

enumerate()

In python `enumerate()` function is used to iterate over a sequence (such as tuple or list) -

While keeping track of the index and the corresponding element.

→ The `enumerate()` Function is handy for situations where you need both the index and the corresponding elements during iteration.

Example:

```
my_list = ['apple', 'banana', 'orange']
```

```
for index, value in enumerate(my_list):
```

```
    print(f"Index: {index}, value: {value}")
```

```
1 my_list = ['apple', 'banana', 'orange']
2
3 for index, value in enumerate(my_list):
4     print(f"Index: {index}, Value:
5           {value}")
```

```
Index: 0, Value: apple  
Index: 1, Value: banana  
Index: 2, Value: orange
```

Syntax of enumerate:

~~~~~

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



reduce()

~~~~~

The reduce () Function is a powerful tool in python that operates on a list (or any iterable), applies a Function to it's elements and 'reduces' them to a single output
→ It is a part of the module, which needs to be imported before you can use reduce ()

→ reduce () stores the intermediate result and only returns the final summation value, whereas.

→ Reduce function, `reduce()` Function works with 3 parameters in python 3 as well as 5 For 2 parameters

→ The 'reduce' Function in python is part of the 'Functools' module.

→ It is used for cumulative operations on a Sequence applying a binary Function to the items & reducing them to a single accumulated result.

Example:

```
from functools import reduce
```

```
numbers = [1, 2, 3, 4, 5]
```

```
product = reduce(lambda x, y: x * y, numbers)
```

```
print(product)
```



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

120

> |

Syntax of reduce:

Syntax: `reduce (Function, iterable, [initializer])`

Map()

Map Function applies a specified Function to all item in a input iterable (e.g, list) & returns an iterators that produces the results.

→ we can also use 'map' with lambda functions for more concise expression.

→ map in python is a Function that works as an iterator to return a result after applying a function to every item of an iterable.

→ It is used when you want to apply a single transformation function to all the iterable elements.

→ The iterable and function are passed as arguments to the map in python.

Example:

```
numbers = [1, 2, 3, 4, 5]
```

```
result = map(lambda x: x**2, numbers)
```

```
result_list = list(result)
```

```
print(result_list)
```

```
[1, 4, 9, 16, 25]
```

```
1 numbers = [1, 2, 3, 4, 5]
2 result = map(lambda x: x ** 2, numbers)
3 result_list = list(result)
4
5 print(result_list)
```



```
[1, 4, 9, 16, 25]
```

Syntax of map():

Syntax: map (Fun, iter)

Filter()

Filter() is a built-in function in Python.

→ The Filter Function can be applied to an iterable such as a list or a dictionary and create a new iterator.

→ This new iterator can filter out certain specific elements based on the condition that you provide very efficiently.

→ The Filter() method creates a new array filled with elements that pass a test provided by a function.

→ The Filter() method does not execute the Function
For empty elements

→ The Filter() method does not change the original
array

→ Python Filter Function is used to Filter the given
Sequence by checking if each element is true or
not.

→ The true elements are presented and the elements
that are False or not do pass the condition given
are discarded with the help of this Function

Example: `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

`even_numbers = list(filter(lambda x: x % 2 ==`

`0, numbers)`

`print(even_numbers)`


```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2 even_numbers = list(filter(lambda x: x %
    2 == 0, numbers))
3 print(even_numbers)
4
```

```
[2, 4, 6, 8, 10]
```

```
> |
```

Syntax of Filter():

Syntax: `Filter (Function, iterable)`

zip()

→ The zip() Function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

→ If the passed iterables have different lengths the iterable with the least items decides the length of the new iterator.

→ zip is a built-in-built function in python used to iterate over multiple iterables.

→ It takes corresponding elements from all the iterable passed to it and merges them in a tuple.

→ The zip() method in python takes one or more arguments or iterables as input parameters and merges each of them element-wise to create a single iterable.

Example:

Example:

```
list1 = [1, 2, 3]
```

```
list2 = ['a', 'b', 'c']
```

```
zipped_result = zip(list1, list2)
```

```
for item in zipped_result:
```

```
    print(item)
```

```
1 list1 = [1, 2, 3]
2 list2 = ['a', 'b', 'c']
3
4 zipped_result = zip(list1, list2)
5
6 for item in zipped_result:
7     print(item)
```

(1, 'a')

(2, 'b')

(3, 'c')

Syntax of zip():

Syntax: (iterator1, iterator2,)

id()

- In python, the `id()` Function is a built-in Function that returns the unique identifier of a object.
- The identifier is an integer, which represents the memory address of the object.
- The `id()` Function is commonly used to check if two variables or objects refer to the same memory location.
- Python `id()` Function returns the "identity" of the object.
- The identity of an object is an integer, which is guaranteed to be unique and constant for this obj during lifetime.
- Two objects with non-overlapping lifetimes may have the same `id()` value.

```
1 x = 42
```

```
2 print(id(x))
```

```
3
```

1

42

Syntax of id():

Syntax: id(object 1)