Python List Comprehensions

In programming, you often need to transform elements of a list and returns a new list

suppose that you have a list of five numbers like this:

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

And you want to get a list of squares based on this numbers list

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

list = []
for i in l:
    list.append(i*i)
    # list.append(i**2)

print(list)
```

Python map() function

syntax of the map() function:

iterator = map(fn, list)

fn is the name of the function that will call on each element of the list.

In fact, you can pass any iterable to the map() function, not just a list or tuple.

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

def get(list):
    return list**2

value=map(get,l)

for i in value:
    print(i)
```

lambda function using

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

value=map(lambda list:list**2,l)
print(value)

for i in value:
    print(i)
```

Another Example

uses the map() function to returns a new list where each element is transformed into the proper case:

```
l=['gautam','vivek','shailesh']

l1=map(lambda list:list.capitalize(),l)
print(list(l1))
```

the map() function returns an iterator, you need to use the list() function to convert the iterator to a list.

list comprehensions.

```
list = [1, 2, 3, 4, 5]
list1 = [i**2 for i in list]

print(list1)
```

A list comprehension consists of the following parts:

- An input list (list)
- A variable that represents the elements of the list (i)
- An output expression (i**2) that returns the elements of the output list from the elements of the input list.

[output_expression for element in list]

Filter() Function

filter() function in python extracts only particular elements from an iterable (like list, tuple, dictionary, etc.)

It takes a function and an iterable as arguments and applies the function passed as an argument to each element of the iterable; once this process of filtering is completed, it returns an iterator as a result.

An iterable is a Python object that can be iterated over.

filter(function, iterable)

The filter() function takes two parameters:

- function: used to check if each element of the iterable holds true for this function or not.
- iterable: iterables like sets, lists, tuples etc. on which filtering will be done.

```
nums = [10, 3, 192, 55, 20, 77 , 91]

def divisible(i):
    return True if i%5==0 else False

v = filter(divisible, nums)

print(list(v))
```

Reduce Function

reduce() function in python performs functional computation by taking a function and an iterable (e.g., list, tuple, dictionary, etc.) as arguments, and the result is returned after computation (the process of applying the function on the iterable).

The reduce() function in python is defined in the **functools** module and doesn't return multiple values or any iterator, it just returns a single value as output which is the result of the whole iterable getting reduced to only a single integer or string or boolean.

reduce() function in python is a part of the functools module, which has to be imported before calling the function in our program. This single value output means that after applying the reduce function on the iterable, only a single integer or string, or boolean is returned.

functools.reduce(function, iterable)

- The first argument in reduce() is a function. This function will be applied to all the elements in an iterable in a cumulative manner to compute the result.
- The second argument is iterable. Iterables are those python objects that can be iterated/looped over, includes like lists, tuples, sets, dictionaries, generators, iterator, etc.

Steps of how to reduce function works in python:

- The function passed as an argument is applied to the first two elements of the iterable.
- After this, the function is applied to the previously generated result and the next element in the iterable.
- This process continues until the whole iterable is processed.
- The single value is returned as a result of applying the reduce function on the iterable.

```
from functools import reduce

nums = [1, 2, 3, 4]

def sum(x,y):
    return x+y
ans = reduce(sum, nums)
print(ans)
```

using lambda function

```
nums = [1, 2, 3, 4]


ans = reduce(lambda x, y: x + y, nums)
print(ans)
```

Product of number

```
def product(x,y):
    return x*y

ans = reduce(product, [2, 5, 3, 7])
print(ans)
```