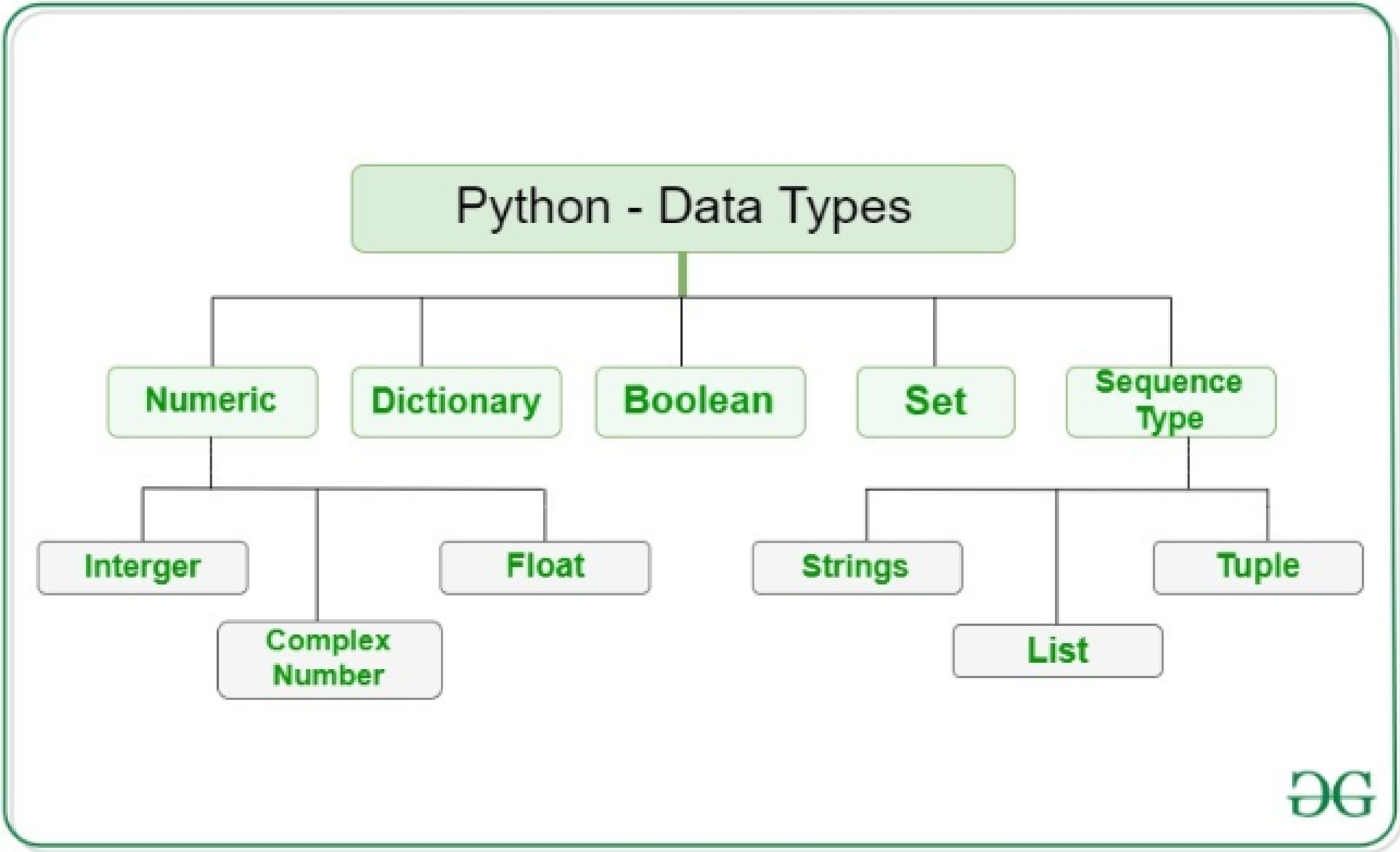
Python

Python Data Types

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


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Interger
Complex Number
Float
Strings
List
Tuple

Numeric

In Python, numeric data type represent 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.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Note – `type()` function is used to determine the type of data type.

Python3

```python
# Python program to
# demonstrate numeric value

a = 5
print("Type of a: ", type(a))

b = 5.0
print("\nType of b: ", type(b))

c = 2 + 4j
print("\nType of c: ", type(c))
```

Output:

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

Type of b:  <class 'float'>

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

Sequence Type

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

- String
- List
- Tuple

1) 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.

```
# 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 Qu
print(String1)

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

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

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

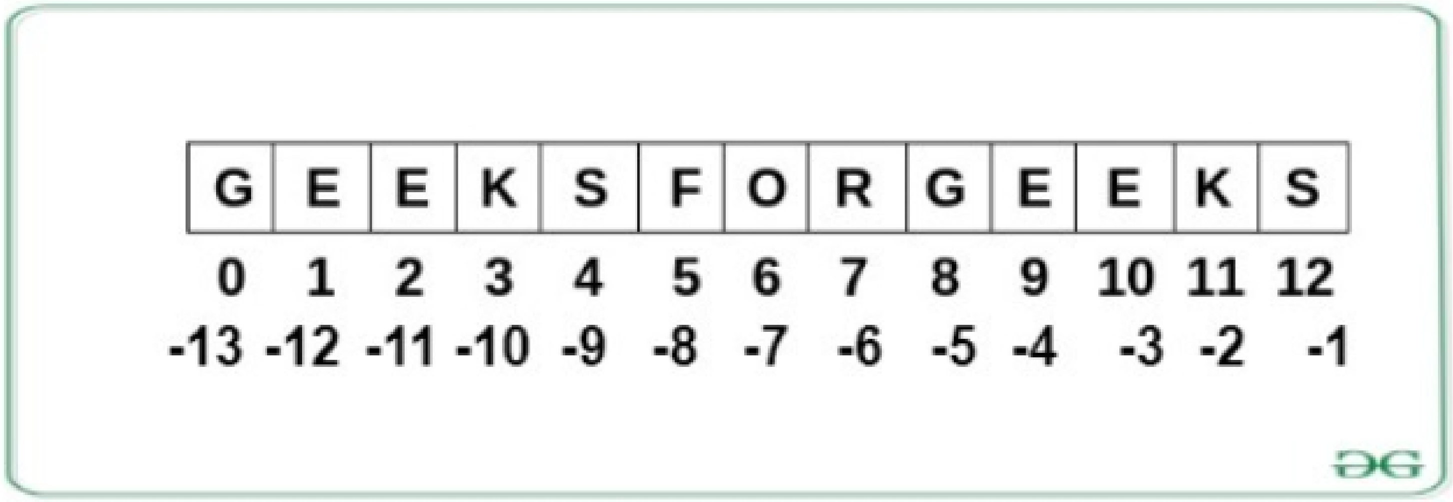
```
# Python Program for
# Creation of String

# Creating a String
# with single Quotes
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Qu
print(String1)

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

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

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

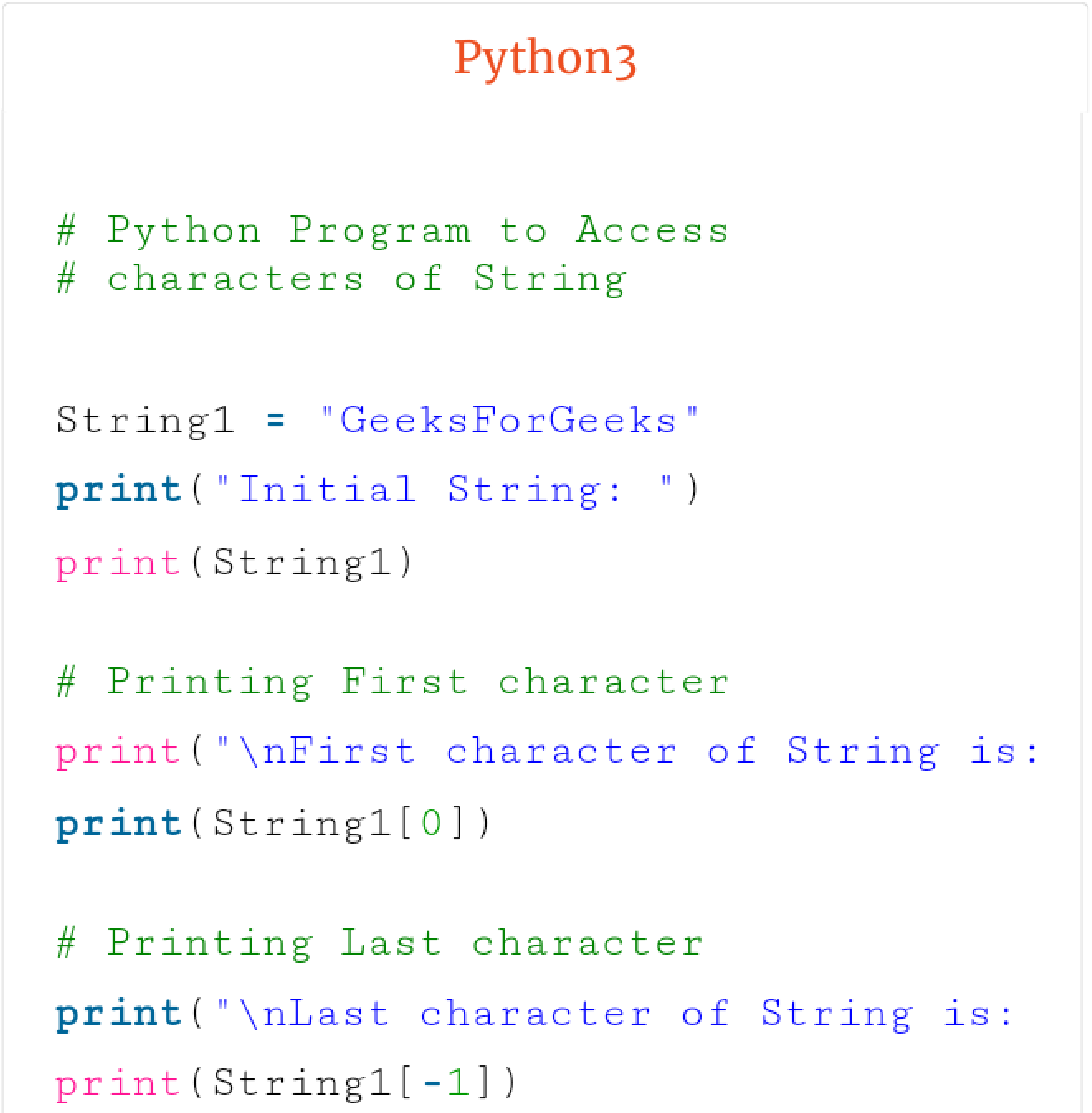
Output:

```
String with the use of Single Quotes:

Welcome to the Geeks World



String with the use of Double Quotes:

I'm a Geek

<class 'str'>



String with the use of Triple Quotes:

I'm a Geek and I live in a world of "Geeks"

<class 'str'>



Creating a multiline String:

Geeks

            For

            Life
```

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.





Python3

```python
# 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])
```

Output:

```
Initial String:

GeeksForGeeks



First character of String is:

G



Last character of String is:

s
```

2) 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.

```
# 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 = ['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 valu
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)
```

Output:

```
Initial blank List:

[]



List with the use of String:

['GeeksForGeeks']



List containing multiple values:

Geeks

Geeks



Multi-Dimensional List:

[['Geeks', 'For'], ['Geeks']]
```

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.

```python
# Python program to demonstrate
# accessing of element from list

# Creating a List with
# the use of multiple values
List = ["Geeks", "For", "Geeks"]

# accessing a element from the
# list using index number
print("Accessing element from the list
print(List[0])
print(List[2])

# accessing a element using
# negative indexing
print("Accessing element using negativ

# print the last element of list
print(List[-1])

# print the third last element of list
print(List[-3])
```

Output:

```
Accessing element from the list

Geeks

Geeks

Accessing element using negative indexing

Geeks

Geeks
```

3) 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.).

Note: 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.

Python3

```
# 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 = ('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 built-in function
Tuple1 = tuple('Geeks')
print("\nTuple with the use of functio
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)
```

Output:

```
Initial empty Tuple:
()


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


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


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


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

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

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.

Python3

```python
# 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])
```

Output:

```
First element of tuple

1



Last element of tuple

5



Third last element of tuple

3
```

Boolean

Data type with one of the two built-in values, `True` or `False`. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class `bool`.

Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

```python
# Python program to
# demonstrate boolean type


print(type(True))

print(type(False))


print(type(true))
```

Output:

```
<class 'bool'>

<class 'bool'>
```

```
Traceback (most recent call last):

  File "/home/7e8862763fb66153d70824099d4f5fb7.py", line 8, in

    print(type(true))

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

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.

```python
# 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("GeeksForGeeks")
print("\nSet with the use of String: ")
print(set1)
```

```python
# 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 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:

```
Initial blank Set:

set()


Set with the use of String:

{'F', 'o', 'G', 's', 'r', 'k', 'e'}


Set with the use of List:

{'Geeks', 'For'}


Set with the use of Mixed Values

{1, 2, 4, 6, 'Geeks', 'For'}
```

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.

Python3

```python
# Python program to demonstrate
# Accessing of elements in a set

# Creating a set
set1 = set(["Geeks", "For", "Geeks"])
print("\nInitial set")
print(set1)

# Accessing element using
# for loop
print("\nElements of set: ")
for i in set1:
    print(i, end =" ")

# Checking the element
# using in keyword
print("Geeks" in set1)
```

Output:

```
Initial set:
{'Geeks', 'For'}


Elements of set:
Geeks For


True
```

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{}.

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

```python
# 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)
```

```python
# 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'}
```

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.

```python
# Python program to demonstrate
# accessing a element from a Dictionary

# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])

# accessing a element using get()
# method
print("Accessing a element using get:")
print(Dict.get(3))
```

Output:

```
Accessing a element using key:

For

Accessing a element using get:

Geeks
```