# [4.1] Set and Dictionaries

June 26, 2022

## 1 Set

**Concept:** A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

[1]:
```
# Different types of sets in Python
# set of integers
my_set = {1, 2, 3}
print(my_set)
```

```
{1, 2, 3}
```

[2]:
```
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

```
{'Hello', 1.0, (1, 2, 3)}
```

[3]:
```
# set cannot have duplicates
my_set = {1, 2, 3, 4, 3, 2}
# Output: {1, 2, 3, 4}
print(my_set)
```

```
{1, 2, 3, 4}
```

[4]:
```
# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1, 2, 3, 2])
print(my_set)
```

```
{1, 2, 3}
```

[5]:
```
# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.

my_set = {1, 2, [3, 4]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4164/3902951068.py in <module>
      3 # this will cause an error.
      4
----> 5 my_set = {1, 2, [3, 4]}

TypeError: unhashable type: 'list'
```

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument.

[6]:
```
# Distinguish set and dictionary while creating empty set

# initialize a with {}
a = {}

# check data type of a
print(type(a))

# initialize a with set()
a = set()

# check data type of a
print(type(a))
```

```
<class 'dict'>
<class 'set'>
```

Sets are mutable. However, since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

We can add a single element using the **add()** method, and multiple elements using the **update()** method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

[7]:
```
# initialize my_set
my_set = {1, 3}
print(my_set)
```

```
{1, 3}
```

[8]:
```
my_set[0]
# you will get an error
# TypeError: 'set' object does not support indexing
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4164/3872634429.py in <module>
----> 1 my_set[0]
      2 # you will get an error
      3 # TypeError: 'set' object does not support indexing

TypeError: 'set' object is not subscriptable
```

[9]:
```
# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)
```

```
{1, 2, 3}
```

[10]:
```
# add multiple elements
my_set.update([2, 3, 4])
# Output: {1, 2, 3, 4}
print(my_set)
```

```
{1, 2, 3, 4}
```

[11]:
```
# add list and set
my_set.update([4, 5], {1, 6, 8})
# Output: {1, 2, 3, 4, 5, 6, 8}
print(my_set)
```

```
{1, 2, 3, 4, 5, 6, 8}
```

A particular item can be removed from a set using the methods **discard()** and **remove()**.

The only difference between the two is that the discard() function leaves a set unchanged if the element is not present in the set. On the other hand, the remove() function will raise an error in such a condition (if element is not present in the set).

The following example will illustrate this.

[12]:
```
# Difference between discard() and remove()

# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)
```

```
{1, 3, 4, 5, 6}
```

[13]:
```
# discard an element
my_set.discard(4)
# Output: {1, 3, 5, 6}
print(my_set)
```

```
{1, 3, 5, 6}
```

[14]:
```
# remove an element
my_set.remove(6)
# Output: {1, 3, 5}
print(my_set)
```

```
{1, 3, 5}
```

[15]:
```
# discard an element
# not present in my_set
my_set.discard(2)
# Output: {1, 3, 5}
print(my_set)
```

```
{1, 3, 5}
```

[16]:
```
# remove an element
# not present in my_set
# you will get an error.

my_set.remove(2)
# Output: KeyError
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4164/2965194814.py in <module>
      3 # you will get an error.
      4
----> 5 my_set.remove(2)
      6 # Output: KeyError

KeyError: 2
```

## 2 Dictionaries

**Concept:** A dictionary is an object that stores a collection of data. Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value.

When you hear the word "dictionary," you probably think about a large book such as the Merriam-Webster dictionary, containing words and their definitions. If you want to know the meaning of a

particular word, you locate it in the dictionary to find its definition.

In Python, a *dictionary* is an object that stores a collection of data. Each element that is stored in a dictionary has two parts: a *key* and a *value*. In fact, dictionary elements are commonly referred to as *key-value pairs*. When you want to retrieve a specific value from a dictionary, you use the key that is associated with that value. This is similar to the process of looking up a word in the Merriam-Webster dictionary, where the words are keys and the definitions are values.

For example, suppose each employee in a company has an ID number, and we want to write a program that lets us look up an employee's name by entering that employee's ID number. We could create a dictionary in which each element contains an employee ID number as the key and that employee's name as the value. If we know an employee's ID number, then we can retrieve that employee's name.

Another example would be a program that lets us enter a person's name and gives us that person's phone number. The program could use a dictionary in which each element contains a person's name as the key and that person's phone number as the value. If we know a person's name, then we can retrieve that person's phone number.

**Note:** Key-value pairs are often referred to as mappings because each key is mapped to a value.

## 2.1 Creating a Dictionary

You can create a dictionary by enclosing the elements inside a set of curly braces ( {} ).An element consists of a key, followed by a colon, followed by a value. The elements are separated by commas. The following statement shows an example:

[17]:
```
phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
```

This statement creates a dictionary and assigns it to the phonebook variable. The dictionary contains the following three elements:

- The first element is 'Chris':'555−1111'. In this element the key is 'Chris' and the value is '555−1111'.
- The second element is 'Katie':'555−2222'. In this element the key is 'Katie' and the value is '555−2222'.
- The third element is 'Joanne':'555−3333'. In this element the key is 'Joanne' and the value is '555−3333'.

In this example the keys and the values are strings. The values in a dictionary can be objects of any type, but the keys must be immutable objects. For example, keys can be strings, integers, floating-point values, or tuples. Keys cannot be lists or any other type of immutable object.

## 2.2 Retrieving a Value from a Dictionary

The elements in a dictionary are not stored in any particular order. For example, look at the following interactive session in which a dictionary is created and its elements are displayed:

[18]:
```
print(phonebook)
```

```
{'Chris': '555-1111', 'Katie': '555-2222', 'Joanne': '555-3333'}
```

**Notice** that the order in which the elements are displayed is different than the order in which they were created. This illustrates how dictionaries are not sequences, like lists, tuples, and strings. As a result, you cannot use a numeric index to retrieve a value by its position from a dictionary. Instead, you use a key to retrieve a value.

To retrieve a value from a dictionary, you simply write an expression in the following general format:

dictionary_name[key]

**KeyError:** If the key exists in the dictionary, the expression returns the value that is associated with the key. If the key does not exist, a KeyError exception is raised. The following interactive session demonstrates:

[19]:
```
print(phonebook['Chris'])
```

```
555-1111
```

[20]:
```
print(phonebook['Joanne'])
```

```
555-3333
```

[21]:
```
print(phonebook['Katie'])
```

```
555-2222
```

[22]:
```
print(phonebook['Kathryn'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4164/1545999959.py in <module>
----> 1 print(phonebook['Kathryn'])

KeyError: 'Kathryn'
```

When the expression **phonebook['Kathryn']** is entered. There is no such key as **'Kathryn'** in the phonebook dictionary, so a **KeyError** exception is raised.

**Note:** Remember that string comparisons are case sensitive. The expression phonebook['katie'] will not locate the key 'Katie' in the dictionary.

### 2.3 Using the in and not in Operators to Test for a Value in a Dictionary

As previously demonstrated, a KeyError exception is raised if you try to retrieve a value from a dictionary using a nonexistent key. To prevent such an exception, you can use the in operator to determine whether a key exists before you try to use it to retrieve a value. The following interactive session demonstrates:

[23]:
```
if 'Chris' in phonebook:
    print(phonebook['Chris'])
```

```
555-1111
```

You can also use the not in operator to determine whether a key does not exist, as demonstrated in the following session:

[24]:
```
if 'Jhon' not in phonebook:
    print("Jhon is not found")
```

```
Jhon is not found
```

### 2.4 Adding Elements to an Existing Dictionary

Dictionaries are mutable objects. You can add new key-value pairs to a dictionary with an assignment statement in the following general format:

dictionary_name[key] = value

In the general format, dictionary_name is the variable that references the dictionary, and key is a key. If key already exists in the dictionary, its associated value will be changed to value. If the key does not exist, it will be added to the dictionary, along with value as its associated value. The following interactive session demonstrates:

[25]:
```
phonebook['Joe'] = '555-0123'
```

[26]:
```
phonebook['Chris'] = '555-4444'
```

[27]:
```
print(phonebook)
```

```
{'Chris': '555-4444', 'Katie': '555-2222', 'Joanne': '555-3333', 'Joe':
'555-0123'}
```

### 2.5 Deleting Elements

You can delete an existing key-value pair from a dictionary with the del statement. Here is the general format:

del dictionary_name[key]

[28]:
```
print(phonebook)
```

```
{'Chris': '555-4444', 'Katie': '555-2222', 'Joanne': '555-3333', 'Joe':
'555-0123'}
```

[29]:
```
del phonebook['Chris']
```

[30]:
```
print(phonebook)
```

```
{'Katie': '555-2222', 'Joanne': '555-3333', 'Joe': '555-0123'}
```

### 2.6 Getting the Number of Elements in a Dictionary

You can use the built-in **len** function to get the number of elements in a dictionary.

[31]:
```
num_items = len(phonebook)
```

[32]:
```
print("Number of items in the dictionary : ",num_items)
```

```
Number of items in the dictionary :  3
```

### 2.7 Mixing Data Types in a Dictionary

As previously mentioned, the keys in a dictionary must be immutable objects, but their associated values can be any type of object. For example, the values can be lists, as demonstrated below:

[33]:
```
test_scores = { 'Kayla' : [88, 92, 100],
 'Luis' : [95, 74, 81],
 'Sophie' : [72, 88, 91],
 'Ethan' : [70, 75, 78] }
```

[34]:
```
print(test_scores)
```

```
{'Kayla': [88, 92, 100], 'Luis': [95, 74, 81], 'Sophie': [72, 88, 91], 'Ethan':
[70, 75, 78]}
```

[35]:
```
print(test_scores['Sophie'])
```

```
[72, 88, 91]
```

[36]:
```
kayla_scores = test_scores['Kayla']
```

[37]:
```
print(kayla_scores)
```

```
[88, 92, 100]
```

### 2.8 Using the for Loop to Iterate over a Dictionary

You can use the for loop in the following general format to iterate over all the keys in a dictionary:

for var in dictionary:

statement

statement

etc.

[38]:
```
print(phonebook)
```

```
{'Katie': '555-2222', 'Joanne': '555-3333', 'Joe': '555-0123'}
```

[39]:
```
# printing keys
for key in phonebook:
    print(key)
```

```
Katie
Joanne
Joe
```

[40]:
```
#printing keys with their values
for key in phonebook:
    print(key,phonebook[key])
```

```
Katie 555-2222
Joanne 555-3333
Joe 555-0123
```

### 2.8.1 The clear Method

The clear method deletes all the elements in a dictionary, leaving the dictionary empty. The method's general format is

dictionary.clear()

[41]:
```
#clear all the elements of the dictionary
phonebook.clear()
```

[42]:
```
print(phonebook)
```

```
{}
```

### 2.8.2 The get Method

You can use the get method as an alternative to the [] operator for getting a value from a dictionary. The get method does not raise an exception if the specified key is not found. Here is the method's general format:

dictionary.get(key, default)

[43]:
```
phonebook = {'Chris':'555-1111', 'Katie':'555-2222'}
value = phonebook.get('Katie', 'Entry not found')
print(value)
```

```
555-2222
```

[44]:
```
value = phonebook.get('Andy', 'Entry not found')
print(value)
```

```
Entry not found
```

### 2.8.3 The items Method

The items method returns all of a dictionary's keys and their associated values. They are returned as a special type of sequence known as a **dictionary view**. Each element in the dictionary view is a tuple, and each tuple contains a key and its associated value.

[45]:
```
print(phonebook.items())
```

```
dict_items([('Chris', '555-1111'), ('Katie', '555-2222')])
```

### 2.8.4 The keys Method

The keys method returns all of a dictionary's keys as a dictionary view, which is a type of sequence.

[46]:
```
print(phonebook)
```

```
{'Chris': '555-1111', 'Katie': '555-2222'}
```

[47]:
```
print(phonebook.keys())
```

```
dict_keys(['Chris', 'Katie'])
```

### 2.8.5 The pop Method

The pop method returns the value associated with a specified key and removes that keyvalue pair from the dictionary. If the key is not found, the method returns a default value. Here is the method's general format:

dictionary.pop(key, default)

[48]:
```
phonebook = {'Chris':'555-1111',
 'Katie':'555-2222',
 'Joanne':'555-3333'}
```

[49]:
```
phone_num = phonebook.pop('Chris', 'Entry not found')
```

[50]:
```
print(phone_num)
```

```
555-1111
```

[51]:
```
print(phonebook) # Elements after removing Chris
```

```
{'Katie': '555-2222', 'Joanne': '555-3333'}
```

[52]:
```
phone_num = phonebook.pop('Andy', 'Element not found')
```

[53]:
```
print(phone_num)
```

```
Element not found
```

[54]:
```
print(phonebook)
```

```
{'Katie': '555-2222', 'Joanne': '555-3333'}
```

### 2.8.6 The popitem Method

The popitem method returns a randomly selected key-value pair, and it removes that keyvalue pair from the dictionary. The key-value pair is returned as a tuple. Here is the method's general format:

dictionary.popitem()

You can use an assignment statement in the following general format to assign the returned key and value to individual variables:

k, v = dictionary.popitem()

[55]:
```
phonebook = {'Chris':'555-1111',
 'Katie':'555-2222',
 'Joanne':'555-3333'}
```

[56]:
```
print(phonebook)
```

```
{'Chris': '555-1111', 'Katie': '555-2222', 'Joanne': '555-3333'}
```

[57]:
```
key, value = phonebook.popitem()
```

[58]:
```
print(key,value)
```

```
Joanne 555-3333
```

[59]:
```
print(phonebook)
```

```
{'Chris': '555-1111', 'Katie': '555-2222'}
```

#### 2.8.7 The values Method

The values method returns all a dictionary's values (without their keys) as a dictionary view, which is a type of sequence.

[60]:
```
phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}
```

[61]:
```
print(phonebook.values())
```

```
dict_values(['555-1111', '555-2222', '555-3333'])
```

# 3 Exercise

## 3.1 Capital Quiz

Write a program that creates a dictionary containing the U.S. states as keys and their capitals as values. (Use the Internet to get a list of the states and their capitals.) The program should then randomly quiz the user by displaying the name of a state and asking the user to enter that state's capital. The program should keep a count of the number of correct and incorrect responses. (As an alternative to the U.S. states, the program can use the names of countries and their capitals.)

## 3.2 Course information

Write a program that creates a dictionary containing course numbers and the room numbers of the rooms where the courses meet. The dictionary should have the following keyvalue pairs:

| Course Number (key) | Room Number (value) |
|---|---|
| CS101 | 3004 |
| CS102 | 4501 |
| CS103 | 6755 |

| Course Number (key) | Room Number (value) |
|---|---|
| NT110 | 1244 |
| CM241 | 1411 |

The program should also create a dictionary containing course numbers and the names of the instructors that teach each course. The dictionary should have the following key-value pairs:

| Course Number (key) | Instructor (value) |
|---|---|
| CS101 | Haynes |
| CS102 | Alvarado |
| CS103 | Rich |
| NT110 | Burke |
| CM241 | Lee |

The program should also create a dictionary containing course numbers and the meeting times of each course. The dictionary should have the following key-value pairs:

| Course Number (key) | Meeting Time (value) |
|---|---|
| CS101 | 8:00 a.m. |
| CS102 | 9:00 a.m. |
| CS103 | 10:00 a.m. |
| NT110 | 11:00 a.m. |
| CM241 | 1:00 p.m. |

### 3.3 Find winner of election

Given an array of names of candidates in an election. A candidate name in the array represents a vote cast to the candidate. Print the name of candidates received Max vote. If there is tie, print a lexicographically smaller name.

**input** = [‘john’,‘johnny’,‘jackie’,‘johnny’, ‘john’,‘jackie’,‘jamie’,‘jamie’, ‘john’,‘johnny’,‘jamie’,‘johnny’, ‘john’]

**Output** : John.
We have four Candidates with name as ‘John’, ‘Johnny’, ‘jamie’, ‘jackie’. The candidates John and Johny get maximum votes. Since John is alphabetically smaller, we print it.

### 3.4 File Encryption and Decryption

Write a program that uses a dictionary to assign “codes” to each letter of the alphabet. For example:

codes = { ‘A’ : ‘%’, ‘a’ : ‘9’, ‘B’ : ‘@’, ‘b’ : ‘#’, etc . . .}

Using this example, the letter A would be assigned the symbol %, the letter a would be assigned the number 9, the letter B would be assigned the symbol @, and so forth.

The program should open a specified text file, read its contents, and then use the dictionary to write an encrypted version of the file's contents to a second file. Each character in the second file should contain the code for the corresponding character in the first file. Write a second program that opens an encrypted file and displays its decrypted contents on the screen.

### 3.5 Shopping Cart

In this exercise you are going to build a shopping cart using python dictionary.Create and initializes an empty dict attribute named items.

Create a method **add_item** that requires item_name, quantity, price and total arguments. This method should add the cost of the added items to the current value of total. It should also add an entry to the items dict such that the key is the item_name and the value is the quantity of the item.

Create a method **remove_item** that requires similar arguments as add_item. It should remove items that have been added to the shopping cart and are not required. This method should deduct the cost of the removed items from the current total and also update the items dict accordingly.

Create a method **checkout** that takes in cash_paid and returns the value of balance from the payment. If cash_paid is not enough to cover the total, return **"Cash paid not enough"**.