1. enumerate( )

enumerate() is a built-in function in python that is used to iterate over a sequence (such as a list, tuple, or string) while keeping track of the index of the current item. It returns a tuple containing the index and the corresponding item from the sequence.

Syntax : enumerate(iterable, start=0)

Iterable : The sequence you want to iterate over.

Start : The index from which the counting should begin. The default if 0.

Example : fruits = ['apple', 'banana', 'orange']

```
for index, fruit in enumerate(fruits, start=1):
print(f"Index {index}: {fruit}")
```

2. Reduce( )

Reduce() is a function in Python that is part of the '**functools'** module. It is used for aggregating elements of a sequence, applying a binary function in a cumulative way to the items of an iterable, and reducing them to a single accumulated result. The typical use case is to successively apply a function to the items in the sequence, reducing the sequence to a single accumulated result.

Syntax : functools.reduce(function, iterable, initializer=None)

Function : The binary function that takes two arguments and is applied cumulatively to the items of the iterable.
It should have the form function(accumulator, element).

Iterable : The iterable sequence of items to be reduced.

Initializer : If provided, it serves as the initial value for the accumulator. If not provided, the first two elements of the iterable are used as the initial values.

Example :

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

3. map( )

map() is a higher-order function that applies a given function to all items in an iterable (e.g., a list, tuple, or dictionary) and returns a new iterable with the results.

Syntax : map(function, iterable, ...)

Function : The function to apply to each item in the iterable.
Iterable : The iterable (e.g., list, tuple) whose elements will be processed by the function.

Example :
```
def square(x):
    return x ** 2

numbers = [1, 2, 3, 4, 5]
# Using map() to square each element in the list
squared_numbers = map(square, numbers)
# Converting the result to a list
result_list = list(squared_numbers)
print(result_list)
```

4. filter( )

In Python, filter() is a built-in function that is used to filter elements of an iterable (such as a list) based on a specified function. The filter() function takes two arguments:

I.A function: This function defines the filtering condition. It should return True for the elements that you want to keep and False for the elements you want to discard.
II.An iterable: This is the sequence of elements that you want to filter.

Syntax : filter(function, iterable)

Example :
```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def is_odd(n):
    return n % 2 != 0
filtered_numbers = filter(is_odd, numbers)
# Convert the iterator to a list (in Python 3, filter() returns an iterator)
result = list(filtered_numbers)
print(result)
```

5. zip( )

In Python, **zip()** is a built-in function that is used to combine elements from multiple iterables (e.g., lists, tuples) into tuples. It takes two or more iterables as input and returns an iterator of tuples where the i-th tuple contains the i-th element from each of the input iterables.

Syntax : zip(iterable1, iterable2, ...)

Example :
```
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30]
zipped_data = zip(names, ages)
result_list = list(zipped_data)
print(result_list)
# Output: [('Alice', 25), ('Bob', 30)]
```

6. id( )

In Python, the **id()** function is a built-in function that returns the identity of an object. The identity of an object is a unique integer that is guaranteed to be unique and constant for the object during its lifetime. Two objects with non-overlapping lifetimes may have the same **id()** value, but the **id()** value of an object will never be reused for another object.

Syntax : id(object)

Example :
```
x = 42
y = "hello"
print(id(x))  # Output: a unique integer representing the identity of x
print(id(y))  # Output: a unique integer representing the identity of y
```