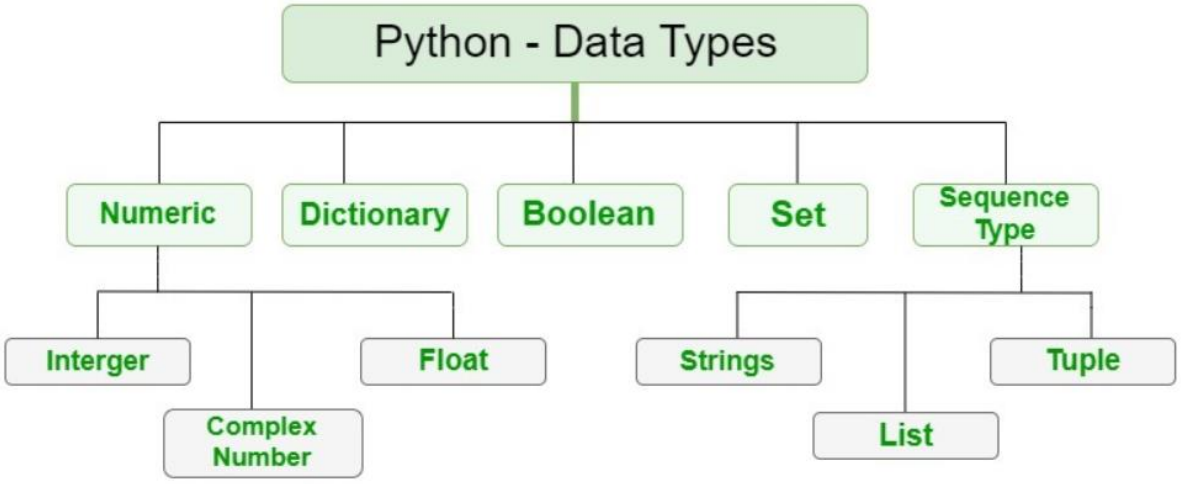
Python Data Types

Python 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, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



1. Numeric Data Types in Python

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int , Python float , and Python complex classes in Python .

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a 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 – A complex number is represented by a 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 Python data type.

Example: This code demonstrates how to determine the data type of variables in Python using the type() function . It prints the data types of three variables : a (integer) , b (float) , and c (complex) . The output shows the respective data type Python for each variable.

```
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'>
```

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

- Python String
- Python List
- Python Tuple

String Data Type

Strings in Python 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 Python, 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, double quotes, or even triple quotes.

Example: This Python code showcases various string creation methods. It uses single quotes, double quotes, and triple quotes to create strings with different content and includes a multiline string. The code also demonstrates printing the strings and checking their data types.

```
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)
String1 = "I'm a Geek"
print("\nString with the use of Double Quotes: ")
print(String1)
print(type(String1))
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes: ")
print(String1)
print(type(String1))

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
```

List Data Type

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

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

Example: This Python code demonstrates list creation and manipulation. It starts with an empty list and prints it. It creates a list containing a single string element and prints it. It creates a list with multiple string elements and prints selected elements from the list. It creates a multi-dimensional list (a list of lists) and prints it. The code showcases various ways to work with lists, including single and multi-dimensional lists.

```
List = []
print("Initial blank List: ")
print(List)
List = ['GeeksForGeeks']
print("\nList with the use of String: ")
print(List)
List = ["Geeks", "For", "Geeks"]
print("\nList containing multiple values: ")
print(List[0])
print(List[2])
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']]
```

Tuple Data Type

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

Creating a Tuple in Python

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

Example: This Python code demonstrates different methods of creating and working with tuples. It starts with an empty tuple and prints it. It creates a tuple containing string elements and prints it. It converts a list into a tuple and prints the result. It creates a tuple from a string using the tuple() function. It forms a tuple with nested tuples and displays the result.

```
Tuple1 = ()
print("Initial empty Tuple: ")
print(Tuple1)
Tuple1 = ('Geeks', 'For')
print("\nTuple with the use of String: ")
print(Tuple1)
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))
Tuple1 = tuple('Geeks')
print("\nTuple with the use of function: ")
print(Tuple1)
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'))
```

3. Boolean Data Type in Python

Python 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). However non-Boolean objects can be evaluated in a 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.

Example: The first two lines will print the type of the boolean values True and False, which is <class 'bool'>. The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters. You need to capitalize the first letter of true to make it a boolean value.

```
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
```

4. Set Data Type in Python

In Python Data Types, a Set is an unordered collection of data types 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.

Create a Set in Python

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 a ‘comma’. The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Example: The code is an example of how to create sets using different types of values, such as strings , lists , and mixed values

```
set1 = set()
print("Initial blank Set: ")
print(set1)
set1 = set("GeeksForGeeks")
print("\nSet with the use of String: ")
print(set1)
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List: ")
print(set1)
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'}
```


5. Dictionary Data Type in Python

A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a 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’.

Create a Dictionary in Python

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. The dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it in curly braces{}. Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

Example: This code creates and prints a variety of dictionaries. The first dictionary is empty. The second dictionary has integer keys and string values. The third dictionary has mixed keys, with one string key and one integer key. The fourth dictionary is created using the dict() function, and the fifth dictionary is created using the [(key, value)] syntax

```
Dict = {}
print("Empty Dictionary: ")
print(Dict)
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer Keys: ")
```

Output:

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