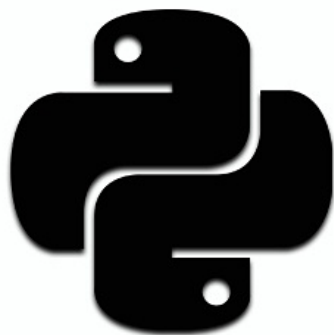


PART - 1

PYTHON ITERTOOLS

Python's Itertool is a module that provides various functions that work on iterators to produce complex iterators. This module works as a fast, memory-efficient tool that is used either by themselves or in combination to form iterator algebra.

[Let's Go ➡](#)

Different types of iterators provided by this module are:

1. Infinite iterators
2. Combinatoric iterators
3. Terminating iterators


Infinite iterators

Iterator in Python is any Python type that can be used with a 'for in loop'. Python lists, tuples, dictionaries, and sets are all examples of inbuilt iterators. But it is not necessary that an iterator object has to exhaust, sometimes it can be infinite. Such type of iterators are known as Infinite iterators.

1. count(start, step): This iterator starts printing from the "start" number and prints infinitely. If steps are mentioned, the numbers are skipped else step is 1 by default.

```
import itertools

# for in loop
for i in itertools.count(5,5):
    if i==35:
        break
    else:
        print(i,end=" ")
```




5 10 15 20 25 30



2. cycle(iterable): This iterator prints all values in order from the passed container. It restarts printing from the beginning again when all elements are printed in a cyclic manner.

```
import itertools


count=0
# for in loop
L = ['PYTHON', 'JAVA']
for i in itertools.cycle(L):
    if count>7:
        break
    else:
        print(i,end=" ")
        count+=1
```



```
PYTHON JAVA PYTHON JAVA PYTHON JAVA
PYTHON JAVA
```

3. repeat(val, num): This iterator repeatedly prints the passed value infinite number of times. If the optional keyword num is mentioned, then it repeatedly prints num number of times.

```
import itertools
# using repeat() to repeatedly print number
print("Printing the numbers repeatedly : ")
print(list(itertools.repeat(20,4)))
```



```
[20, 20, 20, 20]
```



Combinatoric iterators

The recursive generators that are used to simplify combinatorial constructs such as permutations, combinations, and Cartesian products are called combinatoric iterators.


1. Product(): This tool computes the cartesian product of input iterables. To compute the product of an iterable with itself, we use the optional repeat keyword argument to specify the number of repetitions.

```
from itertools import product

print("The cartesian product using repeat:")
print(list(product([1,2],repeat=2)))
print()

print("The cartesian product of the containers:")
print(list(product(['@_PYTHON.PY_', 'AVII'],'2')))
print()

print("The cartesian product of the containers:")
print(list(product('AB',[3,4])))
```



```
The cartesian product using repeat:
[(1, 1), (1, 2), (2, 1), (2, 2)]
```

```
The cartesian product of the containers:
[('@_PYTHON.PY_', '2'), ('AVII', '2')]
```

```
The cartesian product of the containers:
[('A', 3), ('A', 4), ('B', 3), ('B', 4)]
```




2. Permutations(): Permutations() as the name speaks for itself is used to generate all possible permutations of an iterable. All elements are treated as unique based on their position and not their values. This function takes an iterable and group_size, if the value of group_size is not specified or is equal to None then the value of group_size becomes length of the iterable.

```
from itertools import permutations

print("All the permutations of the given list is:")
print(list(permutations([1, '@_python.py_'], 2)))
print()
#Terminating iterators
print("All the permutations of the given string is:")
print(list(permutations('AB')))
print()

print("All the permutations of the given container is:")
print(list(permutations(range(3), 2)))
```



```
All the permutations of the given list is:
[(1, '@_python.py_'), ('@_python.py_', 1)]

All the permutations of the given string is:
[('A', 'B'), ('B', 'A')]

All the permutations of the given container is:
[(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
```



3. Combinations(): This iterator prints all the possible combinations(without replacement) of the container passed in arguments in the specified group size in sorted order.

```
from itertools import combinations

print("All the combination of list in sorted order(without replacement) is:")
print(list(combinations(['A',2],2)))
print()

print("All the combination of string in sorted order(without replacement) is:")
print(list(combinations('AB',2)))
print()

print("All the combination of list in sorted order(without replacement) is:")
print(list(combinations(range(2),1)))
```



```
All the combination of list in sorted order(without replacement) is:
[('A', 2)]

All the combination of string in sorted order(without replacement) is:
[('A', 'B')]

All the combination of list in sorted order(without replacement) is:
[(0,), (1,)]
```



4. Combinations_with_replacement(): This function returns a subsequence of length n from the elements of the iterable where n is the argument that the function takes determining the length of the subsequences generated by the function. Individual elements may repeat itself in combinations_with_replacement function.

```
from itertools import combinations_with_replacement

print("All the combination of string in sorted order(with replacement) is:")
print(list(combinations_with_replacement("AB",2)))
print()

print("All the combination of list in sorted order(with replacement) is:")
print(list(combinations_with_replacement([1,2],2)))
print()

print("All the combination of container in sorted order(with replacement) is:")
print(list(combinations_with_replacement(range(2),1)))
```

All the combination of string in sorted order(with replacement) is:
[('A', 'A'), ('A', 'B'), ('B', 'B')]

All the combination of list in sorted order(with replacement) is:
[(1, 1), (1, 2), (2, 2)]

All the combination of container in sorted order(with replacement) is:
[(0,), (1,)]

