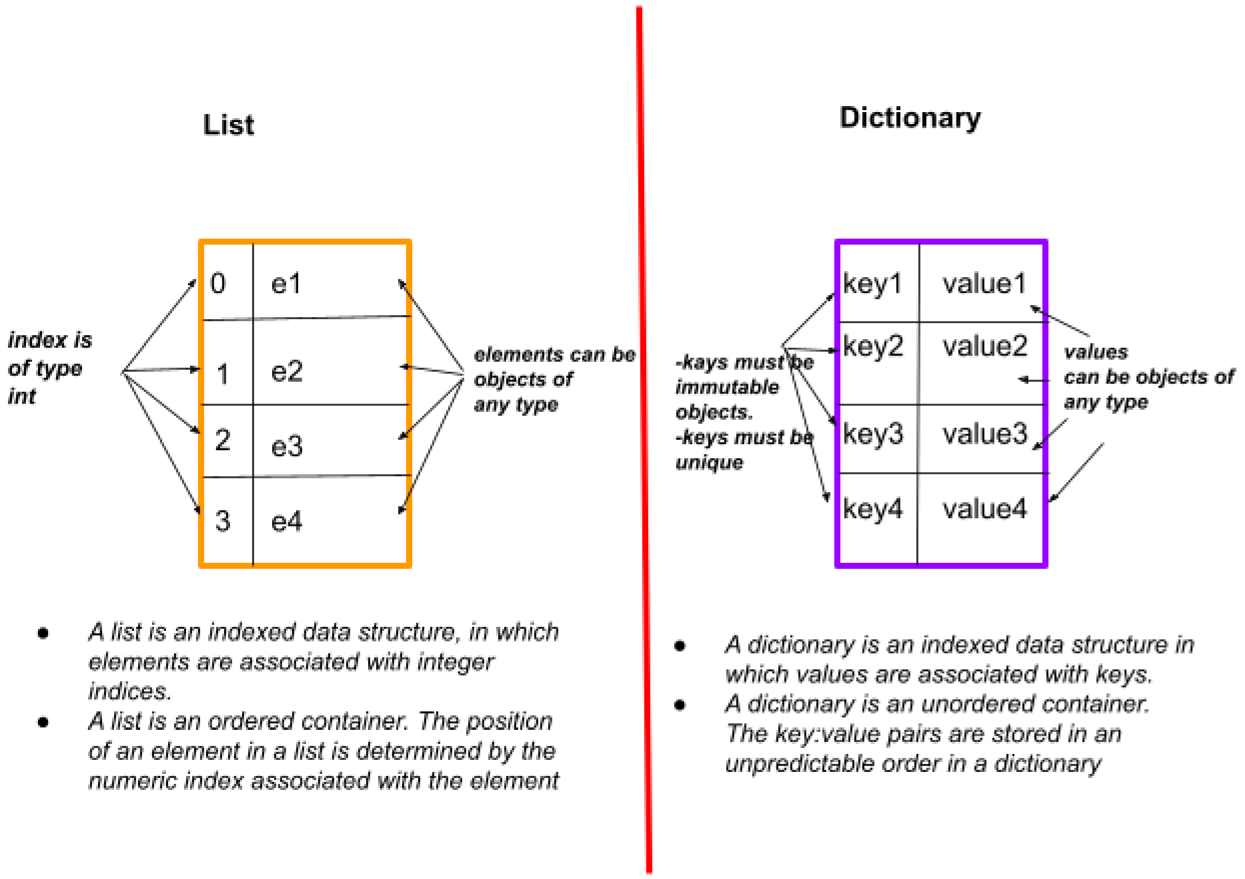
Dictionary

1. What Is A Dictionary?

A dictionary is a ***built-in data type*** in Python. It is a set of ***key:value*** pairs where key and value are objects. Unlike sequences (e.g. str, list, tuple) , which are indexed by a range of numbers, dictionaries are ***indexed by their keys***. Therefore ***keys must be unique*** within one dictionary and they must also have an ***immutable type*** (e,g. str, int, float, tuple).The ***values*** in a dictionary can be of any type. A dictionary, unlike sequences, is also ***not ordered***, i.e., ***key:value*** pairs are stored in an unpredictable order in a dictionary.



- *A list is an indexed data structure, in which elements are associated with integer indices.*
- *A list is an ordered container. The position of an element in a list is determined by the numeric index associated with the element*

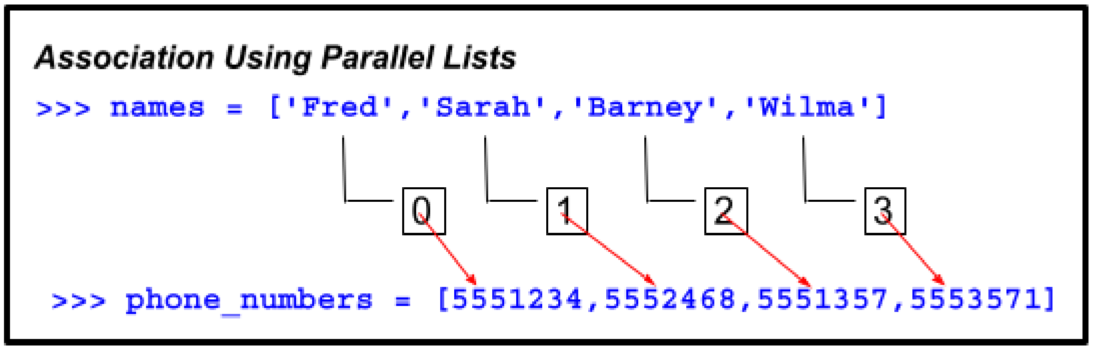
- *A dictionary is an indexed data structure in which values are associated with keys.*
- *A dictionary is an unordered container. The key:value pairs are stored in an unpredictable order in a dictionary*

2. Why Do We Need A Dictionary?

Let's say we have to create a phone book which we can use to look up the phone number of a person by entering their name. We can do this by creating two parallel lists - one for the names and one for the numbers.

```
>>> # Create a list of names
>>> names = ['Fred','Sarah','Barney','Wilma']
>>> # Create a list of numbers
>>> phone_numbers = [5551234,5552468,5551357,5553571]
```

If we want to look up Sarah's number we would first have to look up the position that Sarah is at in the names list. Once we have the position, we can use it to access Sarah's phone number in the phone_numbers list.

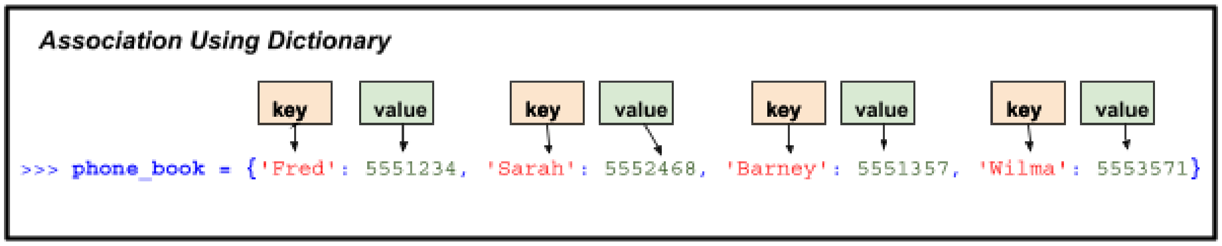


```
>>> names = ['Fred','Sarah','Barney','Wilma']
>>> phone_numbers = [5551234,5552468,5551357,5553571]
```

```
>>> # use index method in list class to retrieve position
>>> position = names.index('Sarah')
>>> phone_numbers[position]
5552468
```

In the above example, looking up a phone number given a name is dependent on both items being at the same position in their respective lists. Parallel lists may work for simple cases, but manipulating indices is cumbersome and error-prone. Any insertion, deletion, or move must always be applied consistently to both lists, or the lists will no longer be synchronized with each other, leading to incorrect outcomes.

In this example of a phonebook, where one object (name of a person) is associated with another object (phone number of the person), it is more convenient, efficient, and less error prone to use a data type like a dictionary that can store ***key:value*** pairs.



```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney': 5551357, 'Wilma': 5553571}
```

Other examples where a dictionary could be used to store association are:

- eclass gradebook where a student's grade is associated with the name of a student.
- the index at the end of the book, where a list of page numbers are associated with a topic in the book.
- counting the occurrence of an object in a container, where the object is associated with the number of times it occurs.

3. Creating A Dictionary

Lets see how we can use a dictionary for our phone book. A pair of empty braces creates an empty dictionary.

```
>>> phone_book = {}
```

The ***syntax*** for adding a key:value pair in the dictionary is as follows:

```
<name of dictionary>[<key>] = <value>
```

We will follow the above syntax to add items in the dictionary:

```
>>> phone_book = {}
>>> phone_book['Fred'] = 5551234
>>> phone_book['Sarah'] = 5552468
>>> phone_book['Barney'] = 5551357
>>> phone_book['Wilma'] = 5553571
```

After adding the items, we can display the content of the dictionary in the Python Shell by writing the name of the dictionary:

```
>>> phone_book
{'Fred': 5551234, 'Sarah': 5552468, 'Barney': 5551357, 'Wilma':
5553571}
```

We can see in this example a dictionary consists of a collection of ***key-value*** pairs. Each ***key:value*** pair maps the ***key*** which is the name of the person to its associated ***value*** which is the phone number of that person.

In the above example we saw how to add ***key:value*** pair to an empty dictionary. We can also create a dictionary with key:value pairs in it .

For example, the phone_book could also be created in the following manner:

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
```

While adding a ***key:value*** pair to the dictionary, we have to bear in mind that the ***key must be an immutable object***. A list cannot be used as a key in a dictionary. For example the following statement would not work as the ***key*** ***`['Fred']`*** ***is a list which is a mutable type:***

```
>>> phone_book = {}
>>> phone_book[['Fred']] = 5551234
Traceback (most recent call last):
  Python Shell, prompt 3, line 1
builtins.TypeError: unhashable type: 'list'
```

4. Accessing A Dictionary

4.1 Accessing a value

A value inside a dictionary can be accessed using the key. For example, in our phone book dictionary example we can access Wilma's phone number by writing the following statement:

```
>>> phone_book['Wilma']
5553571
```

A ***runtime KeyError error*** occurs if we try to access a dictionary with a key which is not in it. For example, in our phone_book dictionary, the name Zoya does not exist as a key. Therefore the following statement results in ***KeyError***.

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> phone_book['Zoya']
Traceback (most recent call last):
  Python Shell, prompt 6, line 1
builtins.KeyError: 'Zoya'
```

There is also a method called ***get*** in the dictionary class that can be used to access a value in the dictionary. Here is the documentation of the ***get method*** in the dictionary class:

```
>>> help(dict.get)
Help on method_descriptor:

get(self, key, default=None, /)
    Return the value for key if key is in the dictionary, else
default.
```

The ***get method*** in the dictionary class returns a default value of None if the key is not in the dictionary. This default value can be replaced with another object provided as a second argument to the get method.

Here are ***some examples of how to use the get method*** in the dictionary class:

Example 1: The ***get method*** returns the value associated with the key, if the key is in the dictionary,

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> phone_book.get('Fred')
5551234
```

Example 2: The ***get method*** returns None, if the key is not in the dictionary and the second argument in the get method call is not specified.

```
 >>> print(phone_book.get('Zoya'))
None
```

Example 3: The ***get method*** returns the second argument If the key is not in the dictionary and the second argument in the get method call is specified.

```
>>> print(phone_book.get('Zoya','Not in dictionary'))
Not in dictionary
```

4.2 Accessing Key Value Pairs In A Dictionary

We can use an iterative statement and the methods in the dictionary class to access multiple ***key:value*** pairs in the dictionary. The two methods in the dictionary class that we can use for this purpose are : ***items*** and ***keys***. Here is the documentation for the two methods:

```
>>> help(dict.items)
Help on method_descriptor:

items(...)
     D.items()  ->  a  set-like  object  providing  a  view  on  D's
items
>>> help(dict.keys)
Help on method_descriptor:

keys(...)
    D.keys() -> a set-like object providing a view on D's keys
```

The ***items method*** returns an iterable object of ***key:value*** pairs that we can access using a for loop.

Example of using the ***items method***:

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> for key,value in phone_book.items():
...     print(key,value)
...
Fred 5551234
Sarah 5552468
Barney 5551357
Wilma 5553571
```

The ***keys method*** returns an iterable object of keys that we can access using a for loop. We can use the keys to retrieve the values associated with the keys.

Example of using the ***keys method***:

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> for key in phone_book.keys():
...     print('key = ',key,'value = ',phone_book[key])
...
key =  Fred value =  5551234
key =  Sarah value =  5552468
key =  Barney value =  5551357
key =  Wilma value =  5553571
```

4.3 Accessing Values In A Dictionary

Sometimes, we want to know all the values in the dictionary. In this case, we can use the ***values method*** in the dictionary class. Here is the documentation of the values method:

```
>>> help(dict.values)
Help on method_descriptor:

values(...)
    D.values() -> an object providing a view on D's values
```

The ***values method*** in the dictionary class returns an iterable object that can be accessed using a for loop.

Example of using the ***values method***:

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> for value in phone_book.values():
...     print('value = ',value)
...
value =  5551234
value =  5552468
value =  5551357
value =  5553571
```

5. Updating The Value Of An Existing Key In A Dictionary

A value associated with a key can be updated by assigning a new value to the same key. Keys are unique in a dictionary, and when a new value is assigned to an existing key in the dictionary, the old value is replaced with a new value. The following example shows how we replace Sarah’s phone number with a new phone number in the dictionary.

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> phone_book['Sarah']
5552468
>>> # replacing existing phone number with a new number
>>> phone_book['Sarah'] = 6662468
>>> phone_book
{'Fred': 5551234, 'Sarah': 6662468, 'Barney': 5551357, 'Wilma':
5553571}
```

6. Deleting Items From A Dictionary

There is a method called ***pop*** in the dictionary class that can remove a key:value pair from the dictionary.. Here is the documentation of the method:

```
help(dict.pop)
Help on method_descriptor:

pop(...)
    D.pop(k[,d]) -> v, remove specified key and return the corresponding
value.
    If key is not found, d is returned if given, otherwise KeyError is
raised
```

The ***pop method*** takes in a key as an argument, ***removes*** the ***key:value*** pair corresponding to the given key from the dictionary, and ***returns*** the value associated with the key. The ***pop method*** also has an optional ***second argument***. If the key specified in the argument is not in the dictionary, then the pop method returns the second argument if it is specified, otherwise, the pop method gives a runtime ***KeyError*** error.

Example 1 key to be deleted exists in the dictionary

```
phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
# deleting 'Barney':5551357
value = phone_book.pop('Barney')
print(value)
5551357
>>> phone_book
{'Fred': 5551234, 'Sarah': 5552468, 'Wilma': 5553571}
```

Example 2 key to be deleted does not exist in the dictionary

```
>>> phone_book.pop('Zoya')
Traceback (most recent call last):
  Python Shell, prompt 31, line 1
builtins.KeyError: 'Zoya'
>>> # pass optional second argument to the pop method
>>> phone_book.pop('Zoya','not in dictionary')
'not in dictionary'
```

7. Checking Membership In A Dictionary

We can check if a key is in the dictionary by using the ***in operator***, The ***in operator*** returns True if the key is in the dictionary and returns False otherwise.

Example 1: Using the in operator to check if a key is in the dictionary.

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> 'Fred' in phone_book
True
>>> 'Wilson' in phone_book
False
```

The ***in operator*** when used to check if a value is in a dictionary ***will return False*** even if the value exists as part of the ***key:value*** pair in the dictionary. In order to check if a particular value exists in a dictionary the ***in operator*** should be used on the iterable object that is returned by the ***values method*** in the dictionary class.

Example 2: Using the ***in operator*** to check if a value is in the dictionary.

```
>>> phone_book = {'Fred': 5551234, 'Sarah': 5552468, 'Barney':
5551357, 'Wilma': 5553571}
>>> 5551234 in phone_book
False
>>> 5551234 in phone_book.values()
True
```

8. Practice Questions

Write the output for each of the following program segments ***without*** using Wing IDE. If an error occurs just write the name of the error.

	Code Segment	Write the output

1.	`speeds={}`
	`speeds['B777']=896`
	`speeds['A330']=840`
	`speeds['E190']=811`
	`print(speeds)`
		{‘B777’:896,
		‘A330’:840,
		‘E190’:811}

2.	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`speeds['E190']=900`
	`speeds['F16']=800`
	`print(speeds)`
		{‘B777’:896,
		‘A330’: 840,
		‘E190’:900,
		‘F16’:800}

3.	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`for k,v in speeds.items():`
	`      print(k,v)`
	`for values in speeds.values():`
	`      print(values)`
	`for keys in speeds.keys():`
	`      print(keys)`
		B777 896
		A330 840
		E190 811
		896
		840
		811
		B777
		A330
		E190

4.	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`print(speeds['B769'])`
		KeyError

5,	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`print(speeds.get('B769',"Not in dictionary"))`
	`print(speeds.get('B769'))`
	`print(speeds.get('E190'))`
		Not in dictionary
		None
		811

6.	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`print(speeds.pop('E194'))`
		KeyError

7.	`speeds = {'B777': 896, 'A330': 840, 'E190': 811}`
	`print(speeds.pop('E194','Item not found'))`
	`print(speeds.pop('A330'))`
	`print(speeds)`
		Item not found
		840
		{‘B777’:896, ‘E190’:811}

8.	`speeds = {'B777': 896, 'A330': 840, 'E190': 900}`
	`print('E190' in speeds)`
	`print(900 in speeds)`
	`print(900 in speeds.values())`
		True
		False
		True