MARRI LAXMAN REDDY
GROUP OF INSTITUTIONS

Python Programming

By Para Upendar
Assistant Professor
Department : CSC

Python Data Types

Variables can hold values, and every value has a data-type. Python is a dynamically typed language;

hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

a = 5

The variable a holds integer value five and we did not define its type. Python interpreter will automatically interpret variables a as an integer type.

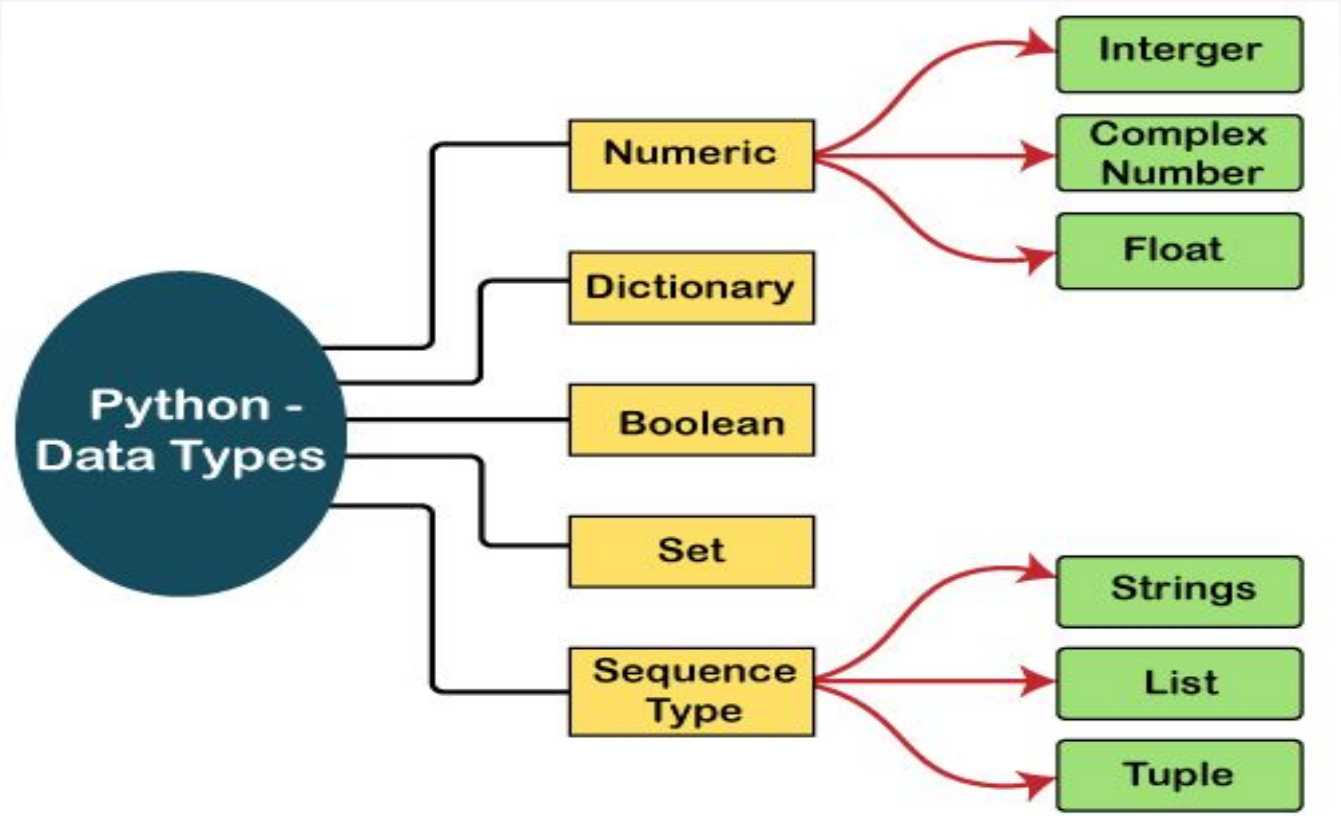
Python enables us to check the type of the variable used in the program. Python provides us the type() function,which returns the type of the variable passed.

```
a=10
b="Hi Python"
c = 10.5
print(type(a))
print(type(b))
print(type(c))
```

<type 'int'>
<type 'str'>
<type 'float'>

The data types defined in Python are given below.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary


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

Numbers

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type.

```
a = 5
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))
print(" c is a complex number",
isinstance(1+3j,complex))
```

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

Int - Integer value can be any length such as integers 2, 29, -20, -150 etc. no restriction on the length of an integer.

Float - Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

complex - A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Sequence Type
String

The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

Python provides built-in functions and operators to perform operations in the string.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+" python" returns "hello python".

The operator * is known as a repetition operator as the operation "Python" *2 returns 'Python Python'.

Example - 1

```
str = "string using double quotes"
print(str)
s = '''A multiline
string'''
print(s)
```

Example - 2

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

List

Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

```
list1  = [1, "hi", "Python", 2]
#Checking type of given list
print(type(list1))
#Printing the list1
print (list1)
# List slicing
print (list1[3:])
# List slicing
print (list1[0:2])
# List Concatenation using + operator
print (list1 + list1)
# List repetation using * operator
print (list1 * 3)
```

<class,'list'>
[1, 'hi', 'Python', 2]
[2]
[1, 'hi']
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]

Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

```
tup  = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))
#Printing the tuple
print (tup)
# Tuple slicing
print (tup[1:])
print (tup[0:1])
# Tuple concatenation using + operator
print (tup + tup)
# Tuple repatation using * operator
print (tup * 3)
# Adding value to tup. It will throw an error.
t[2] = "hi"
```

```
<class 'tuple'>
('hi', 'Python', 2)
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)

Traceback (most recent call last):
  File "main.py", line 14, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item
assignment
```

Dictionary

Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

# Printing dictionary
print (d)

# Accesing value using keys
print("1st name is "+d[1])
print("2nd name is "+ d[4])

print (d.keys())
print (d.values())
```

```
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
1st name is Jimmy
2nd name is mike
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F

```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false)
```

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

Set

Python Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element. The set is created by using a built-in function set(), or a sequence of elements is passed in the curly braces and separated by the comma. It can contain various types of values.

```
# Creating Empty set
set2 = set()
set2 = {'James', 2, 3,'Python'}
 #Printing Set value
print(set2)
# Adding element to the set
set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

{3, 'Python', 'James', 2}
{'Python', 'James', 3, 2, 10}
{'Python', 'James', 3, 10}


Thank you