**USING DICTIONARIES**

**Using Dictionaries**

Now that we know how to create a dictionary, we can start using already created dictionaries to solve problems.

In this lesson, you’ll learn how to:

* Use a key to get a value from a dictionary
* Check for existence of keys
* Iterate through keys and values in dictionaries

**Get A Key**

Once you have a dictionary, you can access the values in it by providing the key. For example, let’s imagine we have a dictionary that maps buildings to their heights, in meters:

building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}

Then we can access the data in it like this:

>>> building\_heights["Burj Khalifa"]  
828  
>>> building\_heights["Ping An"]  
599

**Instructions**

**1.**

We have provided a dictionary that maps the elements of astrology to the zodiac signs. Print out the list of zodiac signs associated with the "earth" element.

Checkpoint 2 Passed

**2.**

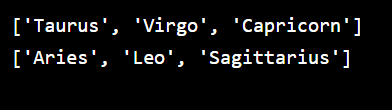
Print out the list of the "fire" signs.

**script.py**

zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

print(zodiac\_elements["earth"])

print(zodiac\_elements["fire"])

****

**Get an Invalid Key**

Let’s say we have our dictionary of building heights from the last exercise:

building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}

What if we wanted to know the height of the Landmark 81 in Ho Chi Minh City? We could try:

print(building\_heights["Landmark 81"])

But "Landmark 81" does not exist as a key in the building\_heights dictionary! So this will throw a KeyError:

KeyError: 'Landmark 81'

One way to avoid this error is to first check if the key exists in the dictionary:

key\_to\_check = "Landmark 81"  
  
if key\_to\_check in building\_heights:  
  print(building\_heights["Landmark 81"])

This will not throw an error, because key\_to\_check in building\_heights will return False, and so we never try to access the key.

**Instructions**

**1.**

Review the code in the editor and predict what the output will be. Run the code to see if you are correct!

Checkpoint 2 Passed

**2.**

Because "energy" is not a key in zodiac\_elements, a KeyError is thrown in the terminal!

Using an if statement, check if "energy" is a key in zodiac\_elements. Nest the existing print() statement within the if statement so that it will only execute if "energy" is a key.

Run your code again. This time, there should be no errors in the terminal!

Checkpoint 3 Passed

**3.**

Add the key "energy" to the zodiac\_elements. It should map to a value of "Not a Zodiac element". Run the code. Since "energy" is now a key, its value prints to the terminal!

**script.py**

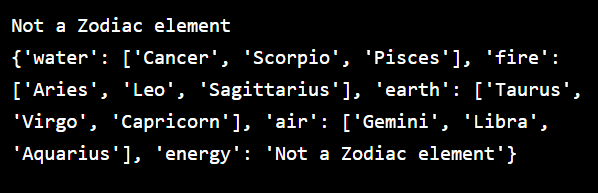
zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

zodiac\_elements["energy"] = "Not a Zodiac element"

if "energy" in zodiac\_elements:

  print(zodiac\_elements["energy"])

print(zodiac\_elements)

****

**Try/Except to Get a Key**

We saw that we can avoid KeyErrors by checking if a key is in a dictionary first. Another method we could use is a try/except:

key\_to\_check = "Landmark 81"  
try:  
  print(building\_heights[key\_to\_check])  
except KeyError:  
  print("That key doesn't exist!")

When we try to access a key that doesn’t exist, the program will go into the except block and print "That key doesn't exist!".

**Instructions**

**1.**

Use a try block to try to print the caffeine level of "matcha". If there is a KeyError, print "Unknown Caffeine Level".

Checkpoint 2 Passed

**2.**

Above the try block, add "matcha" to the dictionary with a value of 30.

**script.py**

caffeine\_level = {"espresso": 64, "chai": 40, "decaf": 0, "drip": 120}

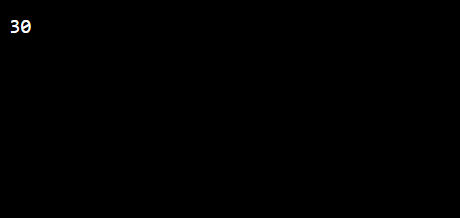
caffeine\_level["matcha"] = 30

try:

  print(caffeine\_level["matcha"])

except KeyError:

  print("Unknown Caffeine Level")

****

**Safely Get a Key**

We saw in the last exercise that we had to add a key:value pair to a dictionary in order to avoid a KeyError. This solution is not sustainable. We can’t predict every key a user may call and add all of those placeholder values to our dictionary!

Dictionaries have a .get() method to search for a value instead of the my\_dict[key] notation we have been using. If the key you are trying to .get() does not exist, it will return None by default:

building\_heights = {"Burj Khalifa": 828, "Shanghai Tower": 632, "Abraj Al Bait": 601, "Ping An": 599, "Lotte World Tower": 554.5, "One World Trade": 541.3}  
  
#this line will return 632:  
building\_heights.get("Shanghai Tower")  
  
#this line will return None:  
building\_heights.get("My House")

You can also specify a value to return if the key doesn’t exist. For example, we might want to return a building height of 0 if our desired building is not in the dictionary:

>>> building\_heights.get('Shanghai Tower', 0)  
632  
>>> building\_heights.get('Mt Olympus', 0)  
0  
>>> building\_heights.get('Kilimanjaro', 'No Value')  
'No Value'

**Instructions**

**1.**

Use .get() to get the value of "teraCoder"‘s user ID, with 100000 as a default value if the user doesn’t exist. Store it in a variable called tc\_id. Print tc\_id to the console.

Checkpoint 2 Passed

**2.**

Use .get() to get the value of "superStackSmash"‘s user ID, with 100000 as a default value if the user doesn’t exist. Store it in a variable called stack\_id. Print stack\_id to the console.

**script.py**

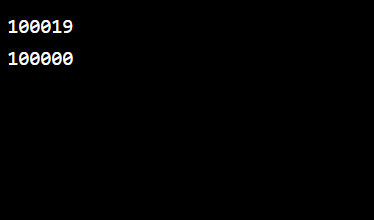
user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

tc\_id = user\_ids.get("teraCoder", 1000)

print(tc\_id)

stack\_id = user\_ids.get("superStackSmash", 100000)

print(stack\_id)

****

**Delete a Key**

Sometimes we want to get a key and remove it from the dictionary. Imagine we were running a raffle, and we have this dictionary mapping ticket numbers to prizes:

raffle = {223842: "Teddy Bear", 872921: "Concert Tickets", 320291: "Gift Basket", 412123: "Necklace", 298787: "Pasta Maker"}

When we get a ticket number, we want to return the prize and also remove that pair from the dictionary, since the prize has been given away. We can use .pop() to do this. Just like with .get(), we can provide a default value to return if the key does not exist in the dictionary:

>>> raffle.pop(320291, "No Prize")  
"Gift Basket"  
>>> raffle  
{223842: "Teddy Bear", 872921: "Concert Tickets", 412123: "Necklace", 298787: "Pasta Maker"}  
>>> raffle.pop(100000, "No Prize")  
"No Prize"  
>>> raffle  
{223842: "Teddy Bear", 872921: "Concert Tickets", 412123: "Necklace", 298787: "Pasta Maker"}  
>>> raffle.pop(872921, "No Prize")  
"Concert Tickets"  
>>> raffle  
{223842: "Teddy Bear", 412123: "Necklace", 298787: "Pasta Maker"}

.pop() works to delete items from a dictionary, when you know the key value.

**Instructions**

**1.**

You are designing the video game Big Rock Adventure. We have provided a dictionary of items that are in the player’s inventory which add points to their health meter. In one line, add the corresponding value of the key "stamina grains" to the health\_points variable and remove the item "stamina grains" from the dictionary. If the key does not exist, add 0 to health\_points.

Checkpoint 2 Passed

Hint

The .pop() method takes a key as an argument and will remove the key-value pair from the dictionary and also return the value:

dictionary\_name.pop(key, default)

In this instance, the key is “stamina grains” and the default is 0.

**2.**

In one line, add the value of "power stew" to health\_points and remove the item from the dictionary. If the key does not exist, add 0 to health\_points.

Checkpoint 3 Passed

**3.**

In one line, add the value of "mystic bread" to health\_points and remove the item from the dictionary. If the key does not exist, add 0 to health\_points.

Checkpoint 4 Passed

**4.**

Print available\_items and health\_points.

**script.py**

available\_items = {"health potion": 10, "cake of the cure": 5, "green elixir": 20, "strength sandwich": 25, "stamina grains": 15, "power stew": 30}

health\_points = 20

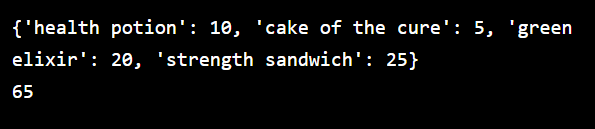
health\_points += available\_items.pop("stamina grains", 0)

health\_points += available\_items.pop("power stew", 0)

health\_points += available\_items.pop("mystic bread", 0)

print(available\_items)

print(health\_points)

****

**Get All Keys**

Sometimes we want to operate on all of the keys in a dictionary. For example, if we have a dictionary of students in a math class and their grades:

test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}

We want to get a roster of the students in the class, without including their grades. We can do this with the built-in list() function:

>>> list(test\_scores)  
["Grace", "Jeffrey", "Sylvia", "Pedro", "Martin", "Dina"]

Dictionaries also have a .keys() method that returns a dict\_keys object. A dict\_keys object is a *view* object, which provides a look at the current state of the dictionary, without the user being able to modify anything. The dict\_keys object returned by .keys() is a set of the keys in the dictionary. You cannot add or remove elements from a dict\_keys object, but it can be used in the place of a list for iteration:

for student in test\_scores.keys():  
  print(student)

will yield:

Grace  
Jeffrey  
Sylvia  
Pedro  
Martin  
Dina

**Instructions**

**1.**

Create a variable called users and assign it to be a dict\_keys object of all of the keys of the user\_ids dictionary.

Checkpoint 2 Passed

**2.**

Create a variable called lessons and assign it to be a dict\_keys object of all of the keys of the num\_exercises dictionary.

Checkpoint 3 Passed

**3.**

Print users to the console.

Checkpoint 4 Passed

**4.**

Print lessons to the console.

**script.py**

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

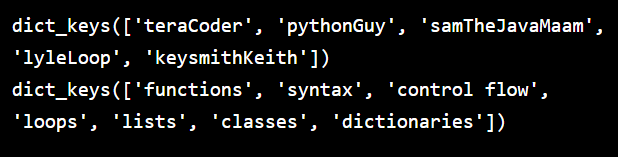
num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

users = user\_ids.keys()

lessons = num\_exercises.keys()

print(users)

print(lessons)

****

**Get All Values**

Dictionaries have a .values() method that returns a dict\_values object (just like a dict\_keys object but for values!) with all of the values in the dictionary. It can be used in the place of a list for iteration:

test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}  
  
for score\_list in test\_scores.values():  
  print(score\_list)

will yield:

[80, 72, 90]  
[88, 68, 81]  
[80, 82, 84]  
[98, 96, 95]  
[78, 80, 78]  
[64, 60, 75]

There is no built-in function to get all of the values as a list, but if you really want to, you can use:

list(test\_scores.values())

However, for most purposes, the dict\_values object will act the way you want a list to act.

**Instructions**

**1.**

Create a variable called total\_exercises and set it equal to 0.

Checkpoint 2 Passed

**2.**

Iterate through the values in the num\_exercises list and add each value to the total\_exercises variable.

Checkpoint 3 Passed

**3.**

Print the total\_exercises variable to the console.

**script.py**

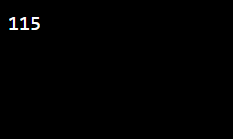
num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

total\_exercises = 0

for exercise in num\_exercises.values():

  total\_exercises += exercise

print(total\_exercises)

****