retrieve data from Python dictionaries.

Videos

Watch this video about dictionaries in Python.

» Dictionaries

Concepts

Here are the Python programming concepts and topics that you should learn during this lesson:

Dictionaries

Within a Python program, a program can store many items in a dictionary. Each item in a dictionary is a key value pair. Each key within a dictionary must be unique. In other words, no key can appear more than once in a dictionary. Values within a dictionary do not have to be unique. Dictionaries are mutable, meaning they can be changed after they are created. Dictionaries were invented to enable computers to find items quickly.

The following table represents a dictionary that contains five items (five key value pairs) Notice that each of the keys is unique.

items		
keys	**values**	
42-039-4736	Clint Huish	↑
61-315-0160	Michelle Davis	5
10-450-1203	Jorge Soares	items
75-421-2310	Abdul Ali	|
07-103-5621	Michelle Davis	↓

We can create a dictionary by using curly braces ({ and }). We can add an item to a dictionary and find an item in a dictionary by using square brackets ([and]) and a key. The following code example shows how to create a dictionary, add an item, remove an item, and find an item in a dictionary.

```python
# Example 1

def main():
    # Create a dictionary with student IDs as
    # the keys and student names as the values.
    students = {
        "42-039-4736": "Clint Huish",
        "61-315-0160": "Michelle Davis",
        "10-450-1203": "Jorge Soares",
        "75-421-2310": "Abdul Ali",
        "07-103-5621": "Michelle Davis"
    }

    # Add an item to the dictionary.
    students["81-298-9238"] = "Sama Patel"

    # Remove an item from the dictionary.
    students.pop("61-315-0160")
```

```
19
20      # Get the number of items in the dictionary and print it.
21      length = len(students)
22      print(length)
23
24      # Print the entire dictionary.
25      print(students)
26
27      # Get a student ID from the user.
28      id = input("Enter a student ID: ")
29
30      # Check if the student ID is in the dictionary.
31      if id in students:
32
33          # Find the student ID in the dictionary and
34          # retrieve the corresponding student name.
35          name = students[id]
36
37          # Print the student's name.
38          print(name)
39      else:
40          print("No such student")
41
42
43  # Call main to start this program.
44  if __name__ == "__main__":
45      main()
```

Line 15 in the previous code example, adds an item to the dictionary. To add an item to an existing dictionary, write code that follows this template:

```
    dictionary_name[key] = value
```

Notice that line 15 follows this template.

Line 35 in the previous code example, finds a key and retrieves its corresponding value from a dictionary. To find a key and retrieve its corresponding value, write code that follows this template:

```
    value = dictionary_name[key]
```

Notice that line 35 follows this template.

Compound Values

A simple value is a value that doesn't contain parts, such as an integer. A compound value is a value that has parts, such as a list. In example 1 above, the *students* dictionary has simple keys and values. Each key is a single string, and each value is a single string. It is possible to store compound values in a dictionary. Example 2 shows a *students* dictionary where each value is a Python list. Because each list contains multiple parts, we say that the dictionary stores compound values.

```
# Example 2

def main():
    # Create a dictionary with student IDs as the keys
    # and student data stored in a list as the values.
    students = {
        # student_ID: [given_name, surname, email_address, credits]
        "42-039-4736": ["Clint", "Huish", "hui20001@byui.edu", 16],
        "61-315-0160": ["Michelle", "Davis", "dav21012@byui.edu", 3],
        "10-450-1203": ["Jorge", "Soares", "soa22005@byui.edu", 15],
        "75-421-2310": ["Abdul", "Ali", "ali20003@byui.edu", 5],
        "07-103-5621": ["Michelle", "Davis" "dav19008@byui.edu", 0]
    }
```

Finding One Item

The reason Python dictionaries were developed is to make finding items easy and fast. As explained in example 1, to find an item in a dictionary, a programmer needs to write just one line of code that follows this template:

```
    value = dictionary_name[key]
```

That one line of code will cause the computer to search the dictionary until it finds the *key*. Then the computer will return the *value* that corresponds to the *key*. Some students forget how easy it is to find items in a dictionary, and when asked to write code to find an item, they write complex code like lines 25–28 in example 3.

```
 1  # Example 3
 2
 3  def main():
 4      # Create a dictionary with student IDs as the keys
 5      # and student data stored in a list as the values.
 6      students = {
 7          # student_ID: [given_name, surname, email_address, credits]
 8          "42-039-4736": ["Clint", "Huish", "hui20001@byui.edu", 16],
 9          "61-315-0160": ["Michelle", "Davis", "dav21012@byui.edu", 3],
10          "10-450-1203": ["Jorge", "Soares", "soa22005@byui.edu", 15],
11          "75-421-2310": ["Abdul", "Ali", "ali20003@byui.edu", 5],
12          "07-103-5621": ["Michelle", "Davis" "dav19008@byui.edu", 0]
13      }
14
15      # Get a student ID from the user.
16      id = input("Enter a student ID: ")
17
18      # This is a difficult and slow way to find an item in a
19      # dictionary. Don't write code like this to find an item
20      # in a dictionary!
21
22      # For each item in the dictionary, check if
23      # its key is the same as the variable id.
24      student = None
25      for key, value in students.items(): # Bad example!
26          if key == id:                   # Don't use a loop like
27              student = value             # this to find an item
28              break                       # in a dictionary.
29
```

Compare the `for` loop at lines 25–28 in the previous example to this one line of code.

```
    value = students[id]
```

Clearly, writing the one line of code is easier for the programmer than writing the `for` loop. Not only is the one line of code easier to write, but the computer will execute it much, much faster than the `for` loop. Therefore, when you need to write code to find an item in a dictionary, don't write a loop. Instead, write one line of code that uses the square brackets ([and]) and a key to find an item. Example 4 shows the correct way to find an item in a dictionary.

```
# Example 4
```

```python

def main():
    # Create a dictionary with student IDs as the keys
    # and student data stored in a list as the values.
    students = {
        # student_ID: [given_name, surname, email_address, credits]
        "42-039-4736": ["Clint", "Huish", "hui20001@byui.edu", 16],
        "61-315-0160": ["Michelle", "Davis", "dav21012@byui.edu", 3],
        "10-450-1203": ["Jorge", "Soares", "soa22005@byui.edu", 15],
        "75-421-2310": ["Abdul", "Ali", "ali20003@byui.edu", 5],
        "07-103-5621": ["Michelle", "Davis" "dav19008@byui.edu", 0]
    }

    # These are the indexes of the elements in the value lists.
    GIVEN_NAME_INDEX = 0
    SURNAME_INDEX = 1
    EMAIL_INDEX = 2
    CREDITS_INDEX = 3

    # Get a student ID from the user.
    id = input("Enter a student ID: ")

    # Check if the student ID is in the dictionary.
    if id in students:

        # Find the student ID in the dictionary and
        # retrieve the corresponding value, which is a list.
        value = students[id]

        # Retrieve the student's given name (first name) and
        # surname (last name or family name) from the list.
        given_name = value[GIVEN_NAME_INDEX]
        surname = value[SURNAME_INDEX]

        # Print the student's name.
        print(f"{given_name} {surname}")
    else:
        print("No such student")


# Call main to start this program.
if __name__ == "__main__":
    main()
```

Processing All Items

Occaisionally, you may need to write a program that processes all the items in a dictionary. Processing all the items in a dictionary is different than finding one item in a dictionary. Processing all the items is done using a `for` loop and the `dict.items()` method as shown in example 5 on line 25.

```python
 1  # Example 5
 2
 3  def main():
 4      # Create a dictionary with student IDs as the keys
 5      # and student data stored in a list as the values.
 6      students = {
 7          "42-039-4736": ["Clint", "Huish", "hui20001@byui.edu", 16],
 8          "61-315-0160": ["Michelle", "Davis", "dav21012@byui.edu", 3],
 9          "10-450-1203": ["Jorge", "Soares", "soa22005@byui.edu", 15],
10          "75-421-2310": ["Abdul", "Ali", "ali20003@byui.edu", 5],
11          "07-103-5621": ["Michelle", "Davis", "dav19008@byui.edu", 0],
12          "81-298-9238": ["Sama", "Patel", "pat21004@byui.edu", 8]
13      }
14
15      # These are the indexes of the elements in the value lists.
16      GIVEN_NAME_INDEX = 0
17      SURNAME_INDEX = 1
18      EMAIL_INDEX = 2
19      CREDITS_INDEX = 3
20
21      total = 0
22
23      # For each item in the list add the number
24      # of credits that the student has earned.
25      for item in students.items():
26          key = item[0]
27          value = item[1]
28
29          # Retrieve the number of credits from the value list.
30          credits = value[CREDITS_INDEX]
31
32          # Add the number of credits to the total.
33          total += credits
34
35      print(f"Total credits earned by all students: {total}")
36
37
38  # Call main to start this program.
39  if __name__ == "__main__":
40      main()
```

As with all the example code in CSE 111, example 5 contains working Python code. Even though the code works, we can combine lines 25–27 into a single line of code by using a Python shortcut called unpacking. Instead of writing lines 25–27, like this:

```
    for item in students.items():
        key = item[0]
        value = item[1]
```

We can write one line of code that combines the three lines of code and unpacks the item in the `for` statement like this:

```
    for key, value in students.items():
```

Dictionaries Are Similar to Lists

Dictionaries are similar to lists in a few ways. The following table shows the similarities and differences of lists and dictionaries.

	Lists	Dictionaries
Similar	A list can store many elements. Each element in a list does not have to be unique.	A dictionary can store many items. Each item is a key value pair. Each key within a dictionary must be unique.
Different	Each element in a list does not have to be unique.	Each item in a dictionary is a key value pair. Each key within a dictionary must be unique. Each value does not have to be unique.

	Lists	Dictionaries
Different	Lists were designed for efficiently storing elements. Lists use less memory than dictionaries. However, finding an element in a list is relatively slow.	Dictionaries were designed for quickly finding items. Finding an item in a dictionary is fast. However, dictionaries use more memory than lists.
Same	Lists are mutable, meaning a program can add and remove elements after a list is created.	Dictionaries are mutable, meaning a program can add and remove items after a dictionary is created.
Same	Lists are passed by reference into a function.	Dictionaries are passed by reference into a function.
Different	A programmer uses square brackets ([and]) to create a list.	A programmer uses curly braces ({ and }) to create a dictionary.
Different	A programmer calls the `append` and `insert` methods to add an element to a list.	A programmer uses square brackets ([and]) to add an item to a dictionary.
Similar	A programmer uses square brackets ([and]) and an index to retrieve an element from a list.	A programmer uses square brackets ([and]) and a key to retrieve a value from a dictionary.
Similar	A programmer uses square brackets ([and]) and an index to replace an element in a list.	A programmer uses square brackets ([and]) and a key to replace a value in a dictionary.

Converting between Lists and Dictionaries

It is possible to convert two lists into a dictionary by using the built-in `zip` and `dict` functions. The contents of the first list will become the keys in the dictionary, and the contents of the second list will become the values. This implies that the two lists must have the same length, and the elements in the first list must be unique because keys in a dictionary must be unique.

It is also possible to convert a dictionary into two lists by using the `keys` and `values` methods and the built-in `list` function. The following code example starts with two lists, converts them into a dictionary, and then converts the dictionary into two lists.

```python
# Example 6

def main():
    # Create a list that contains five student numbers.
    numbers = ["42-039-4736", "61-315-0160",
            "10-450-1203", "75-421-2310", "07-103-5621"]

    # Create a list that contains five student names.
    names = ["Clint Huish", "Michelle Davis",
            "Jorge Soares", "Abdul Ali", "Michelle Davis"]

    # Convert the numbers list and names list into a dictionary.
    student_dict = dict(zip(numbers, names))

    # Print the entire student dictionary.
    print(student_dict)

    # Convert the student dictionary into
    # two lists named keys and values.
    keys = list(student_dict.keys())
    values = list(student_dict.values())

    # Print both lists.
    print(keys)
    print(values)


# Call main to start this program.
if __name__ == "__main__":
    main()
```

Documentation

The following tutorials contain more information about dictionaries in Python.

» Dictionaries in Python.