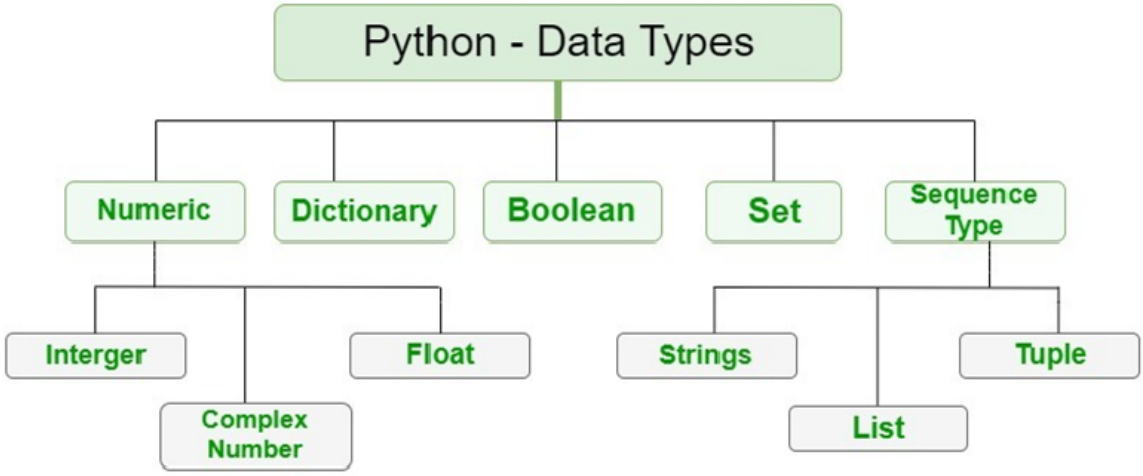
Session 2

- Python datatypes
  - Basic datatypes (int, float and bool)
- type() function
- Strings
- Type casting
- Lists
- String and list indexing

---

1) Python datatypes



We will be learning integer, float, boolean and string datatype in this session

- **integers** are whole numbers (python denotes integers with the keyword `int`)
- **floats** are decimal point numbers (python denotes floats with the keyword `float`)
- **booleans** are datatypes which can hold only 2 values either True or Flase (python denotes Booleans with the keyword `bool`)
- **strings** are a collection of individual characters(python denotes strings with the keyword `str`)

Basic datatypes (int, float and bool)

In [2]:
```
a = 10
```

In the above cell we are assigning the integer value to the variable `a`. So the datatype of variable `a` is `int`

The variable name can be ay combination of characters as long as it follows the rules for defining variable names we discussed earlier.

Usually when we define variable we name it depending on the value that it is going to hold. This helps us recoganise and remember the variable name later in the code.

In [3]:
```
my_varibale = 20
```

In [4]:
```
no_of_lines = 4
```

Since python is dynamically typed, the varibles datatype is determined by the value we assign to it on the right hand side of the assignment operator (`=`)

In [5]:
```
my_float = 12.12
```

In the above line the variable `my_float` is of the type `float` since we are assigning it a decimal point number

---

In [6]:
```
my_bool = True
bool_2 = False
```

When we assign the value of `True` or `False` to the variabel `my_bool` or `bool_2` they are considered as `bool` datatype by python

The T and F in True and False should be capital

---

2) type() function

- Type function is used to check the datatype of the variable declared.
- This fucntion will show the keyword used by the python for that particular datatype
- the type() fucntion returns a string which should be pringted out to see the output

In [19]:
```
a = 500
print(type(a))
```

```
<class 'int'>
```

In [21]:
```
b = 500.12
print(type(b))
```

```
<class 'float'>
```

In [1]:
```
a = True
print(type(a))
```

```
<class 'bool'>
```

In [22]:
```
mystring = "hello world"
print(mystring)
print(type(mystring))
```

```
hello world
<class 'str'>
```

---

3) String datatype

A string is a collection of characters

In other languages we have a separate datatype called as `char` which stores individual characters
In python we can get the same behaviour by defining a string of length 1

In python a string can be defined by enclosing the characters in single double or tripple quotes

In [7]:
```
my_string1 = 'hello world'
```

In [9]:
```
my_string2 = "hello world"
```

In [10]:
```
my_string3 = '''hello world'''
```

The intention behind providing this kind of flexibility is consiidering the cases were we will have to have an actual single or double quote in our sentence
Suppose we want to print the following line `This is Sam's house`

In [11]:
```
my_string1 = 'This is Sam's house'
```

```
  File "<ipython-input-11-6f004d05cb8a>", line 1
    my_string1 = 'This is Sam's house'
                               ^
SyntaxError: invalid syntax
```

In such cases we use the double quotes

In [12]:
```
my_string1 = "This is Sam's house"
```

Whenever we need the double quotes inside a string we declare the string with tripple quotes

In [14]:
```
my_string1 = '''Sam says "hi everyone"'''
```

4) Type Casting

Type casting is the precess of converting one datatype to another datatype.

Lets try to build an application which takes input from the user and return the `number entered + 10`

In [8]:
```
a = input("enter the number : ")
print(a+10)
```

```
enter the number : 20
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-f16ddfb0e55c> in <module>
      1 a = input("enter the number : ")
----> 2 print(a+10)

TypeError: can only concatenate str (not "int") to str
```

We get a type Error. Which says we cannot add Integer datatype with a string datatype.
Remember when we learned the input() fucntion we saw how all things enetered by a keyboard including numbers and symbols are strings by default.
So we need to Type cast the string representaion of number back to integer to do a arithmatic operation.

In [11]:
```
a = input("enter the number : ")
a_int = int(a)
print(a_int+10)
```

```
enter the number : 20
30
```

We can convert any datatype to any other datatype provided that it justifies this operation.
when we convert a float to an int something called as truncation happens. This is different that rounding off as the numbers after the decimal point are simply removed.

In [2]:
```
a = 12.12
print(type(a))
b = int(a)
print(type(b))
print(b)
```

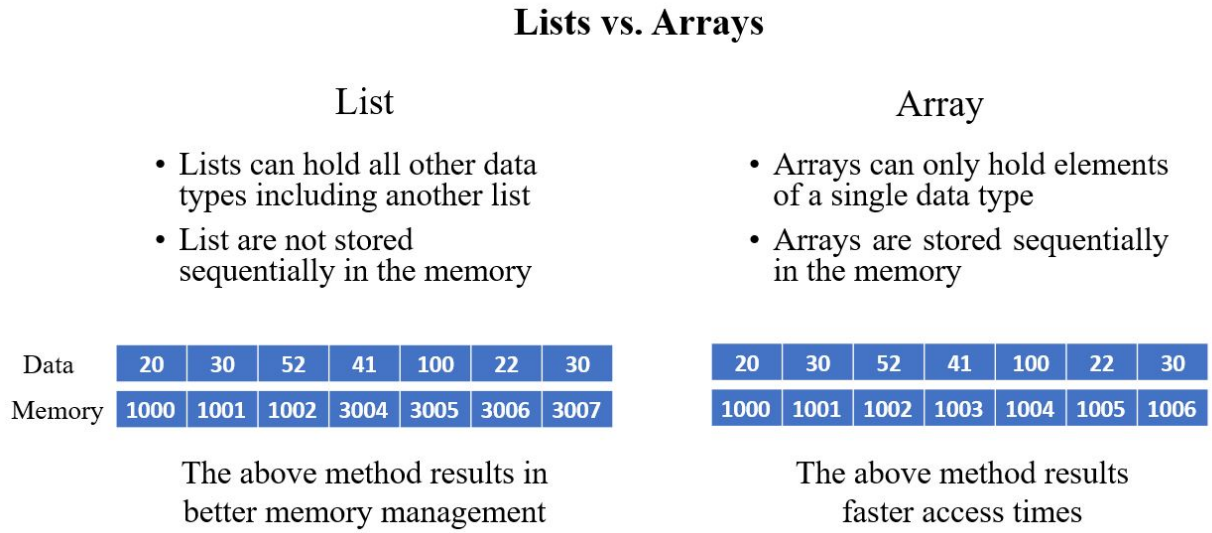
```
<class 'float'>
<class 'int'>
12
```

5) Lists

List is a collection data types in python

- Lists are ordered
- Lists are changeable (mutable)
- Lists can hold another list or another collection data type
- Lists can hold duplicate data

Difference between list and array



Lists are constructed with brackets `[ ]` and commas separating every element in the list.

In [2]:
```
# Assign a list to an variable named my_list
my_list = [1,2,3]
print(my_list)
```

```
[1, 2, 3]
```

We just created a list of integers, but lists can actually hold different object types. For example:

In [3]:
```
my_list = ['A string',23,100.232,'o']
print(my_list)
```

```
['A string', 23, 100.232, 'o']
```

Lists can hold any object in python. Including another list.

Such a list can also be called a multi-dimensional list

In [4]:
```
my_list = [[1,2,3],[4,5,6],[7,8,9]]
print(my_list)
```

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

6) Indexing and Slicing

- Indexing referrs to accessing individual elements on a collection datatype
- Slicing referrs to accessing a group of elements from the collection datatype

Indexing

In [11]:
```
my_list = [11,22,33,44,55,66]
my_list[3]
```

Out[11]: 44

The Index number of elements starts from 0

```
list elements : 11 22 33 44 55 66
index numbers :  0  1  2  3  4  5
```

The Indexing can also be applied to string

In [12]:
```
x = "hello"
x[4]
```

Out[12]: 'o'

The examples we saw was of single dimensional indexing. We can also do indexing on multi-dimensional objects to access elements

In [13]:
```
my_list = [[1,2,3],[4,5,6],[7,8,9]]
my_list[1]
```

Out[13]: [4, 5, 6]

As we can see indexing a multi dimensional object with single index return the entire list itself. We can use an additional index parameter to access individual element

In [14]:
```
my_list = [[1,2,3],[4,5,6],[7,8,9]]
my_list[1][1]
```

Out[14]: 5

In [15]:
```
my_list = [[1,2,3],'hello',[7,8,9]]
my_list[1][3]
```

Out[15]: 'l'

A string inside a list also acts like a muti-dimensional object

---

Home Work

1. create a basic calculator

- Take 2 numbers from the user (using input statement)
- Print out the addition and multiplication of those 2 numbers

2. Try to grab the number 400 using list

In [2]:
```
my_list = [[1,2,3],'hello',[7,8,[100,200,300,400]]]
    #          0      1     0  1  0   1    2   3
```

In [ ]:
```

```