8.2.4 Sets

Sets are also similar to lists that can store multiple
Python sets are little different from lists.

These differences are :

- Sets are created by specifying comma separated values enclosed in curly
- Sets' elements are unordered and unindexed unlike lists.
- Sets do not allow duplicate entries unlike lists.
- Sets cannot contain mutable elements.

Consider below given example which is creating a set of values.

```
>>> myset = { 1, 2, 3, 4}
>>> print(myset)
{1, 2, 3, 4}
>>> myset1 = { 1, 2, 3, 4, 4}
>>> print(myset1)
{1, 2, 3, 4}
>>> type(myset)
<class 'set'>
```

Set created by specifying comma separated values enclosed in curly brackets { }.

Sets do not allow duplicate entries. See, it removed the duplicate entry 4. As you can see that 4 appears only once in the set even though you gave 4 twice while creating.

```
>>> set1 = {1, 2, [3, 4]}
Traceback(most recent call last) :
  File "<pyshell#68>", line 1, in <module>
    set1 = {1, 2, [3, 4]}
TypeError : unhashable type : 'list'
>>> set1 = {1, 2, (3, 4)}
```

Sets do not allow mutable elements like lists. See it gave error when we tried to add a list as element, but it gave no error when we gave a tuple as an element, because tuple is not mutable.

A set cannot contain a mutable element in it. However, a set itself is mutable, *i.e.*, we can add or remove items from it.

NOTE

Python data types like lists, tuples, dictionary are all iterables.

2.5 Dictionary

Dictionary data type is another feature in Python's hat. The *dictionary* is an unordered set of comma-separated **key : value** pairs, within { }, with the requirement that within a dictionary, no two keys can be the same (*i.e.*, there are unique keys within a dictionary). For instance, following are some dictionaries :

```
{'a' : 1 , 'e' : 2, 'i' : 3, 'o' : 4, 'u' : 5}

>>> vowels = {'a' : 1, 'e' : 2, 'i' : 3, 'o' : 4, 'u' : 5}
>>> vowels['a']
1
>>> vowels['u']
```

Here 'a', 'e', 'i', 'o' and 'u' are the keys of dictionary vowels. 1, 2, 3, 4, 5 are values for these keys respectively.

Specifying key inside [ ] after dictionary name gives the corresponding value from the key : value pair inside dictionary.

ITERABLE

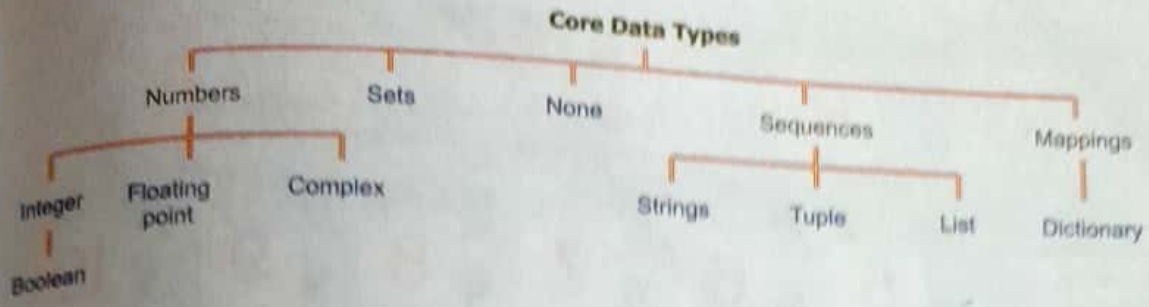
An iterable is any Python object that can return its members, one at a time. Since, you can access elements of strings, lists, tuples and dictionary one at a time, these are all iterable

...ictionaries shall be covered in details in a later ...

8.3

1. Ir

Following figure summarizes the core data types of Python.



8.3 MUTABLE AND IMMUTABLE TYPES

The Python data objects can be broadly categorized into *two – mutable* and *immutable* types, in simple words changeable or modifiable and non-modifiable types.

1. Immutable types

The immutable types are those that can never change their value in place. In Python, the following types are immutable : *integers, floating point numbers, Booleans, strings, tuples.*

Let us understand the concept of immutable types. In order to understand this, consider the code below :

Immutable Types
- integers
- floating point numbers
- booleans
- strings
- tuples

Sample code 8.1

```
P = 5
q = P
r = 5
                    # will give 5, 5, 5
P = 10
r = 7
q = r
```

After reading the above code, you can say that values of integer variables *p, q, r* could be changed effortlessly. Since *p, q, r* are integer types, you may think that integer types can change values.

But hold : It is not the case. Let's see how.

You already know that in Python, variable-names are just the references to value-objects *i.e.*, data values. The variable-names do not store values themselves *i.e.*, they are not storage containers. Recall section 7.5.1 where we briefly talked about it.

Now consider the **Sample code 8.1** given above. Internally, how Python processes these assignments is explained in Fig. 8.2. Carefully go through figure 8.2 on the next page and then read the following lines.

So although it appears that the value of variable *p / q / r* is changing ; values are *not changing "in place"* the fact is that the variable-names are instead made to refer to new immutable integer object. (Changing **in place** means modifying the same value in same memory location.)