Dictionaries

You have earned 0 point(s) out of 0 point(s) thus far.

Definitions

In this topic, we will examine the collection types which store **item pairs**. What does it mean?

Think of a real dictionary. It contains words and their meanings. In Python, you can accept the words as key and the meaning of the words as value.

A dictionary in Python is a collection of key-value pairs called **items** of a dictionary. The dictionary is enclosed by curly braces 👉🏻**{}**. Each pair (item) is separated by a comma and the key and value are separated by a colon.

## Dictionaries

You have earned 0 point(s) out of 0 point(s) thus far.

### Creating a Dictionary

A dictionary also can be created by enclosing pairs, separated by commas, in **curly-braces**. Looks like list or tuple, right?

And of course, we can use a function to create a dictionary : 'dict()' function. Let's create a simple empty dict :



empty\_dict\_1 = {}

empty\_dict\_2 = dict()

This is our first dict in this lesson. Now let's print its type.

input :



empty\_dict\_1 = {}

print(type(empty\_dict\_1))

output :



<class 'dict'>

The basic form of dict looks like :



my\_dict = {'key1': 'value1',

'key2': 'value2',

'key3': 'value3'

}

The syntax for accessing an **item** is very simple. We write a key that we want to access in square brackets. This method works both for adding items to a dict and for reading them from there.

In the following examples, you'll see several methods that allow us to create a dict and add a key-value pair to it.

input :



state\_capitals = {'Arkansas': 'Little Rock',

'Colorado': 'Denver',

'California': 'Sacramento',

'Georgia': 'Atlanta'

}

print(state\_capitals['Colorado']) # accessing method

output :



Denver

input :



state\_capitals = {'Arkansas': 'Little Rock',

'Colorado': 'Denver',

'California': 'Sacramento',

'Georgia': 'Atlanta'

}

state\_capitals['Virginia'] = 'Richmond' # adding a new

    item

print(state\_capitals)

output :



{'Arkansas': 'Little Rock',

'Colorado': 'Denver',

'California': 'Sacramento',

'Georgia': 'Atlanta',

'Virginia': 'Richmond'}

**💡Tips:**

* Note that keys and values can be of different types.



mix\_values = {'animal': ('dog', 'cat'), # tuple type

'planet': ['Neptun', 'Saturn', 'Jupiter'],

                  # list type

'number': 40, # int type

'pi': 3.14, # float type

'is\_good': True} # bool type

mix\_keys = {22 : "integer",

1.2 : "float",

True : "boolean",

"key" : "string"}

And now, let's use dict() function to create a dictionary :

input :



dict\_by\_dict = dict(animal='dog', planet='neptun', number

    =40, pi=3.14, is\_good=True)

print(dict\_by\_dict)

output :



{'animal': 'dog',

'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True}

**⚠️Avoid ! :**

* Do not use quotes for keys when using the dict() function to create a dictionary.
* You cannot use iterables as keys to create a dictionary.

**Q**: What is a dictionary in Python?  
**A**: Python dictionary is one of the supported data types in Python. It is an unordered collection of elements. The elements in dictionaries are stored as key–value pairs. Dictionaries are indexed by keys. For example, below we have a dict named my\_dict. It contains two keys, **fruit** and **vegatable**, along with their corresponding values, **banana** and **onion**.  
  
my\_dict = {'fruit':'banana', 'vegatable':'onion'}

**- Interview Q&A**

Dictionaries

You have earned 0 point(s) out of 0 point(s) thus far.

Main Operations with Dictionaries

There are several methods that allow us to access items, keys, and values. You can access all items using the **.items()** method, all keys using the **.keys()** method, and all values using the **.values()** method:

input :



dict\_by\_dict = {'animal': 'dog',

'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True}

print(dict\_by\_dict.items(), '\n')

print(dict\_by\_dict.keys(), '\n')

print(dict\_by\_dict.values())

output :



dict\_items([('animal', 'dog'), ('planet', 'neptun'),

('number', 40), ('pi', 3.14), ('is\_good', True)])

dict\_keys(['animal', 'planet', 'number', 'pi', 'is\_good'])

dict\_values(['dog', 'neptun', 40, 3.14, True])

You have learned that you can add a new item by assigning value to a key that is not in the dictionary. Likewise, you can add new items using the **.update()** method. Let's see :

input :



dict\_by\_dict = {'animal': 'dog',

'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True}

dict\_by\_dict.update({'is\_bad': False})

print(dict\_by\_dict)

output :



{'animal': 'dog',

'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True,

'is\_bad': False}

You can also remove an item using the **del** function:

**The formula syntax is : del dictionary\_name['key']**.

See the example.

input :



dict\_by\_dict = {'animal': 'dog',

'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True,

'is\_bad': False}

del dict\_by\_dict['animal']

print(dict\_by\_dict)

output :



{'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True,

'is\_bad': False}

Using the **in** and the **not in** operator, you can check if the key is in the dictionary.

* When we use the **in** operator; if the key is in the dictionary, the result will be True otherwise False.
* When we use the **not in**; if the key is not in the dictionary, the result will be True otherwise False.

Look at the example :

input :



dict\_by\_dict = {'planet': 'neptun',

'number': 40,

'pi': 3.14,

'is\_good': True,

'is\_bad': False}

print('pi' in dict\_by\_dict)

print('animal' not in dict\_by\_dict) # remember, we have deleted 'animal'

output :



True

True

Dictionaries

You have earned 0 point(s) out of 0 point(s) thus far.

Nested Dictionaries

In some cases, you need to work with the nested dict. When you decide to specialize in data science, we will work very often with dictionaries in the future.



school\_records={

"personal\_info":

{"kid":{"tom": {"class": "intermediate", "age": 10},

"sue": {"class": "elementary", "age": 8}

},

"teen":{"joseph":{"class": "college", "age": 19},

"marry":{"class": "high school", "age": 16}

},

},

"grades\_info":

{"kid":{"tom": {"math": 88, "speech": 69},

"sue": {"math": 90, "speech": 81}

},

"teen":{"joseph":{"coding": 80, "math": 89},

"marry":{"coding": 70, "math": 96}

},

},

}

We can use square brackets to access internal dicts :

input :



school\_records={

"personal\_info":

{"kid":{"tom": {"class":"intermediate", "age":10},

"sue": {"class":"elementary", "age":8}

},

"teen":{"joseph":{"class":"college", "age":19},

"marry":{"class":"high school", "age":16}

},

},

}

print(school\_records['personal\_info']['teen']['marry']['age'])

output :

2

16

| **nested_dictionary** |
| --- |
| *Diagram of Nested Dictionary* |

**💡Tips:**

* Dictionaries strongly resemble JSON syntax. The native json module in the Python standard library can be used to convert between JSON and dictionaries.

**✏️Homework:**

* What is '**JSON**' and what is it used for?

If you want to go deep into dicts, [**here**](https://docs.python.org/3.8/tutorial/datastructures.html#dictionaries) you will find what you want.

## Sets

### Definitions

A **set** is a collection of elements with no repeats and without insertion order but sorted order.

Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

They can hold multiple data in them, but only one of value. They are used in situations where it is only important that some things are grouped together, and not what order they were included.

Sets

Creating a Set

Curly braces **'{}'** or the **set()** function can be used to create sets. But the only way to create an empty set is: use the set() function.

**⚠️Avoid ! :**

* Note that, to create an empty set you have to use **set()** function. Do not use **{}** to create an empty set. Otherwise, you will create an empty dictionary.

Let's create a simple empty set :



empty\_set = set()

This is our first set in this lesson. Now let's print its type.

input :



empty\_set = set()

print(type(empty\_set))

output :



<class 'set'>

We will now see how sets have unordered and unique objects.

input :



colorset = {'purple', 'orange', 'red', 'darkblue', 'yellow', 'red'}

print(colorset)

print(colorset)

output :



{'darkblue', 'orange', 'purple', 'red', 'yellow'}

{'darkblue', 'purple', 'orange', 'yellow', 'red'}

As you can see in the output, the two 'red' values ​​we have defined in the set have fallen to one. And every time you print the set, the order of the objects in the set changes.

Let's look at another example :

input :



s = set('unselfishness')

print(s)

output :



{'f', 'l', 'i', 'u', 'e', 'n', 'h', 's'}

As you can see, the letters of the string type data are only written once in the set. Within this scope, using sets can help you avoid repetitions. Let's convert a list into a set and look at the repetitions of its elements:

input :



flower\_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']

flowerset = set(flower\_list)

flowerlist = list(flowerset)

print(flowerset)

print(flowerlist)

output :



{'orchid', 'carnation', 'violet', 'rose'}

['orchid', 'carnation', 'violet', 'rose']

**✏️Homework:**

* {'carnation', 'orchid', 'rose', 'violet'} 👈👉 {'rose', 'orchid', 'rose', 'violet', 'carnation'} Do these two sets give the same output and why? (Note: Try to figure out the answer before run on the Playground)

**Q**: Which one of the following is not the correct syntax for creating a set in Python?  
**A**:  
**a. set([[1,2],[3,4],[4,5]])**  
b. set([1,2,2,3,4,5])  
c. {1,2,3,4}  
d. set((1,2,3,4))  
  
**Explanation:** The iterable argument given for the set must be used in a correct way.

Main Operations with Sets

There are several methods that allow us to add and remove items to/from sets. Moreover, we have the methods of intersection, unification, and differentiation of sets :

These methods are :

* **.add()** : Adds a new item to the set.
* **.remove()** : Allows us to delete an item.
* **.intersection()** : Returns the intersection of two sets.
* **.union()** : Returns the unification of two sets.
* **.difference()** : Gets the difference of two sets.

Now, let's do some examples of these methods :

input :



a = set('abracadabra')

print(a)

output :



{'a', 'b', 'c', 'd', 'r'}

input :



a = set('abracadabra')

b = set('alacazam')

print(a - b) # same as '.difference()' method

print(a.difference(b)) # a difference from b

output :



{'b', 'd', 'r'}

{'b', 'd', 'r'}

input :



a = set('abracadabra')

b = set('alacazam')

print(a | b) # same as '.union()' method

print(a.union(b)) # unification of a with b

output :



{'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}

{'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}

input :



a = set('abracadabra')

b = set('alacazam')

print(a & b) # same as '.intersection()' method

print(a.intersection(b)) # intersection of a and b

output :



{'a', 'c'}

{'a', 'c'}

input :



a = set('abracadabra')

a.remove('c') # we delete 'c' from the set

print(a)

output :



{'a', 'b', 'd', 'r'}

input :



a = set('abracadabra')

a.add('c') # we add 'c' again into the set

print(a)

output :



{'a', 'b', 'c', 'd', 'r'}

Additionally, you can:

* Get the number of set’s elements using **len()** function,
* Check if an element belongs to a specific set(in / not in operators), you get the boolean value.

Thus, we have completed this topic which is the most important one in Python.

#### **After finishing the task correctly, then submit your answer (code) as plain text which shows you did correctly.**

**Task :**Find out the most frequent number and its frequency.

Write a program that;

* + Finds out the most frequent number in the given list.
  + Calculates its frequency.
  + Prints out the result such as :

| **Example** | |
| --- | --- |
| **Given list** | **Desired Output** |
| numbers = [1, 3, 7, 4, 3, 0, 3, 6, 3] | the most frequent number is 3 and it was 4 times repeated |

***Note****: You can/should use useful/necessary****built-in functions****and****methods****of the*list*operation.*

* [Assignment-009/2 (Comfortable Words)](https://lms.clarusway.com/mod/assign/view.php?id=10453)

**💡Objective:**

* + To improve your **knowledge** of **collection**types (set), **boolean**logic and raise your awareness of **"ten fingers keyboard"**.

#### **After finishing the task correctly, then submit your answer (code) as plain text which shows you did correctly.**

**Task :**Find out if the given word is "**comfortable words**" in relation to the *ten-finger keyboard use*.

* + A *comfortable word* is a word which you can type always alternating the hand you type with (assuming you type using a Q-keyboard and use of the ten-fingers standard).
  + The word will always be a string consisting of only letters from a to z.
  + Write a program which returns True if it's a comfortable word or False otherwise.

| **Examples** | |
| --- | --- |
| **Given  word** | **Desired Output (explanation)** |
| tester | False (uses only left-hand fingers)\* |
| polly | False (uses only right-hand fingers)\* |
| clarusway | True (uses both hand fingers)\* |

***Note****: Do a quick research on ten-fingers keyboard usage. (\*) the explanation doesn't need to be in the output.*

A comfortable word is a word which you can type always alternating the hand you type with (assuming you type using a QWERTY keyboard and use fingers as shown in the image below).

That being said, create a function which receives a word and returns true/True if it's a comfortable word and false/False otherwise.

The word will always be a string consisting of only ascii letters from a to z.

To avoid problems with image availability, here's the lists of letters for each hand:

Left: q, w, e, r, t, a, s, d, f, g, z, x, c, v, b

Right: y, u, i, o, p, h, j, k, l, n, m