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.

2.

Print out the list of the `"fire"` signs.

script.py

```
zodiac_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "L
eo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini",
"Libra", "Aquarius"]}

print(zodiac_elements["earth"])
print(zodiac_elements["fire"])
```

```
['Taurus', 'Virgo', 'Capricorn']
['Aries', 'Leo', 'Sagittarius']
```

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!

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!

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", "L
eo", "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)
```

```
Not a Zodiac element
{'water': ['Cancer', 'Scorpio', 'Pisces'], 'fire':
['Aries', 'Leo', 'Sagittarius'], 'earth': ['Taurus',
'Virgo', 'Capricorn'], 'air': ['Gemini', 'Libra',
'Aquarius'], 'energy': 'Not a Zodiac element'}
```

Try/Except to Get a Key

We saw that we can avoid `KeyError`s 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"`.

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.

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`.

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`.

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`.

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)
```

```
{'health potion': 10, 'cake of the cure': 5, 'green
elixir': 20, 'strength sandwich': 25}
65
```

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], "Pedr
o":[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:

```
>>>      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.

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.

3.

Print `users` to the console.

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)
```

```
dict_keys(['teraCoder', 'pythonGuy', 'samTheJavaMaam',
'lyleLoop', 'keysmithKeith'])
dict_keys(['functions', 'syntax', 'control flow',
'loops', 'lists', 'classes', 'dictionaries'])
```

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   "Pedr
o"  98  96  95   "Martin"  78  80  78   "Dina"  64  60  75

for score_list in test_scores values
        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.

2.

Iterate through the values in the `num_exercises` list and add each value to the `total_exercises` variable.

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)
```

```
115
```

Get All Items

You can get both the keys and the values with the `.items()` method. Like `.keys()` and `.values()`, it returns a `dict_list` object. Each element of the `dict_list` returned by `.items()` is a tuple consisting of:

```
(key, value)
```

so to iterate through, you can use this syntax:

```
biggest_brands    "Apple"  184  "Google"  141.7  "Microsoft"  80  "Coca-
Cola"  69.7  "Amazon"  64.8

for company  value in biggest_brands items
        company   " has a value of "       value    " billion dollars. "
```

which would yield this output:

```
Apple has a value of 184 billion dollars.
Google has a value of 141.7 billion dollars.
Microsoft has a value of 80 billion dollars.
Coca-Cola has a value of 69.7 billion dollars.
Amazon has a value of 64.8 billion dollars.
```

Instructions

1.

Use a for loop to iterate through the items of `pct_women_in_occupation`. For each key : value pair, print out a string that looks like:

```
Women make up [value] percent of [key]s.
```

script.py

```
pct_women_in_occupation = {"CEO": 28, "Engineering Manager": 9, "Pharmacist": 58,
 "Physician": 40, "Lawyer": 37, "Aerospace Engineer": 9}

for position, number in pct_women_in_occupation.items():
  print("Women make up " + str(number) + " percent of " + position)
```

```
Women make up 28 percent of CEO
Women make up 9 percent of Engineering Manager
Women make up 58 percent of Pharmacist
Women make up 40 percent of Physician
Women make up 37 percent of Lawyer
Women make up 9 percent of Aerospace Engineer
```

Review

In this lesson, you've learned how to go through dictionaries and access keys and values in different ways. Specifically, you have seen how to:

- Use a key to get a value from a dictionary
- Check for existence of keys
- Remove a key: value pair from a dictionary
- Iterate through keys and values in dictionaries

Instructions

1.

We have provided a pack of tarot cards, `tarot`. You are going to do a three card spread of your past, present, and future.

Create an empty dictionary called `spread`.

2.

The first card you draw is card 13. Pop the value assigned to the key `13` out of the `tarot` dictionary and assign it as the value of the `"past"` key of `spread`.

Stuck? Get a hint

3.

The second card you draw is card 22. Pop the value assigned to the key `22` out of the `tarot` dictionary and assign it as the value of the `"present"` key of `spread`.

4.

The third card you draw is card 10. Pop the value assigned to the key `10` out of the `tarot` dictionary and assign it as the value of the `"future"` key of `spread`.

5.

Iterate through the items in the `spread` dictionary and for each key: value pair, print out a string that says:

```
Your {key} is the {value} card.
```

6.

Congratulations! You have learned about how to modify and use dictionaries. Hit the `Run` button one more time when you are ready to continue.

script.py

```
tarot = { 1:  "The Magician", 2:  "The High Priestess", 3:  "The Empress", 4: "Th
e Emperor", 5: "The Hierophant", 6:  "The Lovers", 7:  "The Chariot", 8: "Strengt
h", 9:  "The Hermit", 10: "Wheel of Fortune", 11: "Justice", 12:  "The Hanged Man
", 13: "Death", 14:  "Temperance", 15: "The Devil", 16:  "The Tower", 17:  "The S
tar", 18: "The Moon", 19: "The Sun", 20:  "Judgement", 21:  "The World", 22: "The
 Fool"}

spread = {}
spread["past"] = tarot.pop(13)
spread["present"] = tarot.pop(22)
spread["future"] = tarot.pop(10)

for time, card in spread.items():
  print("Your " + time + " is the " + card + " card.")
```

```
Your past is the Death card.
Your present is the The Fool card.
Your future is the Wheel of Fortune card.
```