PYTHON
EXPERIMENT 1

GROUP MEMBERS NAME:
Harshal Jain - 60004190041
Junaid Girkar - 60004190057
Khushi Chavan - 60004190061
Megh Dedhia - 60004190067

AIM: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.

THEORY:
Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instances (object) of these classes.
There are various data types in Python. Some of the important types are listed below.

Integers: You can use an integer to represent numeric data, and more specifically, whole numbers from negative infinity to infinity, like 4, 5, or -1.

CODE:

```
integer_1 = 100
integer_2 = 50

# various operations on integers
print("TYPE: "+str(type(integer_1)))
print ("Multiplication: ", integer_1*integer_2)
print ("Addition: ",integer_1+integer_2)
print ("Subtraction: ", integer_1-integer_2)
print ("Division: ",integer_1/integer_2)
```

OUTPUT:

```
TYPE: <class 'int'>
Multiplication:  5000
Addition:  150
Subtraction:  50
Division:  2.0
```

Float: "Float" stands for 'floating point number'. You can use it for rational numbers, usually ending with a decimal figure, such as 1.11 or 3.14.

CODE:

```
float_1 = 12.539
float_2 = 6.78

# various operations on floats
print("TYPE: "+str(type(float_1)))
print ("Multiplication: ", float_1*float_2)
print ("Addition: ",float_1+float_2)
print ("Subtraction: ", float_1-float_2)
print ("Division: ",float_1/float_2)
```

OUTPUT:

```
TYPE: <class 'float'>
Multiplication:  85.01442
Addition:  19.319
Subtraction:  5.7589999999999995
Division:  1.849410029498525
```

Boolean: This built-in data type that can take up the values: True and False, which often makes them interchangeable with the integers 1 and 0. Booleans are useful in conditional and comparison expressions, just like in the following examples:

CODE:

```
has_passed = False  #Default Value
marks = 80
if (marks > 50):
    # true has been assigned to has_passed (a boolean)
    has_passed = True
print("Type: ", str(type(has_passed)))
print (has_passed)
```

OUTPUT:

```
Type:  <class 'bool'>
True
```

Strings: Strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.Strings in Python can be created using single quotes or double quotes or even triple quotes.
individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

CODE:

```
#strings

s = 'Hello world!'

print(type(s))

print(s)

print("s[4] = ", s[4])

print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python - TypeError
s[5] ='d'
```

OUTPUT:

```
<class 'str'>
Hello world!
s[4] =  o
s[6:11] =  world


---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-54c802e3dd27> in <module>
     14 # Generates error
     15 # Strings are immutable in Python
---> 16 s[5] ='d'

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

Lists: Lists in Python are one of the most versatile collection object types available. The other two types are dictionaries and tuples, but they are really more like variations of lists.

- Python lists do the work of most of the collection data structures found in other languages and since they are built-in, you don't have to worry about manually creating them.
- Lists can be used for any type of object, from numbers and strings to more lists.
- They are accessed just like strings (e.g. slicing and concatenation) so they are simple to use and they're variable length, i.e. they grow and shrink automatically as they're used.
- In reality, Python lists are C arrays inside the Python interpreter and act just like an array of pointers.

CODE:

```
#lists
l = [10,20,30,40,50,60,70,80]

print(type(l))

print(l)

print("l[2] = ", l[2])

print("l[0:3] = ", l[0:3])

print("l[5:] = ", l[5:])
```

OUTPUT:

```
<class 'list'>
[10, 20, 30, 40, 50, 60, 70, 80]
l[2] =  30
l[0:3] =  [10, 20, 30]
l[5:] =  [60, 70, 80]
```

Array: An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array.

A user can treat lists as arrays. However, users cannot constraint the type of elements stored in a list. If you create arrays using the array module, all elements of the array must be of the same type.

Arrays in Python can be created by importing array modules. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

In-built functionalities of an array:

- insert() - Add one or more data elements in any position
- append() - Add the value at the end of the array
- Index operator [] - Used to access the array items.
- remove () - Remove one array item from any position
- pop() - Removes one array time from the end
- Slicing operator [Start Index:End Index] - This creates a sub-array list from the main array.
- sort() - Sorts the list

CODE:

```
# Python program to demonstrate Creation of Array

# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

print ("Array before insertion : ", end =" ")
for i in range (0, 3):
    print (b[i], end =" ")
print()

# adding an element using append()
b.append(4.4)

print ("Array after insertion : ", end =" ")
for i in (b):
    print (i, end =" ")
print()

# accessing element of array
```

```
print("Access element index 0: ", a[0])

# initializing array with array values and signed integers
arr = arr.array('i', [1, 2, 3, 1, 5, 6, 8, 10, 12, 55, 99, 2])

# printing original array
print ("The new created array is : ", end ="")
for i in range (0, 5):
    print (arr[i], end =" ")
print ("\r")

# using pop() to remove element at 2nd position
print ("The popped element is : ", end ="")
print (arr.pop(2))

# printing array after popping
print ("The array after popping is : ", end ="")
for i in range (0, 4):
    print (arr[i], end =" ")
print("\r")

# using remove() to remove 1st occurrence of 1
arr.remove(1)

# printing array after removing
print ("The array after removing is : ", end ="")
for i in range (0, 3):
    print (arr[i], end =" ")
```

OUTPUT:

```
The new created array is :  1 2 3
Array before insertion :  2.5 3.2 3.3
Array after insertion :  2.5 3.2 3.3 4.4
Access element index 0:  1
The new created array is : 1 2 3 1 5
The popped element is : 3
The array after popping is : 1 2 1 5
The array after removing is : 2 1 5
```

Dictionary: In python, dictionaries are similar to hash or maps in other languages. It consists of key value pairs. The value can be accessed by a unique key in the dictionary.

- Keys are unique & immutable objects.

Syntax:

```
dictionary = {"key name": value}
```

CODE:

```
#dictionaries

d = {1:'val','key':2}

print(type(d))

print(d)

print("d[1] = ", d[1])

print("d['key'] = ", d['key'])

# Generates error - KeyError
print("d[2] = ", d[2])
```

OUTPUT:

```
<class 'dict'>
{1: 'val', 'key': 2}
d[1] =  val
d['key'] =  2
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-23-d7fc8db7cd00> in <module>
     12
     13 # Generates error - KeyError
---> 14 print("d[2] = ", d[2])
KeyError: 2
```

Tuple: Python tuples work exactly like Python lists except they are immutable, i.e. they can't be changed in place. They are normally written inside parentheses to distinguish them from lists (which use square brackets), but as you'll see, parentheses aren't always necessary. Since tuples are immutable, their length is fixed. To grow or shrink a tuple, a new tuple must be created.

CODE:

```
#tuples

t = (1,'program', 3+4j)

print(type(t))

print(t)

print("t[1] = ", t[1])

print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable - TypeError
t[0] = 10
```

OUTPUT:

```
<class 'tuple'>
(5, 'program', (1+3j))
t[1] =  program
t[0:3] =  (5, 'program', (1+3j))
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-79b203763893> in <module>
     13 # Generates error
     14 # Tuples are immutable - TypeError
---> 15 t[0] = 10

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

Sets: Unordered collection of unique objects.

- Set operations such as union (|) , intersection(&), difference(-) can be applied on a set.
- Frozen Sets are immutable i.e once created further data can't be added to them
- {} are used to represent a set.Objects placed inside these brackets would be treated as a set.

CODE:

```
#sets

s =  {1, 2, 3, 4, 5}

print(type(s))

print(s)

#Generates Error - TypeError

print(s[1])
```

OUTPUT:

```
<class 'set'>
{1, 2, 3, 4, 5}


---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-043a1c74b72c> in <module>
      8
      9 #Generates Error - TypeError
---> 10 print(s[1])

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

If Statement:

CODE:

```
a=False
if a:
    print("Inside if")
else:
    print("In else")
```

OUTPUT:

```
In else
```

Elif statement:

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

CODE:

```
# printing the biggest of 3

a=int(input())
b=int(input())
c=int(input())

if(a>=b and a>=c):
    print(a)
elif(b>=c):
    print(b)
else:
    print(c)
```

OUTPUT:

```
>>10
>>1
>>11
11
```

While Loop:

Syntax :

```
while expression:
    statement(s)
```

CODE:

```
n=int(input())
count=1
while n>=count:
    print(count)
    count=count+1;
```

OUTPUT:

```
>>5
1
2
3
4
5
```

For Loop:

Syntax:

```
for iterator_var in sequence:
    statements(s)
```

CODE:

```
animals = ['cat', 'dog', 'mouse', 'lion', 'tiger']
for animal in animals:
    print(animal)
```

OUTPUT:

```
cat
dog
mouse
lion
tiger
```

Nested Loops:
Python programming language allows one loop inside another loop. Following section shows a few examples to illustrate the concept.
Syntax:
The syntax for a nested for loop statement in Python programming language is as follows:

```
for iterator_var in sequence:
    for iterator_var in sequence:
        statements(s)
        statements(s)
```

The syntax for a nested while loop statement in Python programming language is as follows:

```
while expression:
    while expression:
        statement(s)
        statement(s)
```

CODE:

```
for i in range(1, 6):
    str = ""
    for j in range(i):
        str += "*"
    print(str)
```

OUTPUT:

```
*
**
***
****
*****
```

CODE:

```
#checking prime in a Range

n=int(input())
k=2
while k<=n:
    d=2
    flag=False
    while d<k:
        if(k%d ==0):
```

```
            flag=True
        d=d+1
    if not flag:
        print(k)
    k=k+1
```

OUTPUT:

```
>>20
2
3
5
7
11
13
17
19
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Break Statement:

The break statement is used to terminate the loop containing it, the control of the program will come out of that loop.

CODE:

```
i=1
while i<10:
    if i == 4:
        break
    print(i)
    i=i+1
```

OUTPUT:

```
1
2
3
```

Continue Statement:
When the program encounters a continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the next iterations.

CODE:

```
# even which are not divisible by 7
n=int(input())
for i in range(2,n+1,2):
    if i%7==0:
        continue
    print(i)
```

OUTPUT:

```
>>10
2
4
6
8
10
```

Pass Statement:
Pass statement is a null operation, which is used when the statement is required syntactically.

CODE:

```
i=3
if i<7:
    pass
print("end")
```

OUTPUT:

```
end
```

CONCLUSION: We learnt about the different data types available in python and implemented each of them. We also learnt and implemented the different control statements in python. Overall it was an amazing learning experience and we also found out about this new tool Overleaf which is very helpful while working in groups and increases the team efficiency by a wide margin.