Language Fundamentals

July 11, 2022

In our previous blogs, we have discussed about we have discussed "Why Python is recommended for beginners" and "The features of Python and the different flavors of it".

Now, it's time to start with little hands on experience

The Python Langauge Fundamentals include:

- Identifiers
- Reserved Words
- Data Types
- TypeCasting

0.1 Identifiers

A name in python program which is used to identify a class,function,module or variable is called Identifiers

[1]:
```
a=10
```

0.2 Rules to define Identifiers in Python:

- The only allowed characters in python for identifiers are alphabets symbols(either lowercase or uppercase), digits(0 to 9),underscore symbol (_)

  By mistake if we are using any other symbol like $then we will gwt syntax error.

  - Short = 20 (Valid)
  - $hort = 30 (Invalid)
- Identifier should not starts with digit
  - 123number = 123 (Inalid)
  - number123 = 123 (Valid)
- Identifiers are case sensitive in python
  - apple = 9
  - Apple = 99

[2]:
```
Short = 20
```

[3]:
```
$hort = 30
```

```
  File "<ipython-input-3-0ac4b7f2f9d7>", line 1
    $hort = 30
    ^
SyntaxError: invalid syntax
```

[4]:
```
123number = 123
```

```
  File "<ipython-input-4-3f326582a07e>", line 1
    123number = 123
       ^
SyntaxError: invalid syntax
```

[5]:
```
number123 =123
```

[6]:
```
apple = 9
```

[7]:
```
Apple = 99
```

0.3 We can not use reserve words as identifiers

0.4 Which of the following are vaild Python identifiers ?

1. 123abc
2. abc123
3. python2learn
4. cash
5. ___hello_
6. if
7. def

Kindly execute the above variables by assigning some values to it and enhance your knowledge.

0.5 In Python, some words are reserved to represent some meaning or functionality. These reserved words are called Keywords

0.5.1 There are 35 reserved words available in Python

- True, False, None
- and, or, not
- if, elif, else
- while, for, break, continue, return, in, yield, async, await
- try, except, finally, raise, assert
- import, from, as, class, def, pass, global, nonlocal, lambda,del,with

0.6 Note:

1. All Reserved wordsin python contains only alphabet symbols
2. Except the following 3 reserved words, all contain only lower case alphabet symbols
    - True
    - False
    - None

a= true (Invalid)

A= True (Valid)

[8]:
```
a= true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-3240e43fdcb9> in <module>
----> 1 a= true

NameError: name 'true' is not defined
```

[9]:
```
d = True
```

0.7 Check which of the keywords are available in python through code ?

[10]:
```
import keyword
keyword.kwlist
```

[10]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
'if',
'import',
'in',
'is',
'lambda',
'nonlocal',
'not',
'or',
'pass',
'raise',
'return',
'try',
'while',
'with',
'yield']
```

0.8 Data Types

Datatype represents the type of data present inside a variable. In Python, we are not required to specify the type explicitly.Based on value provided, the type will be assigned automatically. Hence, Python is dynamically typed language.

0.9 Python contains the following inbuilt data types

1. Int
2. Float
3. Complex
4. Bool
5. Str
6. Bytes
7. Bytearray
8. Range
9. List
10. Tuple
11. Set
12. Frozenset
13. Dict
14. None

1 Everything in python is object

1.1 Python contains several inbuilt functions

- type() : to check the type of variable.
- id() : to get address of object
- print() : to print the value

1.2 int Data type:

int data type is used to represnt whole numbers(integral values)

- a =10
- type(a)

[11]:
```
a = 10
```

[12]:
```
type(a)
```

[12]:
```
int
```

1.3 We can represnt the int values in the following ways:

- Decimal Form
- Binary Form
- Octal Form
- Hexa Decimal Form

1.4 DecimalForm (Base-10) :

- It is the default number system in Python
- The allowed digits are: 0 to 9

Example : a = 10

[13]:
```
a =10
```

[14]:
```
type(a)
```

[14]:
```
int
```

1.5 Binary Form (Base-2) :

- The allowed digits are: 0 & 1
- Literal value should be prefixed with 0b or 0B

Example : - a = 0B1111 - b = 0B11 - c = 0b111

[15]:
```
a = 0B1111
```

[16]:
```
type(a)
```

[16]:
```
int
```

[17]:
```
print(a)
```

```
15
```

[18]:
```
b = 0B11
```

[19]:
```
type(b)
```

[19]:
```
int
```

[20]:
```
print(b)
```

```
3
```

[21]:
```
c=0b111
```

[22]:
```
print(c)
```

```
7
```

1.6 Octal Form (Base-8):

- The allowed digits are: 0 to 7
- Literal value should be prefixed with 0O or 0o

[23]:
```
a = 0O745
```

[24]:
```
print(a)
```

```
485
```

[25]:
```
b = 0o473
```

[26]:
```
print(b)
```

```
315
```

1.7 Hexa Decimal Form (Base-16)

- The allowed digits are 0-9, a-f(both lowercase and uppercases are allowed)
- Literal value should be prefixed with 0x or 0X

[27]:
```
a=0XFACE
```

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

```
64206
```

[29]:
```
b=0xbeef
```

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

```
48879
```

Being a programmer we can specify literal values in decimal, binary, octal and hexadecimal forms But PVM (Python Virtual Machine) will always provide values only in decimal form.

[31]:
```
a = 10
```

[32]:
```
b = 0B10
```

[33]:
```
c = 0O10
```

[34]:
```
d = 0X10
```

[35]:
```
print(a)
```

```
10
```

[36]:
```
print(b)
```

```
2
```

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

```
8
```

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

```
16
```

1.8 Base conversions

- bin(): We can use bin() to convert any base to binary
- oct(): We can use oct() to convert from any base to octal
- hex(): We can use hex() to covert from any base to hexa decimal

[39]:
```
bin(15)
```

[39]:
```
'0b1111'
```

[40]:
```
bin(0O11)
```

[40]:
```
'0b1001'
```

[41]:
```
bin(0X10)
```

[41]:
```
'0b10000'
```

[42]:
```
oct(10)
```

[42]:
```
'0o12'
```

[43]:
```
oct(0b11)
```

[43]:
```
'0o3'
```

[44]:
```
oct(0Xbeef)
```

[44]:
```
'0o137357'
```

[45]:
```
hex(10)
```

[45]:
```
'0xa'
```

[46]:
```
hex(0b11)
```

[46]:
```
'0x3'
```

[47]:
```
hex(0O10)
```

[47]:
```
'0x8'
```

1.9 Float Data Type:

We can use float data type to represent floating point values(decimal values)

Example: - f = 1.725 - type(f)

[48]:
```
f = 1.725
```

[49]:
```
type(f)
```

[49]:
```
float
```

1.10 We can also represent floating point values by using exponential form (Scientific Notation)

Example:

- g = 1.2e3
- h = 1.2E3

[50]:
```
g = 1.2e3
```

[51]:
```
type(g)
```

[51]:
```
float
```

[52]:
```
h = 1.2E3
```

```
[53]: type(h)
```

```
[53]: float
```

The main advantage of exponential form is we can represent big values in less memory.

1.11 Note:

We can represent int values in decimal, binary, octal and hexa decimal forms. But we can not represent float values only by using decimal form.

```
[54]: a = 0B11.01
```

```
  File "<ipython-input-54-71112486705c>", line 1
    a = 0B11.01
            ^
SyntaxError: invalid syntax
```

```
[55]: a = 0o123.456
```

```
  File "<ipython-input-55-fb634c0241c5>", line 1
    a = 0o123.456
             ^
SyntaxError: invalid syntax
```

```
[56]:  a= 0X123.456
```

```
  File "<ipython-input-56-f5255c11e9bd>", line 1
    a= 0X123.456
            ^
SyntaxError: invalid syntax
```

1.12 Complex Data type :

- A complex number is of the form : a + ij where j 2 = -1
- ‘a’ and ‘b’ contain Integers or Floating Point Values
- example: a = 3 + 5j

```
[57]: a = 3 + 5j
```

```
[58]: type(a)
```

```
[58]: complex
```

1.13 Note :

- In the real part of complex data type if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form
- But imaginary partshould be specified by using decimal form

```
[59]: a = 0B11 + 5j
```

```
[60]: a
```

```
[60]: (3+5j)
```

```
[61]: a = 3 + 0b101j # imaginary part should be specified by using decimal form
```

```
  File "<ipython-input-61-8cf4209719c8>", line 1
    a = 3 + 0b101j # imaginary part should be specified by using decimal form
                ^
SyntaxError: invalid syntax
```

1.14 Even we can perform operations on complex type variables

```
[62]: a = 10 +1.5j
```

```
[63]: b= 20 + 2.5j
```

```
[64]: c = a + b
```

```
[65]: type(c)
```

```
[65]: complex
```

```
[66]: print(c)
```

```
(30+4j)
```

1.15 Note: Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

- c = 13.5+3.6j
- c.real -> 13.5
- c.imag -> 3.6

1.15.1 We can use complex type generally in scientific Applications and electrical engineering Applications.

```
[67]: c = 13.5 + 3.6j
```

```
[68]: c.real
```

```
[68]: 13.5
```

```
[69]: c.imag
```

```
[69]: 3.6
```

1.16 Bool Data Type:

- We can use this data type to represent boolean values.
- The only allowed values for this data type are: True and False.
- Internally Python represnts True as 1 and False as 0

```
[70]: b = True
```

```
[71]: type(b)
```

```
[71]: bool
```

```
[72]: a = 10
```

```
[73]: b= 20
```

```
[74]: c = a < b
```

```
[75]: print(c)
```

```
True
```

```
[76]: True + True
```

```
[76]: 2
```

```
[77]: True - False
```

```
[77]: 1
```

1.17 Str Data Type

- Str represents String data type
- A string is a sequence of characters enclosed within single quotes or double quotes

- s = 'flower'
- s = "Flower"
- By using single quotes or double quotes we can not represent multiline string literals. For this requirement, we should go for triple single quotes (' ' ') or triple quotes (" " ")
- Syntax: name [start:stop:step]

```
[78]: s= 'flower'
```

```
[79]: s = "Flower"
```

```
[80]: k = ''' This is written in
               multiple line '''
```

```
[81]: print(k)
```

```
 This is written in
          multiple line
```

```
[82]: j =''' This is " double quotes '''
```

```
[83]: print (j)
```

```
 This is " double quotes
```

1.18 Slicing of Strings:

- Slice means a piece
- [] operator is used for slicing (for retrieving some part of strings)
- In python Strings folows zero based index
- The index can be either positive (+ ve) or negative (- ve)
- Positive (+ ve) index means forward direction from Left to Right
- Negative (- ve) index means backward direction from Right to Left

```
[84]: s = 'apple'
```

```
[85]: s[0]
```

```
[85]: 'a'
```

```
[86]: s[1]
```

```
[86]: 'p'
```

```
[87]: s[-1]
```

```
[87]: 'e'
```

```
[88]: s[40]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-88-465d2612c03b> in <module>
----> 1 s[40]

IndexError: string index out of range
```

```
[89]: s[1:40]
```

```
[89]: 'pple'
```

```
[90]: s[1:]
```

```
[90]: 'pple'
```

```
[91]: s[:4]
```

```
[91]: 'appl'
```

```
[92]: s[0].upper() +s[1:]
```

```
[92]: 'Apple'
```

```
[93]: s[:len(s)-1] +s[-1].upper()
```

```
[93]: 'applE'
```

```
[94]: len(s)
```

```
[94]: 5
```

In Python the below mentioned data types are considered as Fundamental Data types

- int
- float
- complex
- bool
- str

1.19 Typecasting

- We can convert one type value to another type.This conversion is called Typecasting.
- The following are various inbuilt functions for type casting

1. int()
2. float()

3. complex()
4. bool()
5. str()

1.20 int () : We can use this function to convert values from other types to int.

[95]:
```
int(23421.4353423)
```

[95]:
```
23421
```

[96]:
```
int(10 + 5j) # we can't convert complex to int
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-96-af1b02f59688> in <module>
----> 1 int(10 + 5j) # we can't convert complex to int

TypeError: can't convert complex to int
```

[97]:
```
int(True)
```

[97]:
```
1
```

[98]:
```
int(False)
```

[98]:
```
0
```

[99]:
```
int("10")
```

[99]:
```
10
```

[100]:
```
int ("10.5") # string only can't conert to int if it contain integer number␣
↪only in decimal form
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-100-a837000a101b> in <module>
----> 1 int ("10.5") # string only can't conert to int if it contain integer␣
 ↪number only in decimal form

ValueError: invalid literal for int() with base 10: '10.5'
```

[101]:
```
int("twenty")  # string only can't conert to int if it contain integer number␣
↪only in decimal form
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-101-93163b80294e> in <module>
----> 1 int("twenty")  # string only can't conert to int if it contain integer␣
 ↪number only in decimal form

ValueError: invalid literal for int() with base 10: 'twenty'
```

[102]:
```
int("0B111")  # string only can't conert to int if it contain integer number␣
↪only in decimal form
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-102-12a82d649d08> in <module>
----> 1 int("0B111")  # string only can't conert to int if it contain integer␣
 ↪number only in decimal form

ValueError: invalid literal for int() with base 10: '0B111'
```

[103]:
```
int(0B111)
```

[103]:
```
7
```

1.21 Note:

1. We can convert from any type to int except complex type.
2. If we want to convert str type to int type, compulsary str should contain only integral value and should be specified in base-10.

1.22 Float(): We can use float() function to convert other type values to float type.

[104]:
```
 float(10)
```

[104]:
```
10.0
```

[105]:
```
 float(10+5j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-105-d2f956539d9b> in <module>
----> 1 float(10+5j)

TypeError: can't convert complex to float
```

```
[106]: float(True)

[106]: 1.0

[107]: float(False)

[107]: 0.0

[108]: float("10")

[108]: 10.0

[109]: float("10.5")

[109]: 10.5

[110]: float("ten")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-110-6277cc9af495> in <module>
----> 1 float("ten")

ValueError: could not convert string to float: 'ten'
```

```
[111]: float("0B1111")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-111-d6e6d8207df6> in <module>
----> 1 float("0B1111")

ValueError: could not convert string to float: '0B1111'
```

```
[112]: float(0B1111)

[112]: 15.0
```

1.23 Note:

1. We can convert any type value to float type except complex type.
2. Whenever we are trying to convert str type to float type compulsary str should be either integral or floating point literal and should be specified only in base-10.

1.24 complex():

- We can use complex() function to convert other types to complex type.
- Form-1: complex(x)
- We can use this function to convert x into complex number with real part x and imaginary part 0

```
[113]: complex(10)
```

```
[113]: (10+0j)
```

```
[114]: complex(11.5)
```

```
[114]: (11.5+0j)
```

```
[115]: complex(True)
```

```
[115]: (1+0j)
```

```
[116]: complex(False)
```

```
[116]: 0j
```

```
[117]: complex("10")
```

```
[117]: (10+0j)
```

```
[118]: complex("11.5")
```

```
[118]: (11.5+0j)
```

```
[119]: complex("ten")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-119-8871b5951e94> in <module>
----> 1 complex("ten")

ValueError: complex() arg is a malformed string
```

1.25 Form-2: complex(x,y)

- We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

```
[120]: complex(90, -3)
```

```
[120]: (90-3j)
```

```
[121]: complex(True, False)
```

```
[121]: (1+0j)
```

1.26 Bool(): We can use this function to convert other type values to bool type

```
[122]: bool(0)
```

```
[122]: False
```

```
[123]: bool(1)
```

```
[123]: True
```

```
[124]: bool(10)
```

```
[124]: True
```

```
[125]: bool(12.5)
```

```
[125]: True
```

```
[126]: bool(0.231)
```

```
[126]: True
```

```
[127]: bool(0.0)
```

```
[127]: False
```

```
[128]: bool(10-2j)
```

```
[128]: True
```

```
[129]: bool(0+1.5j)
```

```
[129]: True
```

```
[130]: bool( 0 + 0j)
```

```
[130]: False
```

```
[131]: bool("True")
```

```
[131]: True
```

```
[132]: bool("False")
```

```
[132]: True
```

```
[133]: bool("")
```

```
[133]: False
```

1.27 NOTE:

- If x is int datatype, 0 means False , Non-zero means true
- If x is float datatype, when the total value is zero then the result is False otherwise the result is True
- If the x is complex datatype, when the realand imaginory part are zero, then the result is False otherwise True
- If x is str datatype, xwhen x is empty string then the result is False otherwise the result is True

1.28 str() : We can use this method to convert other type values to str type

```
[134]: str(20)
```

```
[134]: '20'
```

```
[135]: str(10.5)
```

```
[135]: '10.5'
```

```
[136]: str(0o1331)
```

```
[136]: '729'
```

```
[137]: str(10+5j)
```

```
[137]: '(10+5j)'
```

```
[138]: str(True)
```

```
[138]: 'True'
```

```
[139]: str(False)
```

```
[139]: 'False'
```

```
[140]: str(0b1110)
```

```
[140]: '14'
```

```
[141]: str(0xface)
```

```
[141]: '64206'
```

1.29 bytes: byte data type represnts a group of byte numbers just like an array

```
[142]: A =[5,10,15,25]
```

```
[143]: b=bytes(A)
```

```
[144]: type(b)
```

```
[144]: bytes
```

```
[145]: print(b[0])
```

```
5
```

```
[146]: print(b[-1])
```

```
25
```

1.30 Note:

- The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.
- Once we creates byte type value,we cannot change its values, otherwise we will get TypeError

```
[147]: A = [5,1,0,15,25]
```

```
[148]: A = bytes(A)
```

```
[149]: A[0]=10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-149-fb9b1772b7f1> in <module>
----> 1 A[0]=10

TypeError: 'bytes' object does not support item assignment
```

1.31 bytearray DataType : bytearray is exactly same as bytes data type except that its elements can be modified

```
[150]: A = [5,10,35,25]
```

```
[151]: A=bytearray(A)
```

```
[152]: print(A[1])
```

```
10
```

```
[153]: A[1]=20
```

```
[154]: print(A[1])
```

```
20
```

```
[155]: A=[10,256]
```

```
[156]: A=bytearray(A)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-156-f0b06574f387> in <module>
----> 1 A=bytearray(A)

ValueError: byte must be in range(0, 256)
```

```
[157]: A=[10,255]
```

```
[158]: A=bytearray(A)
```

```
[159]: print(A[0],A[1])
```

```
10 255
```

1.32 List Data Type:

- If we want to represnt group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.
  - Insertion order is preserved
  - Heterogeneous Objects are allowed
  - Duplicates are allowed
  - Growable in Nature
  - Values should be enclosed within square brackets

[160]:
```
l = [20,20.5,'apple',True,20]
```

[161]:
```
print(l)
```

```
[20, 20.5, 'apple', True, 20]
```

[162]:
```
print(l[1])
```

```
20.5
```

[163]:
```
print(l[1:3])
```

```
[20.5, 'apple']
```

[164]:
```
print(l)
```

```
[20, 20.5, 'apple', True, 20]
```

1.33 List is growable in nature that is based on our requirement we can increase or decrease the size

[165]:
```
l =[10,20,30,50]
```

[166]:
```
l.append("learn")
```

[167]:
```
print(l)
```

```
[10, 20, 30, 50, 'learn']
```

[168]:
```
l.remove(20)
```

[169]:
```
print(l)
```

```
[10, 30, 50, 'learn']
```

[170]:
```
l = l * 2
```

[171]:
```
print(l)
```

```
[10, 30, 50, 'learn', 10, 30, 50, 'learn']
```

1.34 An ordered, mutuable, heterogenous collection of elements is nothing but list, where duplicates are also allowed

1.35 Tuple Data Type :

- Tuple data type is exactly same as list data type except that it is immutable that is we can not change values
- Tuple elements can be represented with in paranthesis

```
[172]: t= (10,20,30,40)
```

```
[173]: type(t)
```

```
[173]: tuple
```

```
[174]: t[0]=100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-174-3382f43ca263> in <module>
----> 1 t[0]=100

TypeError: 'tuple' object does not support item assignment
```

```
[175]: t.append("you can not append to tupple")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-175-acfb21cc08ab> in <module>
----> 1 t.append("you can not append to tupple")

AttributeError: 'tuple' object has no attribute 'append'
```

```
[176]: t.remove(10)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-176-c0ded720561a> in <module>
----> 1 t.remove(10)

AttributeError: 'tuple' object has no attribute 'remove'
```

1.36 tuple is the read only version of list

1.37 Range Data Type:

- range datatype represents a sequence of numburs
- The elements present in range Data type can not be modified that is Data type is immutable

```
[177]: r= range (10) # generates numbers from 0 to 9
```

```
[178]: for num in r:       # to display range of numbers in r
           print(num)
```

```
0
1
2
3
4
5
6
7
8
9
```

[179]:
```
r = range(11,21) # generates numbers from 11 to 20
```

[180]:
```
for num in r:      # to display range of numbers in r
    print(num)
```

```
11
12
13
14
15
16
17
18
19
20
```

[181]:
```
r = range(10,21,2) # increment value=2
```

[182]:
```
for num in r:      # to display range of numbers in r
    print(num)
```

```
10
12
14
16
18
20
```

1.38 We can access elements present in the range Data type by using index

[183]:
```
r =range (10,20)
```

[184]:
```
print(r[0])
```

```
10
```

[185]:
```
print(r[15])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-185-18937ed0e28f> in <module>
----> 1 print(r[15])

IndexError: range object index out of range
```

1.39 We can not modify the values of range data type

[186]:
```
r[0]=99
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-186-0355d00592fa> in <module>
----> 1 r[0]=99

TypeError: 'range' object does not support item assignment
```

[187]:
```
list(range(10))
```

[187]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1.40 set Data Type:

1.40.1 If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

- Insertion order is not preserved
- Duplicates are not allowed
- Heterogeneous objects are allowed
- Index concept is not applicable
- It is mutable collection
- Growable in nature

[188]:
```
s ={100,200,100,10,0 ,'Learn to code Python'}
```

[189]:
```
print(s)
```

```
{0, 100, 200, 10, 'Learn to code Python'}
```

[190]:
```
s[0] #set does not support indexing
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-190-c9cd4b0a2b3d> in <module>
----> 1 s[0] #set does not support indexing
```

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

1.40.2 set is growable in nature, based on our requirement we can increase or decrease the size.

[191]:
```
s.add(99)
```

[192]:
```
print(s)
```

```
{0, 99, 100, 200, 10, 'Learn to code Python'}
```

[193]:
```
s.remove(100)
```

1.41 frozenset Data Type:

- It is exactly same as set except that it is immutable.
- Hence we cannot use add or remove functions.

[194]:
```
s ={5,10,15,20}
```

[195]:
```
fs= frozenset(s)
```

[196]:
```
type(fs)
```

[196]:
```
frozenset
```

[197]:
```
fs
```

[197]:
```
frozenset({5, 10, 15, 20})
```

[198]:
```
for num in fs:
    print(num)
```

```
10
20
5
15
```

[199]:
```
fs.add(70)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-199-a1cfd82111ea> in <module>
----> 1 fs.add(70)
```

```
AttributeError: 'frozenset' object has no attribute 'add'
```

[200]:
```
fs.remove(10)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-200-b658f8c94e09> in <module>
----> 1 fs.remove(10)

AttributeError: 'frozenset' object has no attribute 'remove'
```

1.42 dict Data Type:

- If we want to represent a group of values as key-value pairs then we should go for dict data type.
- Eg: d = {101:'learn',102:'python',103:'programming'}
- Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

[201]:
```
d = {101:'learn',102:'python',103:'programming'}
```

[202]:
```
d[101]='new'
```

[203]:
```
print(d)
```

```
{101: 'new', 102: 'python', 103: 'programming'}
```

[204]:
```
d ={} # create empty dictionary
```

[205]:
```
d['a']='Apple'
```

[206]:
```
d['b']='Banana'
```

[207]:
```
print(d)
```

```
{'a': 'Apple', 'b': 'Banana'}
```

1.43 Note: dict is mutable and the order won't be preserved

1.44 Note: In general we can use bytes and bytearray data types to represent binary information like images, video files etc

1.45 None Data Type:

- None means nothing or No value associated.
- If the value is not available, then to handle such type of cases None introduced.
- It is something like null value in Java.

[208]:
```
def m1():
 a=10
```

[209]:
```
print(m1())
```

```
None
```

[ ]:
```

```

[ ]:
```

```