Functional features

Assignment Date	18 november 2022
Student Name	k.ananthavalli
Student Roll Number	815819104004
Maximum Marks	2 Marks

Functional Programming in Python Functional programming is a programming paradigm in which we try to bind everything pure mathematical functions style. It is a declarative type of programming style. Its main focus is on “what to solve” in contrast to an imperative style where the main focus is “how to solve“. It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables.

Example:

Python program to demonstrate # pure functions

A pure function that does Not # changes the input list and # returns the new List def pure_func(List):

New_List = []

for i in List: New_List.append(i**2)

return New_List

Driver's code Original_List = [1, 2, 3, 4] Modified_List = pure_func(Original_List)

print("Original List:", Original_List) print("Modified List:", Modified_List)

Output:

Original List: [1, 2, 3, 4] Modified List: [1, 4, 9, 16]

Recursion

During functional programming, there is no concept of for loop or while loop, instead recursion is used. Recursion is a process in which a function calls itself directly or indirectly. In the recursive program, the solution to the base case is provided and the solution to the bigger problem is expressed in terms of smaller

problems. A question may arise what is base case? The base case can be considered as a condition that tells the compiler or interpreter to exit from the function.

Example:

Let’s consider a program that will find the sum of all the elements of a list without using any for loop.

Python program to demonstrate # recursion

Recursive Function to find # sum of a list def Sum(L, i, n, count):

Base case if n <= i: return count

count += L[i]

Going into the recursion count = Sum(L, i + 1, n, count)

return count

Driver's code L = [1, 2, 3, 4, 5] count = 0 n = len(L) print(Sum(L, 0, n, count))

Output:

15

Functions are First-Class and can be Higher-Order
First-class objects are handled uniformly throughout. They may be stored in data structures, passed as arguments, or used in control structures. A programming language is said to support first-class functions if it treats functions as first-class objects.

Properties of first class functions:

● A function is an instance of the Object type. ● You can store the function in a variable. ● You can pass the

function as a parameter to another function. ● You can return the function from a function. ● You can store

them in data structures such as hash tables, lists, …

Python program to demonstrate # higher order functions

def shout(text): return text.upper()

def whisper(text): return text.lower()

def greet(func): # storing the function in a variable greeting = func("Hi, I am created by a function passed as

an argument.") print(greeting)

greet(shout) greet(whisper)

Output:

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT. hi, I am created by a function passed

as an argument.

Map(): map() function returns a list 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. Return Type: Returns an iterator of map class. Example:

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) results = map(addition, numbers)

Does not Print the value print(results)

For Printing value for result in results: print(result, end = " ")

Output: <map object at 0x7fae3004b630> 2 4 6 8

Note: For more information, refer to Python map() function

filter(): 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: a 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. Return Type: returns an iterator that is already filtered. Example:

Python program to demonstrate working # of the filter.

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 code to demonstrate # lambda

cube = lambda x: x * x*x print(cube(7))

L = [1, 3, 2, 4, 5, 6] is_even = [x for x in L if x % 2 == 0]

print(is_even)

Output: 343 [2, 4, 6]

Example:

Python program to demonstrate # immutable data types

String data types immutable = "GeeksforGeeks"

changing the values will # raise an error immutable[1] = 'K'

Output:

Traceback (most recent call last): File "/home/ee8bf8d8f560b97c7ec0ef080a077879.py", line 10, in

immutable[1] = 'K' TypeError: 'str' object does not support item assignment