

Assignment - 1

Python Methods

① enumerate():

— enumerate() method adds a counter to an iterable and returns it in the form of an enumerating object.

This enumerated object can then be used directly for loops or converted into a list of tuples using the list() function.

Syntax: `enumerate(iterable, start=0)`

Parameters: iterable: any object that supports iteration

start: the index value from which the counter is to be started; by default it is 0

Returns an iterator with index, and element pairs from the original iterable

Example for enumerate() with both list and a string:

```
L1 = ["c", "java", "python"]
```

```
S1 = "System"
```

```
# Creating enumerate objects
```

```
obj1 = enumerate(L1)
```

```
obj2 = enumerate(S1)
```

```
print("return type:", type(obj1))
```

```
print(list(enumerate(L1)))
```

```
print ("return type:", type (s1))
```

```
# changing the start index to 1 from 0
```

```
print (list (enumerate (s1, 1)))
```

o/p : return type: <class 'enumerate'>

[(0, 'c'), (1, 'java'), (2, 'python')]

return type: <class 'str'>

[(1, 's'), (2, 'y'), (3, 's'), (4, 't'), (5, 'e'), (6, 'm')]

Example using enumerate object in loops

```
l1 = ["c", "java", "python"]
```

```
# printing the tuples in object directly
```

```
for Sys in enumerate (l1):
```

```
    print (Sys)
```

```
# changing index and printing separately
```

```
for Count, Sys in enumerate (l1, 10):
```

```
    print (Count, Sys)
```

```
# getting desired output from tuple
```

```
for Count, Sys in enumerate (l1):
```

```
    print (Count)
```

```
    print (Sys)
```

Q/p : (0, 'c')
(1, 'java')
(2, 'python')

10, c
11 java
12 python
0
c
1
java
2
python

output: 10, c
11, java
12, python

enumerate() example 2

```
In [8]: l1=["c","java","python"]

#printing the tuples in object directly
for sys in enumerate(l1):
    print(sys)

#changing index and printing seperately
for count,sys in enumerate(l1,10):
    print(count,sys)

#getting desired output from tuple
for count,sys in enumerate(l1):
    print(count)
    print(sys)
```

```
(0, 'c')
(1, 'java')
(2, 'python')
10 c
11 java
12 python
0
c
1
java
2
python
```


② `reduce()` is a built-in function that applies a particular function to the elements of an iterable, reducing them to a single value.

Working:

- i) At first step, first 2 elements of sequence are picked and the result is obtained.
- ii) Next step is to apply the same function to the previously attained result and the number just succeeding the 2nd element and the result is again stored.
- iii) This process continues till no more elements are left and the final returned result is returned.

reduce function is defined in "functools" module

Syntax: `functools.reduce(function, iterable[, initializer])`

Parameters:

- function argument is a function that takes 2 arguments where 1st argument is accumulated value and the second argument is current value from iterable
- iterable argument is sequence of values to be reduced
- initializer is optional

Example for `reduce()`:

```
from functools import reduce
```

```
# function that returns the sum of 2 numbers
```

```
def add(a,b):
```

```
    return a+b
```

```
# iterable
```

```
num_list = [1,2,3]
```

```
# add function is passed as the 1st argument  
and num_list
```

```
sum = reduce(add, num_list)
```

```
print("sum of integers of num_list:", (sum))
```

```
# passing 10 as initial value
```

```
sum = reduce(add, num_list, 10)
```

```
print("sum of integers of num_list with initial  
value 10:", (sum))
```

Q1 : sum of integers of num_list : 6
sum of integers of num_list with initial value 10: 16

reduce()

```
In [6]: from functools import reduce

#function that returns the sum of two numbers
def add(a,b):
    return a+b

#iterable
num_list=[1,2,3,4,5,6,7,8,9,10]

#add function is passed as the first argument and num_list
sum=reduce(add,num_list)
print("sum of the integers of num_list:",(sum))

#passing 10 as an initial value
sum=reduce(add,num_list,10)
print("sum of the integers of num_list with intial value 10:",(sum))

sum of the integers of num_list: 55
sum of the integers of num_list with intial value 10: 65
```


③ map():

map() function returns a map object (which is an iterator) of results after applying the given function to each item of a given iterable (list, tuple etc...)

Syntax: map(function, iterable)

parameters:

function: To which map passes each element of given iterable

iterable: which is to be mapped

We can pass 1 or more iterable to the map() function

Returns from map():

- Returns a list of results after applying the given function to each item of a given iterable (list, tuple)
- The returned value from map() (map object), then can be passed to functions like list() (to create list), set() (to create set)

Example for map() :

1) # Return double of n

```
def addition(n):
```

```
    return n+n
```

Double all numbers using map()

```
numbers = (1,2,3,4)
```

```
result = map(addition, numbers)
```

```
print(list(result))
```

O/p : [2, 4, 6, 8]

2) # Double all the numbers using map and lambda

```
numbers = (1,2,3,4)
```

```
result = map(lambda x: x+x, numbers)
```

```
print(list(result))
```

O/p : [2, 4, 6, 8]

3) # Add two lists using map and lambda

```
numbers1 = [1,2,3]
```

```
numbers2 = [4,5,6]
```

```
result = map(lambda x,y: x+y, numbers1, numbers2)
```

```
print(list(result))
```

O/p : [5, 7, 9]

map() example

```
In [1]: #return double of n

def addition(n):
    return n+n

#double all numbers using map()

numbers=(1,2,3,4)
result=map(addition,numbers)
print(list(result))

[2, 4, 6, 8]
```

map() example2

```
In [2]: #list of strings

l=['sat','bat','mat']

#map() can listify the list of strings individually

test=list(map(list,l))
print(test)

[['s', 'a', 't'], ['b', 'a', 't'], ['m', 'a', 't']]
```


④ filter() :

— filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not

Syntax : filter(function, sequence)

Parameters :

function : Tests if each element of a sequence is true or not

Sequence : which needs to be filtered, it can be sets, lists, tuples (or) containers of any iterators

Returns an iterator that is already filtered

Example for filter() :

function that filters Vowels

```
def fun (Variable):
```

```
    letters = ['a', 'e', 'i', 'o', 'u']
```


if (variable in letters):

return True

else:

return False

Sequence

Sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'x']

using filter function

filtered = filter(fun, Sequence)

print('The filtered letters are:')

for s in filtered:

print(s)

O/p: The filtered letters are:

e
e

example for filter()

In [4]: *#function that filters vowels*

```
def fun(variable):
    letters=['a','e','i','o','u']
    if(variable in letters):
        return True
    else:
        return False
#sequence
sequence=['g','e','e','j','k','s','p','r']

#using filter function
filtered=filter(fun,sequence)
print('The filtered letters are:')

for s in filtered:
    print(s)
```

The filtered letters are:

e
e

⑤ `zip()`:

It takes iterable containers and returns a single iterator object, having mapped values from all the containers — It is used to map the similar index of multiple containers so that they can be used just using a single entity.

Syntax: `zip(iterator1, iterator2, ...)`

Parameters: `iterator1`, `iterator2` etc., These are

iterables that want to Combine. , (1)2, roll no. 3
We can provide multiple iterables as arguments and
zip() will pair up elements at corresponding positions
— Returns a single iterator object

Example for zip():

Create two lists

name = ["Manjeet", "Nikhil", "Shambavi", "Astha"]

rollno = [4, 1, 3, 2]

using zip() to map the values

mapped = zip(name, rollno)

print(set(mapped))

O/p : {('Nikhil', 1), ('Shambavi', 3), ('Manjeeth', 4),
('Astha', 2)}

example for zip()

```
In [8]: #create two lists

name=["Manjeeth","Nikhil","shambavi","Ashtha"]
rollno=[4,1,3,2]

#using zip()to map the values

mapped=zip(name,rollno)
print(set(mapped))

{('Nikhil', 1), ('Manjeeth', 4), ('shambavi', 3), ('Ashtha', 2)}
```

⑥ `id()` :

It returns the unique identifier of an object

The identifier is an integer, which represents the memory address of the object.

— `id()` function is used to check if two variables or objects refer to the same memory location

Syntax: `id(Object)`

Returns a unique integer for a given object

Example for id():

$x = 42$

$y = x$

$z = 42$

`print (id(x))`

`print (id(y))` # (Same as x)

`print (id(z))` # (Same as x & y)

O/p: ~~10731304~~ 140713214187592

~~10731304~~ 140713214187592

~~10731304~~ 140713214187592

In [9]: *#example for id()*

In [13]: `x=42
y=x
z=42
print(id(x))
print(id(y))
print(id(z))`

140713214187592
140713214187592
140713214187592