Functional Programming

In Python, an anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

How to use lambda Functions in Python?

A lambda function in python has the following syntax.

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

Example of Lambda Function in python

double = lambda x: x * 2

print(double(5))

1.Write a function to print cube of a number

2.Write a python function to reverse a string with uppercase.

3.Write a python function to find max of two numbers.

4.Write a function to sort a list based on the scores

```
[('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]
```

In-built functions :

a.filter

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

Here is an example use of filter() function to filter out only even numbers from a list.

5.Program to filter out only the even items from a list

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))

print(new_list)

6. Filter all people having age more than 18, using lambda and filter() function

7.Write a python program to filter a number which is divisible by both 2 and 3

8. Given a list of strings, return a list containing all strings with three consecutive characters “**a**” in it.user_list = ["asfdc", "aabbaaac", "cdfdccc", "baaab", "baab"]

map()

The map() function in Python takes in a function and a list.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Here is an example use of map() function to double all the items in a list.

9.Program to double each item in a list using map()

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(map(lambda x: x * 2 , my_list))

print(new_list)

10.Write a program that reads the sentence and gives the output as the length of each word in a sentence in the form of a list.

11.Write a program to add each element of two lists

reduce()

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

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

With list comprehension you can do all that with only one line of code:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

newlist = [*expression* for *item* in *iterable* if *condition* == True]

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to True.

Example

Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

The condition if x != "apple" will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

Example

With no if statement:

newlist = [x for x in fruits]

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

Same example, but with a condition:

Example

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]

You can set the outcome to whatever you like:

Example

Set all values in the new list to 'hello':

newlist = ['hello' for x in fruits]

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "orange" instead of "banana":

newlist = [x if x != "banana" else "orange" for x in fruits]

The *expression* in the example above says:

"Return the item if it is not banana, if it is banana return orange".