Introduction

Prof. Joseph Eremondi CS 350

Programming Languages

- What are they made of?

Programming Languages

- What are they made of?
- How do they work?

Interpreters

- How to make a programming language

Interpreters

- How to make a programming language
- Parts of an interpreter

Interpreters

- How to make a programming language
- Parts of an interpreter
    - Parsing

Interpreters

- How to make a programming language
- Parts of an interpreter
  - Parsing
  - Desugaring

Interpreters

- How to make a programming language
- Parts of an interpreter
  - Parsing
  - Desugaring
  - Typechecking

Interpreters

- How to make a programming language
- Parts of an interpreter
  - Parsing
  - Desugaring
  - Typechecking
  - Evaluation

What is Racket?

- LISP-like language

What is Racket?

- LISP-like language
  - parentheses

What is Racket?

- LISP-like language
  - parentheses
- A language for writing programming languages

Will I Ever Use Racket in Industry?

Will I Ever Use Racket in Industry?

No

Will I Ever Use Racket in Industry?

No

(probably)

Future Proofing

- Don’t know what you’ll use in industry in 10 years

Future Proofing

- Don’t know what you’ll use in industry in 10 years
  - If you know how languages work, you can learn *any* language quickly

Future Proofing

- Don’t know what you’ll use in industry in 10 years
  - If you know how languages work, you can learn *any* language quickly
  - Racket is effective for learning how languages work

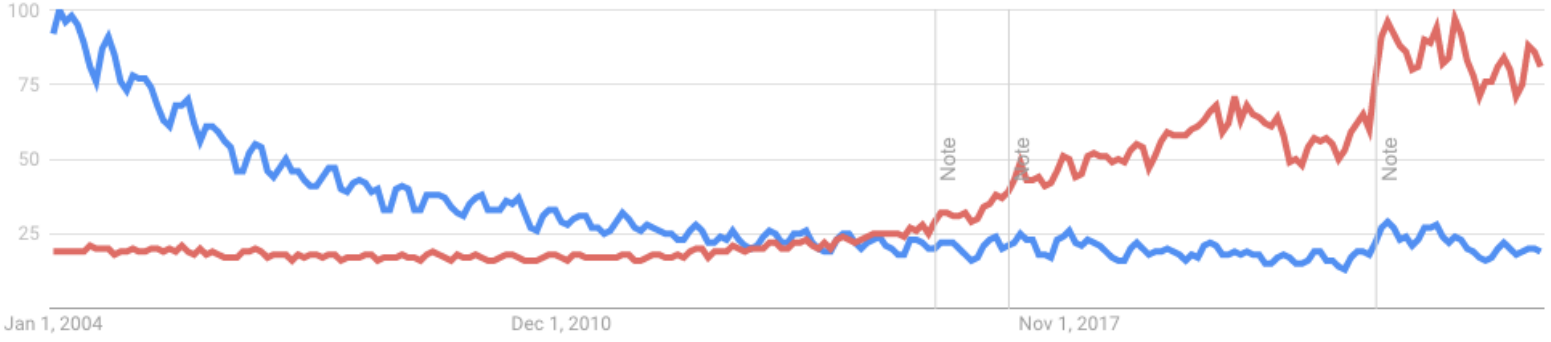
Future Proofing

Objective C vs Swift

Future Proofing

Objective C vs Swift

C++ vs Python



Syntax Vs Semantics

- Semantics

Syntax Vs Semantics

- Semantics
  - What a program *means*

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics
- Racket looks very different from other languages

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics
- Racket looks very different from other languages
  - Expressions, not statements

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics
- Racket looks very different from other languages
  - Expressions, not statements
  - Recursion, not loops

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics
- Racket looks very different from other languages
  - Expressions, not statements
  - Recursion, not loops
  - Parentheses + functions, not operators

Syntax Vs Semantics

- Semantics
  - What a program *means*
  - How a program behaves
- Different syntaxes can have identical semantics
- Course goal: Learning to see past syntax and understand a program as its semantics
- Racket looks very different from other languages
  - Expressions, not statements
  - Recursion, not loops
  - Parentheses + functions, not operators
- Changes how you think about programs

Seeing Past Syntax

By the end of the course, you should be able to look at these programs and intuitively know that they’re doing the same thing:

Seeing Past Syntax

By the end of the course, you should be able to look at these programs and intuitively know that they're doing the same thing:

```
int pow (int x, int y){
  int ret = 1;
  for (int i = 0; i < y; i++){
    ret *= x;
  }
  return ret;
}
```

Seeing Past Syntax

By the end of the course, you should be able to look at these programs and intuitively know that they're doing the same thing:

```c
int pow (int x, int y){
  int ret = 1;
  for (int i = 0; i < y; i++){
    ret *= x;
  }
  return ret;
}
```

```racket
(define (pow x y)
  (if
    (<= y 0)
    0
    (* x (pow x (- y 1)))))
```

Why Functional Programming

- Sum types

Why Functional Programming

- Sum types
  - Perfect for modelling syntax

Why Functional Programming

- Sum types
  - Perfect for modelling syntax
  - Missing/hard in most imperative languages

Functional Programming in Practice

- Anonymous functions

Functional Programming in Practice

- Anonymous functions
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.

Functional Programming in Practice

- Anonymous functions
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11

Functional Programming in Practice

- Anonymous functions
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11
  - Added in Java 8

Functional Programming in Practice

- Anonymous functions
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11
  - Added in Java 8
  - C

Is this a hard course?

Why this course is hard?

Is this a hard course?

Why this course is hard?

- By the end if this course, you will be able to write a program that is powerful enough to simulate every other computer program that ever has or ever will be written

Is this a hard course?

Why this course is hard?

- By the end if this course, you will be able to write a program that is powerful enough to simulate every other computer program that ever has or ever will be written

Why this course is easy

Is this a hard course?

Why this course is hard?

- By the end if this course, you will be able to write a program that is powerful enough to simulate every other computer program that ever has or ever will be written

Why this course is easy

- It's just a bunch of tree traversals