Functional Programming

Functional programming is a style of programming *based on evaluation of expressions built by composition of (typically pure) functions over immutable data*.

Functional programming often uses *recursion*, and depends on functions being *first-class data values*. (*First-class data values* are values that can be passed as arguments, stored in variables and data structures, or returned by functions.)

Imperative vs. Functional Programming

C++ and Java were designed for object-oriented ***imperative*** programming.

In ***imperative*** programming we write code that is thought of as specifying a ***sequence of actions*** that the machine is to perform when the code is executed.

Imperative vs. Functional Programming

C++ and Java were designed for object-oriented ***imperative*** programming.

In ***imperative*** programming we write code that is thought of as specifying a ***sequence of actions*** that the machine is to perform when the code is executed.

The specified actions often ***update values*** stored in variables, data structures, and components of objects.

Functional programming is different.
In pure functional programming:

- The code ***isn't*** thought of as specifying a sequence of actions: We think of it as specifying ***what*** is to be computed rather than ***how*** the computation is to be done.

  Note: Writing code that's thought of as specifying the ***effect*** we want to achieve rather than how to achieve it is called ***declarative programming***: Functional programming is an example; other examples are database query programming (using, e.g., SQL) and logic programming (using, e.g., Prolog).

Imperative vs. Functional Programming

C++ and Java were designed for object-oriented ***imperative*** programming.

In ***imperative*** programming we write code that is thought of as specifying a ***sequence of actions*** that the machine is to perform when the code is executed.

The specified actions often ***update values*** stored in variables, data structures, and components of objects.

Functional programming is different.
In pure functional programming:

- The code ***isn't*** thought of as specifying a sequence of actions: We think of it as specifying ***what*** is to be computed rather than ***how*** the computation is to be done.
- Execution of the code will ***never update*** values stored in variables, data structures, and components of objects.

Functional programming is different.
In pure functional programming:

- The code ***isn't*** thought of as specifying a sequence of actions: We think of it as specifying ***what*** is to be computed rather than ***how*** the computation is to be done.
- Execution of the code will ***never update*** values stored in variables, data structures, and components of objects.

A common way to state the second property above is to say that, *in pure functional programming, variables, data structures, and objects are* ***immutable***:

- Once a variable has been given a value, its value *stays the same for as long as the variable exists*.
- Once a data structure or object has been created, values stored in its components *stay the same for as long as the data structure/object exists*.

Functional programming is a style of programming in which the code we write consists of:

- Definitions of *functions that have* ***no side-effects***.
- Expressions that call such programmer-defined functions or call library functions that have *no side-effects*.

We say a function f ***has no side-effects*** if a call of f does nothing except return a value, which implies:

- On return from any call of f, the values stored in variables, data structures, and object components are exactly the same as they were before that call of f.
- This implies execution of f doesn't initialize variables that can be used after f returns (though f may return a new or an existing data object).
- Execution of f does ***not*** do any I/O.
- Execution of f does ***not*** throw an exception.