NAME : SHRUTI THOOL

AICTE Internship ID :

STU64e0e03a9dafa1692459066

(1) enumerate ()

The Python enumerate () is a function that converts a data collection object into an enumerate object. The Python enumerate () function works faster as compared to the iteration with a loop. It is because the function creates an enumerate object which yields the result one by one and requires less memory. We can use the Python enumerate () function to access any given list or iterable item. This helps us keep track of the index and the element while iterating. We can use Python enumerate () to reduce the compilation time and the memory requirement.

Syntax :

enumerate (iterable, start = 0)

The Python enumerate () function takes the following two parameters:

1. iterable - It is a sequence or an iterator which enables iteration. It is the mandatory parameter.

2. start - It is used to set the starting of the index. It is optional, i.e., it may or may not be present. zero is the default value.

Example :

```
fruits = ['apple', 'banana', 'cherry']
for i, fruit in enumerate(fruits):
    print(i, fruit)
```

```
>>> fruits=['apple','banana','cherry']
>>> for i,fruit in enumerate(fruits):
...     print(i,fruit)
...
0 apple
1 banana
2 cherry
>>>
```

In this example, the enumerate() function is utilized to emphasize over the fruits list, and for every component in the rundown, a tuple containing the index and the relating value is delivered. The for loop then prints each tuple, bringing about the result displayed previously.

(2) zip()

Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes iterables, makes it an iterator that aggregates the elements based on iterables passed, and returns an iterator of tuples.

Python's zip() function is a built-in function that is used to combine two or more iterables into a single iterable. This function takes in any number of iterables (lists, tuples, sets, etc.) as arguments and returns an iterator that aggregates elements from each of the iterables into tuples. Each tuple contains the i-th element from each of the input iterables.

Syntax :

```
zip (*iterables)
```

Example :

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
zipped = zip( list1, list2)
result = list (zipped)
print (result)
```

```
C:\Users\SHRUTI>python
Python 3.12.0 (tags/v3.12.0:0fb18b0,
Type "help", "copyright", "credits"
>>> list1=[1,2,3]
>>> list2=['a','b','c']
>>> zipped=zip(list1,list2)
>>> result=list(zipped)
>>> print(result)
[(1, 'a'), (2, 'b'), (3, 'c')]
>>>
```

In this example, zip() pairs the elements of list1 and list2 element by element, creating a list of tuples.

(3) map()

The map() function in Python is a built-in function used to apply a specified function to all the items in an iterable (eg., a list) and return an iterator that yields the results.

Syntax:

map(function, iterable)

function : This is the function that you want to apply to each element in the iterable.

iterable : This is the iterable (eg., a list, tuple, etc.) whose elements you want to apply the function to.

The map() function returns a map object, which is an iterator. You can convert it to a list or another iterable if needed.

Example:

```
def square(x):
    return x*x
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
result = list(squared_numbers)
print(result)
```

```
>>> def square(x):
...     return x*x
...
>>> numbers=[1,2,3,4,5]
>>> squared_numbers=map(square,numbers)
>>> result=list(squared_numbers)
>>> print(result)
[1, 4, 9, 16, 25]
>>>
```

In this example, the square() function is applied to each element in the numbers list using map(), and the result is obtained as a new list.

(4) reduce()

In Python, reduce is a built-in function that applies a given function to the elements of an iterable, reducing them to a single value.

Syntax:

functools.reduce (function, iterable [initializer])

- The function argument is a function that takes two arguments and returns a single value. The first argument is the accumulated value, and the second argument is the current value from the iterable.
- The iterable argument is the sequence of values to be reduced.
- The optional initializer argument is used to provide an initial value for the accumulated result. If no initializer is specified, the first element of the iterable is used as the initial value.

Example:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
product = reduce (lambda x, y : x * y, numbers)
print (product)
```

```
>>> from functools import reduce
>>> numbers=[1,2,3,4,5]
>>> product=reduce(lambda x,y:x*y,numbers)
>>> print(product)
120
>>>
```