INTRODUCTION TO DATA TYPES

- STRINGS
- LIST
- TUPLES
- SETS
- DICTIONARY

STRINGS

In Python, 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.

Creating a String

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

String in single quotes cannot hold any other single quoted character in it otherwise an error arises because the compiler won't recognize where to start and end the string. To overcome this error, use of double quotes is preferred, because it helps in creation of Strings with single quotes in them. For strings which contain Double quoted words in them, use of triple quotes is suggested. Along with this, triple quotes also allow the creation of multiline strings.

```
# Python Program for

# Creation of String


# Creating a String

# with single Quotes

String1 = 'Welcome to the Geeks World'
```

```
print("String with the use of Single Quotes: ")
print(String1)

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

# Creating a String
# with triple Quotes
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Geeks
                For
                Life'''
print("\nCreating a multiline String: ")
print(String1)
```

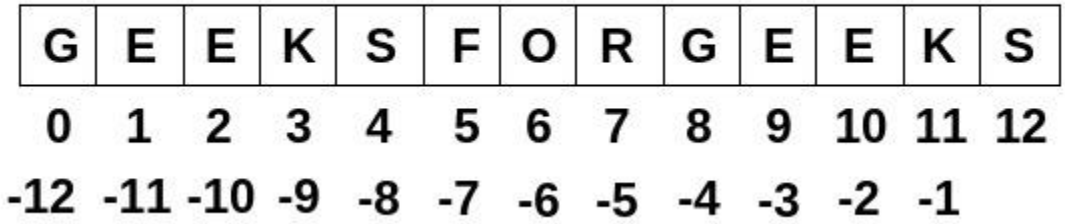
Output:

```
String with the use of Single Quotes:
```

```
Welcome to the Geeks World

String with the use of Double Quotes:
I'm a Geek

String with the use of Triple Quotes:
I'm a Geek and I live in a world of "Geeks"

Creating a multiline String:
Geeks
        For
        Life
```

Accessing characters in python

In Python, individual characters of a String can be accessed by using the method of Indexing, to access a range of characters in the String, method of slicing is used. Slicing in a String is done by using a Slicing operator (colon). 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. While accessing an index out of the range will cause an Index Error. Only Integers are allowed to be passed as an index, float or other types will cause a Type Error.




```
# Python Program to Access
# characters of String

String1 = "GeeksForGeeks"
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])

# Printing 3rd to 12th character
print("\nSlicing characters from 3-12: ")
print(String1[3:12])
```

```
# Printing characters between
# 3rd and 2nd last character
print("\nSlicing characters between " +
      "3rd and 2nd last character: ")
print(String1[3:-2])
```

Output:

```
Initial String:
GeeksForGeeks

First character of String is:
G

Last character of String is:
s

Slicing characters from 3-12:
ksForGeek

Slicing characters between 3rd and 2nd last character:
ksForGee
```

LISTS

Lists are just like the arrays, declared in other languages. Lists need not be homogeneous always which makes it a most powerful tool in Python. A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are also very

useful for implementing stacks and queues. Lists are mutable, and hence, they can be altered even after their creation.
In Python, list is a type of container in Data Structures, which is used to store multiple data at the same time. Unlike Sets, the list in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

Note- Lists are a useful tool for preserving a sequence of data and further iterating over it.
Creating a List
Lists in Python can be created by just placing the sequence inside the square brackets[]. Unlike Sets, list doesn't need a built-in function for creation of list. A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

```
# Python program to demonstrate

# Creation of List


# Creating a List

List = []

print("Intial blank List: ")

print(List)


# Creating a List with

# the use of a String

List = ['GeeksForGeeks']
```

```
print("\nList with the use of String: ")
print(List)

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

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

# Creating a List with
# the use of Numbers
# (Having duplicate values)
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
```

```
print(List)


# Creating a List with

# mixed type of values

# (Having numbers and strings)

List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']

print("\nList with the use of Mixed Values: ")

print(List)
```

Output:

```
Intial blank List:
[]

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

List containing multiple values:
Geeks
Geeks

Multi-Dimensional List:
[['Geeks', 'For'], ['Geeks']]

List with the use of Numbers:
[1, 2, 4, 4, 3, 3, 3, 6, 5]
```

List with the use of Mixed Values:

[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']

TUPLES

Tuple is a collection of Python objects much like a list. The sequence of values stored in a tuple can be of any type, and they are indexed by integers. The important difference between a list and a tuple is that tuples are immutable. Also, Tuples are hashable whereas lists are not.
Values of a tuple are syntactically separated by 'commas'. Although it is not necessary, it is more common to define a tuple by closing the sequence of values in parentheses. This helps in understanding the Python tuples more easily.
Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

Creating a Tuple

In Python, tuples are created by placing sequence of values separated by 'comma' with or without the use of parentheses for grouping of data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.). Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Note – Creation of Python tuple without the use of parentheses is known as Tuple Packing.

Python program to demonstrate

Addition of elements in a Set

Creating an empty tuple

Tuple1 = ()

```
print("Initial empty Tuple: ")

print (Tuple1)


# Creating a Tuple with

# the use of Strings

Tuple1 = ('Geeks', 'For')

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 loop

Tuple1 = ('Geeks')

n = 5

print("\nTuple with a loop")

for i in range(int(n)):
```

```
    Tuple1 = (Tuple1,)
    print(Tuple1)


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


# Creating a Tuple with
# Mixed Datatypes
Tuple1 = (5, 'Welcome', 7, 'Geeks')
print("\nTuple with Mixed Datatypes: ")
print(Tuple1)


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

```
print(Tuple3)


# Creating a Tuple

# with repetition

Tuple1 = ('Geeks',) * 3

print("\nTuple with repetition: ")

print(Tuple1)
```

Output:

```
Initial empty Tuple:
()

Tuple with the use of String:
('Geeks', 'For')

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

Tuple with a loop
('Geeks',)
(('Geeks',),)
((('Geeks',),),)
(((('Geeks',),),),)
(((((('Geeks',),),),),),)
```

```
Tuple with the use of function:
('G', 'e', 'e', 'k', 's')

Tuple with Mixed Datatypes:
(5, 'Welcome', 7, 'Geeks')

Tuple with nested tuples:
((0, 1, 2, 3), ('python', 'geek'))

Tuple with repetition:
('Geeks', 'Geeks', 'Geeks')
```

SETS

In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements.
Set in Python is equivalent to sets in mathematics. The order of elements in a set is undefined though it may consist of various elements. Elements of a set can be added and deleted, elements of the set can be iterated, various standard operations (union, intersection, difference) can be performed on sets. Besides that, the major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Creating a Set

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’. A set contains only unique elements but at the time of set creation, multiple duplicate values can also be passed. Order of elements in a set is undefined and is unchangeable. Type of elements in a set need not be the same, various mixed up data type values can also be passed to the set.

Note – A set cannot have mutable elements like a list, set or dictionary, as its elements.

```
# Python program to demonstrate

# Creation of Set in Python


# Creating a Set

set1 = set()

print("Intial blank Set: ")

print(set1)


# Creating a Set with

# the use of a String

set1 = set("GeeksForGeeks")

print("\nSet with the use of String: ")

print(set1)


# Creating a Set with

# the use of Constructor

# (Using object to Store String)

String = 'GeeksForGeeks'

set1 = set(String)
```

```
print("\nSet with the use of an Object: " )

print(set1)


# Creating a Set with

# the use of a List

set1 = set(["Geeks", "For", "Geeks"])

print("\nSet with the use of List: ")

print(set1)


# Creating a Set with

# a List of Numbers

# (Having duplicate values)

set1 = set([1, 2, 4, 4, 3, 3, 3, 6, 5])

print("\nSet with the use of Numbers: ")

print(set1)


# Creating a Set with

# a mixed type of values

# (Having numbers and strings)

set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])

print("\nSet with the use of Mixed Values")
```

```
print(set1)
```

Output:

```
Intial blank Set:
set()

Set with the use of String:
{'e', 'r', 'k', 'o', 'G', 's', 'F'}

Set with the use of an Object:
{'r', 'o', 'e', 'F', 's', 'k', 'G'}

Set with the use of List:
{'Geeks', 'For'}

Set with the use of Numbers:
{1, 2, 3, 4, 5, 6}

Set with the use of Mixed Values
{1, 2, 4, 'Geeks', 6, 'For'}
```

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’.

A Dictionary in Python works similar to the Dictionary in a real world. Keys of a Dictionary must be unique and of *immutable* data type such as Strings, Integers and tuples, but the key-values can be repeated and be of any type.
Note – Keys in a dictionary doesn’t allows Polymorphism.

Creating a Dictionary

In Python, a Dictionary can be created by placing sequence of elements within curly **{}**braces, separated by ‘comma’. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its **Key:value**. 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 to curly braces{}.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

```
# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)



# Creating a Dictionary

# with Integer Keys

Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

```
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary
# with Mixed keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)

# Creating a Dictionary
# with dict() method
Dict = dict({1: 'Geeks', 2: 'For', 3:'Geeks'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary
# with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

Output:

```
Empty Dictionary:
{}

Dictionary with the use of Integer Keys:
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

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

Dictionary with the use of dict():
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with each item as a pair:
{1: 'Geeks', 2: 'For'}
```