

Python – Experiment – 1

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

Theory & Implementation:

Datatypes: Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary

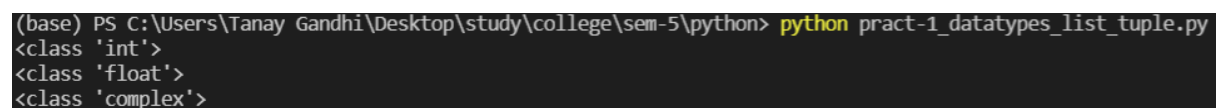
Numeric: In Python, numeric data type represents the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python. The type keyword tells us type of the datatype.

Code:

```
# Print datatypes

print(type(30))
print(type(30.5))
print(type(30+5j))
```

Output:



```
(base) PS C:\Users\tanay.gandhi\Desktop\study\college\sem-5\python> python pract-1_datatypes_list_tuple.py
<class 'int'>
<class 'float'>
<class 'complex'>
```

Sequence Type: In Python, sequence is the ordered collection of similar or different data types. Sequences allow to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

- String

- List
- Tuple

String: In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Creating String: Strings in Python can be created using single quotes or double quotes or even triple quotes.

Accessing elements of String: In Python, 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.

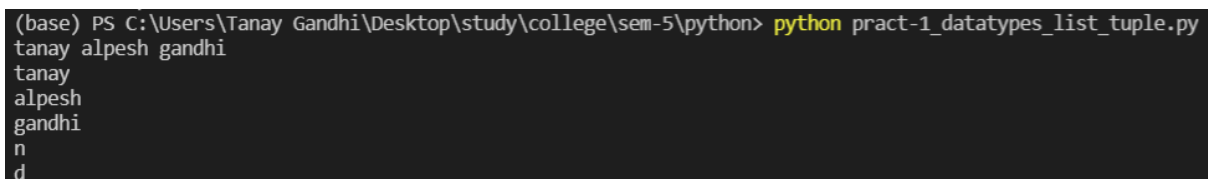
Code:

```
# Print string
string1 = 'tanay alpesh gandhi'
print(string1)

# 3 single quotes allows to print on the next line
string2 = '''tanay
alpesh
gandhi'''
print(string2)

# Access single letter in a string
print(string1[2])
print(string1[-3])
```

Output:



```
(base) PS C:\Users\tanay.gandhi\Desktop\study\college\sem-5\python> python pract-1_datatypes_list_tuple.py
tanay alpesh gandhi
tanay
alpesh
gandhi
n
d
```

List: Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating List: Lists in Python can be created by just placing the sequence inside the square brackets []. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

```
a = [1, 2.2, 'python']
```

We can use the slicing operator [] to extract an item or a range of items from a list. The index starts from 0 in Python.

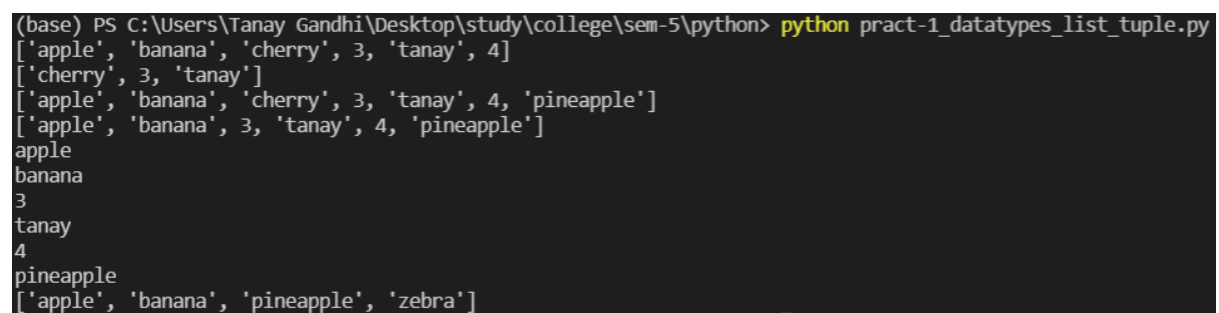
Accessing elements of List: In order to access the list items, refer to the index number. Use the index operator [] to access an item in a list. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

Code:

```
# List -> can be modified after being created
list1 = ['apple', 'banana', 'cherry', 3]
list2 = ['tanay', 4]
list1.extend(list2)
print(list1)
print(list1[2:-1])
list1.append('pineapple')
print(list1)
list1.remove('cherry')
print(list1)
for i in range(len(list1)):
    print(list1[i])

list1.remove(3)
list1.remove(4)
list1[2] = 'zebra'
list1.sort()
print(list1)
```

Output:



```
(base) PS C:\Users\tanay\OneDrive\Desktop\study\college\sem-5\python> python pract-1_datatypes_list_tuple.py
['apple', 'banana', 'cherry', 3, 'tanay', 4]
['cherry', 3, 'tanay']
['apple', 'banana', 'cherry', 3, 'tanay', 4, 'pineapple']
['apple', 'banana', 3, 'tanay', 4, 'pineapple']
apple
banana
3
tanay
4
pineapple
['apple', 'banana', 'pineapple', 'zebra']
```

Tuple: Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple: In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

Accessing elements of Tuple: In order to access the tuple items refer to the index number. Use the index operator [] to access an item in a tuple. The index must be an integer. Nested tuples are accessed using nested indexing.

Code:

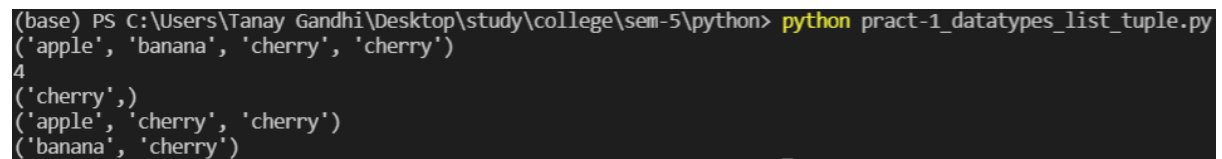
```
# Tuples -> Cannot be modified after created
tuple1 = ('apple', 'banana', 'cherry', 'cherry')
print(tuple1)
print(len(tuple1))
print(tuple1[2:-1])

list3 = list(tuple1)
list3.remove('banana')
tuple1 = tuple(list3)
print(tuple1)

thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)

print(thistuple)
```

Output:



```
(base) PS C:\Users\Tanay Gandhi\Desktop\study\college\sem-5\python> python pract-1_datatypes_list_tuple.py
('apple', 'banana', 'cherry', 'cherry')
4
('cherry',)
('apple', 'cherry', 'cherry')
('banana', 'cherry')
```

Set: In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets: Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by 'comma'. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

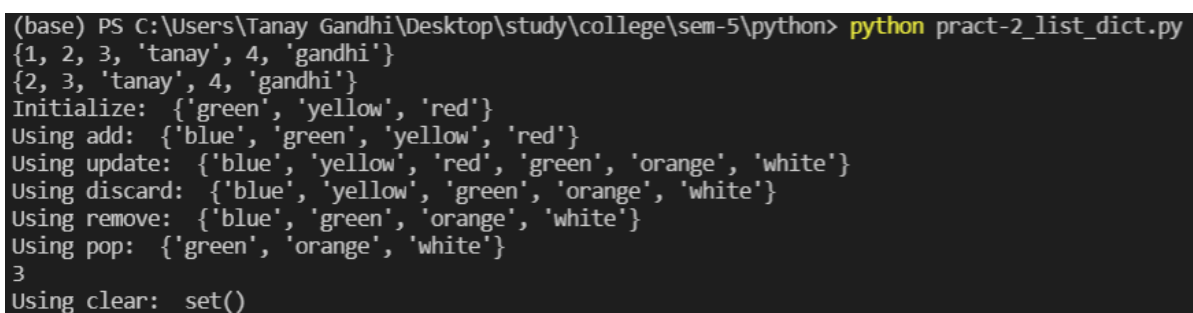
Accessing elements of Sets: Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Code:

```
# Set & List operations

list1 = [1, 2, 3, 4]
set1 = {'tanay', 'gandhi'}
set1.update(list1)
print(set1)
set1.pop()
print(set1)

set1 = {'red', 'yellow', 'green'}
print('Initialize: ', set1)
set1.add('blue')
print('Using add: ', set1)
set2 = {'white', 'orange'}
set1.update(set2)
print('Using update: ', set1)
set1.discard('red')
print('Using discard: ', set1)
set1.remove('yellow')
print('Using remove: ', set1)
set1.pop()
print('Using pop: ', set1)
print(len(set1))
set1.clear()
print('Using clear: ', set1)
```

Output:

```
(base) PS C:\Users\tanay\OneDrive\Desktop\study\college\sem-5\python> python pract-2_list_dict.py
{1, 2, 3, 'tanay', 4, 'gandhi'}
{2, 3, 'tanay', 4, 'gandhi'}
Initialize: {'green', 'yellow', 'red'}
Using add: {'blue', 'green', 'yellow', 'red'}
Using update: {'blue', 'yellow', 'red', 'green', 'orange', 'white'}
Using discard: {'blue', 'yellow', 'green', 'orange', 'white'}
Using remove: {'blue', 'green', 'orange', 'white'}
Using pop: {'green', 'orange', 'white'}
3
Using clear: set()
```

Dictionary: Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon: whereas each key is separated by a 'comma'.

Creating Dictionary: In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces {}.

Accessing elements of Dictionary: In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. There is also a method called get() that will also help in accessing the element from a dictionary.

Code:

```
# Dictionary and its operations

dict1 = {1: "Maharashtra", 2: "Himachal P.", 3: "Madhya P."}
print(len(dict1))
print(dict1)
print(dict1.keys())
print(dict1.values())
print(dict1.items())
print(dict1.get(1))
dict1[1] = 'Kerala'
print(dict1)
dict1.update({1: "Maharashtra"})
print(dict1)
dict1[4] = 'Gujarat'
print(dict1)
dict1.pop(4)
print(dict1)
print(2 in dict1)
del dict1[3]
print(dict1)
dict1.popitem()
print(dict1)
```

Output:

```
(base) PS C:\Users\tanay.gandhi\Desktop\study\college\sem-5\python> python pract-2_list_dict.py
3
{1: 'Maharashtra', 2: 'Himachal P.', 3: 'Madhya P.'}
dict_keys([1, 2, 3])
dict_values(['Maharashtra', 'Himachal P.', 'Madhya P.'])
dict_items([(1, 'Maharashtra'), (2, 'Himachal P.'), (3, 'Madhya P.')])
Maharashtra
{1: 'Kerala', 2: 'Himachal P.', 3: 'Madhya P.'}
{1: 'Maharashtra', 2: 'Himachal P.', 3: 'Madhya P.'}
{1: 'Maharashtra', 2: 'Himachal P.', 3: 'Madhya P.', 4: 'Gujarat'}
{1: 'Maharashtra', 2: 'Himachal P.', 3: 'Madhya P.'}
True
{1: 'Maharashtra', 2: 'Himachal P.'}
{1: 'Maharashtra'}
```

Operators: Operators are used to perform operations on variables and values. Examples of some operators are shown in the table below:

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Code:

```
# Operators in python
```

```
a = 50
b = 90
print(a//b)
print(a/b)
print(a+b)
print(a-b)
print(a*b)
print(a**2)
l = [1, 2, 3, 4, 5]
if (6 not in l):
    print('Not there')
if 1 in l:
    print('Found')
print(a/5 == b/10)
print(a/5)
print(b/10)
print(a+40 == b)
print(a != b)
print(a is b)
print(not a is b)
print(a-40 is b)
```

Output:

```
(base) PS C:\Users\tanay.gandhi\Desktop\study\college\sem-5\python> python pract-2_list_dict.py
0
0.5555555555555556
140
-40
4500
2500
Not there
Found
False
10.0
9.0
True
True
False
True
False
```

Loops and Control Statements (continue, break and pass) in Python:

Python programming language provides following types of loops to handle looping requirements.

While Loop**Syntax:**

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

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

For Loop

In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to for each loop in other languages.

Syntax:

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

It can be used to iterate over iterators and a range.

We can use for in loop for user defined iterators.

Nested Loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

```
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)

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement

It returns the control to the beginning of the loop.

Break Statement

It brings control out of the loop

Pass Statement

We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

Code:

```
import math
r = float(input("Enter the radius: "))
area = math.pi*r**2
print("Area of the circle: ", area)
```

```
n = int(input("Enter digit: "))
```

```
if n == 0:
    print("Zero")
elif n == 1:
    print("One")
elif n == 2:
    print("Two")
elif n == 3:
    print("Three")
elif n == 4:
    print("Four")
elif n == 5:
    print("Five")
elif n == 6:
    print("Six")
elif n == 7:
    print("Seven")
elif n == 8:
    print("Eight")
elif n == 9:
```

```
print("Nine")

print("Printing nos from 100 to 200: ")
for i in range(100, 200):
    if (i % 2 != 0):
        print(i, end=" ")

print("Sum is: ")
my_list = [1, 2, 3, 4, 5, 6]
sum = 0
i = 0

while (i < len(my_list)):
    sum += my_list[i]
    i += 1

print("Sum = {}".format(sum))

print("Printing star pattern")
n = 20
for i in range(1, 11):
    print(' '*n, end='') # repet space for n times
    print('* '*(i)) # repeat stars for i times
    n -= 1

for i in range(1, 101):
    print(i, end=' ')
    if(i % 10 == 0):
        print()

my_list = [1, 2, 3, 4, 5, 6]
a = int(input("No to be searched: "))
for i in range(0, len(my_list)):
    if (my_list[i] == a):
        print("Number found at index", i)
        break
    else:
        continue

mylist = []
print("Enter 5 elements for the list: ")
for i in range(5):
    val = int(input())
    mylist.append(val)

print("Enter an element to be search: ")
elem = int(input())
```

```
for i in range(5):
    if elem == mylist[i]:
        print("\nElement found at Index:", i)
        print("Element found at Position:", i+1)
    else:
        continue
```

Output:

[illegible]

Arrays: 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 (generally denoted by the name of the array). For simplicity, we can think of an array a fleet of stairs where on each step is placed a value (let's say one of your friends). Here, you can identify the location of any of your friends by simply knowing the count of the step they are on. Array can be handled in Python by a module named array. They can be useful when we have to manipulate only a specific data type values. A user can treat lists as arrays. However, user 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.

Creating an Array: Array 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.

Adding Elements to a Array: Elements can be added to the Array by using built-in insert() function. Insert is used to insert one or more data elements into an array. Based on

the requirement, a new element can be added at the beginning, end, or any given index of array. `append()` is also used to add the value mentioned in its arguments at the end of the array.

Accessing elements from the Array: In order to access the array items refer to the index number. Use the index operator `[]` to access an item in a array. The index must be an integer.

Removing Elements from the Array: Elements can be removed from the array by using built-in `remove()` function but an Error arises if element doesn't exist in the set. `Remove()` method only removes one element at a time, to remove range of elements, iterator is used. `pop()` function can also be used to remove and return an element from the array, but by default it removes only the last element of the array, to remove element from a specific position of the array, index of the element is passed as an argument to the `pop()` method. `Remove` method in List will only remove the first occurrence of the searched element.

Slicing of a Array: In Python array, there are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation. Slice operation is performed on array with the use of colon(`:`). To print elements from beginning to a range use `[:Index]`, to print elements from end use `[:-Index]`, to print elements from specific Index till the end use `[Index:]`, to print elements within a range, use `[Start Index:End Index]` and to print whole List with the use of slicing operation, use `[:]`. Further, to print whole array in reverse order, use `[::-1]`.

Searching element in a Array: In order to search an element in the array we use a python in-built `index()` method. This function returns the index of the first occurrence of value mentioned in arguments.

Updating Elements in a Array: In order to update an element in the array we simply reassign a new value to the desired index we want to update.

Code:

```
from array import *
from numpy import *

a = array('i', [10, 20, 30, 40, 50])
print(len(a))
for element in a:
    print(element)
for i in range(len(a)):
    y1 = a[:4]
    y2 = a[0:]

print(y1)
print(y2)

arr1 = array('i', [30, 40, 10, 20, 50, 35])
arr2 = array('u', ['a', 'b', 'c', 'd'])
print('Int array: ')
for element in arr1:
    print(element)
```

```
print('Char array: ')
for element in arr2:
    print(element)

print("From 2-4: ", arr1[2:4])
sum = 0
for element in arr1:
    sum += element
print("Sum of elements in arr1 is: ", sum)
print("Percentage is: ", round(sum/len(arr1), 3))

print("Array before sorting: ", arr1)

def bubbleSort(arr, len):
    for i in range(len - 1):
        for j in range(0, len-i-1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

bubbleSort(arr1, len(arr1))
print("Array after sorting: ", arr1)

def search(arr, elem):
    index = -1
    for element in arr1:
        if (element == elem):
            index = arr1.index(elem)
            break
    return index

elem = int(input("Element to be found in array: "))
pos = search(arr1, elem)
if pos == -1:
    print("Element not found!")
else:
    print("Element found at position", (pos+1))

print(arr1)
arr1.append(100)
print(arr1)
arr1.insert(200, 3)
print(arr1)
arr1.remove(30)
```

```
print(arr1)
p = arr1.pop()
print(arr1)
print("Removed elem: ", p)
q = arr1.index(40)
print("Index of 40 is: ", q)
```

Output:

```
(base) PS C:\Users\tanay\OneDrive\Desktop\study\college\sem-5\python> python pract_5_array.py
5
10
20
30
40
50
[10, 20, 30, 40]
[10, 20, 30, 40, 50]
Int array:
30
40
10
20
50
35
Char array:
a
b
c
d
From 2-4: [10, 20]
Sum of elements in arr1 is: 185
Percentage is: 30.833
Array before sorting: [30, 40, 10, 20, 50, 35]
Array after sorting: [10, 20, 30, 35, 40, 50]
Element to be found in array: 10
Element found at position 1
[10, 20, 30, 35, 40, 50]
[10, 20, 30, 35, 40, 50, 100]
[10, 20, 30, 35, 40, 50, 100, 3]
[10, 20, 35, 40, 50, 100, 3]
[10, 20, 35, 40, 50, 100]
Removed elem: 3
Index of 40 is: 3
```

String Methods: The following are the inbuilt methods in strings.

- **lower():** Converts all uppercase characters in a string into lowercase
- **upper():** Converts all lowercase characters in a string into uppercase
- **title():** Convert string to title case
- **replace():** Replaces a specified phrase with another specified phrase.
- **casefold():** Returns a string where all the characters are lower case. This method is similar to the lower() method, but the casefold() method is stronger, more aggressive, meaning that it will convert more characters into lower case, and will find more matches when comparing two strings and both are converted using the casefold() method.

Code:

```
text = "Python is widely used, python is an interpreted, object-  
oriented, and high-level programming language ß"  
print(text)  
print(text.lower())  
print(text.casefold())  
print(text.upper())  
print(len(text))  
print(text.lower().count("an", 25, 35))  
print(text.find("an"))  
print(text.replace("Python", "PYTHON"))  
print(text.replace("Python", "PYTHON", 1))  
  
dct = {"1": "A", "2": "B"}  
X="@".join(dct)  
print(X)  
  
list = ["X", "Y"]  
Y="#".join(list)  
print(Y)
```

Output:

```
(base) PS C:\Users\tanay.gandhi\Desktop\study\college\sem-5\python> python pract-4_string.py  
Python is widely used, python is an interpreted, object-oriented, and high-level programming language ß  
python is widely used, python is an interpreted, object-oriented, and high-level programming language ß  
python is widely used, python is an interpreted, object-oriented, and high-level programming language ss  
PYTHON IS WIDELY USED, PYTHON IS AN INTERPRETED, OBJECT-ORIENTED, AND HIGH-LEVEL PROGRAMMING LANGUAGE SS  
103  
1  
33  
PYTHON is widely used, python is an interpreted, object-oriented, and high-level programming language ß  
PYTHON is widely used, python is an interpreted, object-oriented, and high-level programming language ß  
1@2  
X#Y
```

Conclusion: Hence, with the help of the above experiment, we learnt about the basics of python including datatypes, strings, list, tuples, sets, dictionaries, arrays etc. We also learnt inbuilt methods for the datatypes and implemented them in python.