Computer Science 340
Programming Languages
Siena College
Fall 2019

Topic Notes: Functional Programming

Our next topic is to look at a different programming paradigm: *functional programming*. Recall from the first week of class, the following brief description of functional programming languages:

“The primary mechanism for computing in a functional language is, unsurprisingly, the application of (often recursive) functions to parameters.

- pure functional programming has no variables or assignment statements!
- very convenient in some contexts
- not well-suited for others
- functional languages are usually interpreted rather than compiled”

Functional languages generally have a much simpler and “cleaner” syntax than object-oriented/imperative languages. This has led to the adoption of functional languages as a first language for introductory computer science courses at some schools.

The ideas of functional programming have gained popularity in recent years with the growing use of languages like Scala and Clojure, in part due to he fact that concurrency is now a very common feature of programs.

From `http://clojure.org/rationale`: “Customers and stakeholders have substantial investments in, and are comfortable with the performance, security and stability of, industry-standard platforms like the JVM. While Java developers may envy the succinctness, flexibility and productivity of dynamic languages, they have concerns about running on customer-approved infrastructure, access to their existing code base and libraries, and performance. In addition, they face ongoing problem dealing with concurrency using native threads and locking. Clojure is an effort in pragmatic dynamic language design in this context. It endeavors to be a general-purpose language suitable in those areas where Java is suitable. It reflects the reality that, for the concurrent programming future, pervasive, unmoderated mutation simply has to go.”

We will examine functional programming with an older, but still very relevant language called Scheme.

Mathematical Basis of Functional Languages

Those who have taken some college mathematics have probably seen the idea of *composite functions*:

$$F(x) \circ G(x) = F(G(x))$$

We would read this “F follows G” or “F of G of x”. It means we form a new function by first applying G then F.

In functional programming, we combine often simple functions to build more complex ones.

We will also see *higher-order functions* (or, *functional forms*), where a function receives functions as parameters and/or returns a function as a result.

This can be done in some conventional programming languages (passing of function pointers in C), but it is much more natural in a functional programming language.