CSE M226: Programming Languages

Lecture 10: Introduction to Functional Languages

Rokan Uddin Faruqui

Dept. of Computer Science and Engineering
University of Chittagong, Bangladesh
Email: rufaruqui@cu.ac.bd

Historical Origin

- The imperative and functional models grew out of work undertaken by mathematicians **Alan Turing**, **Alonzo Church**, **Stephen Kleene**, **Emil Post**, and others in the 1930s.
- Working largely independently, these individuals developed several very different formalizations of the notion of
  - an algorithm
  - effective procedure, based on automata
  - symbolic manipulation
  - recursive function definitions
  - combinatorics
- **Outcome**: **anything that could be computed in one could be computed in the others**
- Church Thesis: – **any intuitively appealing model of computing would be equally powerful as well**

Alan Turing

- Alan Turing (1912 – 1954), after whom the Turing Award is named, was a British mathematician, philosopher, and computer visionary.
- As intellectual leader of Britain's cryptanalytic group during World War II, he was instrumental in cracking the German *Enigma* code and turning the tide of the war.
- He also
  - helped lay the theoretical foundations of modern computer science,
  - conceived the general-purpose electronic computer,
  - pioneered the field of Artificial Intelligence

.

Alonzo Church

- Alonzo Church (1903?1995) was a member of the mathematics faculty
  - at Princeton University from 1929 to 1967
  - at UCLA from 1967 to 1990
- While at Princeton he supervised the doctoral theses of, among many others, Alan Turing, Stephen Kleene, Michael Rabin, and Dana Scott
- His codiscovery, with Turing, of **undecidable problems** was a major breakthrough in understanding the limits of mathematics.

Models of Computation

Turing Machine

- an automaton with the ability to access arbitrary cells of an unbounded storage "tape".
- computes in an imperative way, by changing the values in cells of its tape, just as a high-level imperative program computes by changing the values of variables

Models of Computation

Lambda Calculus

Church defined a formal computation model (a formal system for function definition, function application, and recursion) using lambda expressions.

- is based on the notion of parameterized expressions
- with each parameter introduced by an occurrence of the letter λ
- was the inspiration for functional programming
- computation by substituting parameters into expressions, just as one computes in a high-level functional program by passing arguments to functions.

Functional Programming

Functional programming defines the outputs of a program as a **mathematical function** of the inputs, with no notion of **internal state**, and thus no **side effects**.

- Functional programming sees **programs as functions**, and **functions are decomposed into other functions**.
- In pure functional programming the only value that the function computes is what it returns, there are not side effects, and the input values are not modified.

Characteristics of FL

- Expressions without side effects

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions
- Higher-order functions

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions
- Higher-order functions
- Recursion

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions
- Higher-order functions
- Recursion
- Immutable data structures

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions
- Higher-order functions
- Recursion
- Immutable data structures
- Lazy evaluation

Characteristics of FL

- Expressions without side effects
- First-class functions. This includes
  - Pass functions as arguments
  - Return functions
  - Assign them to variables and to data structures
  - Anonymous functions
- Higher-order functions
- Recursion
- Immutable data structures
- Lazy evaluation

Characteristics of Imperative Languages

- Commands are the main components of the language

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures
- Iteration and loops

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures
- Iteration and loops
- Mutable objects

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures
- Iteration and loops
- Mutable objects
- Eager evaluation

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures
- Iteration and loops
- Mutable objects
- Eager evaluation
- (mostly) Recursion is not supported

Characteristics of Imperative Languages

- Commands are the main components of the language
- Functions and procedures
- Iteration and loops
- Mutable objects
- Eager evaluation
- (mostly) Recursion is not supported

First-Class Functions

First-Class Status

In general, a value in a programming language is said to have first-class status if it can be **passed** as a parameter, **returned** from a subroutine, or **assigned** into a variable.

- First-class status also requires the ability to create (compute) new values at run time.
- Subroutines are second-class values in most imperative languages, but first-class values in all functional programming languages.
- A **higher-order function** takes a function as an argument, or returns a function as a result.

Functional and Imperative Paradigms

	Functional	Imperative
A program as a	Function	Command
Building blocks	Expressions (evaluation)	Assignment (execution)
Program Construction	Composing functions	Sequences of commands
Variables	Immutable (let x be)	Mutable (memory cell)
Repetition	Recursion	Loop