Functional Programming in Python

Functional programming is a programming paradigm in which we try to bind everything in 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.

Concepts of Functional Programming

Any Functional programming language is expected to follow these concepts.

- **Pure Functions:** These functions have two main properties. First, they always produce the same output for the same arguments irrespective of anything else. Secondly, they have no side-effects i.e. they do modify any argument or global variables or output something.
- **Recursion:** There are no “for” or “while” loop in functional languages. Iteration in functional languages is implemented through recursion.
- **Functions are First-Class and can be Higher-Order:** First-class functions are treated as first-class variable. The first-class variables can be passed to functions as a parameter, can be returned from functions or stored in data structures.

Functional Programming in Python

Python too supports Functional Programming paradigms without the support of any special features or libraries.

Pure Functions

As Discussed above, pure functions have two properties.

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

print("Original List:", Original_I
print("Modified List:", Modified_I
```

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.

Note: For more information, refer Recursion

Example: Let's consider a program that

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)
```

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.

- 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 va
    greeting = func("Hi, I am crea
    print(greeting)

greet(shout)
greet(whisper)
```

Output:

```
HI, I AM CREATED BY A FUNCTION PASSED AS

hi, I am created by a function passed as
```

Note: For more information, refer to First Class functions in Python.

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/ee8bf8d8f560b97c7ec0ef080a
    immutable[1] = 'K'
TypeError: 'str' object does not support
```