Getting started with Python-

Using Indexing With Data Objects

Objects and Zero Based Indexing

Python has 5 standard data types

Python Data Types

Numbers	Numeric values: •• integers (positive or negative whole numbers) : -2, 44, 4645, -245 •• floating point numbers : 5.0, 22.2, 1.4556, -55555.675
Strings	Sequence of characters represented in either pairs of single or double quotes.
Lists	Sequence of elements which do not need to be of same data type. Elements and size can be updated.
Tuples	Sequence data type similar to the list. Unlike lists, tuples are enclosed within parentheses (( )) and cannot be updated.
Dictionary	Unordered collection of key-value pairs. Dictionaries are defined within curly braces ({ }). Each KEY must be unique, but the VALUE may be the same for two or more keys.

* Note : Use type() function to know the data type of the object.

Numbers

Both integers and Floating point numbers are supported by Python and are defined as **`int`** and **`float`**.

```
x=10
x
```

```
10
```

```
type(x)
```

```
<type 'int'>
```

```
y= 15.20
type(y)
```

```
<type 'float'>
```

```
z= -32.54e100
type(z)
```

```
<type 'float'>
```

Numbers

- To convert **float** to **int**, you need to use **int()** function.

```
x=int(-99999.675)
x
```

```
-99999
```

- To c

```
type(x)
```

```
<type 'int'>
```

```
y=float(25)
y
```

```
25.0
```

```
type(y)
```

```
<type 'float'>
```

Strings

- Python strings are sequence (left-to-right order) of characters enclosed in single or double quotes.
- Python strings are immutable i.e. they cannot be modified but we can run expressions to make new string objects.

```
x='welcome to the Python world'
x
'welcome to the Python world'
z='3948'
z
'3948'
type(z)
<class 'str'>
x = x + ", John"
x
'welcome to the Python worl
```

String outputs are displayed in quotes

Using '+' operator you can concatenate two or more strings to make a new string object

Strings

Accessing elements of the String:

- Use [ ] brackets with expression as index or indices.

```
x[0]
'w'
x[2:5]
'lco'
```

access multiple elements by giving a range of their index

- To check the length of the string, use **`len()`** function.

```
# length of the string object ‘x’
len(x)
33
```

- Python methods for handling and processing strings are covered later.

Strings

- To convert a value to a string type, you need to use str() function.

```
a=1567
a
1567
y=str(a)
type(y)
<type 'str'>
```

Any value can be converted into string with the help of **str()** function.

Lists

- List can store elements of different data types.
- List elements are separated by commas (,) and are enclosed within square brackets ([ ]).
- Lists are mutable unlike strings.

```
list1=['python', 1998, 'list', 12]
list1
['python', 1998, 'list', 12]
len(list1)
4
list2=[2001,2005,2010,2016]
list2
[2001, 2005, 2010, 2016]
list3=["red","blue", "white", "black"]
list3
['red', 'blue', 'white', 'black']
```

list1 has elements of number and string type

len() length of the list1

Lists

Accessing elements of the List:

- Use [ ] brackets with expression as index or indices.

```
list1[0]
'python'
list2[1:4]
[2005, 2010, 2016]
```

Updating Lists:

```
# Change the value of 3rd element of list2 to '2006'
list2[2]=2006
list2
[2001, 2005, 2006, 2016]

# Add an element '2012' to list2
list2.append(2012)
list2
[2001, 2005, 2006, 2016, 2012]
```

To change the value, you need to specify the index no. of the element enclosed in **[] brackets**.

append() function modifies the original list by adding a single element to the end of the list.

Lists

Deleting List elements:

- To remove a list element if you know exactly which element(s) to be deleted, use **`del`** operator.

```
# Delete 2nd element of list2
del list2[1]
list2
```

```
[2001, 2006, 2016, 2012]
```

In case you do not know which element to delete use **`remove()`** method

```
# Delete ‘2012’ from list2
list2.remove(2012)
list2
```

```
[2001, 2006, 2016]
```

Lists

Some other common List functions:

```
# Insert new element at the given index
list2.insert(1,2019)
list2
```

```
[2001, 2019, 2006, 2016]
```

insert() can insert an item at given position. Here **1** is the position **2019** is the item

```
# Search index of an element from the list
list2.index(2016)
```

```
3
```

```
# Sort the list in ascending order.
list2.sort()
list2
```

For descending order specify **reverse=True**

```
[2001, 2006, 2016, 2019]
```

```
# Reverse the list in place
list2.reverse()
list2
```

```
[2019, 2016, 2006, 2001]
```

Tuples

- Lists are mutable and tuples are immutable.
- The main difference between mutable and immutable is memory
- Tuples are useful when you know that you are not frequently adding new elements.
- Tuples are heterogeneous data structures

Tuples

Creating a Tuple:

```
# Create a tuple
tuple1=('math','physics', 'chemistry')
tuple1
('math', 'physics', 'chemistry')
```

Accessing elements of the Tuple:

```
tuple1[0]
'math'
tuple1[0]="mathematics"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not suppor
```

Use [] brackets with expression as index or indices.

An exception is raised because modifying tuple is not allowed.

Tuples

Updating Tuples:

- Tuples are immutable that means there elements cannot be changed. However, you can take portions of existing tuples to create new tuples.

```
tup1=('one', 'two', 'three')
tup2=(11,22,44.5)
tup3=tup1+tup2
tup3
('one', 'two', 'three', 11, 22, 44.5)
```

New tuple ‘tup3’ is created containing elements of ‘tup2’ and ‘tup3’.

Tuples

Deleting Tuple elements:

- Individual elements cannot be removed from a tuple. To explicitly remove an entire tuple, use **`del`** command.

```
del tuple1
tuple1
```

```
Traceback (most recent call last):

  File "<ipython-input-7-97b79749fdf6>", line 1, in <module>
    tuple1

NameError: name 'tuple1' is not defined
```

An exception is raised because tuple1 doesn't exists any more.

Dictionary

- A dict (dictionary) type object can store a collections of elements just like a list, but the elements are in the form of key-value pairs and to retrieve the values, you can use keys. This way a dict can be treated like a database for storing and organising data
- Key-Value pairs are separated by a colon (:) and pairs themselves by commas (,) and all this is defined in a pair of curly braces ({ }).
- The key-value pairs in dictionary objects are not ordered in any manner.
- Dictionary keys are Case Sensitive.

Dictionary

Creating a Dictionary:

```
# Create a Dictionary
Dict1={'Name': 'Ruchi', 'Age': '18', 'Class': 'Twelfth'}
Dict1
```

```
{'Name': 'Ruchi', 'Age': '18', 'Class': 'Twelfth'}
```

Accessing values of the Dictionary:

- Use [ ] brackets along with the key as index to obtain its value.

```
Dict1['Name']
```

```
'Ruchi'
```

*

For the keys of the dictionary you can use only immutable objects but for the values you can use either mutable or immutable objects. Dictionaries can be nested with other dictionaries.

Dictionary

Updating Dictionary:

- Dictionaries are mutable; you can add or modify elements. Just the way you access values using keys, you can modify values using keys.

```
# Change the value of ‘Age’ to ‘19’
Dict1['Age']='19'
Dict1
```

```
{'Name': 'Ruchi', 'Age': '19', 'Class': 'Twelfth'}
```

```
# Adding a new element
Dict1['School']='Kendriya Vidyalaya'
Dict1
```

```
{'Name': 'Ruchi',
 'Age': '19',
 'Class': 'Twelfth',
 'School': 'Kendriya Vidyalaya'}
```

Dictionary

Deleting Dictionary elements:

- You can delete the individual key-value pairs from the dictionary using **del** command

```
del Dict1['Name']
Dict1
```

```
{'Age': '19', 'Class': 'Twelfth', 'School': 'Kendriya
Vidyalaya'}
```

- Delete all key-value pairs at once using **clear()** function.

```
Dict1.clear()
Dict1
```

```
{}
```

Dictionary

- Delete entire dictionary using **del** command.

```
del Dict1
Dict1
```

```
Traceback (most recent call last):
  File "<ipython-input-24-6dd8ab4d7b02>",
    Dict1
NameError: name 'Dict1' is not defined
```

An exception is raised because Dict1 doesn't exists anymore.

Quick Recap

5 Standard Native Data Types: Numbers, Strings, Lists, tuples and Dictionary

Mutable Data Types	Lists and Dictionaries
Immutable Data Types	Strings and Tuples
Accessing	Use [ ] brackets along with the expression as index for Strings, Lists & Tuples and key for Dictionary
Updating	Specify index no. (for lists) and key (for Dictionary) between [ ] brackets and assign a new value
Deleting	Use **`del`** command for Strings, Lists, Tuples and Dictionaries