Dictionaries, List and Tuples:

Concept of dictionary, list and tuple: accessing, updating, deletion and basic operations of dictionary, list and tuple, Applications of dictionary, list and tuple

Python Data Types

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its 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.

Consider the following example to define the values of different data types and checking its type.

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

Output:

<type 'int'>

<type 'str'>

<type 'float'>

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

List

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.

A list can be defined as follows.

1. L1 = ["John", 102, "USA"]
2. L2 = [1, 2, 3, 4, 5, 6]
3. L3 = [1, "Ryan"]

Example:

emp = ["John", 102, "USA"]

print(type(emp))

```
In [1]: emp = ["John", 102, "USA"]
   ...: print(type(emp))
<class 'list'>
```

print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))

```
Name : John, ID: 102, Country: USA
```

print("Name :"+ emp[0]+" ID :"+str(emp[1])+" Country :"+emp[2])

```
Name :John ID :102 Country :USA
```

List indexing and splitting

The indexing are processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

Consider the following example.

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

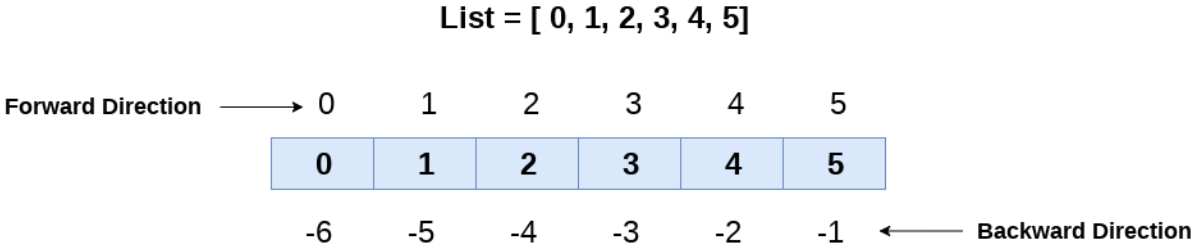
List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

Unlike other languages, python provides us the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (right most) of the list has the index -1, its adjacent left element is present at the index -2 and so on until the left most element is encountered.



Updating List values

Lists are the most versatile data structures in python since they are mutable and their values can be updated by using the slice and assignment operator.

Python also provide us the append() method which can be used to add values to the string.

Consider the following example to update the values inside the list.

List = [1, 2, 3, 4, 5, 6]

print(List)

List[2] = 10;

print(List)

List[1:3] = [89, 78]

print(List)

Output:

[1, 2, 3, 4, 5, 6]

[1, 2, 10, 4, 5, 6]

[1, 89, 78, 4, 5, 6]

The list elements can also be deleted by using the **del** keyword. Python also provides us the remove() method if we do not know which element is to be deleted from the list.

emp = ["John", 102, "USA"]

del emp[0]

print(emp)

```
In [11]: print(emp)
[102, 'USA']
```

emp.remove('USA')

print(emp)

```
In [18]: print(emp)
[102]
```

Adding elements to the list

Python provides append() function by using which we can add an element to the list. However, the append() method can only add the value to the end of the list.

x = ['Karan', 27, 'USA']

print(x)

x.append('Vadodara')

print(x)

```
['Karan', 27, 'USA']
['Karan', 27, 'USA', 'Vadodara']
```

Python List Built-in functions

Python provides the following built-in functions which can be used with the lists.

SN	Function	Description
1	len(list)	It is used to calculate the length of the list.
2	max(list)	It returns the maximum element of the list.
3	min(list)	It returns the minimum element of the list.

```
y = ['Karan', 27, 'USA']
z = [15, 27, 18]
print(len(y))
print(max(z))
print(min(z))
```

```
3
27
15
```

```
y = ['Karan', 27, 'USA']
print(y)
y.reverse()
print(y)
```

```
['Karan', 27, 'USA']
['USA', 27, 'Karan']
```

```
courses = ['CPU', 'JAVA', '.net', 'Python']
print(courses)
print(len(courses))
print(courses[0])
print(courses[-1])
print(courses[-2])
print(courses[0:2])

courses.append('PHP')
print(courses)

courses.insert(1, 'HTML')
print(courses)

courses_2 = ['Art','Science']
courses.insert(0,courses_2)
print(courses)

print(courses[0])

courses.append(courses_2)
print(courses)

courses_3 = ['Maths','Physics']
courses.extend(courses_3)
print(courses)
```

```
['CPU', 'JAVA', '.net', 'Python']
4
CPU
Python
.net
['CPU', 'JAVA']
['CPU', 'JAVA', '.net', 'Python', 'PHP']
['CPU', 'HTML', 'JAVA', '.net', 'Python', 'PHP']
[['Art', 'Science'], 'CPU', 'HTML', 'JAVA', '.net', 'Python', 'PHP']
['Art', 'Science']
[['Art', 'Science'], 'CPU', 'HTML', 'JAVA', '.net', 'Python', 'PHP',
['Art', 'Science']]
[['Art', 'Science'], 'CPU', 'HTML', 'JAVA', '.net', 'Python', 'PHP',
['Art', 'Science'], 'Maths', 'Physics']
```

mutable object can be changed after it is created, and an **immutable** object can't.

So, List is mutable.

Python Tuple

Python Tuple is used to store the sequence of immutable python objects. Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets. A tuple can be defined as follows.

```
list_1 = ['CPU', 'JAVA', 'Python', 'Android', 'HTML']
list_2 = list_1

print(list_1)
print(list_2)

list_1[0] = 'C#'        # list is mutable

print(list_1)
print(list_2)

tuple_1 = ('CPU', 'JAVA', 'Python', 'Android', 'HTML')
tuple_2 = tuple_1

print(tuple_1)
print(tuple_2)

tuple_1[0] = 'C#'   # tuple is immutable

print(tuple_1)
print(tuple_2)
```

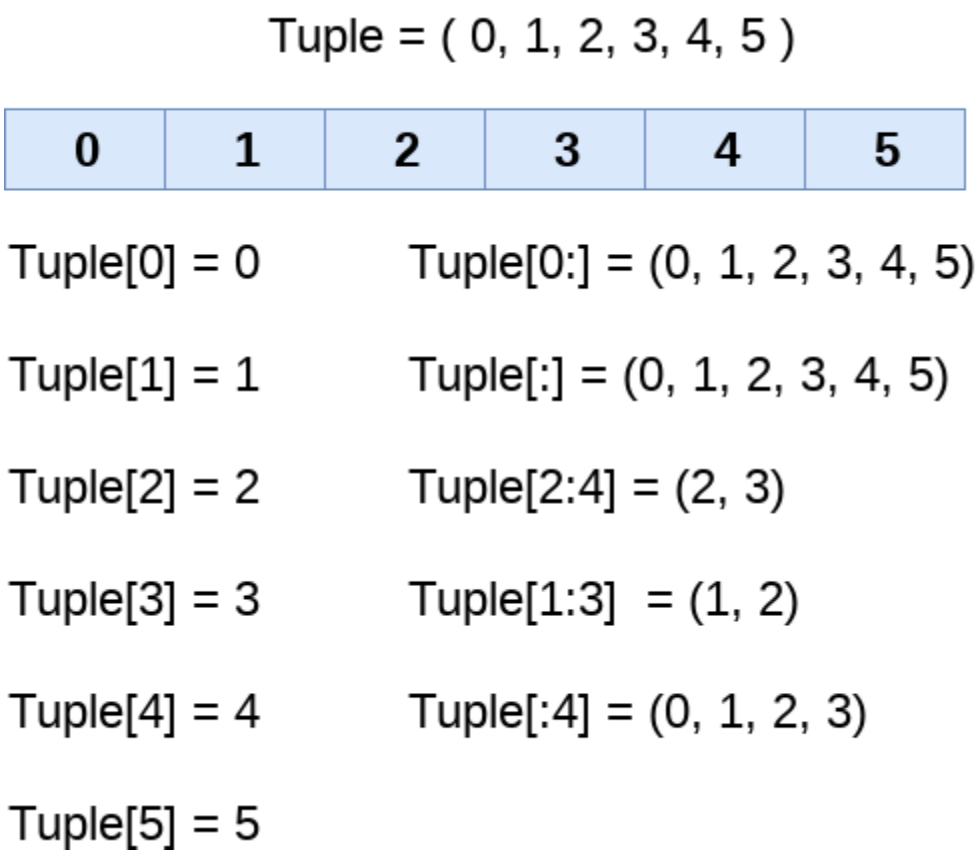
```
['CPU', 'JAVA', 'Python', 'Android', 'HTML']
['CPU', 'JAVA', 'Python', 'Android', 'HTML']
['C#', 'JAVA', 'Python', 'Android', 'HTML']
['C#', 'JAVA', 'Python', 'Android', 'HTML']
('CPU', 'JAVA', 'Python', 'Android', 'HTML')
('CPU', 'JAVA', 'Python', 'Android', 'HTML')

In [2]: tuple_1[0] = 'C#'
Traceback (most recent call last):

  File "<ipython-input-2-8682ada4e197>", line 1, in <module>
    tuple_1[0] = 'C#'

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

Tuple indexing and splitting

The indexing and slicing in tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.

The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

Consider the following image to understand the indexing and slicing in detail.



List VS Tuple

List	Tuple
The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
The List is mutable.	The tuple is immutable.
The List has the variable length.	The tuple has the fixed length.
The list provides more functionality than tuple.	The tuple provides less functionality than the list.

The list Is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Python Dictionary

Dictionary is used to implement the key-value pair in python. The dictionary is the data type in python which can simulate the real-life data arrangement where some specific value exists for some particular key.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs separated by the colon (:). The collections of the key-value pairs are enclosed within the curly braces {}.

The syntax to define the dictionary is given below.

Dict = {"Name": "Ayush","Age": 22}

In the above dictionary Dict, The keys Name, and Age are the string that is an immutable object.

Let's see an example to create a dictionary and printing its content.

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

print(type(Employee))

print("printing Employee data .... ")

print(Employee)

```
<class 'dict'>
printing Employee data ....
{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}
```

Accessing the dictionary values

We have discussed how the data can be accessed in the list and tuple by using the indexing.

However, the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

```
print("Name : "+Employee["Name"])
print("Age : "+str(Employee["Age"]))
print("Salary : "+str(Employee["salary"]))
print("Company : "+Employee["Company"])
```

```
Name : John
Age : 29
Salary : 25000
Company : GOOGLE
```

Updating and Deleting dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys.

Let's see an example to update the dictionary values.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

```
<class 'dict'>
printing Employee data ....
{'Name': 'John', 'Age': 29, 'salary': 25000, 'Company': 'GOOGLE'}
Enter the details of the new employee....

Name: Manan

Age: 26

Salary: 40

Company:UVPCE
printing the new data
{'Name': 'Manan', 'Age': 26, 'salary': 40, 'Company': 'UVPCE'}
```

del Employee["salary"]

print(Employee)

```
{'Name': 'Manan', 'Age': 26, 'Company': 'UVPCE'}
```

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
print(x);

```
Name
Age
salary
Company
```

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
print(Employee[x]);

```
John
29
25000
GOOGLE
```

Python Set

The set in python can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set can not be duplicate. The elements of the python set must be immutable.

Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday"}
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
   print(i)
```

```
{'Friday', 'Monday', 'Thursday', 'Sunday', 'Wednesday', 'Tuesday',
'Saturday'}
<class 'set'>
looping through the set elements ...
Friday
Monday
Thursday
Sunday
Wednesday
Tuesday
Saturday
```

You can use a List to store the steps necessary to cook a chicken, because Lists support sequential access and you can access the steps in order.

1. It is used to store multiple kind of data types and perform numerous action over these data.
2. To provide easy access to, and manipulation of, a data set.

You can use a Tuple to store the latitude and longitude of your home, because a tuple always has a predefined number of elements (in this specific example, two). The same Tuple type can be used to store the coordinates of other locations.

You can use a Set to store passport numbers, because a Set enforces uniqueness among its elements. Passport numbers are always unique and two people can't have the same one.

1. A Dictionary (or "**dict**") is a way to store data just like a list, but instead of using only numbers to get the data, you can use almost anything.
2. in order to store unordered key-value-pairs.