Lecture 1. Overview

August 27, 2018

Syllabus

Please check Syllabus on Canvas for more detailed information

Discuss With Your Neighbors

What do you know about programming languages?
Why do we learn programming languages?
What do you want to learn about programming languages?

Practical Values of COM S 342

- ability to quickly learn a new language
  - Apple: Swift
  - Google: Go and Dart
  - Microsoft: F# and TypeScript
- work as a language/compiler person: design and implementation of domain-specific language (DSL)
- better programmer: select an appropriate language for the task
- foundation of computer science, problem solving skills and mindset: help write efficient code, design data types, application programming interface (API)

Content Keywords

Functional programming: Racket [1]
Logic programming: SWI-prolog [2]
Language features and interpreter implementation
Theoretical foundations: Grammar, Formal semantics, Lambda calculus

[1] https://racket-lang.org/
[2] http://www.swi-prolog.org/

Let's beginning ...

Language: a Tool for communication

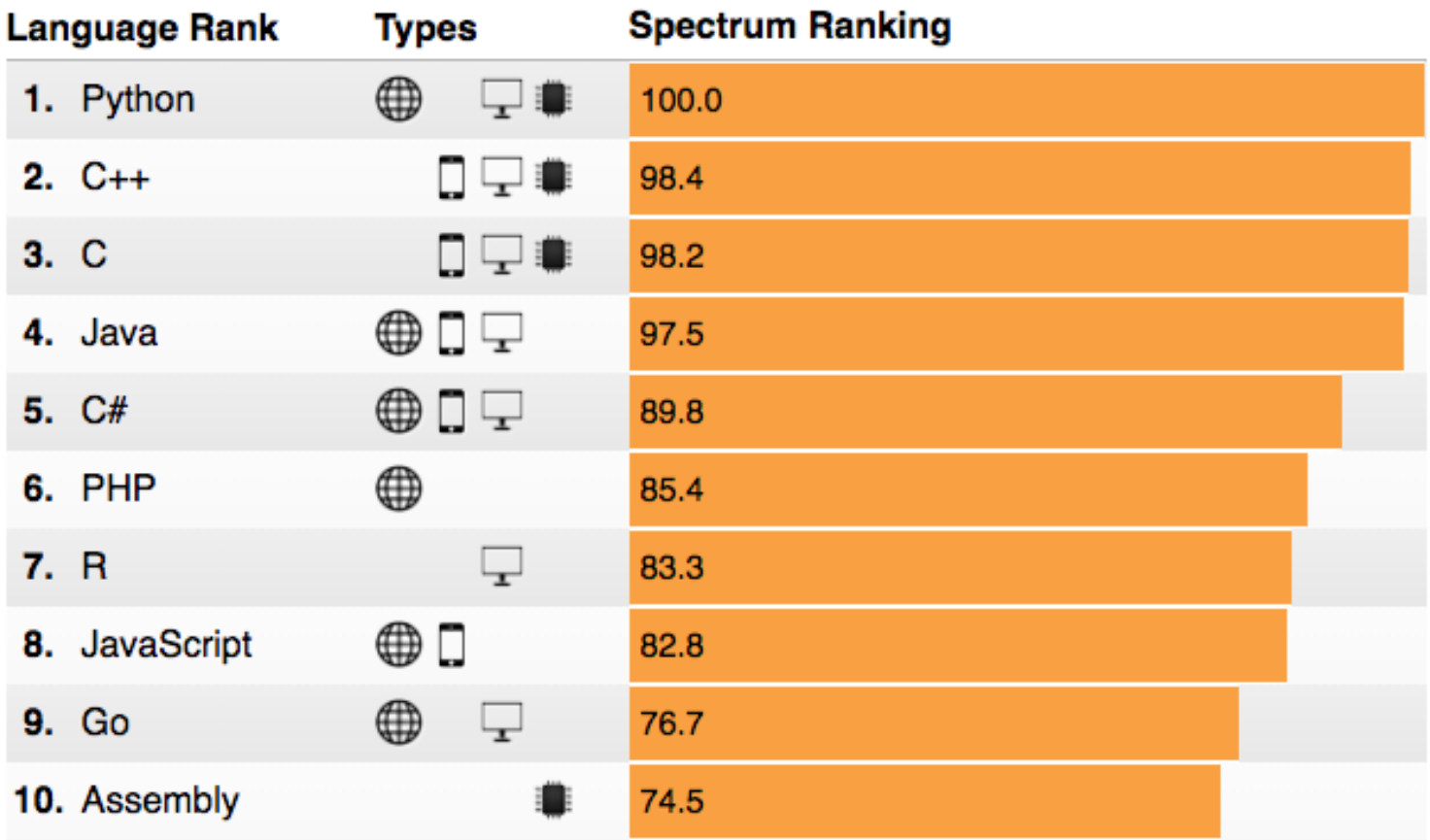
- Syntax for validity
- Semantics for understanding

Programming Languages

languages that express computations

History of Programming Languages (further reading [3])

1950s: FORTRAN, LISP, COBOL (NASA, ATMs, credit card)
1970s: PASCAL, C (Unix)
1980s: C++ (Firefox, Chrome, Adobe, IE)
1990s: Python, Java (Android)
2018 top programming languages



[3] http://www.cs.umd.edu/class/spring2017/cmsc330/lectures/history.pdf

Types of a Programming Languages

- **general-purpose language**: express all computation
- **domain-specific language**: 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

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

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

1. English prose and examples in a careful, expository document (ambiguous, corner cases)
2. compiler [4] ,interpreter [5] implementation
3. Formal, mathematical tools: grammar, semantics

[4] Interpreter: Translates program one statement at a time.

[5] Compiler: Scans the entire program and translates it as a whole into machine code.

Programming Paradigms, Programming Styles

Ways of thinking about computation:

- Imperative: Fortran, Pascal, C
- Object-oriented: Smalltalk, C++, Java
- Functional: ML, Ocaml, Haskel, Scheme, Scala
- 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, 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? [6]

[6] 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

Detailed Main Idea [7]

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

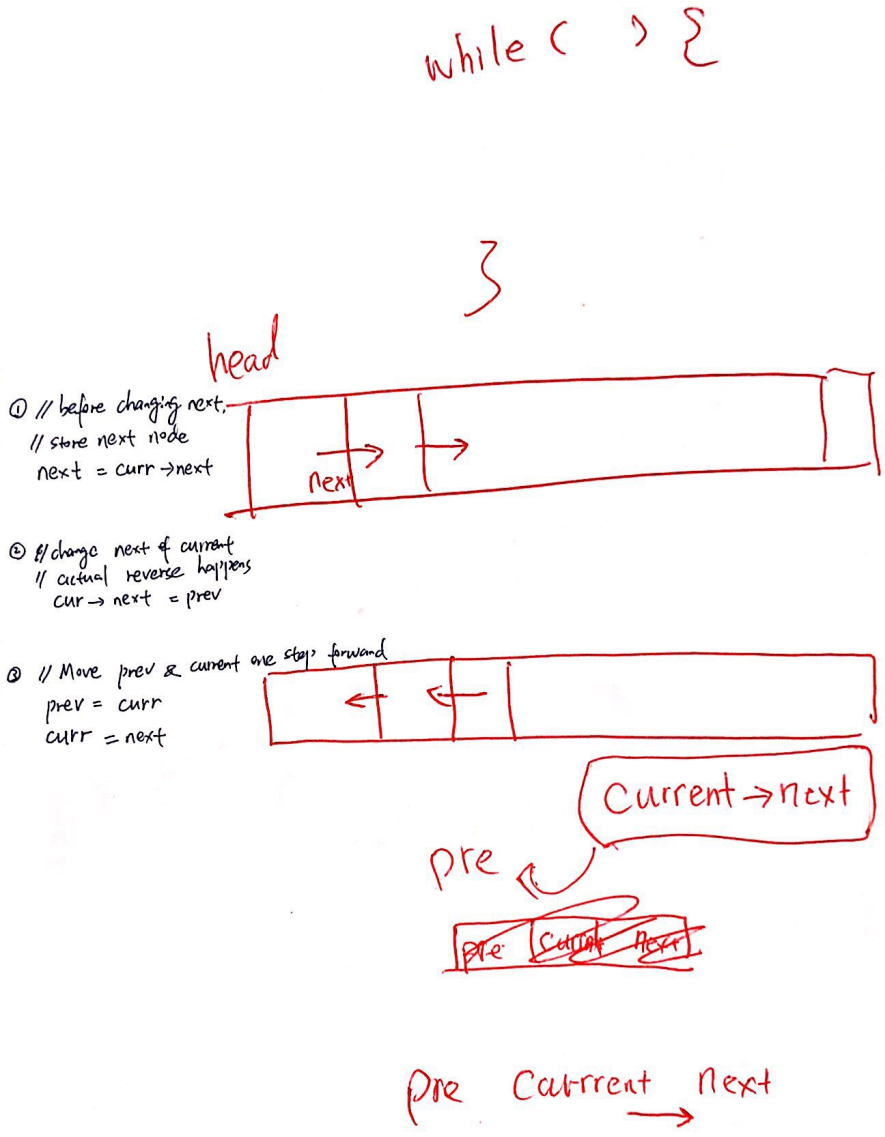
```
(define (rev lst)
   (if (null? lst)
     lst
     (append (rev (cdr lst))
             (list (car lst))
     )
   )
)
```

Logic Programming

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

[7] https://www.geeksforgeeks.org/reverse-a-linked-list/

Reverse a list – Functional Programming



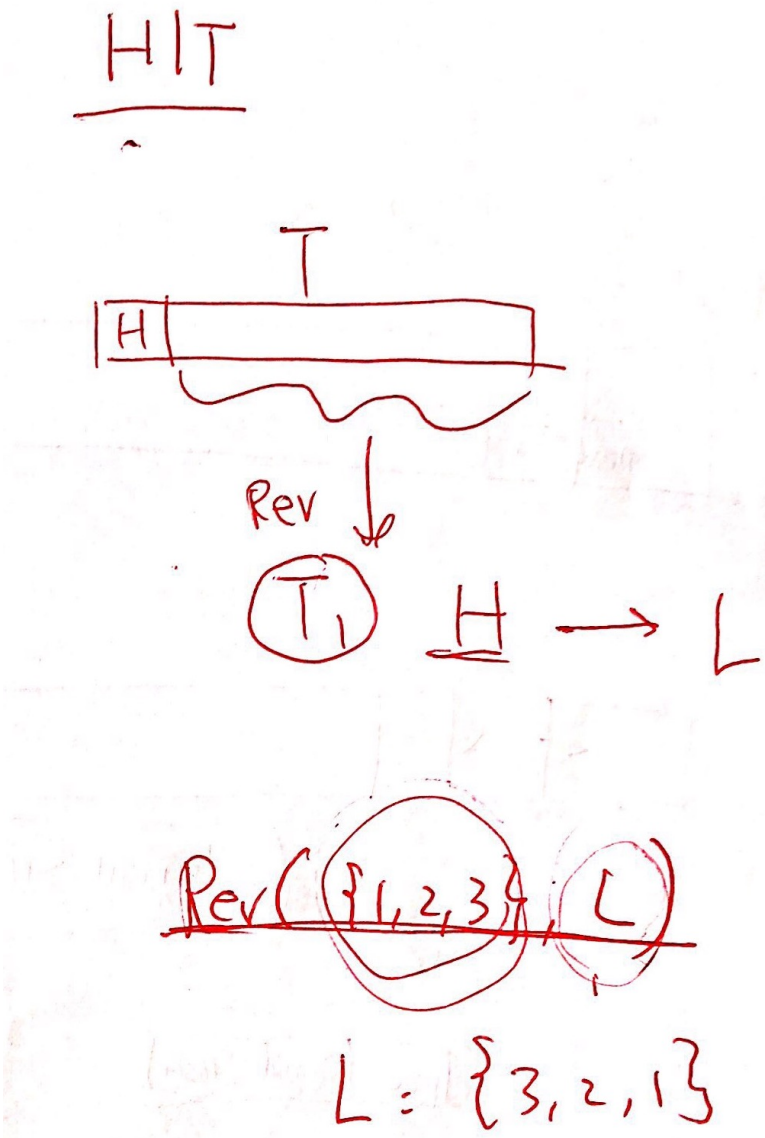
Reverse a list – Functional Programming

Simple Notes on Racket Syntax:

- First element in the list: `(car lst)`
- Tail of the list: `(cdr lst)`
- if-then-else: `(if condition then-stmt else-stmt)`

```
(if (equal? 'a x)
    1
    0)
```

Reverse a list – Logic Programming



Ray Tracer

http://www.ffconsultancy.com/languages/ray_tracer/index.html

Review

1. What is Programming Language?
2. Programming Language has two components:
   - Syntax
   - Semantics
3. Parts of a Program:
   - Computation
   - Composition
   - Abstraction
4. Programming Paradigms and Programming Styles