Functional Programming Fundamentals

- Brief history of functional programming
- The benefits of functional Programming
- Introducing functional programming

Brief history of functional programming

Rise

Lambda calculus was introduced by the mathematician Alonzo Church in the 1930s.

Lisp was created in 1958, and was heavily influenced by lambda calculus.

Brief history of functional programming

- 1951 – Regional Assembly Language
- 1952 – Autocode
- 1954 – (forerunner to LISP)
- 1955 – FLOW-MATIC (led to COBOL)
- 1957 – FORTRAN (first compiler)
- 1957 – COMTRAN (precursor to COBOL)
- 1958 –

AutoLisp is used in Autocad

Brief history of functional programming

Fall

Somewhere between 1970 and 1980, the way that software was composed drifted away from simple algebraic math to procedure(C) and OO language(C++,Java)

> For most of us, creating software was a bit of a nightmare for 30 years. Dark times.

---Composing Software by Eric Elliott

Brief history of functional programming

Rise

Around 2010, something great began to happen: JavaScript exploded.

Before about 2006, JavaScript was widely considered a toy language used to make cute animations happen in web browsers.

It is functional feature such as lambda calculus make it powerful.

List of functional programming languages

- Lisp
- Clojure
- Erlang
- Haskell
- F#
- Javascript
- Java
- ……

Programming paradigms: imperative

Imperative in which the programmer instructs the machine how to change its state:

- which groups instructions into procedures,
- which groups instructions together with the part of the state they operate on.

Programming paradigms: declarative

in which the programmer merely declares properties of the desired result, but not how to compute it:

- in which the desired result is declared as the value of a series of function applications,
- in which the desired result is declared as the answer to a question about a system of facts and rules,
- in which the desired result is declared as the solution of an optimization problem

Introducing functional programming

In a programming paradigm such as OOP, the main building blocks that we use to create an application are objects (objects are declared using classes).

In FP, we use functions as the main building block in our applications.

Introducing functional programming

> makes code understandable by encapsulating moving parts. makes code understandable by minimizing moving parts.

-–– Michael Feathers

The benefits of functional programming

- Code is simple
- Code is testable
- Code is easy to reason about
- Concurrency
- Simpler caching

The benefits of functional programming

A functional programmer is an order of magnitude more than its conventional counterpart ---Why Functional Programming Matters by John Hughes

:functional and nonfunctional

Pure functions

A pure function is a function where the return value is only determined by its input values, without observable side effects.

An impure permits something to happen in a dark corner rather than along the input/output boundaries.   : pure.ts

Pure functions : Same Input, Same Output

Pure functions : Same Input, Same Output

Pure functions : No Side Effects

A function is said to have a side effect if it modifies some state variable value(s) outside its local environment, that is to say has an observable effect besides returning a value (the main effect) to the invoker of the operation.

Pure functions : No Side Effects

Pure functions : No Side Effects

Pure functions : No Side Effects

> It is not that we're forbidden to use them, rather we want to contain them and run them in a controlled way.

---drboolean. mostly-adequate-guide (old)

Pure functions : Immutability

Immutability refers to the inability to change the value of a variable after a value has been assigned to it.

How to design Immutable class/datatype?

Pure functions: Cacheable

pure functions can always be cached by input

Pure functions: Portable / Self-Documenting

Pure functions are completely self contained, a pure function's dependencies are explicit and therefore easier to see and understand - no funny business going on under the hood.

Pure functions: Portable / Self-Documenting

Impure

Pure functions: Portable / Self-Documenting

Pure

Pure functions : Testable

Pure functions make testing much easier. We don't have to mock a "real" payment gateway or setup and assert the state of the world after each test. We simply give the function input and assert output.

Pure functions : Reasonable

If the program is expressed in a mathematical form, then each stage of the computation can be described by a formula. Programs can be verified — proved correct — or derived from a specification.

Pure functions : Parallel Code

we can run any pure function in parallel since it does not need access to shared memory and it cannot, by definition, have a race condition due to some side effect.

: firstclass.js

Referential transparency

In functional programming, referential transparency is generally defined as the fact that an expression, in a program, may be replaced by its value (or anything having the same value) without changing the result of the program.

Stateless versus stateful

means the computer or program keeps track of the state of interaction, usually by setting values in a storage field designated for that purpose.

means there is no record of previous interactions and each interaction request has to be handled based entirely on information that comes with it.

Stateless versus stateful

The more pieces of your program are stateless, the more ways there are to put pieces together without having anything break. The power of the stateless paradigm lies not in statelessness (or purity), but the ability it gives you to write powerful, reusable functions and combine them.

Declarative versus imperative programming

focuses on describing how a program operates.

expresses the logic of a computation without describing its control flow.

Functions as first-class citizens

A function is a first-class citizen when it can do everything that a variable can do, which means that functions can be passed to other functions as an argument, or return as values from a function.

Higher-order functions

A higher-order function is a function that does at least on of the following:

- Takes one or more functions as arguments
- Returns a function as its result

Pointfree style

Pointfree style means functions that never mention the data upon which they operate. First class functions, currying, and composition all play well together to create this style.

Pointfree code can again, help us remove needless names and keep us concise and generic. For example, Can't compose a while loop, conditional if/else. We can focus on transformation itself.

pointfree is a double-edged sword and can sometimes obfuscate intention

Lambda expressions

Lambda expressions are just expressions that can be used to declare anonymous functions (functions without a name).

Function arity

The arity of a function is the number of arguments that the function takes.

A function is a function that only takes a single argument.Unary functions are very important in functional programming because they facilitate utilization of the function composition pattern.

A function is a function that takes two arguments.

There are also functions with three (functions) or more arguments.

Laziness

Many functional programming languages feature lazy-evaluated APIs. The idea behind lazy evaluation is that operations are not computed until doing so can no longer be postponed.

Composition

The essence of software development is composition.

- Object Composition
- Function composition

Composition : Object

Composition : Function

- javascript

- clojure

Composition : How to make it possible

it is simplicity that make composition possible.

- A horrible example
- The gorilla/banana problem

  “…the problem with object-oriented languages is they’ve got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.”
  Joe Armstrong