Data Types in Python

In Python, the following are the built-in data types:

int: Integer numbers like 4,55,66

float: Floating point numbers like 45.5 99.5, 234.5

complex: Complex numbers like 3 + 4j

bool: Boolean values which can be either True or False.

str: String of characters as "Habibi"

list: Ordered collection of items, which can be of different data types.

tuple: Immutable ordered collection of items, which can be of different data types.

set: Unordered collection of unique items, which can be of different data types.

dict: Unordered collection of key-value pairs, where keys must be unique and of different data types.

None: Special constant representing absence of a value.

These data types allow you to store and manipulate different kinds of data in your Python programs.

Boolean Data Types

In [3]:
```
# a data types which has two possible valeus either Ture or False
male = True;
female = False;
print("isMale {}, isFemale {}".format(male,female))
print("type of male variable:{}, type of female variable:{}".format(type(male),type(female)))
```

```
isMale True, isFemale False
type of male variable:<class 'bool'>, type of female variable:<class 'bool'>
```

In python there are comparison operators which are used to compare two objects and return either true or false, let see some of them

In [5]:
```
a = 55;
b = 98;
c = 98;
```

In [6]:
```
#Equal Operator
print(a==b)
```

```
False
```

In [7]:
```
#Not Equal Operator
print(a!=b)
```

```
True
```

In [8]:
```
#Greater than and Less than operator
print(a>b)
print(a<b)
```

```
False
True
```

In [11]:
```
#Greater than or equal to and Less than or equal to
print(b>=a)
print(b<=c)
```

```
True
True
```

Logical Operators

Logical operators are used to combine conditional statements let see some examples,

In [14]:
```
#and operator
counteryCode = "+92"
age = 18
if(counteryCode=="+92" and age>=18):
    print("You're eligible to cast a vote")
else:
    print("You're not eligible")
```

```
You're eligible to cast a vote
```

In [16]:
```
#or operator
isMember = False;
paymentReceived = True;
if(isMember or paymentReceived):
```

```
    print("Order Delivered")
else:
    print("Please buy subscription or send payment to receive your order")
```

```
Order Delivered
```

In [17]:
```
#or operator
isMember = False;
paymentReceived = False;
if(isMember or paymentReceived):
    print("Order Delivered")
else:
    print("Please buy subscription or send payment to receive your order")
```

```
Please buy subscription or send payment to receive your order
```

In [20]:
```
#not operator
#Reverse the result, returns False if the result is true
print (not True)
print(not(4<5))
```

```
False
False
```

Python Identity Operators

```
In Python, identity operators are used to compare the memory locations of two objects to determine if they are
the same object in memory. There are two identity operators:

is: Returns True if the two operands are the same object in memory, and False otherwise.

is not: Returns True if the two operands are not the same object in memory, and False otherwise.
```

In [25]:
```
a = 45;
b = a;
print(a)
print(b)
print(b is a)
```

```
45
45
True
```

In [24]:
```
a = 99;
print(a)
print(b)
print(b is a)
```

```
99
45
False
```

In [29]:
```
x = [1,2,3]
y = [1,2,3]
z = x
print("x is y :{}".format(x is y))
```

```
x is y :False
```

this returns false because x and y are containing same values but the object they are refering in memory location are different that is the reason x is y returns false, but in z we assign same object of x so it will return true as shown below,

In [30]:
```
print(x is z)
```

```
True
```

In [32]:
```
x[0] = 34;
print(x is z)
print(x)
print(z)
```

```
True
[34, 2, 3]
[34, 2, 3]
```

```
Why this behaviour changes when assigning a
x = 45;
y = x;
and changing x value does not reflect in y ????
but when we use a list like
w = [1,2,3]
and
s = w
and changes in w also reflect in s why?????
```

```
let's understand this. Basically there are some data types which are mutable and there are some data types which
are immutable, almost all simple data type in each language are immutable which are mostly strings, integers,
float, bool and etc.
Now what happen actually when we assign a immutable objet (everything in python is object) to another immutable
object like we do so in above example as,
x =45;
y = x;
both are immutables, so when we do this a new refernce will be created which is y here and it refers to the same
object the x reference is refering to, but when we do
x = 9;
a new integer object will be created because integers are immutables and at that time the x reference will refer to
that new integer object while the y reference is refering to same previous object which holds value 4 this is the
reason when we assign
x = 5
y = x
```

```
x = 9
the value of x does not reflect in y.
```

```
Now if we talk about list or any data types which are mutable, basically the same scanerio is happend like
x = [1,2,3]
and
y = x
now one object of list will be created and both x and y are refering to same object, when we made any
changes in x like x[0] = 99, because object is mutable the value will be changed/ added/ deleted according to action
in same object which mean as compare to immutable data type where new object is created when ever change is made a
new object will not be created here and only the current object will be changed and since both x and y are
refering to same object the change will be reflected in both variables since they are referring same mutable
object.
```

In [35]:
```
#now let see again
x  =5 #an integer object is created and its reference is stored in x
y = x #the refer of integer object create at time x = 5 will be stored in variable y
x = 9 # a new integer object will be created and x refers to that integer object but y refer to same
#previous integer object which contains value 4. a object can't be removed from memory whenever it has
#refers by a single or more variables. when there is no reference of object in program the
#object is removed from memory

w = [1,2,3] # a new list object will be created
x = w  # x referring to same list object created at time w = [1,2,3]
w[0] = 45 #the list object is mutable so the value inside x object will be changed instead of creating
#new object and since both w and x referring to same object the change will reflect in both variables.
print(x is y) #false
print(w is x) #true
```

```
False
True
```

Python Membership Operators

```
In Python, membership operators are used to check if a value is a member of a sequence, such as a list, tuple,
or string. There are two membership operators in Python: in and not in.
The in operator returns True if a value is found in the sequence and False otherwise.
```

```
The not in operator returns True if a value is not found in the sequence and False otherwise.
```

In [36]:
```
x = "Habibi"
print ('H' in x)
```

```
True
```

In [38]:
```
friendList = ["Nasir","Hamza","Chan","GM","Awais","Jillani","Jibran","Jahantab","Saad","Taimoor"]
print("Nasir" in x)
print("Jillani" in x)
print("Jibran" in x)
print("GM" in x)
print("Chan" in x)
print("Awais" in x and "Jahantab" in x)
print("Snake" in friendList) #this will return false as i did not put my snake friends (although they are not
#friends ) in list ).
```

```
True
True
True
True
True
True
False
```

In [40]:
```
print ("Snake" not in friendList) #not in will return true if the given argument is not in list/any other sequence
```

```
True
```

In [ ]:
```
```