In [25]:

```
# Datatypes:
# Data type represents the type of data present inside a variable
```

In [26]:

```
# In python we are not required to specify the type explicitly. Based on the value provided, the type will be ass
igned
# automatically, Hence python is called Dynamically typed language
```

In [27]:

```
# Python contains the following inbuilt datatypes
```

In [28]:

```
# 1. int datatype : we can use int datatype to represent whole numbers (integral values)
#eg:
emp1_salary=20000
type(a)
```

Out[28]:

```
int
```

In [29]:

```
# 2. Float datatype: We can use float datatype to represent floating point values (decimal values)
#eg:
emp1_pf=1210.10
type(emp1_pf)
```

Out[29]:

```
float
```

In [31]:

```
# 3. Complex datatype: A complex number is the form a+bj here is real part and b is imaginary part
#eg:
emp1=10+16j
type(emp1)
# Note: we can use complex type generally in scientific applications
```

Out[31]:

```
complex
```

In [38]:

```
# 4. bool datatype: We can use bool datatype to represent boolean values.
# The only allowed values for this datatype are True and False.
# Internally python represents True as 1 and False as 0.
#eg:
emp1_salary=20000
emp1_pf=1210.0
c=emp1_salary>emp1_pf
print(c)

True+True  # here python internally represents True as 1 and False as 0
```

```
True
```

Out[38]:

```
2
```

In [39]:

```
# 5. str type: str represents String data type.
# A String is a sequence of characters enclosed with in either single quotes or double quotes.
# eg: emp1_name='jake' -->single quotes
# eg: emp2_name="Mike" -->double quotes

# By using single quotes ot double quotes we cannot represent multi line string literals. For this requirement we
should
# go for triple single quotes(''') or triple double quotes(""")
```

In [40]:

```
#eg for strings
emp1_name='jake'
emp2_name='mike'
```

In [41]:

```
type('emp1_name')
```

Out[41]:

```
str
```

In [42]:

```
type(emp2_name)
```

Out[42]:

```
str
```

In [47]:

```
# example for multiline string literals.
emp1_description='''he is a python developer.
he loves python'''
```

In [48]:

```
print(emp1_description)
```

```
he is a python developer.
he loves python
```

In [67]:

```
#slicing of strings
# slice means piece. [] operator is called slice operator which can be used to retrieve parts of string.
# In python string follows zero based index.
# The index can be either + or -
# +ve index means forward direction from left to right
# -ve index means backward direction from left to right
```

In [76]:

```
#eg:
emp1_name='aravind'
# emp1_name[0] -o/p:'a'
# emp1_name[-1] -o/p:'d'
# emp1_name[2] -o/p:'a'
# emp1_name[60] -o/p: IndexError:string index out of range
# emp1_name[1:30] -o/p: 'ravind'
# emp1_name[1:]  -o/p:'ravind'
# emp1_name[:5] o/p:'aravi'
# emp1_name[:] o/p: 'aravind'
# emp1_name*3 o/p: 'aravindaravindaravind'
# len(emp1_name) o/p: 7
```

Out[76]:

```
7
```

In [51]:

```
#bytes datatype: byte data type represents a group of byte numbers just like array
# The only allowed values in python are 0 to 256. If we provide the values greater than 256 we will get value err
or.
# once we creates byte data type, we cannot change it's values, otherwise we will get Type Error
employee_id=[10,20,30,40]
b=bytes(employee_id)
type(b)
```

Out[51]:

```
bytes
```

In [52]:

```
# the only allowed values in python are 0 to 256. If we provide the values greater than 256 we will get value err
or.
employee_id=[10,20,30,40,267]
b=bytes(employee_id)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-52-2f93b74ccef5> in <module>
      1 # the only allowed values in python are 0 to 256. If we provide the values greater than 256
we will get value error.
      2 employee_id=[10,20,30,40,267]
----> 3 b=bytes(employee_id)

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

In [54]:

```
# once we creates byte data type, we cannot change it's values, otherwise we will get Type Error
employee_id=[10,20,30,40,267]
b=bytes(employee_id)
b[0]=100
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-54-ae273c67e1c1> in <module>
      1 # once we creates byte data type, we cannot change it's values, otherwise we will get Type E
rror
      2 employee_id=[10,20,30,40,267]
----> 3 b=bytes(employee_id)
      4 b[0]=100

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

In [61]:

```
# printing the byte values
for i in b:
    print(i)
```

```
10
20
30
40
```

In [65]:

```
#bytearray Data type : bytearray is exactly same as byte datatype except its elements can be modified.
# The only allowed values in bytearray datatype are 0 to 256.
employee_id=[10,20,30,40,50]
b=bytearray(employee_id)
for i in b:
    print(i)
```

```
10
20
30
40
50
```

In [66]:

```
# we can change the element values in byte array datatype
b[0]=100
for i in b:
    print(i)
```

```
100
20
30
40
50
```

In [79]:

```
# list datatype: If we want to represent a group of values as a single entity where insertion order is preserved
and
# duplicates are allowed then we should go for list data type.

# 1. Insertion order is preserved
# 2. heterogenous objects are allowed.
# 3. duplicates are allowed
# 4. Growable in nature
# 5. values should be enclosed in square brackets.
#eg:
employee_list=['aravind','reddy','malluriaravindreddy@gmail.com',85000,'Python']
print(employee_list)
type(employee_list)

#note: lists are mutable
```

```
['aravind', 'reddy', 'malluriaravindreddy@gmail.com', 85000, 'Python']
```

Out[79]:

```
list
```

In [83]:

```
#tuple data type: tuple data type is exactly same as list datatype except that it is immutable. we cannot change
the values
# Tuple elements can be represented within paranthesis.
# tuple is read only version of list
employee_list=('aravind','reddy','malluriaravindreddy@gmail.com',85000,'Python')
employee_list[0]='sushanth'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-83-893af4d0ce99> in <module>
      4 employee_list=('aravind','reddy','malluriaravindreddy@gmail.com',85000,'Python')
      5 type(employee_list)
----> 6 employee_list[0]='sushanth'

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

In [84]:

```
type(employee_list)
```

Out[84]:

```
tuple
```

In [89]:

```
# range Data type: range data type represents a sequence of numbers
# The elements in the range data type are not modifiable. range datatype is immutable.
#eg1:
#range(10) # generate numbers from 0 to 9
#eg2:
#range(startindex,stopindex,2)
#range(20,30,2)
```

In [92]:

```
# set Datatype: If we want to represent a group of values without duplicates where order is not important then we
should
# go for set data type.

# 1.Insertion order is not preserved.
# 2. duplicates are not allowed
# 3. heterogenous objects are allowed.
# 4. index concept is not applicable.
# 5. It is mutable collection.
# 6. Growable in nature.

#eg:
employee_list={'aravind','reddy','malluriaravindreddy@gmail,com',85000,'Python','Python'}
print(employee_list)
type(employee_list)
```

```
{85000, 'Python', 'malluriaravindreddy@gmail,com', 'aravind', 'reddy'}
```

Out[92]:

```
set
```

In [95]:

```
# frozenset datatype:
# it is exactly same as set but it is immutable.
employee_list={'aravind','reddy','malluriaravindreddy@gmail,com',85000,'Python','Python'}
employee_list=frozenset(employee_list)
print(employee_list)
type(employee_list)
```

```
frozenset({'Python', 85000, 'malluriaravindreddy@gmail,com', 'aravind', 'reddy'})
```

Out[95]:

```
frozenset
```

In [97]:

```
# dict datatype:

# if we want to represent group of values as key-value pairs then we should go for dict datatype.
employee={'firstname':'aravind','last':'reddy','email':'malluriaravindreddy@gmail.com','salary':85000,'Prog_langu
age':'C'}
print(employee)
```

```
{'firstname': 'aravind', 'last': 'reddy', 'email': 'malluriaravindreddy@gmail.com', 'salary': 85000,
'Prog_language': 'C'}
```

In [98]:

```
# dict is mutable and order wont be preserved.
# In dictionary duplicate keys are not allowed values can be duplicated.If we are trying to insert an entry with
duplicate
# key then old value will be replaced with new value.
```

In [99]:

```
#Note: In general we can use byte and byte array data types to represent binary information like images,videofile
s etc.
```

In [100]:

```
# None datatype: None means nothing or no value associated.
# If the value is not available then to handle such cases None introduced.
def m1():
    a=10
print(m1())
None
```

```
None
```