UNITED
GROUP OF INSTITUTIONS

Python Programming
Unit 3
(KNC-302)

Prepared By
Abhishek Kesharwani
Assistant Professor,UCER Naini,Allahabad

UNIT III

- Python Data Structure:
- Tuples,
- Unpacking Sequences,
- Lists
- Mutable Sequences,
- List Comprehension,
- Sets
- Dictionaries
- Higher Order Functions:
- Functions as first-class Objects,
- Lambda Expressions

What is List

- List is ordered and allows duplicate entries as well.
- Lists in python need not be homogeneous, which means it can contain different data types like integers, strings and other collection data types.
- It is mutable in nature and allows indexing to access the members in a list.
- The items in the list are separated with the comma (,) and enclosed with the square brackets [].

A list can be defined as follows.

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

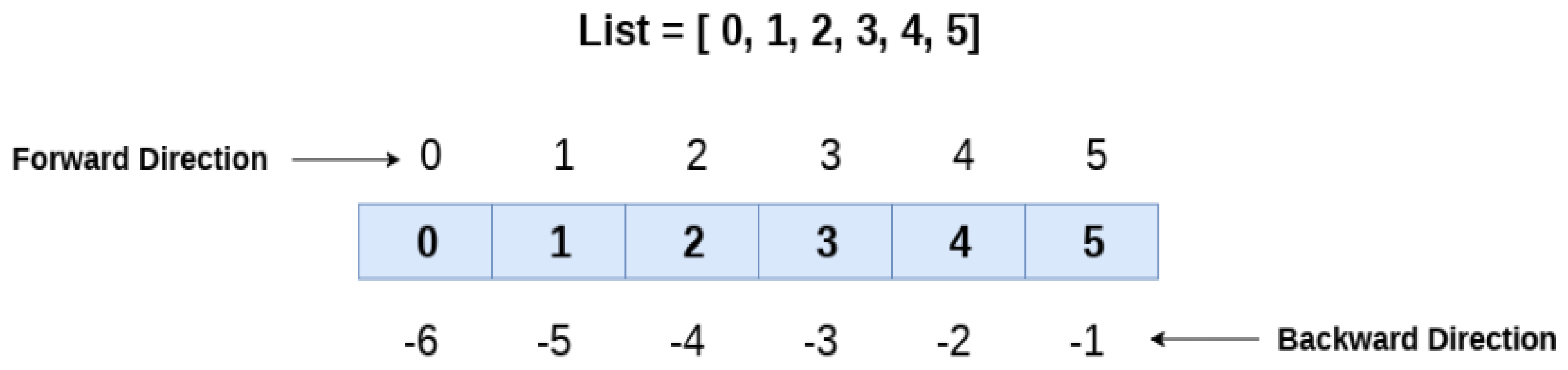
Why it is important

- Lists are small but essential element in data science.
- No need to worry about size and type of the list.
- List can be used to solve complex problems with reduced lines of codes.

List indexing and splitting

- The elements of the list can be accessed by using the slice operator [:].
- The index starts from 0 and goes to length - 1.
- Unlike other languages, python provides us the flexibility to use the negative indexing also.
- The negative indices are counted from the right.

Positive and Negative indexing



Splitting

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]

Find the Output of the Program

```
l1=[1,2,3,6,7]
print(l1[3:])
print(l1[1:3])
print(l1[:3])
```

Output

- [6, 7]
- [2, 3]
- [1, 2, 3]

List Built-in functions

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.
4	list(seq)	It converts any sequence to the list.

List built-in methods

S. No	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	list.count(obj)	It returns the number of occurrences of the specified object in the list.
4	list.insert(index, obj)	The object is inserted into the list at the specified index.
5	list.pop(obj=list[-1])	It removes and returns the last object of the list.
6	list.remove(obj)	It removes the specified object from the list.
7	list.reverse()	It reverses the list.
8	list.sort([func])	It sorts the list by using the specified compare function if given.
9	list.extend(seq)	The sequence represented by the object seq is extended to the list.

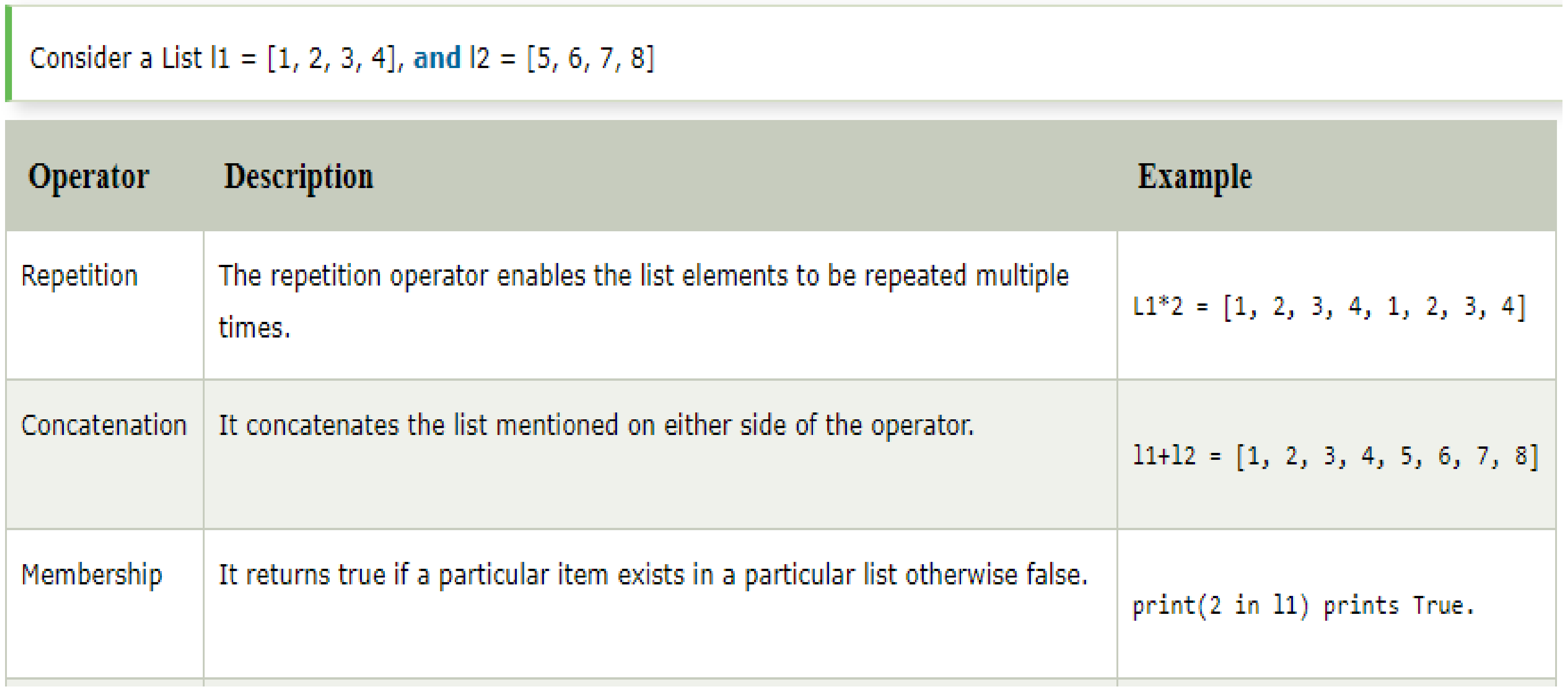
Find the Output of the Program

```
l1=[1,2,3]
l1.append(0)
print(l1)
l1.insert(2,7)
print(l1)
print(l1.pop(2))
print(l1)
l1.remove(1)
print(l1)
l1.reverse()
print(l1)
l1.sort()
print(l1)
l1.sort(reverse=True)
print(l1)
```

Output

- [1, 2, 3, 0]
- [1, 2, 7, 3, 0]
- 7
- [1, 2, 3, 0]
- [2, 3, 0]
- [0, 3, 2]
- [0, 2, 3]
- [3, 2, 0]

Python List Operations



Consider a List l1 = [1, 2, 3, 4], **and** l2 = [5, 6, 7, 8]

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	`L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]`
Concatenation	It concatenates the list mentioned on either side of the operator.	`l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]`
Membership	It returns true if a particular item exists in a particular list otherwise false.	`print(2 in l1) prints True.`

Iteration	The for loop is used to iterate over the list elements.	`for i in l1:`
		`    print(i)`
		Output
		`1`
		`2`
		`3`
		`4`

Length	It is used to get the length of the list	`len(l1) = 4`

```
if __name__ == '__main__':
    N = int(input())
    L=[]
    for i in range(N):
        x=input().split(" ")
        command=x[0]
        if command=="insert":
            L.insert(int(x[1]),int(x[2]))
        if command=="remove":
            L.remove(int(x[1]))
        if command=="append":
            L.append(int(x[1]))
        if command=="sort":
            L.sort()
        if command=="pop":
            if(len(L)!=0):
             L.pop()
        if command=="reverse":
            L.reverse()
        if command=="print":
            print(L)
```

Introduction to the list unpacking

```
colors = ['red', 'blue', 'green’]
```

To assign the first, second, and third elements of the list to variables, you may assign individual elements to variables like this:

```
red = colors[0]
blue = colors[1]
green = colors[2]
```

```
red, blue, green = colors
```

If you use a fewer number of variables on the left side, you’ll get an error. For example:

```
colors = ['red', 'blue', 'green']
red, blue = colors
```

```
Error
```

```python
colors = ['cyan', 'magenta', 'yellow', 'black']
cyan, magenta, *other = colors

print(cyan)
print(magenta)
print(other)
```

Here the first and second elements of the colors list to the cyan and magenta variables. And it assigns the last elements of the list to the other variable

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.

Example

```
tuple1=(11,12,1,4,56,32)
count=0;
for i in tuple1:
    print("tuple1[%d]=%d"%(count,i))
    count=count+1
```

Output

```
tuple1[0]=11
tuple1[1]=12
tuple1[2]=1
tuple1[3]=4
tuple1[4]=56
tuple1[5]=32
```

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.

Tuple indexing and splitting

Tuple = (0, 1, 2, 3, 4, 5)

0	**1**	**2**	**3**	**4**	**5**

Tuple[0] = 0

Tuple[1] = 1

Tuple[2] = 2

Tuple[3] = 3

Tuple[4] = 4

Tuple[5] = 5

Tuple[0:] = (0, 1, 2, 3, 4, 5)

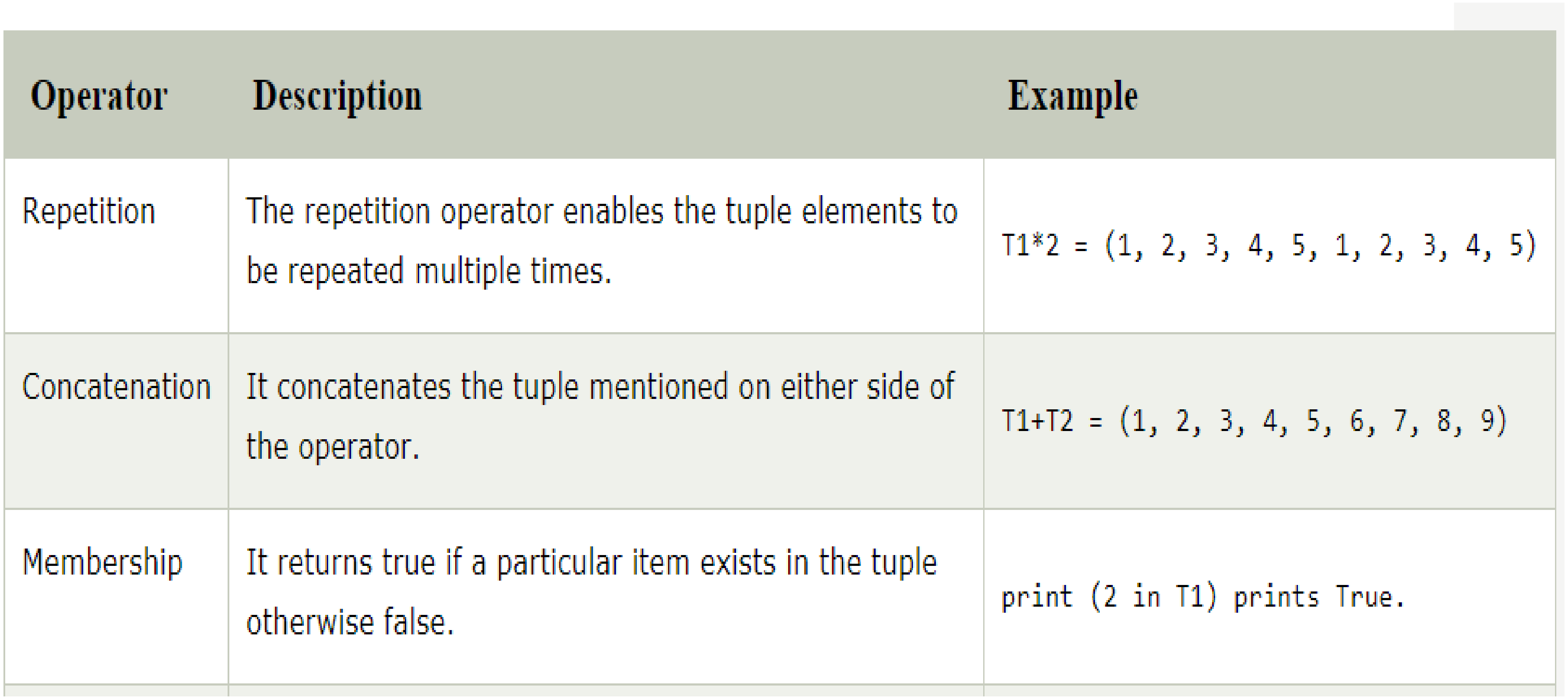
Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2:4] = (2, 3)

Tuple[1:3] = (1, 2)

Tuple[:4] = (0, 1, 2, 3)

Basic Tuple operations

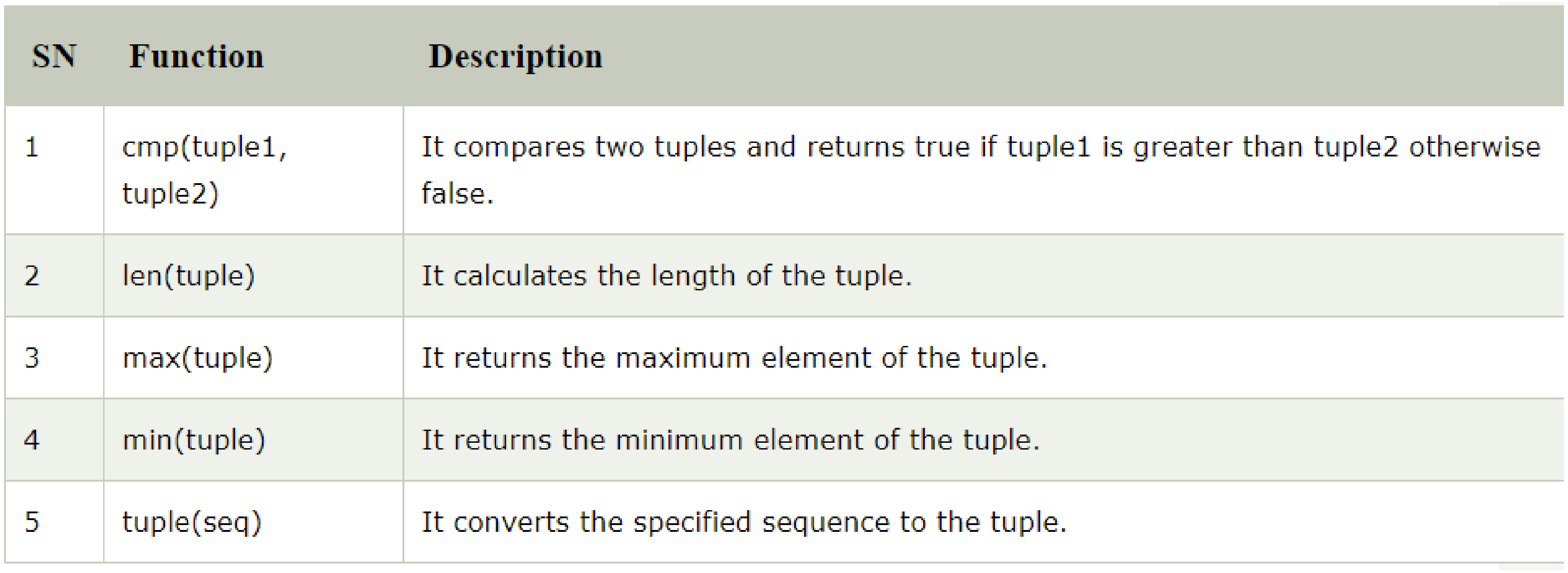


Operator	Description	Example
Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	`T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)`
Concatenation	It concatenates the tuple mentioned on either side of the operator.	`T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)`
Membership	It returns true if a particular item exists in the tuple otherwise false.	`print (2 in T1) prints True.`

Iteration	The for loop is used to iterate over the tuple elements.	`for i in T1:`
		`    print(i)`
		Output
		1
		2
		3
		4
		5

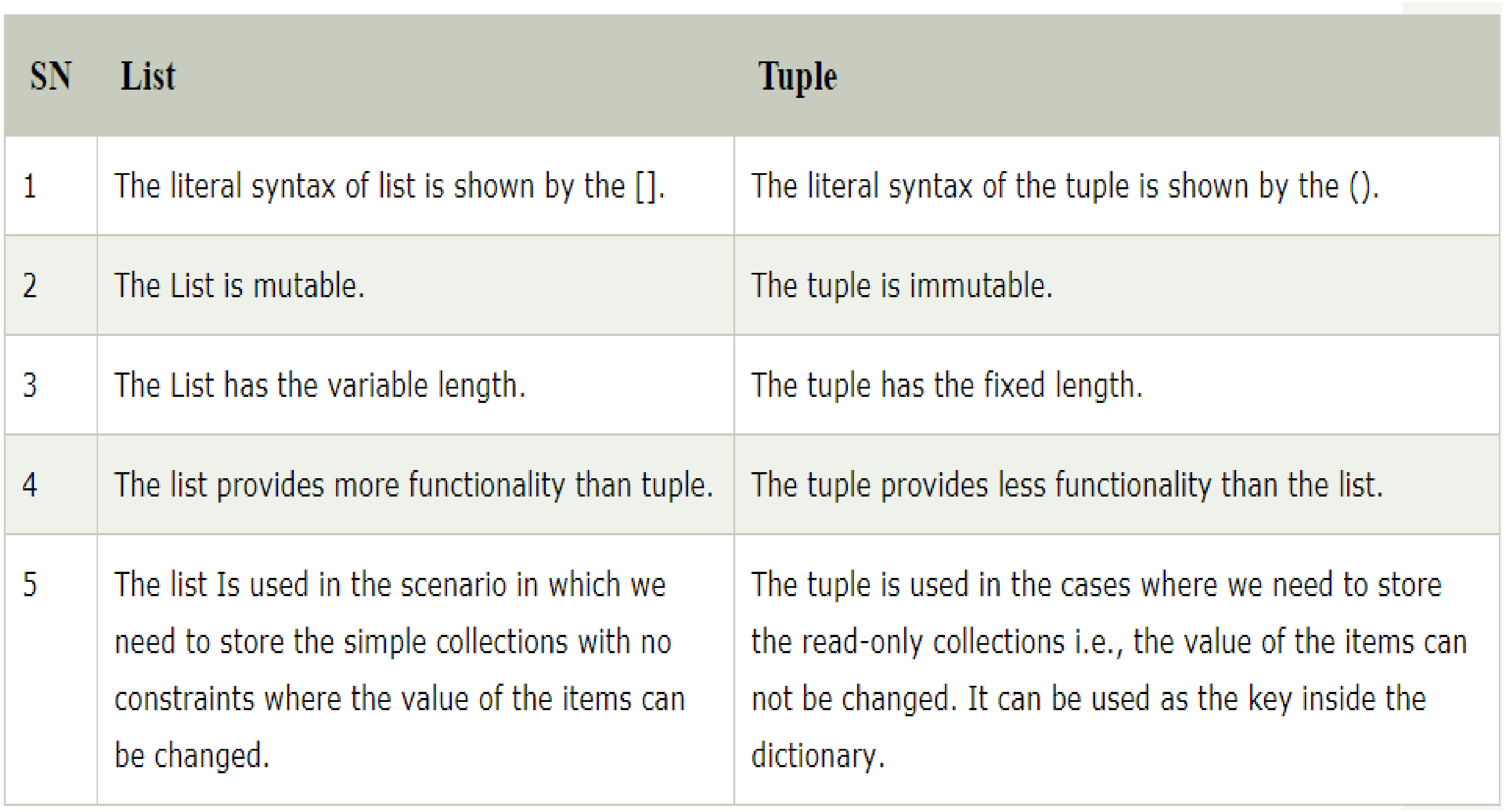
Length	It is used to get the length of the tuple.	`len(T1) = 5`

Python Tuple inbuilt functions



SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple.
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

List VS Tuple



SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	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 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.

Creating a set

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

Output

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

set() method

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

Python Set operations

Adding items to the set

- Python provides the add() method which can be used to add some particular item to the set.

```
Months = set(["January","February", "March", "
   April", "May", "June"])
Months.add("July");
Months.add("August");
```

To add more than one item in the set, Python provides the **update()** method.

```
Months = set(["January","February", "March", "
April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nupdating the original set ... ")
Months.update(["July","August","September","
October"]);
print(Months)
```

Removing items from the set

- Python provides **discard()** method which can be used to remove the items from the set.
- Python also provide the **remove()** method to remove the items from the set.
- We can also use the **pop()** method to remove the item. However, this method will always remove the last item.
- Python provides the **clear()** method to remove all the items from the set.
- To add more than one item in the set, Python provides the **update()** method.

Difference between discard() and remove()

- If the key to be deleted from the set using discard() doesn't exist in the set, the python will not give the error. The program maintains its control flow.
- On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.

Example

```
Months = set(["January","February", "March", "
   April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nAdding other months to the set...")
Months.add("July")
Months.add("August")
Months.update(["July","August","September","October"]);
print("\nprinting the modified set ... ")
print(Months);
Months.discard("January");
Months.discard("May");
print("\nPrinting the modified set...");
print(Months)
```

```
Months.remove("January");
Months.remove("May");
print("\nPrinting the modified set...");
print(Months)
Months.pop();
print("\nPrinting the modified set...");
print(Months)
Months.clear()
print("\nPrinting the modified set...")
print(Months)
```

Union of two Sets

- The union of two sets are calculated by using the or (|) operator.
- The union of the two sets contains the all the items that are present in both the sets.

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2)
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2))
#printing the union of the sets
```

Output:

```
{'Friday', 'Monday', 'Tuesday', 'Thursday', 'Wednesday', 'Sunday', 'Saturday'}
```

Intersection of two sets

- The & (intersection) operator is used to calculate the intersection of the two sets in python.
- The intersection of the two sets are given as the set of the elements that common in both sets.

using & operator

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steve","Milan","David", "Martin"}
print(set1&set2)
#prints the intersection of the two sets
```

Output:

```
{'Martin', 'David'}
```

using intersection() method

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steave","Milan","David", "Martin"}
print(set1.intersection(set2))
 #prints the intersection of the two sets
```

Output:

```
{'Martin', 'David'}
```

Difference of two sets

- The difference of two sets can be calculated by using the subtraction (-) operator.
- The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

using subtraction (-) operator

Days1 = {"Monday", "Tuesday", "Wednesday", "
Thursday"}

Days2 = {"Monday", "Tuesday", "Sunday"}

print(Days1-
Days2) #{"Wednesday", "Thursday" will be pri
nted}

Output:

{'Thursday', 'Wednesday'}

using difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "
    Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2))
# prints the difference of the two sets Days1 and
    Days2
```

Output:

```
{'Thursday', 'Wednesday'}
```

Set comparisons

- Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is subset, superset, or equivalent to other set.
- The boolean true or false is returned depending upon the items present inside the sets.

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thu
    rsday"}
Days2 = {"Monday", "Tuesday"}
Days3 = {"Monday", "Tuesday", "Friday"}
#Days1 is the superset of Days2 hence it will print true.
print (Days1>Days2)
 #prints false since Days1 is not the subset of Days2
print (Days1<Days2)
 #prints false since Days2 and Days3 are not equivalent
print (Days2 == Days3)
```

Output:

```
True
False
False
```

FrozenSets

- The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set can not be changed and therefore it can be used as a key in dictionary.
- The elements of the frozen set can not be changed after the creation.
- We cannot change or append the content of the frozen sets by using the methods like add() or remove().

```
Frozenset = frozenset([1,2,3,4,5])
print(type(Frozenset))
print("\nprinting the content of frozen set...")
for i in Frozenset:
    print(i);
Frozenset.add(6)
```

Output

```
<class 'frozenset'>
printing the content of frozen set...
1
2
3
4
5
'frozenset' object has no attribute 'add'
```

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.
- A dictionary is the collection of key-value pairs where the value can be any python object whereas the keys are the immutable python object, i.e., Numbers, string or tuple.

Creating the dictionary

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

```
Employee = {"Name": "John", "Age": 29, "salary"
    :25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Output

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

Accessing the dictionary values

```
Employee = {"Name": "John", "Age": 29, "salary"
    :25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Output

```
<class 'dict'>
printing Employee data ....
Name : John Age : 29 Salary : 25000 Company :
    GOOGLE
```

Updating dictionary values

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

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)
```

Deleting elements using del keyword

```
Employee = {"Name": "John", "Age": 29, "salary":25
    000,"Company":"GOOGLE"}
del Employee["Name"]
del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
del Employee
print("Lets try to print it again ");
print(Employee)
```

Output

- printing the modified information
- {'Age': 29, 'salary': 25000}
- Deleting the dictionary: Employee Lets try to print it again Traceback (most recent call last): File "list.py", line 13, in <module> print(Employee)
- NameError: name 'Employee' is not defined

Iterating Dictionary

for loop to print all the keys of a dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25
    000,"Company":"GOOGLE"}
for x in Employee:
    print(x)
```

Output

```
Name
Company
Salary
Age
```

for loop to print the values of the dictionary by using values() method.

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

Output:

```
GOOGLE
25000
John
29
```

#for loop to print the items of the dictionary by using items() method.

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

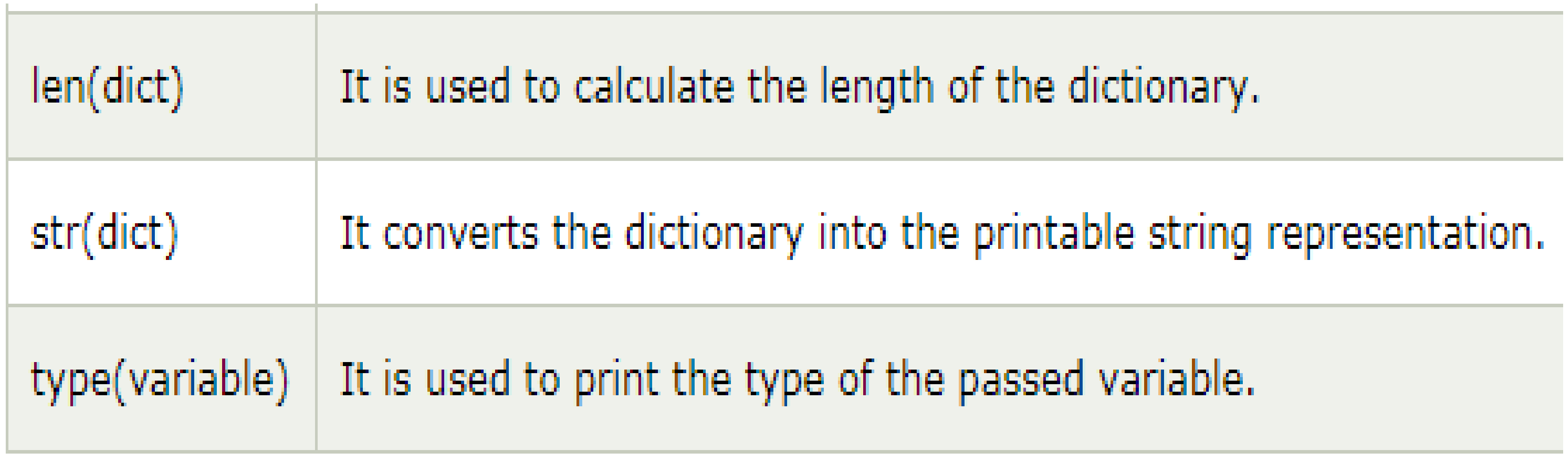
Output:

```
('Name', 'John')
('Age', 29)
('salary', 25000)
('Company', 'GOOGLE')
```

Properties of Dictionary keys

1. In the dictionary, we can not store multiple values for the same keys. If we pass more than one values for a single key, then the value which is last assigned is considered as the value of the key.
2. In python, the key cannot be any mutable object. We can use numbers, strings, or tuple as the key but we can not use any mutable object like the list as the key in the dictionary.

Built-in Dictionary functions



len(dict)	It is used to calculate the length of the dictionary.
str(dict)	It converts the dictionary into the printable string representation.
type(variable)	It is used to print the type of the passed variable.

Dictionary setdefault() Method

- Python setdefault() method is used to set default value to the key.
- It returns value, if the key is present. Otherwise it insert key with the default value.
- Default value for the key is None.

Example

```
coursefee = {'B,Tech': 400000, 'BA':2500, 'B.COM
    ':50000}
p = coursefee.setdefault('BCA',100000)
print("default",p)
print(coursefee)
```

Dictionary update() Method

- Python update() method updates the dictionary with the key and value pairs.
- It inserts key/value if it is not present.
- It updates key/value if it is already present in the dictionary.
- It also allows an iterable of key/value pairs to update the dictionary. like: update(a=10,b=20) etc.

Example

```
inventory = {'Fan': 200, 'Bulb':150, 'Led':1000}
inventory.update({'cooler':50})
inventory.update(Switch=300,case=200)
print("Updated inventory:",inventory)
```

```
Updated inventory:
{'Fan': 200, 'Bulb': 150, 'Led': 1000, 'cooler': 50,
 'Switch': 300, 'case': 200}
```

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

- Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
- Without list comprehension you will have to write a for statement with a conditional test inside.

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Output

```
['apple', 'banana', 'mango']
```

Syntax

```
newlist =
[expression for item in iterable if
 condition == True]
```

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if x != "apple"]

print(newlist)
```

Example

```
fruits = ["apple", "banana", "cherry", "kiwi",
"mango"]

newlist = [x for x in fruits]

print(newlist)
```

Example

You can use the range() function to create an iterable:

```
newlist = [x for x in range(10)]
```

Example

```
newlist = [x for x in range(10) if x < 5]
```

Example

```
fruits = ["apple", "banana", "cherry", "kiwi",
"mango"]

newlist = [x if x != "banana" else "orange" for x
in fruits]

print(newlist)
```

Python Lambda Functions

- Python allows us to not declare the function in the standard manner, i.e., by using the def keyword.
- Rather, the anonymous functions are declared by using lambda keyword.
- However, Lambda functions can accept any number of arguments, but they can return only one value in the form of expression.

lambda arguments : expression

Example 1

x = **lambda** a:a+10

a is an argument and a+10 is an expression which got evaluated and returned.

print("sum = ",x(20))

Output

sum = 30

Example 2

Multiple arguments to Lambda function

x = **lambda** a,b:a+b

# a and b are the arguments and a+b is the expr
ession which gets evaluated and returned.

print("sum = ",x(20,10))

Output:

sum = 30

Example 2

Use of lambda function with filter

```
#program to filter out the list which contains nu
   mber divisible by 3 numbers
List = {1,2,3,4,10,123,22}
Oddlist = list(filter(lambda x:(x%3 == 0),List))
print(Oddlist)
```

Output:

```
[3, 123]
```

Example 3

```
List = {1,2,3,4,10,123,22}
new_list = list(map(lambda x:x*3,List))

print(new_list)
```

Output:

```
[3, 6, 9, 12, 30, 66, 369]
```