Python Data Types

In this class, you will learn about different data types you can use in Python.

Data types in Python

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

1. Python Numbers

Integers, floating point numbers and complex numbers fall under **Python numbers (https://github.com/milaan9/02_Python_Datatypes/blob/main/001_Python_Numbers.ipynb)** category. They are defined as `int` , `float` and `complex` classes in Python.

We can use the `type()` function to know which class a variable or a value belongs to.

Similarly, the `isinstance()` function is used to check if an object belongs to a particular class.

In [1]:
```python
a = 6
print(a, "is of type", type(a))
print(a, "is integer number?", isinstance(5,int))

a = 3.0
print(a, "is of type", type(a))
print(a, "is float number?", isinstance(2.0,float))

a = 1+2j  # '1' is real part and '2j' is imaginary part
print(a, "is of type", type(a))
print(a, "is complex number?", isinstance(1+2j,complex))
```

```
6 is of type <class 'int'>
6 is integer number? True
3.0 is of type <class 'float'>
3.0 is float number? True
(1+2j) is of type <class 'complex'>
(1+2j) is complex number? True
```

Integers can be of any length, it is only limited by the memory available.

A floating-point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. `1` is an integer, `1.0` is a floating-point number.

Complex numbers are written in the form, `x + yj` , where `x` is the real part and `y` is the imaginary part. Here are some examples.

In [2]:
```
a = 1234567890123456789
print (a)

b = 0.1234567890123456789  # total of only 17 numbers after decimal can be printed.
print (b)

c = 1+2j
print (c)
```

```
1234567890123456789
0.12345678901234568
(1+2j)
```

Notice that the `float` variable `b` got truncated.

2. Python List [ ]

[List (https://github.com/milaan9/02_Python_Datatypes/blob/main/003_Python_List.ipynb)](https://github.com/milaan9/02_Python_Datatypes/blob/main/003_Python_List.ipynb) is an **ordered sequence** of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets `[ ]`.

```
>>>a = [1, 3.3, '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.

In [3]:
```
x = [6, 99, 77, 'Apple']
print(x, "is of type", type(x))
```

```
[6, 99, 77, 'Apple'] is of type <class 'list'>
```

In [4]:
```
a = [5, 10, 15, 20, 25, 30, 35, 40]  # Total elemnets is 8
#   [0   1   2   3   4   5   6   7]  ⟵ Index forward
#   [-8 -7  -6  -5  -4  -3  -2  -1]  ⟶ Index backward

# index '0' is element '1' = 5,
# index '1' is element '2' = 10,
# index '2' is element '3' = 15,
# .
# .
# .
# index '7' is element '8' = 40,

a[1] # To access the elements in the list

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])  # [0:3] means elements from 0 uptil 2 index (not include last ele
ment)
                            # [0:3] means from index 0 to 3 - 1
                            # [0:3] means from index 0 to 2

# a[5:] = [30, 35, 40]  # [5:] means all the elements from 5 till end
print("a[5:] = ", a[5:])
```

```
a[2] =  15
a[0:3] =  [5, 10, 15]
a[5:] =  [30, 35, 40]
```

In [5]:
```
a[1:-2]
```

Out[5]:
```
[10, 15, 20, 25, 30]
```

```
In [6]: a[5:9]
Out[6]: [30, 35, 40]
```

```
In [7]: a[:5]
Out[7]: [5, 10, 15, 20, 25]
```

Lists are **mutable**, meaning, the value of elements of a list can be altered.

```
In [8]: # Change the element of the List
        a = [1, 2, 3]
        #   [0  1  2] ➞ Index forward

        a[2] = 4  # Change my third element from '3' to '4' # [2] is the index number
        print(a)

        [1, 2, 4]
```

3. Python Tuple `()`

[Tuple (https://github.com/milaan9/02_Python_Datatypes/blob/main/004_Python_Tuple.ipynb)](https://github.com/milaan9/02_Python_Datatypes/blob/main/004_Python_Tuple.ipynb) is an **ordered sequence** of items same as a list. The only difference is that tuples are **immutable**. Tuples once created cannot be modified.

Tuples are used to **write-protect data** and are usually faster than lists as they cannot change dynamically.

It is defined within parentheses `()` where items are separated by commas.

```
>>>t = (6,'program', 1+3j)
```

We can use the slicing operator `[]` to extract items but we cannot change its value.

```
In [9]: # Tuple 't' have 3 elements
        t = (6,'program', 1+3j)
        #   (0        1      2) ➞ Index forward

        # index '0' is element '1'= 6
        # index '1' is element '2'= program
        # index '2' is elemtnt '3'= 1+3j

        # t[1] = 'program'
        print("t[1] = ", t[1])

        # t[0:3] = (6, 'program', (1+3j))
        print("t[0:3] = ", t[0:3])

        # Generates error
        # Tuples are immutable
        t[0] = 10  # trying to change element 0 from '6' to '10'

        t[1] =  program
        t[0:3] =  (6, 'program', (1+3j))

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        <ipython-input-9-c92d9f2b36de> in <module>
             15 # Generates error
             16 # Tuples are immutable
        ---> 17 t[0] = 10  # trying to change element 0 from '6' to '10'

        TypeError: 'tuple' object does not support item assignment
```

In [10]:
```
list1 =  [9, 'apple', 3 + 6j]  # list
tuple1 = (9, 'apple', 3 + 6j)  # tuple

list1[1] = 'banana'  # List is mutable
print(list1)         # No error
tuple1[1]= 'banana'  # Tuple is immutable
print(tuple1)        # error
```

```
[9, 'banana', (3+6j)]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-e32e417070a1> in <module>
      4 list1[1] = 'banana'  # List is mutable
      5 print(list1)         # No error
----> 6 tuple1[1]= 'banana'  # Tuple is immutable
      7 print(tuple1)        # error

TypeError: 'tuple' object does not support item assignment
```

4. Python Strings

[String (https://github.com/milaan9/02_Python_Datatypes/blob/main/002_Python_String.ipynb)](https://github.com/milaan9/02_Python_Datatypes/blob/main/002_Python_String.ipynb) is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, `'''` or `"""` .

In [11]:
```
s = '''Apple'''
print(s)
s = """Apple"""
print(s)
s = 'Apple'
print(s)
s = "Apple"
print(s)
s = Apple   # cannot write string with out quotes ('', " ", """ """, ''' ''')
print(s)
```

```
Apple
Apple
Apple
Apple

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-7fce570bf337> in <module>
      7 s = "Apple"
      8 print(s)
----> 9 s = Apple   # cannot write string with out quotes ('', " ", """ """, ''' ''')
     10 print(s)

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

In [12]:
```
s = "This is a string"  # s is my variable
print(s)
s = '''A multiline
string'''
print(s)
```

```
This is a string
A multiline
string
```

Just like a list and tuple, the slicing operator `[ ]` can be used with strings. Strings, however, are **immutable**.

In [13]:
```python
s = 'Hello world!' # total 12 elements. Index start from '0' to '11'

# s[4] = 'o'
print("s[4] = ", s[4])

# s[6:11] = 'world' # index '6' to '11' means element from 6 to 10
print("s[6:11] = ", s[6:11])
```

```
s[4] =  o
s[6:11] =  world
```

In [14]:
```python
a = "apple"

a[0]='o'

# Simiar to TUPLE, STRING is immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-466b90e7ef2f> in <module>
      1 a = "apple"
      2 
----> 3 a[0]='o'
      4 
      5 # Simiar to TUPLE, STRING is immutable

TypeError: 'str' object does not support item assignment
```

5. Python Set `{}`

Set (https://github.com/milaan9/02_Python_Datatypes/blob/main/006_Python_Sets.ipynb) is an **unordered collection** of unique items. Set is defined by values separated by comma inside braces `{ }`. Items in a set are not ordered.

In [15]:
```python
a = {7,1,3,6,9}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```

```
a =  {1, 3, 6, 7, 9}
<class 'set'>
```

We can perform set operations like union, intersection on two sets. Sets have unique values. They eliminate duplicates.

In [16]:
```python
a = {1,2,2,3,3,3} # we can see total 6 elements
print(a)
```

```
{1, 2, 3}
```

Since, set are unordered collection, indexing has no meaning. Hence, the slicing operator `[]` does not work.

```
In [17]: a = {1,2,3}  # in Set data type we cannot access the elements because set is unordered collec
         tion
         a[1]  # Index [1] means element 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-242b77ef2a87> in <module>
      1 a = {1,2,3}  # in Set data type we cannot access the elements because set is unordere
d collection
----> 2 a[1]  # Index [1] means element 2

TypeError: 'set' object is not subscriptable
```

6. Python Dictionary `{}`

Dictionary (https://github.com/milaan9/02_Python_Datatypes/blob/main/005_Python_Dictionary.ipynb) is an **unordered collection** of **key-value pairs**.

It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces `{}` with each item being a pair in the form key:value. Key and value can be of any type.

```
In [18]: d = {1: 'Apple', 2: 'Cat', 3: 'Food'}  # 'Apple' is element and 1 is the key of element.
         print(d, type(d))

         d[3]
```

```
{1: 'Apple', 2: 'Cat', 3: 'Food'} <class 'dict'>
```

```
Out[18]: 'Food'
```

```
In [19]: d = {1:'value','key':2} # d is my variable, 'value' and 'key' are the element and 1 and 2 are
         the key.
         type(d)
```

```
Out[19]: dict
```

We use key to retrieve the respective value. But not the other way around.

```
In [20]: d = {1:'value','key':2} # '1' is the key to access 'value' and 'key' is the key to access '2'
         print(type(d))

         print("d[1] = ", d[1]); # try to find the element from key.

         print("d['key'] = ", d['key']);  # try to find the key from the element.
```

```
<class 'dict'>
d[1] =  value
d['key'] =  2
```

```
In [21]: print(type(zip([1,2],[3,4])))
```

```
<class 'zip'>
```

```
In [ ]:
```