Lecture 1. Overview

September 1, 2019

COM S 342 Fall 2019: Principles of Programming Languages

Goal: maximum your learning on the topic of programming languages

Instructor: Wei Le

TAs:
Samantha-Syed Khairunnesa
Olukorede Fakorede (Joseph)
Wenfei Song

Lecture time, recitations, office hours, piazza, self-study time and homework

Syllabus

FAQs

1.What are the frameworks/libraries that will be used in this course?
A: In case of functional Programming,
Environment: IntelliJ; Defining Language: Java; Build System: Gradle; Parser Generator: Antlr.
In case of logic Programming,
Environment: SWI-Prolog.

2. Is this course going to introduce highly popular languages in industries?
A: In advanced topic section, the course aims to introduce features of popular languages such as Rust, Swift etc.

3. Will there be practice exam provided?
A: There will be review classes before exams and mock exams will be provided as well.

4. Are the exams open book?
A: No, exams are closed book.

Discuss With Your Neighbors

Any questions about syllabus
What do you know about programming languages?
What do you want to learn about programming languages?

What is a programming language?

a language that expresses computations

a language in which developers write code/instructions for computers

It is a foundational course in Computer Science and Software Engineering

- help you find jobs (functional programming, language and compiler design for domain-specific languages)
- make you a better programmer: select an appropriate language for the task, help write efficient code
- improve your ability for future learning
  - Apple: Swift
  - Google: Go and Dart
  - Microsoft: F# and TypeScript
  - Mozilla: Rust
- provoke formal and deep thinking in computing

Programming Language vs. Natural Language

- Both have certain structure i.e., consists of syntax and semantics.
- Like natural language, there are a lot of naturalness in programming languages, e.g., certain grammar rules are used more than the other resulting in certain frequent patterns.
- In contrast, natural language may contain ambiguity, but, it is not a desired property in case of programming languages.

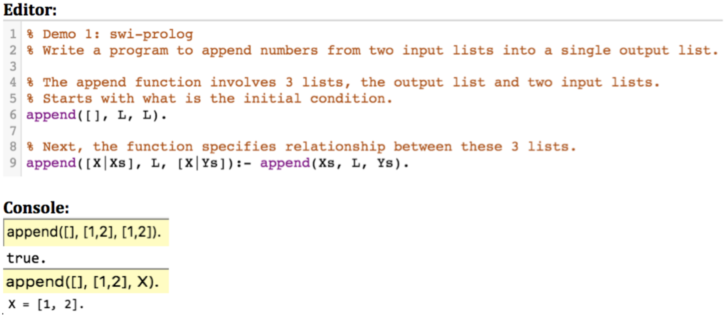
Functional Programming vs. Imperative Programming

- Functional programing aims to extend a person's horizon on how to use functions to achieve tasks.
- It essentially requires a different way of thinking about the way one writes code compared to imperative styled programming.
- However, absence of strong library support is one of the reason that makes it difficult to gain popularity over imperative styled programming.

Topics: write code beyond C, Java and Python

Scheme (Racket) and Prolog (SWI-Prolog)
Example: Appending two lists

Append a List: SWI-Prolog



Editor:

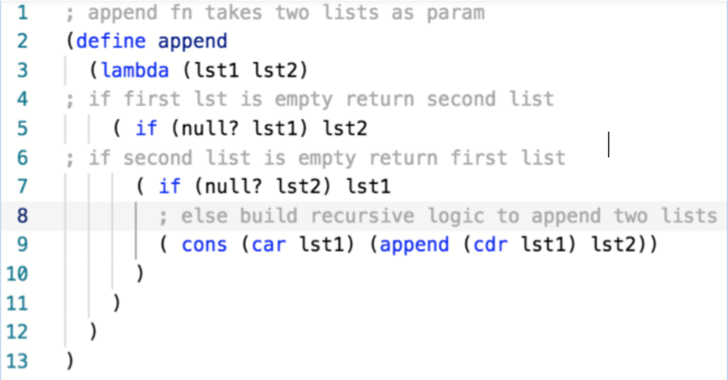
```
% Demo 1: swi-prolog
% Write a program to append numbers from two input lists into a single output list.

% The append function involves 3 lists, the output list and two input lists.
% Starts with what is the initial condition.
append([], L, L).

% Next, the function specifies relationship between these 3 lists.
append([X|Xs], L, [X|Ys]):- append(Xs, L, Ys).
```

Console:

```
append([], [1,2], [1,2]).
true.
append([], [1,2], X).
X = [1, 2].
```

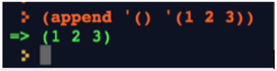
Append a List: Racket

Editor:



```
; append fn takes two lists as param
(define append
  (lambda (lst1 lst2)
; if first lst is empty return second list
    ( if (null? lst1) lst2
; if second list is empty return first list
      ( if (null? lst2) lst1
        ; else build recursive logic to append two lists
        ( cons (car lst1) (append (cdr lst1) lst2))
      )
    )
  )
)
```

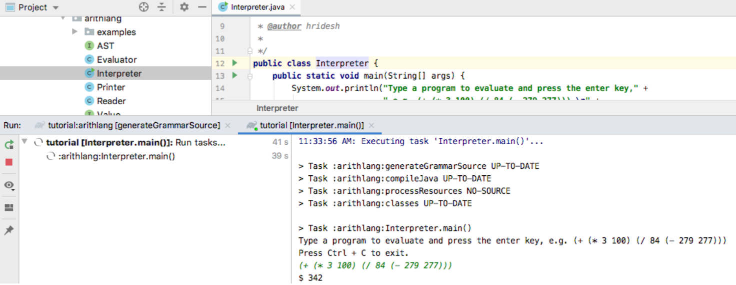
Console:



```
 > (append '() '(1 2 3))
=> (1 2 3)
 >
```

Topics: implement your own programming languages

Interpreter demo


Project
arithlang
examples
AST
Evaluator
Interpreter
Printer
Reader
Value
Interpreter.java
Interpreter
Run:
tutorial:arithlang [generateGrammarSource]
tutorial [Interpreter.main()]
tutorial [Interpreter.main()]: Run tasks...
41 s
:arithlang:Interpreter.main()
39 s

```java
 * @author hridesh
 *
 */
public class Interpreter {
    public static void main(String[] args) {
        System.out.println("Type a program to evaluate and press the enter key," +
```

```
11:33:56 AM: Executing task 'Interpreter.main()'...

> Task :arithlang:generateGrammarSource UP-TO-DATE
> Task :arithlang:compileJava UP-TO-DATE
> Task :arithlang:processResources NO-SOURCE
> Task :arithlang:classes UP-TO-DATE

> Task :arithlang:Interpreter.main()
Type a program to evaluate and press the enter key, e.g. (+ (* 3 100) (/ 84 (- 279 277)))
Press Ctrl + C to exit.
(+ (* 3 100) (/ 84 (- 279 277)))
$ 342
```

Taste of formalism: mathematical aspects of programming languages

- context free grammars
- operational semantics
- lambda calculus: using functions to represent computation

Given
true: $\lambda x.(\lambda y.x)$
false: $\lambda x.(\lambda y.y)$
$\neg$: $\lambda x.((x\ \mathit{false})\mathit{true})$

what some keywords mean?
λ *is the keyword to state that a function is coming up,*
x *or,* y *are the parameters,*
. *means function body is coming up*
and, parenthesis contains the function body

Prove the following:

1. $\neg\ \mathit{false} = \mathit{true}$
2. $\neg(\neg \mathit{true}) = \mathit{true}$

Language: a Tool for communication

There are two important parts about a language:

- Syntax for validity: is this sentence valid?
- Semantics for understanding: what does this sentence mean? to computers: what is the value of this sentence?

History of Programming Languages

1950s: FORTRAN, LISP, COBOL (NASA, ATMs, credit card)
1970s: PASCAL, C (Unix)
1980s: C++ (Firefox, Chrome, Adobe, IE)
1990s: Python, Java (Android)

10 top programming languages:

- 2019: JavaScript, Python, Java, C/C++, PHP, Swift, C#, Ruby, Objective-C, SQL (Geeks for Geek)
- 2018: Python, C++, C, Java, C#, PHP, R, JavaScript, Go, Assembly

Types of Programming Languages

There are mainly two ways to classify programming languages. First,

- **general-purpose language**: express all computation
- **domain-specific language (DSL)** : support data types, relations, operations in domain
  - the Dot language for Graphviz – purpose: graph visualization, special concepts: nodes/edges
  - the HTML language for browsers – purpose: display web pages, special concepts: markup or typesetting related concepts
  - the SQL language for database – purpose: query database, special concepts: support query, join database

Finally,

- **assembly language**
- **high level language**: programs in high-level languages are eventually translated to machine level via *Compilation*, *Interpretation* or *Hybrid*

[Note.] A compiler processes all statements together, and goals are to produce executables (e.g., .exe/.out), whereas, an intepreter program takes statements one by one and evaluates it to produce values.

Parts of a Programming Language

- **Computation**: to actually compute, e.g., primitive expressions, addition, subtraction, multiplication
- **Composition**: to put together computation, e.g., sequential (order), choice, or repeat
- **Abstraction**: to make programming scalable, e.g., function, name, that can be repeatedly used to refer to a complex piece of computation

How to Specify a Language

Specifying a language creates a medium between the language designer and the language user.

Three ways to specify a language:

1. English prose and examples in a careful, expository document (ambiguous, corner cases)
2. compiler/interpreter implementation
3. Formal, mathematical tools: grammar, semantics

Programming Paradigms, Programming Styles

Ways of thinking about computation:

- Imperative: Fortran, Pascal, C
- Object-oriented: Smalltalk, C++, Java
- Functional: ML, Ocaml, Haskel, Scheme, Scala, Lisp, R ...
- Logic: Prolog

functional programming (FP) is a programming style in which mathematical (partial) functions are used as the core programming abstraction. Functional languages make this programming style more natural.

Imperative Programming

- + Easier to learn, taught more often
- + Better development environments (IDE) and libraries
- + Typically faster
- - Side effect (e.g., aliasing), hard to reason
- - Hard to parallel?

Functional Programming

- + side-effect Free and easy to reason: Input and Output completely describes the behavior of any function
- + less code
- - less efficient?
- - less support for IDE and libraries
- - hard to learn, not taught in school often

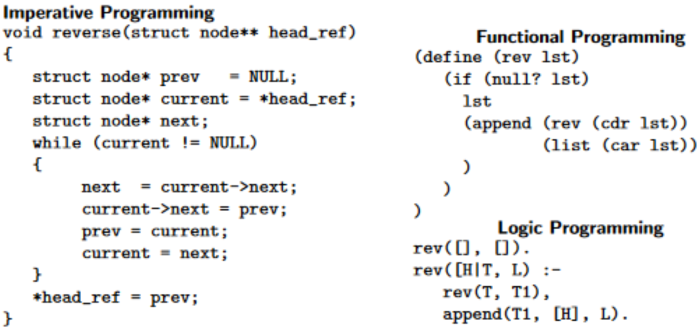
Why teach/learn FL? [1]

[1] http://www.pl-enthusiast.net/2018/07/24/teaching-programming-languages/

Logic Programming

- Data as facts and relations
- Computations as logical inferences
- Control constructs: if-then-else and recursion

Reverse a list



Imperative Programming

```
void reverse(struct node** head_ref)
{
   struct node* prev    = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL)
   {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
   }
   *head_ref = prev;
}
```

Functional Programming

```
(define (rev lst)
   (if (null? lst)
     lst
     (append (rev (cdr lst))
             (list (car lst))
     )
   )
)
```

Logic Programming

```
rev([], []).
rev([H|T, L) :-
   rev(T, T1),
   append(T1, [H], L).
```

Interesting reading

Ray Tracer Language Comparison