Common Lisp and Introduction to Functional Programming
Lecture 1: Introduction

Yuri Zhykin

Feb 4, 2021

Contacts

- @rodentrabies on Telegram
- https://github.com/rodentrabies

Course structure 1/2

- Basic concepts
  - Intensional vs. extensional ways of viewing things
  - Turing's vs. Church's views of computations
  - Imperative vs. declarative approaches to programming
- Introduction to Common Lisp
  - Short introduction to λ-calculus
  - Language primitives
  - Detailed study of Lisp functions
  - Common Lisp Object System (CLOS)
  - Discussion of Lisp's homoiconicity and macros
- Functional programming basics
  - Functions, state, mutation, procedures
  - Referential transparency
  - Recursion and composition of functions
  - Functions as first class objects in programming languages
  - When and why functional approach is better

...

Course structure 2/2

...

- Case study: sequence processing with recursive operators
  - Map, Reduce and Filter
- "Advanced" functional programming
  - Introduction to type theory
  - Type inference
  - Purely functional programming languages
  - Lazy evaluation
  - Introduction to category theory
  - Interactive theorem provers
- Other declarative programming paradigms
  - Constraint programming
  - Domain-specific languages
  - Logic programming

Intensional and extensional definitions

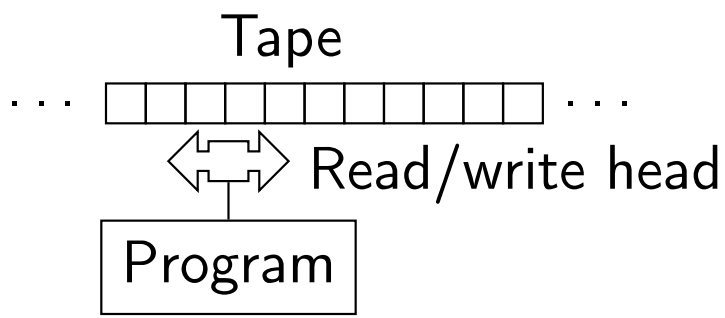
- **Intensional** definition states *necessary and sufficient conditions* for the concept to be applied
  - "Actor is a person that portrays a character in a performance."
- **Extensional** definition provides *a set of all instances* of the concept being applied
  - "Actors: Ryan Gosling, Harrison Ford, Rutger Hauer..."
- Intensional definitions deal with internals of the concept
- Extensional definitions ignore internals of the concepts

Turing, Church and computable functions

- Kurt Gödel, 1933: general recursive functions
- Alan Turing, 1936: Turing-computable functions - all functions that can be computed by a Turing machine
- Alonzo Church, 1936: λ-computable function - all functions that can be described by a term of the λ-calculus
- Church-Turing, 1936-1937: function is λ-computable iff it is Turing-computable and iff it is general recursive

Imperative programming

- Imperative programming describes **how** to achieve the final result by sequentially executing a set of primitive steps
- Steps explicitly modify the state of the program
- Turing machine is the simplest imperative programming model
  - clearly defined state
  - a set of instructions to be executed



Declarative programming

- Declarative programming defines **what** the final result is and leaves the "how" to the implementation of the programming environment
- Since there are no steps that modify the state, there is no explicit state
- Declarative programming is often described as any programming that is not imperative:
  - markup languages (HTML)
  - query languages (SQL, SPARQL)
  - logic programming (Prolog)
  - functional programming (building programs out of composable functions)

Why is declarative programming important?

- Programs written declaratively are easier to read and analyze, since there is no need to keep the state in mind
- Computers are much better at keeping lots of details in memory than humans, and declarative programming leaves maintaining of state to the programming environment
- Most of modern programming languages provide tools for functional programming, and these tools are often the best tools for the job, so learning to use these tools properly is essential to writing idiomatic programs in modern languages

The end

Thank you!