**Task#9**

A different approach for data collection and manipulation is used for both functional programming and object-oriented programming. Data can not be contained in objects in functional programming and it can only be converted by constructing functions. Data is stored into objects in object-oriented programming.

Functional programming says that, for consistency, data and behavior are distinctly different and should be kept apart. In comparison, OOP claims that it is easier to explain how a program operates by putting together data and actions in a single location.

The main distinction between the two approaches is that functional programming emphasizes on what we are doing, while the emphasis on how we are doing it is object-oriented. Although functional programming allows high performance for applications in the processing of large data, an object-oriented application cannot be used for the processing of big data.

|  |  |
| --- | --- |
| Functional | Object Oriented Programming (OOP) |
| Uses PURE functions | Encapsulates functions using classes |
| Avoids joint states and  mutable data (uses immutable) | State is stored in the class properties(uses mutable data) |
| Declarative | Imperative |
| Concise and easier to test | Conceived to take advantage of  Tricks for programming to be testable  as a "injection of dependence" |
| Functional programming emphasizes on functon evoluation | Based on concept of objects |
| Supports Parallel programming | Does not support parallel programming |
| The statements might be executed in an any order | In oops, statements should be executed in particular order |

Example:

In **OOP**, we have the flexibility to represent real world objects(animals, cars, person etc). Overall, an object is something that possess some of the characteristics and can-do certain functions.

class bestcar:

# attributes

year = 1999 # car model's year

mpg = 25 # mileage

speed = 255 # today` speed

# methods

def faster(self):

return car.speed + 20

def slower(self):

return car.speed - 50

As we stated earlier, functional programming is pure.

We may start coding in a more practical way without any special Python features or libraries. If you would like functions to be pure, then do not alter the input value or other data that occurs beyond the meaning of the function.

This makes it far smoother to test the work we write. As it doesn't change the state of any element, any time we run the function with the same input, we are guaranteed to get the same output.

Let's create a pure-function to multiply numbers by3:

def multiply\_3\_pure(numbers): #we will create a function which multiplies numbers by 3

new\_numbers = [] #it is empty for now

for n in numbers:

new\_numbers.append(n \* 3) #now, 3 will be multiplied to each of the number in the list

return new\_numbers

original\_numbers = [1, 3, 5, 10] #original set of numbers

changed\_numbers = multiply\_3\_pure(original\_numbers)

print(original\_numbers) *# [1, 3, 5, 10]*

print(changed\_numbers) *# [3, 9, 15, 30]*

The original set of numbers is unchanged, and outside of the function, we do not apply to any other variables, so it is pure.