

Python for Data Science

Lesson 11

Dictionaries

```
In [2]: list = ["abc", "xyz", "abc"]  
print(list)  
print(set(list))
```

```
['abc', 'xyz', 'abc']  
{'abc', 'xyz'}
```

```
In [4]: sampleSet = { "Yellow", "Orange", "Black"}  
sampleSet.update(["Blue", "Green", "Red"])  
print(sampleSet)
```

```
{'Yellow', 'Red', 'Orange', 'Black', 'Blue', 'Green'}
```

Adapted from:

How to Think Like a Computer Scientist: Interactive Edition

Resources:

- [How to Think Like a Computer Scientist: Interactive Edition](#)

-
- [Dictionaries in Python](#)
 - [12 Examples to Master Python Dictionaries \(Toward Data Science\)](#)
 - [Python Dictionary \(Programiz\)](#)
 - [Python Dicts \(Stanford Computer Science\)](#)

TABLE OF CONTENTS

Overview

1. [Dictionaries Defined](#)
 - a. [Creating dictionaries](#)
2. [Dictionary Operations](#)
 - a. [del statement](#)
 - b. [Modifying a dictionary value](#)
 - c. [len\(\) function with dictionaries](#)
3. [Dictionary Methods](#)
 - a. [keys method](#)

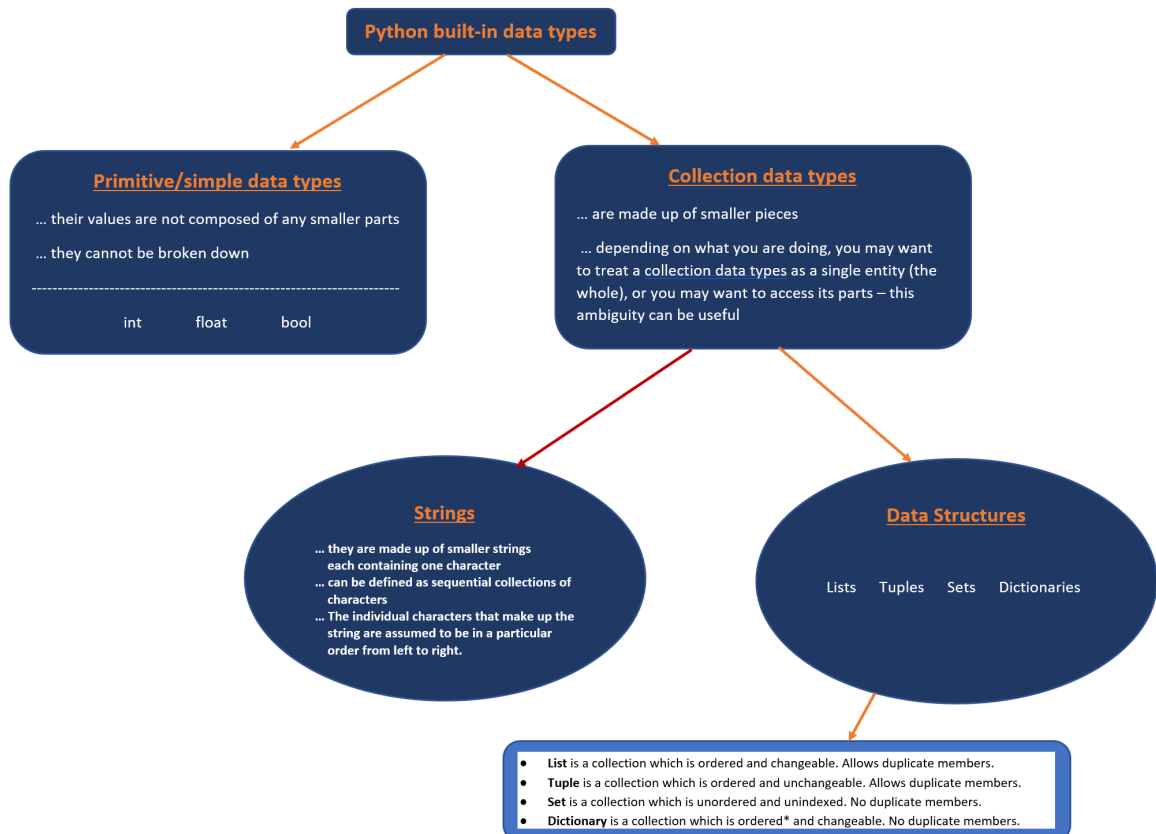
- b. values method
 - c. items method
 - d. in and not in operators
 - e. get(key) method & get(key, alt) method
4. Aliasing and Copying Dictionaries
- a. Aliasing with dictionaries
 - b. Copying a dictionary
5. Sparse Matrices
- a. Representing a sparse matrix using a list of lists
 - b. Representing a sparse matrix using a dictionary

For this interactive presentation, you will read through this notebook then complete the Practice Problems that are found in the green cells below.

```
In [5]: # set up notebook to display multiple output in one cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Overview



1. Dictionaries Defined

- Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates

a. Creating dictionaries

- All of the compound data types we have studied in detail so far — strings, lists, and tuples — are sequential collections.
- This means that the items in the collection are ordered from left to right and they use integers as indices to access the values they contain.
- Dictionaries are a different kind of collection.
- They are Python's built-in mapping type.
- A map is an unordered, associative collection.
- The association, or mapping, is from a key, which can be any immutable type, to a value, which can be any Python data object.
- As an example, we will create a dictionary that associates a state with its capital. For this dictionary, the keys are strings and the values will also be strings.
- One way to create a dictionary is to start with the empty dictionary and add key-value pairs. The empty dictionary is denoted {}

In [6]: *# Creating a dictionary by adding key-value pairs*

```
capitals = {}  
print(capitals)  
  
capitals['Wisconsin'] = "Madison"  
print(capitals)  
  
capitals["Illinois"] = "Springfield"  
print(capitals)  
  
capitals["Texas"] = "Austin"  
print(capitals)  
  
capitals["Ohio"] = "Columbus"  
print(capitals)
```

```
{}
```

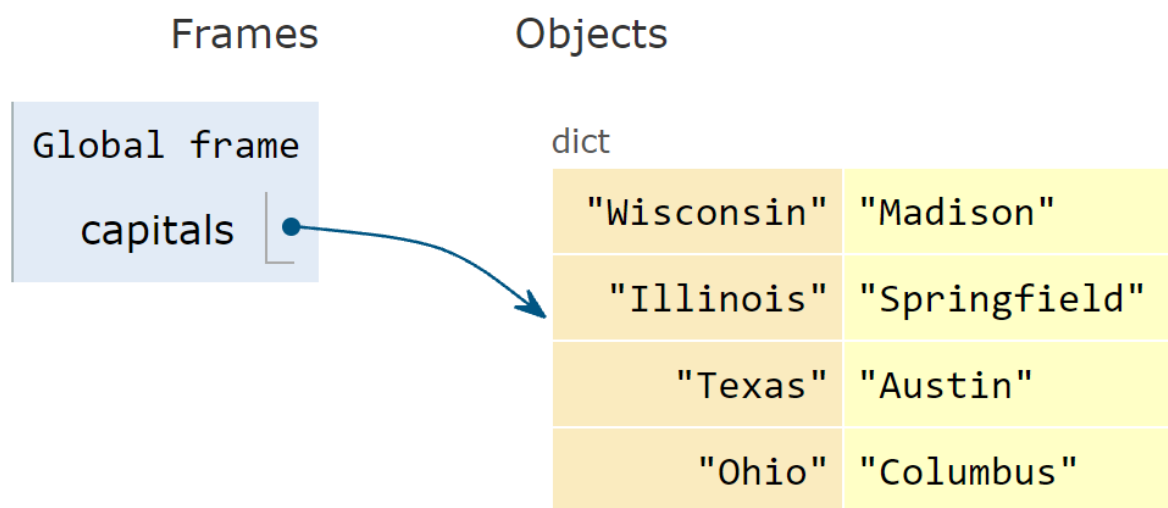
```
{'Wisconsin': 'Madison'}
```

```
{'Wisconsin': 'Madison', 'Illinois': 'Springfield'}
```

```
{'Wisconsin': 'Madison', 'Illinois': 'Springfield', 'Texas': 'Austin'}
```

```
{'Wisconsin': 'Madison', 'Illinois': 'Springfield', 'Texas': 'Austin', 'Ohio': 'Columbus'}
```

- The first assignment creates an empty dictionary named capitals.
- The other assignments add new key-value pairs to the dictionary.
- The left hand side gives the dictionary name and the key being associated. The right hand side gives the value being associated with that key.
- We can print the current value of the dictionary in the usual way.
- The key-value pairs of the dictionary are separated by commas.
- Each pair contains a key and a value separated by a colon.



- Another way to create a dictionary is to provide a list of key-value pairs using the same syntax as the previous output.

```
In [8]: # Creating a dictionary by providing a list of key-value pairs
```

```
capitals = {"Wisconsin": "Madison", "Illinois": "Springfield", "Texas": "Austin", "Ohio": "Columbus"}
```

```
print(capitals)
```

```
{'Wisconsin': 'Madison', 'Illinois': 'Springfield', 'Texas': 'Austin', 'Ohio': 'Columbus'}
```

```
In [7]: # Using a key to look up the corresponding value ... use [] notation
```

```
capitals = {"Wisconsin": "Madison", "Illinois": "Springfield", "Texas": "Austin", "Ohio": "Columbus"}
```

```
value = capitals["Texas"]
```

```
print(value)
```

Austin

Practice Problem 1: 1. Create a dictionary based on the **Compact SUVs** table found below by starting with an empty dictionary and then adding key-value pairs. Write a print

statement for each stage of the dictionary construction. 2. Create a dictionary based on the **Ivy League Schools** table found below by providing a list of key-value pairs. Consider using a multiline statement to create the dictionary. Print out the resulting dictionary. 3. Write code to print out the value of the model that corresponds to the Kia make. 4. Write code to print out the value of the state that corresponds to Princeton. ![dicts-2.PNG](attachment:dicts-2.PNG)

```
In [7]: compact_suvs = {
    "Honda": "CR-V",
    "Mazda": "CX-5",
    "Kia": "Sportage",
    "Subaru": "Forester",
    "Nissan": "Rogue",
    "Toyota": "Rav4"
}

ivy_league_schools = {
    "Brown": "Rhode Island",
    "Columbia": "New York",
    "Cornell": "New York",
    "Dartmouth": "New Hampshire",
    "Harvard": "Massachusetts",
    "Penn": "Pennsylvania",
    "Princeton": "New Jersey",
    "Yale": "Connecticut"
}

print(f"{compact_suvs} \n\n{ivy_league_schools}")
print("\n")
print({compact_suvs["Kia"]},{ivy_league_schools["Princeton"]} )

{'Honda': 'CR-V', 'Mazda': 'CX-5', 'Kia': 'Sportage', 'Subaru': 'Forester', 'Nissan': 'Rogue', 'Toyota': 'Rav4'}

{'Brown': 'Rhode Island', 'Columbia': 'New York', 'Cornell': 'New York', 'Dartmouth': 'New Hampshire', 'Harvard': 'Massachusetts', 'Penn': 'Pennsylvania', 'Princeton': 'New Jersey', 'Yale': 'Connecticut'}

{'Sportage'} {'New Jersey'}
```

2. Dictionary Operations

a. del statement

- The del statement removes a key-value pair from a dictionary.
- For example, the following dictionary contains the names of various candy bars and the number of each candy bar in stock. If someone buys all of the Snickers, we can remove the entry from the dictionary.

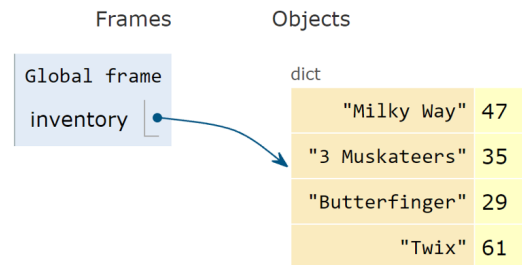
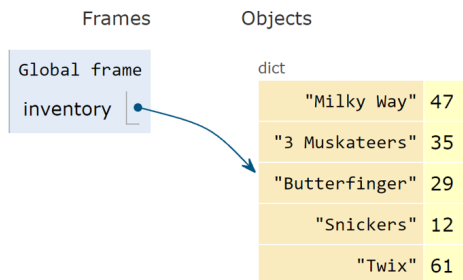
In [9]: *# del statement to remove a key-value pair from a dictionary*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
print(inventory)

del inventory["Snickers"]
print(inventory)
```

```
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Snickers': 12, 'Twix': 61}
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Twix': 61}
```

del inventory["Snickers"]



- Dictionaries are also mutable.
- As we've seen before with lists, this means that the dictionary can be modified by referencing an association on the left hand side of the assignment statement.
- For example, instead of deleting the entry for Snickers, we could have set its inventory to 0.

b. Modifying a dictionary value

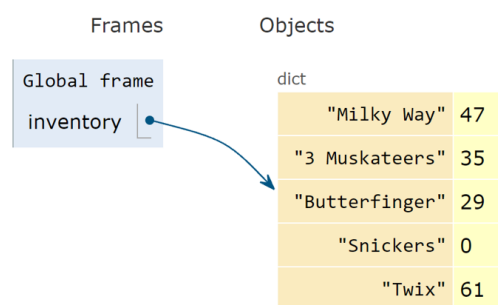
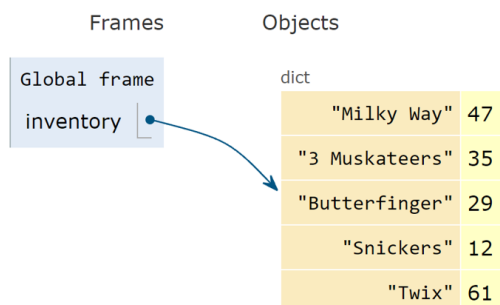
In [15]: *# Modifying a dictionary value by referencing an association on the left hand side of the assignment statement*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
print(inventory)

inventory["Snickers"] = 0
print(inventory)
```

```
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Snickers': 12, 'Twix': 61}
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Snickers': 0, 'Twix': 61}
```

inventory["Snickers"] = 0



- Similarly, a new shipment of 75 Twix arriving could be handled as follows:

In [11]: *# Modifying a dictionary value*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, ""}
print(inventory)
```

```
inventory["Twix"] = inventory["Twix"] + 75
print(inventory)
```

```
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Snickers': 12, 'Twix': 61}
{'Milky Way': 47, '3 Muskateers': 35, 'Butterfinger': 29, 'Snickers': 12, 'Twix': 136}
```

```
inventory["Twix"] = inventory["Twix"] + 75
```

Global frame	dict
inventory	
	"Milky Way" 47
	"3 Muskateers" 35
	"Butterfinger" 29
	"Snickers" 12
	"Twix" 61

Global frame	dict
inventory	
	"Milky Way" 47
	"3 Muskateers" 35
	"Butterfinger" 29
	"Snickers" 12
	"Twix" 136

c. len() function with dictionaries

- The len function also works on dictionaries. It returns the number of key-value pairs.

In [13]: *# Using the len() function to find the number of items (i.e. key-value pairs) in a dict*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, ""}
print(len(inventory))
```

5

Practice Problem 2: For this problem, use the **Toothpaste Inventory** table found below. 1. Write code to create a toothpaste inventory dictionary. Print out the results. 2. Write code to eliminate the key-value pair that corresponds to **Aquafresh**. Print out the results. 3. Write code to indicate that **Aim** has sold out. Print out the results. 4. Write code to indicate that the **Crest** inventory has increased by 15 and the **Colgate** inventory has decreased by 8. Print out the results. 5. Write code to print out the number of items in your current dictionary. !
[toothpaste.PNG](attachment:toothpaste.PNG)

```
In [10]: toothpaste ={
    "Colgate": 24,
    "Sensodyne": 18,
    "Crest": 20,
    "Aquafresh": 12,
    "Close Up": 14,
```

```

    "Pepsodent": 17,
    "Arm and Hammer": 16,
    "Aim": 10
}

del toothpaste["Aquafresh"]
print(toothpaste)
print("\n")

toothpaste["Aim"] = "Sold Out"
print(toothpaste)

toothpaste["Crest"] = 15
toothpaste["Colgate"] = 8
print(toothpaste)
print(f"\n{len(toothpaste)} ")

```

```
{'Colgate': 24, 'Sensodyne': 18, 'Crest': 20, 'Close Up': 14, 'Pepsodent': 17, 'Arm and Hammer': 16, 'Aim': 10}
```

```
{'Colgate': 24, 'Sensodyne': 18, 'Crest': 20, 'Close Up': 14, 'Pepsodent': 17, 'Arm and Hammer': 16, 'Aim': 'Sold Out'}
```

```
{'Colgate': 8, 'Sensodyne': 18, 'Crest': 15, 'Close Up': 14, 'Pepsodent': 17, 'Arm and Hammer': 16, 'Aim': 'Sold Out'}
```

7

3. Dictionary Methods

- Dictionaries have a number of useful built-in methods. The following table provides a summary and more details can be found in the [Python Documentation](#).

Method	Parameters	Description
keys	none	Returns a view of the keys in the dictionary
values	none	Returns a view of the values in the dictionary
items	none	Returns a view of the key-value pairs in the dictionary
get	key	Returns the value associated with key; None otherwise
get	key,alt	Returns the value associated with key; alt otherwise

keys method

a. keys method

- The keys method returns what Python 3 calls a **view** of its underlying keys.
- We can iterate over the view (we will cover iteration in Unit 14 Loops & List Comprehensions)
- We can also turn the view into a list by using the list conversion function.

In [23]: *# keys method for dictionaries*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
print(inventory.keys())

dict_keys(['Milky Way', '3 Muskateers', 'Butterfinger', 'Snickers', 'Twix'])
```

In [29]: *# Iterating over the keys of a dictionary*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
for key in inventory.keys():
    print("This is the key", key, "which maps to the value", inventory[key])
```

```
This is the key Milky Way which maps to the value 47
This is the key 3 Muskateers which maps to the value 35
This is the key Butterfinger which maps to the value 29
This is the key Snickers which maps to the value 12
This is the key Twix which maps to the value 61
```

- It is so common to iterate over the keys in a dictionary that you can omit the keys method call in the for loop — iterating over a dictionary implicitly iterates over its keys.

In [34]: *# Iterating over the keys of a dictionary implicitly*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
for x in inventory:
    print("This is the key", x, "which maps to the value", inventory[x])
```

```
This is the key Milky Way which maps to the value 47
This is the key 3 Muskateers which maps to the value 35
This is the key Butterfinger which maps to the value 29
This is the key Snickers which maps to the value 12
This is the key Twix which maps to the value 61
```

In [32]: *# Converting the keys of a dictionary to a List*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Twix": 61}
keys_list = list(inventory.keys())
print(keys_list)
```

```
['Milky Way', '3 Muskateers', 'Butterfinger', 'Snickers', 'Twix']
```

- As we saw earlier with strings and lists, dictionary methods use dot notation, which specifies the name of the method to the right of the dot and the name of the object on which to apply the method immediately to the left of the dot.
- The empty parentheses in the case of keys indicate that this method takes no parameters.

Practice Problem 3: For this problem, use the **Top Women Tennis Players** table found

below. 1. Write code to create a dictionary for the top 3 women tennis players. Use the player names for the keys. Print out the results. 2. Write code that prints out the keys of the dictionary that you created in Part (1). 3. Write code to convert the keys of the dictionary that you created in Part (1) to a list. Print out the results. ![tennis.PNG] (attachment:tennis.PNG)

```
In [15]: top_women_tennis_players = {
    "Ashleigh Barty": 1,
    "Aryna Sabalenka": 2,
    "Karolina Pliskova": 3,
    "Iga Swiatek": 4,
    "Sei Young Kim": 4,
    "Barbors Krejčíková": 5,
    "Elina Svitolina": 6,
    "Naomi Osaka": 7
}
print(top_women_tennis_players.keys())
print("\n")
print(list(top_women_tennis_players.keys()))
```

```
dict_keys(['Ashleigh Barty', 'Aryna Sabalenka', 'Karolina Pliskova', 'Iga Swiatek', 'Sei Young Kim', 'Barbors Krejčíková', 'Elina Svitolina', 'Naomi Osaka'])
```

```
['Ashleigh Barty', 'Aryna Sabalenka', 'Karolina Pliskova', 'Iga Swiatek', 'Sei Young Kim', 'Barbors Krejčíková', 'Elina Svitolina', 'Naomi Osaka']
```

b. values method

- The values method is similar to the keys method.
- It returns a view object which can be turned into a list or iterated over directly.

```
In [39]: # values method for dictionaries

inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "M&M's": 61}
values = inventory.values()
print(values)

values_list = list(inventory.values())
print(values_list)

dict_values([47, 35, 29, 12, 61])
[47, 35, 29, 12, 61]
```

Practice Problem 4: For this problem, use the top 3 women tennis player dictionary that you created in Problem 3 above. 1. Write code that prints out the values of the dictionary. 2. Write code to convert the values of the dictionary to a list. Print out the results.

```
In [16]: print(top_women_tennis_players.values())
print("\n")
```

```
print(list(top_women_tennis_players.values()))
```

```
dict_values([1, 2, 3, 4, 4, 5, 6, 7])
```

```
[1, 2, 3, 4, 4, 5, 6, 7]
```

c. items method

- The items method is similar to the keys and values methods.
- It returns a view objects which can be turned into a list or iterated over directly.
- Note that the items are shown as tuples containing the key and the associated value.

In [40]: *# items method for dictionaries*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, ""  
items = inventory.items()  
print(items)
```

```
items_list = list(inventory.items())  
print(items_list)
```

```
dict_items([('Milky Way', 47), ('3 Muskateers', 35), ('Butterfinger', 29), ('Snickers',  
12), ('Twix', 61)])  
[('Milky Way', 47), ('3 Muskateers', 35), ('Butterfinger', 29), ('Snickers', 12), ('Twix', 61)]
```

In [18]: *# iterating over the items of a dictionary*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, ""  
items = inventory.items()  
print(items)
```

```
print()
```

```
for (k,v) in inventory.items():  
    print("This is the key", k, "which maps to the value", v)
```

```
print()
```

```
for k in inventory:  
    print("This is the key", k, "which maps to the value", inventory[k])
```

```
dict_items([('Milky Way', 47), ('3 Muskateers', 35), ('Butterfinger', 29), ('Snickers', 12), ('Twix', 61)])
```

```
This is the key Milky Way which maps to the value 47
This is the key 3 Muskateers which maps to the value 35
This is the key Butterfinger which maps to the value 29
This is the key Snickers which maps to the value 12
This is the key Twix which maps to the value 61
```

```
This is the key Milky Way which maps to the value 47
This is the key 3 Muskateers which maps to the value 35
This is the key Butterfinger which maps to the value 29
This is the key Snickers which maps to the value 12
This is the key Twix which maps to the value 61
```

- Note that tuples are often useful for getting both the key and the value at the same time while you are looping.
- The two loops above do the same thing.

Practice Problem 5: For this problem, use the top 3 women tennis player dictionary that you created in Problem 3 above. 1. Write code that prints out the items of the dictionary. 2. Write code to convert the items of the dictionary to a list. Print out the results.

```
In [17]: print(top_women_tennis_players.items())
print("\n")
print(list(top_women_tennis_players.items()))
```

```
dict_items([('Ashleigh Barty', 1), ('Aryna Sabalenka', 2), ('Karolina Pliskova', 3),
('Iga Swiatek', 4), ('Sei Young Kim', 4), ('Barbors Krejckikova', 5), ('Elina Svitolina', 6), ('Naomi Osaka', 7)])
```

```
[('Ashleigh Barty', 1), ('Aryna Sabalenka', 2), ('Karolina Pliskova', 3), ('Iga Swiatek', 4), ('Sei Young Kim', 4), ('Barbors Krejckikova', 5), ('Elina Svitolina', 6), ('Naomi Osaka', 7)]
```

d. in and not in operators

- The in and not in operators can test if a key is in the dictionary:

```
In [48]: # in and not in operators with dictionaries

inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, "Almond Joy": 61}
print("3 Muskateers" in inventory)
print("Almond Joy" in inventory)

print()

# in operator with a dictionary in a conditional statement

if "3 Muskateers" in inventory:
```

```
print("There are", inventory["3 Muskateers"], "3 Muskateers in inventory.")
else:
    print("There are no 3 Muskateers in inventory.")
```

True
False

There are 35 3 Muskateers in inventory.

- These operators can be very useful since looking up a non-existent key in a dictionary causes a runtime error.

e. get(key) & get(key, alt)

- The get method allows us to access the value associated with a key, similar to the [] operator.
- The important difference is that get will not cause a runtime error if the key is not present.
- It will instead return None.
- There exists a variation of get that allows a second parameter that serves as an alternative return value in the case where the key is not present. This can be seen in the example below. In this case, since "Baby Ruth" is not a key, return "We do not stock Baby Ruth's." (instead of None).

In [53]: *# get method for dictionaries*

```
inventory = {"Milky Way": 47, "3 Muskateers": 35, "Butterfinger": 29, "Snickers": 12, ""
print(inventory.get("Butterfinger"))
print(inventory.get("Baby Ruth"))

print(inventory.get("Baby Ruth", "We do not stock Baby Ruth's."))
```

29
None
We do not stock Baby Ruth's.

Practice Problem 6: For this problem, use the top 3 women tennis player dictionary that you created in Problem 3 above. 1. Write code that prints out that indicates whether or not Naomi Osaka is a member of the top 3 women tennis player dictionary. 2. Write code that prints out that indicates whether or not Aryna Sabalenka is a member of the top 3 women tennis player dictionary. 3. Write code that uses the get() method to print out the values associated with Karolina Pliskova. 4. Write code that uses the get() method to print out the values associated with Barbora Krejčíková. 5. Write code that uses the get() method to print out the values associated with Barbora Krejčíková, but if Barbora Krejčíková is not a top 3 player, the code should return "She is not a top 3 player."

In [23]:

```
print("Naomi Osaka" in top_women_tennis_players)
print("Aryna Sabalenka" in top_women_tennis_players)
```

```
print(top_women_tennis_players.get("Karolina Pliskova"))
print(top_women_tennis_players.get("Barbors Krejcikova"))
if top_women_tennis_players.get("Barbors Krejcikova") > 3:
    print("Barbors Krejcikova is not a top 3 player")
```

```
True
True
3
5
Barbors Krejcikova is not a top 3 player
```

4. Aliasing and Copying Dictionaries

a. Aliasing with dictionaries

- Because dictionaries are mutable, you need to be aware of aliasing (as we saw with lists).
- Whenever two variables refer to the same dictionary object, changes to one affect the other.
- For example, opposites is a dictionary that contains pairs of opposites.

In [23]: *# aliasing with dictionaries*

```
homonyms = {"dear" : "deer", "hoarse" : "horse", "cent" : "scent", "won" : "one"}
print(homonyms)

print()

alias = homonyms
print(alias)

print()

print(alias == homonyms)
print(alias is homonyms)

print()

print(id(alias))
print(id(homonyms))

print()

alias["cent"] = "sent"

print(alias)
print(homonyms)
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'scent', 'won': 'one'}
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'scent', 'won': 'one'}
```

```
True
```

```
True
```

```
1828063810880
```

```
1828063810880
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'sent', 'won': 'one'}
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'sent', 'won': 'one'}
```

- As you can see from the `is` operator and the `id()` built-in function `alias` and `homonyms` refer to the same object.

Run the code above using the [Python Tutor -- Visualize Code Execution](#) tool.

Practice Problem 7: For this problem, use the **Synonyms** table found below. 1. Write code to create a dictionary based on the **Synonyms** table. Use the words for the keys. Print out the results. 2. Write code to create a second dictionary that is equal to the dictionary that you created in Part (1). Print out the results. 3. Write out code that would allow you to determine if your two dictionaries contain the same items. Print out the results. 4. Write out code that would allow you to determine if your two dictionaries reference the same object. Do this two different ways and print out the results. 5. In your second dictionary, write out code that changes the value that corresponds to the key "beautiful" to "pretty". Print out both of your dictionaries. 6. Use a comment to write out what you observed in Part (5). !

[Synonyms.PNG](attachment:Synonyms.PNG)

```
In [34]: synonyms = {
    "good": "fine",
    "beautiful": "attractive",
    "important": "key",
    "sad": "sorrowful",
    "happy": "content"
}
print(synonyms, end="\n")
duplicate = synonyms
print(duplicate)

print(synonyms == duplicate)
print(synonyms is duplicate)
print(id(synonyms) == id(duplicate))
duplicate["beautiful"] = "pretty"
print(duplicate, synonyms, sep="\n")

print("Because duplicate and synonyms point to the same object, both dictionaries get cl
```

```
{'good': 'fine', 'beautiful': 'attractive', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
{'good': 'fine', 'beautiful': 'attractive', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
True
True
True
{'good': 'fine', 'beautiful': 'pretty', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
{'good': 'fine', 'beautiful': 'pretty', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
```

Because duplicate and synonyms point to the same object, both dictionaries get changed when you only change one of them.

Copying a dictionary

b. Copying a dictionary

- If you want to modify a dictionary and keep a copy of the original, use the dictionary copy method.
- Since `homonyms_copy` below is a copy of the dictionary `homonyms`, changes to it will not effect the original.

```
In [63]: # copying a dictionary

homonyms = {"dear" : "deer", "hoarse" : "horse", "cent" : "scent", "won" : "one"}
print(homonyms)

print()

homonyms_copy = homonyms.copy()
homonyms_copy["cent"] = "sent"

print(homonyms_copy == homonyms)
print(homonyms_copy is homonyms)

print()

print(id(homonyms_copy))
print(id(homonyms))

print()

print(homonyms_copy)
print(homonyms)
```



```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'scent', 'won': 'one'}
```

```
False
```

```
False
```


```
1433111983168
```

```
1433111982976
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'sent', 'won': 'one'}
```

```
{'dear': 'deer', 'hoarse': 'horse', 'cent': 'scent', 'won': 'one'}
```

Run the code above using the [Python Tutor -- Visualize Code Execution](#) tool.

Practice Problem 8: For this problem, use the **Synonyms** table found below. 1. Write code to create a dictionary based on the **Synonyms** table. Use the words for the keys. Print out the results. 2. Write code to create a second dictionary that is copy of the dictionary that you created in Part (1). Print out the results. 3. In your second dictionary (i.e., in your copy), write code that changes the value that corresponds to the key "beautiful" to "pretty". Print out the results 4. Write code that would allow you to determine if your two dictionaries contain the same items. Print out the results. 5. Write code that would allow you to determine if your two dictionaries reference the same object. Do this two different ways and print out the results. 6. Write code to print out both of your dictionaries. 7. Use a comment to write out what you observed in Part (6). ! (attachment:Synonyms.PNG)

```
In [41]: synonyms = {
    "good": "fine",
    "beautiful": "attractive",
    "important": "key",
    "sad": "sorrowful",
    "happy": "content"
}
print(synonyms)
copy = synonyms.copy()
print(copy)
copy["beautiful"] = "pretty"
print(copy)

print(copy == synonyms)
print(copy is synonyms)
print(id(copy) == id(synonyms))
print(copy, synonyms, sep="\n")

print("Because copy and synonyms do not point to the same object, when you change one d:
```

```
{'good': 'fine', 'beautiful': 'attractive', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
{'good': 'fine', 'beautiful': 'attractive', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
{'good': 'fine', 'beautiful': 'pretty', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
False
False
False
{'good': 'fine', 'beautiful': 'pretty', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
{'good': 'fine', 'beautiful': 'attractive', 'important': 'key', 'sad': 'sorrowful', 'happy': 'content'}
```

Because copy and synonyms do not point to the same object, when you change one dictionary, the other dictionary does not automatically change.

5. Sparse Matrices

a. Representing a sparse matrix using a list of lists

- A matrix is a two dimensional collection, typically thought of as having rows and columns of data.
- One of the easiest ways to create a matrix is to use a list of lists. For example, consider the matrix shown below.

$$\begin{bmatrix} 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- We can represent this collection as five rows, each row having five columns.
- Using a list of lists representation, we will have a list of five items, each of which is a list of five items.
- The outer items represent the rows and the items in the nested lists represent the data in each column.

In [72]: *# Using a List of Lists to represent a matrix*

```
matrix = [[0, 0, 0, 4, 0],
           [0, 0, 0, 0, 0],
           [0, 3, 0, 0, 0],
           [0, 0, 0, 0, 5],
           [0, 0, 0, 0, 0]]
print(matrix)
```

```
[[0, 0, 0, 4, 0], [0, 0, 0, 0, 0], [0, 3, 0, 0, 0], [0, 0, 0, 0, 5], [0, 0, 0, 0, 0]]
```

In [73]: *# Accessing an element of a matrix being represented as a List of Lists*

```
print(matrix[3][4])
```

5

- One thing that you might note about this example matrix is that there are many items that are zero.
- In fact, only three of the data values are nonzero. This type of matrix has a special name. It is called a **sparse matrix**.
- Since there is really no need to store all of the zeros, the list of lists representation is considered to be inefficient.
- An alternative representation is to use a dictionary.
- For the keys, we can use tuples that contain the row and column numbers.
- Here is the dictionary representation of the same matrix.

b. Representing a sparse matrix using a dictionary

In [75]: *# Using a dictionary to represent a sparse matrix*

```
matrix = {(0, 3): 4, (2, 1): 3, (3, 4): 5}
print(matrix)
```

```
{(0, 3): 4, (2, 1): 3, (3, 4): 5}
```

- We only need three key-value pairs, one for each nonzero element of the matrix.
- Each key is a tuple, and each value is an integer.
- To access an element of the matrix, we could use the [] operator:

In [1]: *# Accessing an element of a matrix being represented by a dictionary*

```
matrix = {(0, 3): 4, (2, 1): 3, (3, 4): 5}
matrix[(3, 4)]
```


Out[1]: 5

- Notice that the syntax for the dictionary representation is not the same as the syntax for the nested list representation. - - Instead of two integer indices, we use one index, which is a tuple of integers.
- There is one problem. If we specify an element that is zero, we get an error, because there is no entry in the dictionary with that key. The alternative version of the get method solves this problem. The first argument will be the key. The second argument is the value get should return if the key is not in the dictionary (which would be 0 since it is sparse).

In [76]: *# Using the get() method when accessing an element of a matrix being represented by a dictionary*

```
matrix = {(0, 3): 4, (2, 1): 3, (3, 4): 5}
print(matrix.get((2, 1)))
print(matrix.get((4, 2), 0))
```

```
3
0
```

Practice Problem 9: For this problem, use the **matrix** found below.  (attachment:matrix2.PNG) 1. Write code to represent the given matrix as a list of lists and print out the results. 2. Write code to access the element of the matrix that is in row 3, column 5. Print out the results. 3. Write code that uses a dictionary to represent the given matrix and print out the results. 4. Write code that accesses the element of the matrix that is in row 4, column 2 from the dictionary representation of the matrix. 5. Use the get() method to access the element of the matrix that is in row 2, column 4 from the dictionary representation of the matrix. Your code should return a 0 if there is no entry in the dictionary that corresponds to the key that you use.

In [46]:

```
matrix = [[0, 0, 3, 0, 0],
          [1, 0, 0, 0, 0],
          [0, 0, 0, 0, 6],
          [0, 7, 0, 0, 0],
          [0, 0, 0, 5, 0]]
print(matrix)
print(matrix[2][4])

matrix = {(0, 2): 3, (1, 0): 1, (2, 4): 6, (3,1):7, (4,4):5}
print(matrix[(3,1)])
print(matrix.get((1, 3), 0))
```

```
[[0, 0, 3, 0, 0], [1, 0, 0, 0, 0], [0, 0, 0, 0, 6], [0, 7, 0, 0, 0], [0, 0, 0, 5, 0]]
6
7
0
```

In []: