Aim: To review fundamentals of Python programming for Data science and machine learning programming. Here, we focus on concepts such as

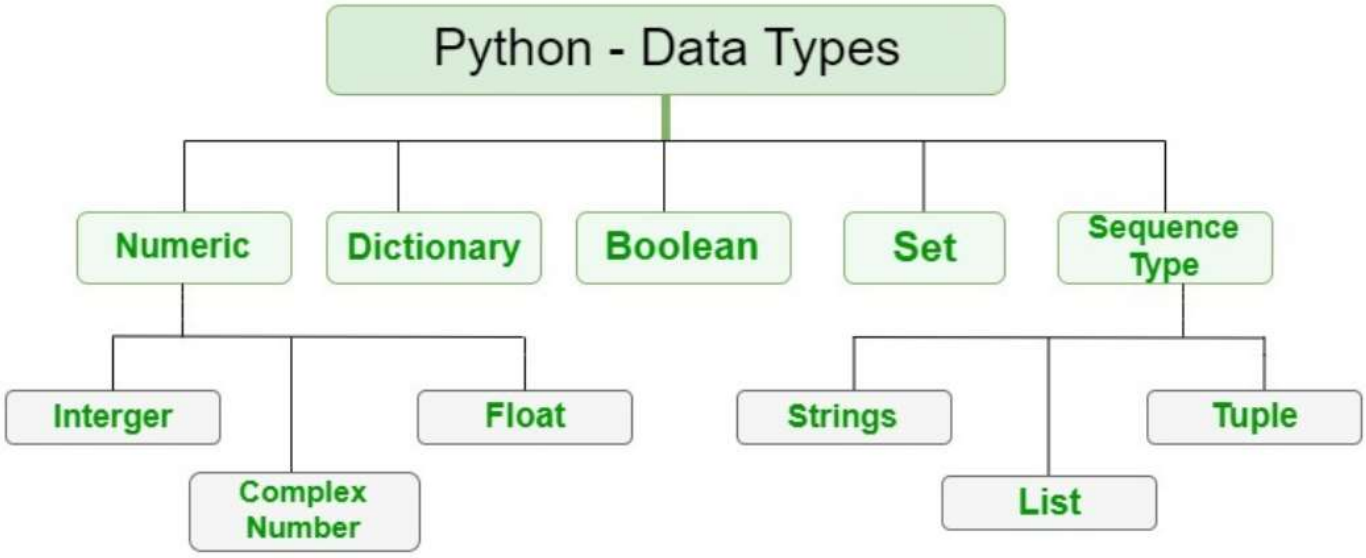
(a) Data types

(b) Containers: List, Tuple, Dictionary, Sets

(c) Functions and return statement

Program

(a) **Data types**




```
1 a = 5
2 print("Type of a: ", type(a))
3
4 b = 5.0
5 print("\nType of b: ", type(b))
6
7 c = 2 + 4j
8 print("\nType of c: ", type(c))
```

```
Type of a:  <class 'int'>

Type of b:  <class 'float'>

Type of c:  <class 'complex'>
```

```
1 # Python Program for
2 # Creation of String
3
4 # Creating a String
5 # with single Quotes
6 String1 = 'Welcome to the Data Science Lab'
```

```
String1 = "Welcome to the Data Science Lab"
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm happy to program"
print("\nString with the use of Double Quotes: ")
print(String1)
print(type(String1))

# Creating a String
# with triple Quotes
String1 = '''I'm happy to learn "Python"'''
print("\nString with the use of Triple Quotes: ")
print(String1)
print(type(String1))

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Programming
            is
            fun'''
print("\nCreating a multiline String: ")
print(String1)


# Python Program to Access
# characters of String

String1 = "Programming"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])

```

```
String with the use of Single Quotes:
Welcome to the Data Science Lab

String with the use of Double Quotes:
I'm happy to program
<class 'str'>

String with the use of Triple Quotes:
I'm happy to learn "Python"

<class 'str'>

Creating a multiline String:
Programming
```

```
                                is
                                fun
Initial String:
Programming

First character of String is:
P

Last character of String is:
g
```

(b) **Containers**: List, Tuple Dictionary, Sets, Numpy Arrays

```
# Python program to demonstrate
# Creation of List

# Creating a List
List = []
print("Initial blank List: ")
print(List)

# Creating a List with
# the use of a String
List = ['Python']
print("\nList with the use of String: ")
print(List)

# Creating a List with
# the use of multiple values
List = ["Python", "C", "Java"]
print("\nList containing multiple values: ")
print(List[0])
print(List[2])

# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Python', 'Java'], ['C']]
print("\nMulti-Dimensional List: ")
print(List)

```

```
Initial blank List:
[]

List with the use of String:
['Python']

List containing multiple values:
Python
Java



Multi-Dimensional List:
[['Python', 'Java'], ['C']]
```

```
# Python program to demonstrate
```

```
# creation of Set

# Creating an empty tuple
Tuple1 = ()
print("Initial empty Tuple: ")
print (Tuple1)

# Creating a Tuple with
# the use of Strings
Tuple1 = ('Java', 'Python')
print("\nTuple with the use of String: ")
print(Tuple1)

# Creating a Tuple with
# the use of list
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))

# Creating a Tuple with the
# use of built-in function
Tuple1 = tuple('Python')
print("\nTuple with the use of function: ")
print(Tuple1)

# Creating a Tuple
# with nested tuples
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('data', 'science')
Tuple3 = (Tuple1, Tuple2)
print("\nTuple with nested tuples: ")
print(Tuple3)


# Python program to
# demonstrate accessing tuple

tuple1 = tuple([1, 2, 3, 4, 5])

# Accessing element using indexing
print("First element of tuple")
print(tuple1[0])

# Accessing element from last
# negative indexing
print("\nLast element of tuple")
print(tuple1[-1])

print("\nThird last element of tuple")
print(tuple1[-3])

```

```
Initial empty Tuple:
()
```

```
Tuple with the use of String:
('Java', 'Python')

Tuple using List:
(1, 2, 4, 5, 6)

Tuple with the use of function:
('P', 'y', 't', 'h', 'o', 'n')

Tuple with nested tuples:
((0, 1, 2, 3), ('data', 'science'))
First element of tuple
1

Last element of tuple
5

Third last element of tuple
3
```

```
# Python program to
# demonstrate boolean type

print(type(True))
print(type(False))

print(type(true)) #Error for case sensitive t in True

```

```
<class 'bool'>
<class 'bool'>
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-d732ccf9e5f8> in <module>()
      5 print(type(False))
      6
----> 7 print(type(true)) #Error for case sensitive t in True

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

SEARCH STACK OVERFLOW

```
# Python program to demonstrate
# Creation of Set in Python

# Creating a Set
set1 = set()
print("Initial blank Set: ")
print(set1)

# Creating a Set with
# the use of a String

set1 = set("Programming")
print("\nSet with the use of String: ")
print(set1)
```

```
14
15 # Creating a Set with
16 # the use of a List
17 set1 = set(["Python", "Programming", "Java", "Programming"])
18 print("\nSet with the use of List: ")
19 print(set1)
20
21 # Creating a Set with
22 # a mixed type of values
23 # (Having numbers and strings)
24 set1 = set([1, 2, 'C', 4, 'Java', 6, 'Java'])
25 print("\nSet with the use of Mixed Values")
26 print(set1)
27
28 # Python program to demonstrate
29 # Accessing of elements in a set
30
31 # Creating a set
32 set1 = set(["Java", "Program", "Python", "Java"])
33 print("\nInitial set")
34 print(set1)
35
36 # Accessing element using
37 # for loop
38 print("\nElements of set: ")
39 for i in set1:
40     print(i, end =" ")
41
42 # Checking the element
43 # using in keyword
44 print("Java" in set1)
45
46
```

```
Initial blank Set:
set()

Set with the use of String:
{'r', 'P', 'o', 'n', 'g', 'a', 'm', 'i'}

Set with the use of List:
{'Java', 'Programming', 'Python'}

Set with the use of Mixed Values
{1, 2, 4, 6, 'Java', 'C'}

Initial set
{'Java', 'Python', 'Program'}

Elements of set:
Java Python Program True
```

```
1 # Creating an empty Dictionary
2 Dict = {}
3 print("Empty Dictionary: ")
4 print(Dict)
```

```
 5
 6 # Creating a Dictionary
 7 # with Integer Keys
 8 Dict = {1: 'Java', 2: 'C', 3: 'Python'}
 9 print("\nDictionary with the use of Integer Keys: ")
10 print(Dict)
11
12 # Creating a Dictionary
13 # with Mixed keys
14 Dict = {'Name': 'Java', 1: [1, 2, 3, 4]}
15 print("\nDictionary with the use of Mixed Keys: ")
16 print(Dict)
17
18 # Creating a Dictionary
19 # with dict() method
20 Dict = dict({1: 'Java', 2: 'C', 3:'Python'})
21 print("\nDictionary with the use of dict(): ")
22 print(Dict)
23
24 # Creating a Dictionary
25 # with each item as a Pair
26 Dict = dict([(1, 'Java'), (2, 'C'),  ('program', 'Python'), (3, 'Machine Learning')])
27 print("\nDictionary with each item as a pair: ")
28 print(Dict)
29
30 print("Accessing a element using key:")
31 print(Dict['program'])
32
33 # accessing a element using get()
34 # method
35 print("Accessing a element using get:")
36 print(Dict.get(3))
37
```

```
Empty Dictionary:
{}

Dictionary with the use of Integer Keys:
{1: 'Java', 2: 'C', 3: 'Python'}

Dictionary with the use of Mixed Keys:
{'Name': 'Java', 1: [1, 2, 3, 4]}

Dictionary with the use of dict():
{1: 'Java', 2: 'C', 3: 'Python'}

Dictionary with each item as a pair:
{1: 'Java', 2: 'C', 'program': 'Python', 3: 'Machine Learning'}
Accessing a element using key:
Python
Accessing a element using get:
Machine Learning
```

(c) Functions and return statement

```
def my_function():
  print("Hello from a simple function")

my_function()

def my_function(fname):
  print("Hello " + fname + " from a fuction with one argument")

my_function("Amy")
my_function("Sara")
my_function("Thomas")

#function with 2 arguments
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Amy", "Joseph")

#function with arbritrary number of arguments
# If the number of arguments is unknown, add a * before the parameter name:
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Amy", "Sara", "Stephen")

#Arguments with the key = value syntax.
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Amy", child2 = "Sara", child3 = "Stephen")

#Keyword Arguments
# If the number of keyword arguments is unknown, add a double ** before the parameter n

def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Sara", lname = "Peter")

#Default parameter value
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")

#List as argument

def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]
```

```
fruits = ["apple", "banana", "cherry"]

my_function(fruits)

#return statement

def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))

#Pass statement
#function definitions cannot be empty, but if you for some reason have a function
#definition with no content, put in the pass statement to avoid getting an error
def myfunction():
  pass

#recursion Example

def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)


#Lambda function
#A lambda function is a small anonymous function.
#A lambda function can take any number of arguments, but can only have one expression.

#Example 1: Multiply argument a with argument b and return the result:
x = lambda a, b : a * b
print(x(5, 6))

#Fuction within another fuction
#The power of lambda is better shown when you use them as an anonymous function
#inside another function.

print("\n\n\n Function with function")
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print("My doubler: %d" %mydoubler(11))
print("My Tripler: %d" %mytripler(11))

```

```
Hello from a simple function
Hello Amy from a fuction with one argument
Hello Sara from a fuction with one argument
Hello Thomas from a fuction with one argument
Amy Joseph
The youngest child is Stephen
The youngest child is Stephen
His last name is Peter
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45


Recursion Example Results
1
3
6
10
15
21
30



 Function with function
My doubler: 22
My Tripler: 33
```