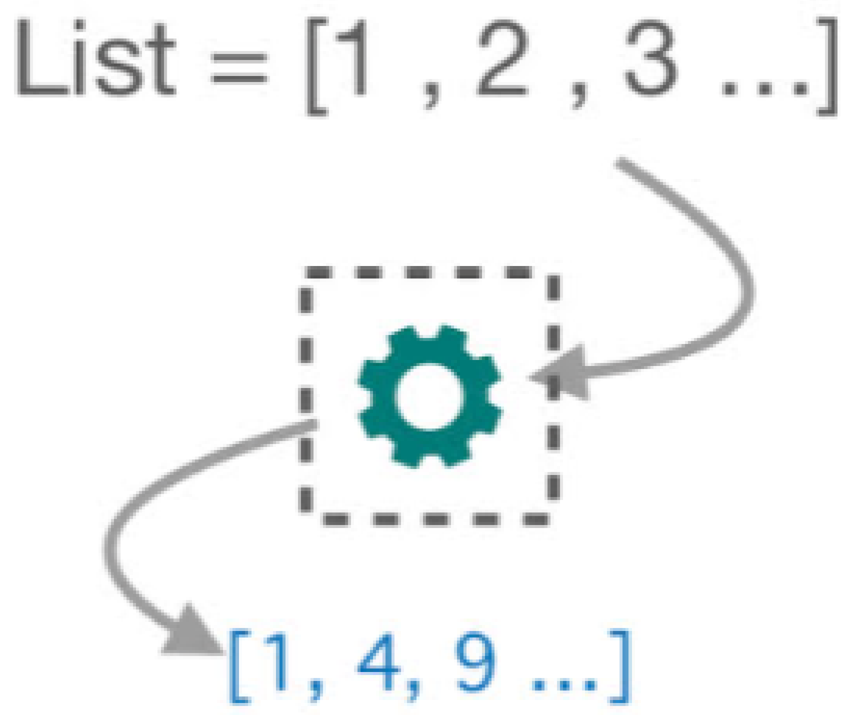
Python

Besant Technologies

Map, Filter and Reduce Functions in Python

Python map( ) Function



Python map() function

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)
Syntax :

```
map(fun, iter)
```

Parameters :

> **fun :** It is a function to which map passes each element of given iterable. **iter** : It is a iterable which is to be mapped.

NOTE : You can pass one or more iterable to the map() function. **Returns :**

```
Returns a list of the results after applying the given function

to each item of a given iterable (list, tuple etc.)
```

```python
# Python program to demonstrate working
# of map.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

Output:

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

CODE 2 We can also use lambda expressions with map to achieve above result

Python3

```python
# Double all numbers using map and lambda

numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))
```

Output:

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

CODE 3

Python3

```python
# Add two lists using map and lambda
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]
result = map(lambda x, y: x + y, numbers1, numbers2)
print(list(result))
```

Output:

```
[5, 7, 9]
```

CODE 4

Python3

```python
# List of strings
l = ['sat', 'bat', 'cat', 'mat']

# map() can listify the list of strings individually
test = list(map(list, l))
print(test)
```

Output:

```
[['s', 'a', 't'], ['b', 'a', 't'], ['c', 'a', 't'], ['m', 'a', 't']]
```

Time complexity: O(n), where n is the number of elements in the input list l.

Auxiliary space: O(n), as the resulting list test contains n sublists, each of which contains a single character.

using if statement with map():

In the example, the double_even() function doubles even numbers and leaves odd numbers unchanged. The map() function is used to apply this function to each element of the numbers list, and an if statement is used within the function to perform the necessary conditional logic.

Time complexity analysis:

The map function applies the double_even function to each element of the list. The time complexity of the map function is O(n), where n is the number of elements in the list. The time complexity of the double even function is constant, O(1), since it only performs a single arithmetic operation and a comparison. Therefore, the overall time complexity of the program is O(n).

Space complexity analysis: The program uses a list to store the result of the map function, so the space complexity is proportional to the number of elements in the list. In the worst case, if all elements are even, the resulting list will have the same number of elements as the input list. Therefore, the space complexity is O(n) in the worst case.

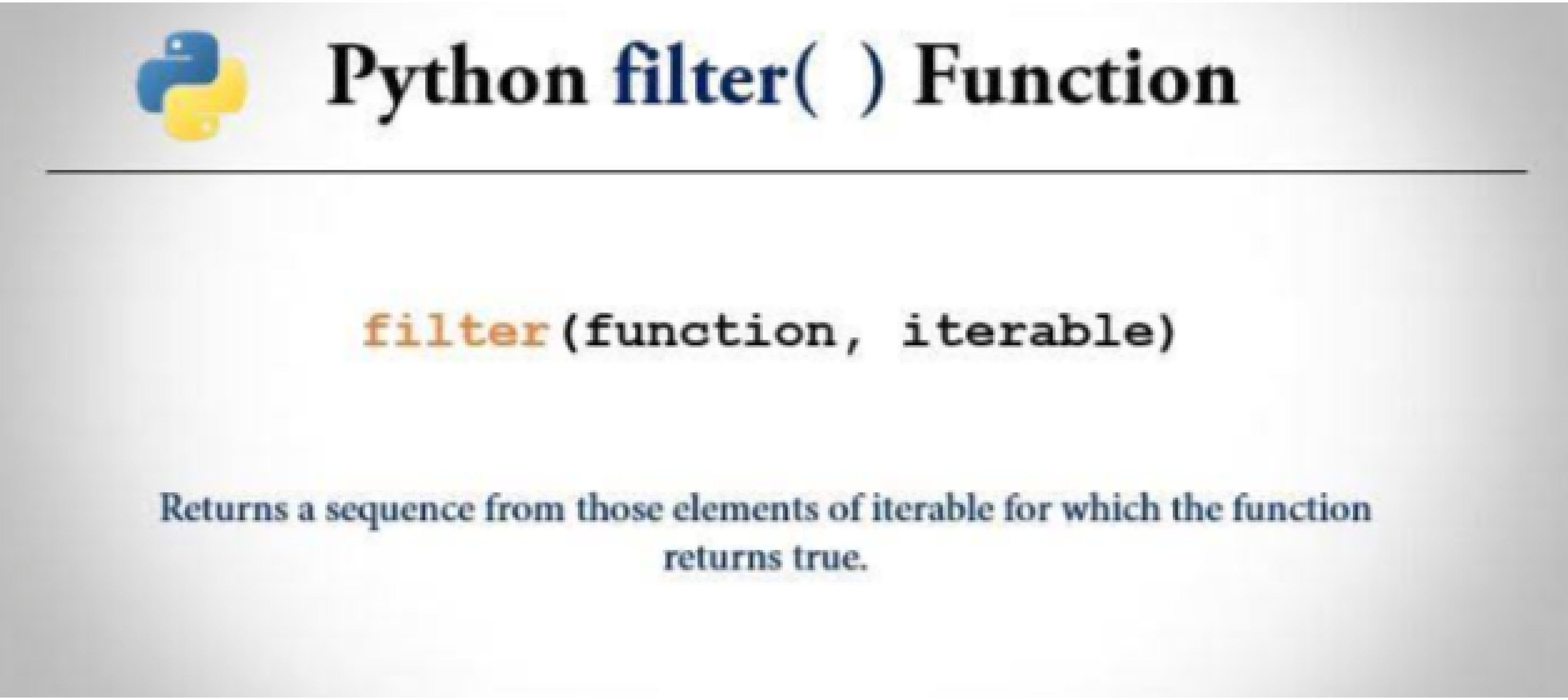
```python
# Define a function that doubles even numbers and leaves odd numbers as is
def double_even(num):
    if num % 2 == 0:
        return num * 2
    else:
        return num

# Create a list of numbers to apply the function to
numbers = [1, 2, 3, 4, 5]

# Use map to apply the function to each element in the list
result = list(map(double_even, numbers))

# Print the result
print(result)  # [1, 4, 3, 8, 5]
```

Output

```
[1, 4, 3, 8, 5]
```

Time complexity: O(n), where n is the length of the input list. The map() function applies the double_even() function to each element in the list, which takes constant time. Therefore, the overall time complexity is proportional to the length of the input list.

Auxiliary space complexity: O(n), where n is the length of the input list. The map() function creates a new list to store the output, which takes up space proportional to the length of the input list. Therefore, the auxiliary space complexity is also proportional to the length of the input list.



Python filter( ) Function

```
filter(function, iterable)
```

Returns a sequence from those elements of iterable for which the function returns true.

filter() in python

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not

syntax:

```
filter(function, sequence)

Parameters:

function: function that tests if each element of a

sequence true or not.

sequence: sequence which needs to be filtered, it can

be sets, lists, tuples, or containers of any iterators.

Returns:

returns an iterator that is already filtered.
```

```python
# function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False

# sequence
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

Output:

```
The filtered letters are:

e

e
```

```python
# a list contains both even and odd numbers.
seq = [0, 1, 2, 3, 5, 8, 13]

# result contains odd numbers of the list
result = filter(lambda x: x % 2 != 0, seq)
print(list(result))

# result contains even numbers of the list
result = filter(lambda x: x % 2 == 0, seq)
print(list(result))
```

Output:

```
[1, 3, 5, 13]

[0, 2, 8]
```

python filter() functions:

In this program, the is_multiple_of_3() function checks if a number is a multiple of 3. The filter() function is used to apply this function to each element of the numbers list, and a for statement is used within the lambda function to iterate over each element of the list before applying the condition. This way, we can perform additional operations on each element before applying the condition.

```python
#Define a function to check if a number is a multiple of 3
def is_multiple_of_3(num):
    return num % 3 == 0
# Create a list of numbers to filter
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Use filter and a lambda function to filter the list of numbers
# and only keep the ones that are multiples of 3
result = list(filter(lambda x: is_multiple_of_3(x), numbers))
# Print the result
print(result)  # [3, 6, 9]
```

Output

```
[3, 6, 9]
```

Time complexity analysis:
1. The filter function is used to filter the list of numbers, and it applies the lambda function to each element of the list. The time complexity of the filter function is O(n), where n is the number of elements in the list.
2. The time complexity of the lambda function is constant, O(1), since it only performs a single arithmetic operation. Therefore, the overall time complexity of the program is O(n).

Auxiliary Space analysis:
The program uses a list to store the filtered numbers, so the space complexity is proportional to the number of filtered numbers. In the worst case, if all numbers are multiples of 3, the filtered list will have n/3 elements. Therefore, the space complexity is O(n/3), which simplifies to O(n) in big O notation.

Python

Higher Order Function

reduce ( ) Function

reduce() in Python

The **reduce(fun,seq)** function is used to **apply a particular function passed in its argument to all of the list elements** mentioned in the sequence passed along.This function is defined in “**functools**” module.

Working :

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

Python3

```
# python code to demonstrate working of reduce()

# importing functools for reduce()
import functools

# initializing list
lis = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a, b: a+b, lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lis))
```

Output

```
The sum of the list elements is : 17

The maximum element of the list is : 6
```

Using Operator Functions

reduce() can also be combined with operator functions to achieve the similar functionality as with lambda functions and makes the code more readable.

Python3

```python
# python code to demonstrate working of reduce()
# using operator functions

# importing functools for reduce()
import functools

# importing operator for operator functions
import operator

# initializing list
lis = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
# using operator functions
print("The sum of the list elements is : ", end="")
print(functools.reduce(operator.add, lis))

# using reduce to compute product
# using operator functions
print("The product of list elements is : ", end="")
print(functools.reduce(operator.mul, lis))

# using reduce to concatenate string
print("The concatenated product is : ", end='')
print(functools.reduce(operator.add, ["geeks", "for", "geeks"]))
```

Output

```
The sum of the list elements is : 17

The product of list elements is : 180

The concatenated product is : geeksforgeeks
```

reduce() vs accumulate()

Both reduce() and accumulate() can be used to calculate the summation of a sequence elements. But there are differences in the implementation aspects in both of these.

- reduce() is defined in “functools” module, accumulate() in “itertools” module.
- reduce() stores the intermediate result and only returns the final summation value. Whereas, accumulate() returns a iterator containing the intermediate results. The last number of the iterator returned is summation value of the list.
- reduce(fun, seq) takes function as 1st and sequence as 2nd argument. In contrast accumulate(seq, fun) takes sequence as 1st argument and function as 2nd argument.

Python3

```python
# python code to demonstrate summation
# using reduce() and accumulate()

# importing itertools for accumulate()
import itertools

# importing functools for reduce()
import functools

# initializing list
lis = [1, 3, 4, 10, 4]

# printing summation using accumulate()
print("The summation of list using accumulate is :", end="")
print(list(itertools.accumulate(lis, lambda x, y: x+y)))

# printing summation using reduce()
print("The summation of list using reduce is :", end="")
print(functools.reduce(lambda x, y: x+y, lis))
```

Output

```
The summation of list using accumulate is :[1, 4, 8, 18, 22]

The summation of list using reduce is :22
```

Map, filter and reduce

Python provides this as

- Map – transform everything in the iterable
- Filter – filter those that return the value true
- Reduce – a two argument function

```
ls1 = map(lambda x: x +2, a)
ls2 = filter(lambda x: x%2 == 0, a)
ls3 = reduce(lambda x, y : x+y, a )
```

Key characteristics	Map	Filter	Reduce
Syntax	map(function, iterable object)	filter(function, iterable object)	reduce(function, iterable object)
Effect of the function on the iterable	Applies the function evenly to all the items in the iterable object	Function is a boolean condition that rejects all the items in the iterable object that are not true	Breaks down the entire process of applying the function into pair-wise operations
Input to output variables	N to N	N to M where N>=M	N to 1
Use Case	Transformation/Mapping	Splitting the data	Single output operations
Example	List of the square of all numbers in a list	All even numbers from a list	Product of all the items in the list