Introduction

CS 350

Dr. Joseph Eremondi

Last updated: June 18, 2024

Course Overview

Course Objectives

To learn:

- Functional programming

Course Objectives

To learn:

- Functional programming
    - Recursion

Course Objectives

To learn:

- Functional programming
    - Recursion
    - Immutable data

Course Objectives

To learn:

- Functional programming
  - Recursion
  - Immutable data
  - Programming by cases

Course Objectives

To learn:

- Functional programming
  - Recursion
  - Immutable data
  - Programming by cases
  - Higher-order functions

Course Objectives

To learn:

- Functional programming
    - Recursion
    - Immutable data
    - Programming by cases
    - Higher-order functions
- How to write your own programming language

Course Objectives

To learn:

- Functional programming
  - Recursion
  - Immutable data
  - Programming by cases
  - Higher-order functions
- How to write your own programming language
  - Parsing/Abstract Syntax

Course Objectives

To learn:

- Functional programming
    - Recursion
    - Immutable data
    - Programming by cases
    - Higher-order functions
- How to write your own programming language
    - Parsing/Abstract Syntax
    - Desugaring

Course Objectives

To learn:

- Functional programming
    - Recursion
    - Immutable data
    - Programming by cases
    - Higher-order functions
- How to write your own programming language
    - Parsing/Abstract Syntax
    - Desugaring
    - Typechecking

Course Objectives

To learn:

- Functional programming
  - Recursion
  - Immutable data
  - Programming by cases
  - Higher-order functions
- How to write your own programming language
  - Parsing/Abstract Syntax
  - Desugaring
  - Typechecking
  - Evaluation

Course Objectives

To learn:

- Functional programming
  - Recursion
  - Immutable data
  - Programming by cases
  - Higher-order functions
- How to write your own programming language
  - Parsing/Abstract Syntax
  - Desugaring
  - Typechecking
  - Evaluation
- To change how you *think* about programming

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi
    - aka PLAI

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi
    - aka PLAI
    - Freely avaliable online, pdf in UR Courses

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi
    - aka PLAI
    - Freely avaliable online, pdf in UR Courses
- 3rd edition also available

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi
    - aka PLAI
    - Freely avaliable online, pdf in UR Courses
- 3rd edition also available
    - Optional additional reference

Textbook

- *Programming Languages: Application and Interpretation*, 2nd edition, by Shriram Krishnamurthi
    - aka PLAI
    - Freely avaliable online, pdf in UR Courses
- 3rd edition also available
    - Optional additional reference
    - Similar content but very different approach

Course Communication

- Everything on URCourses

Course Communication

- Everything on URCourses
    - Announcements

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin

Course Communication

- Everything on URCourses
    - Announcements
    - Assignments and Handin
    - Textbook, Slides, Videos

Course Communication

- Everything on URCourses
    - Announcements
    - Assignments and Handin
    - Textbook, Slides, Videos
    - Email

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin
  - Textbook, Slides, Videos
  - Email
  - Discussion Forum

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin
  - Textbook, Slides, Videos
  - Email
  - Discussion Forum
- Do NOT ask programming/conceptual questions by email

Course Communication

- Everything on URCourses
    - Announcements
    - Assignments and Handin
    - Textbook, Slides, Videos
    - Email
    - Discussion Forum
- Do NOT ask programming/conceptual questions by email
    - Use the discussion forum

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin
  - Textbook, Slides, Videos
  - Email
  - Discussion Forum
- Do NOT ask programming/conceptual questions by email
  - Use the discussion forum
  - If you’re wondering, others are too

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin
  - Textbook, Slides, Videos
  - Email
  - Discussion Forum
- Do NOT ask programming/conceptual questions by email
  - Use the discussion forum
  - If you’re wondering, others are too
  - EXCEPTION: when you can’t ask your question without revealing your solution to the assignment

Course Communication

- Everything on URCourses
  - Announcements
  - Assignments and Handin
  - Textbook, Slides, Videos
  - Email
  - Discussion Forum
- Do NOT ask programming/conceptual questions by email
  - Use the discussion forum
  - If you're wondering, others are too
  - EXCEPTION: when you can't ask your question without revealing your solution to the assignment
- Announcements on URCourses

Grading Scheme

- 25% assignments

Grading Scheme

- 25% assignments
- 25% midterm

Grading Scheme

- 25% assignments
- 25% midterm
  - In-class

Grading Scheme

- 25% assignments
- 25% midterm
    - In-class
    - Thursday, July 25

Grading Scheme

- 25% assignments
- 25% midterm
    - In-class
    - Thursday, July 25
- 50% final

Grading Scheme

- 25% assignments
- 25% midterm
  - In-class
  - Thursday, July 25
- 50% final
  - Aug 19

Grading Scheme

- 25% assignments
- 25% midterm
  - In-class
  - Thursday, July 25
- 50% final
  - Aug 19
  - 2pm-5pm

Grading Scheme

- 25% assignments
- 25% midterm
  - In-class
  - Thursday, July 25
- 50% final
  - Aug 19
  - 2pm-5pm
  - This room

Assignments

- Six weekly assignments

Assignments

- Six weekly assignments
- Due Tuesday at 5pm

Assignments

- Six weekly assignments
- Due Tuesday at 5pm
    - No extensions

Assignments

- Six weekly assignments
- Due Tuesday at 5pm
    - No extensions
    - Lowest grade dropped

Assignments

- Six weekly assignments
- Due Tuesday at 5pm
    - No extensions
    - Lowest grade dropped
- Submitted over UR Courses

Assignments (ctd.)

- Mostly programming

Assignments (ctd.)

- Mostly programming
    - Some conceptual questions

Assignments (ctd.)

- Mostly programming
  - Some conceptual questions
- Score based on running tests

Assignments (ctd.)

- Mostly programming
    - Some conceptual questions
- Score based on running tests
    - Some public (included in assignment)

Assignments (ctd.)

- Mostly programming
    - Some conceptual questions
- Score based on running tests
    - Some public (included in assignment)
    - Some private (only known by me)

Assignments (ctd.)

- Mostly programming
    - Some conceptual questions
- Score based on running tests
    - Some public (included in assignment)
    - Some private (only known by me)
    - Code doesn’t run → no marks

Assignments (ctd.)

- Mostly programming
  - Some conceptual questions
- Score based on running tests
  - Some public (included in assignment)
  - Some private (only known by me)
  - Code doesn’t run → no marks
- Some points for style/documentation/etc.

Assignments (ctd.)

- Mostly programming
    - Some conceptual questions
- Score based on running tests
    - Some public (included in assignment)
    - Some private (only known by me)
    - Code doesn’t run → no marks
- Some points for style/documentation/etc.
    - Sample based marking

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

- Considered a violation of Academic Integrity

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

- Considered a violation of Academic Integrity

ChatGPT has trouble with Racket/plait

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

- Considered a violation of Academic Integrity

ChatGPT has trouble with Racket/plait

- Don’t expect sympathy if you copy/paste code from an LLM that doesn’t work

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

- Considered a violation of Academic Integrity

ChatGPT has trouble with Racket/plait

- Don’t expect sympathy if you copy/paste code from an LLM that doesn’t work

Don’t set yourself up for failure on the exams

LLM Policy

Use of ChatGPT, GitHub Copilot, or any other Large Language Model or Generative AI is forbidden when completing the assignments for this class

- Considered a violation of Academic Integrity

ChatGPT has trouble with Racket/plait

- Don’t expect sympathy if you copy/paste code from an LLM that doesn’t work

Don’t set yourself up for failure on the exams

- Doing the assignments is the best way to study

Motivation: Functional Programming

Programming in This Class

- In plait

Programming in This Class

- In plait
    - i.e., “PLAI-typed”

Programming in This Class

- In plait
    - i.e., “PLAI-typed”
- Plait is

Programming in This Class

- In plait
    - i.e., “PLAI-typed”
- Plait is
    - a programming language

Programming in This Class

- In plait
    - i.e., “PLAI-typed”
- Plait is
    - a programming language
    - a library for the Racket programming language

Programming in This Class

- In plait
    - i.e., “PLAI-typed”
- Plait is
    - a programming language
    - a library for the Racket programming language
- We’ll learn more why this distinction is fuzzy

What is Racket

- A programming language for writing programming languages

What is Racket

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

What is Racket

- A programming language for writing programming languages
- LISP-like
    - parentheses `(((((((((())))))))))`

What is Racket

- A programming language for writing programming languages
- LISP-like
    - parentheses (((((((((())))))))))
    - functions are values just like anything else

What is Racket

- A programming language for writing programming languages
- LISP-like
    - parentheses (((((((((())))))))))
    - functions are values just like anything else
- Immutable: once a variable has a value, it never changes

What is Racket

- A programming language for writing programming languages
- LISP-like
    - parentheses `(((((((((())))))))))`
    - functions are values just like anything else
- Immutable: once a variable has a value, it never changes
    - Racket does let you mutate variables, but those parts of the language are **forbidden** in this class

What is Racket

- A programming language for writing programming languages
- LISP-like
  - parentheses (((((((((())))))))))
  - functions are values just like anything else
- Immutable: once a variable has a value, it never changes
  - Racket does let you mutate variables, but those parts of the language are **forbidden** in this class
    - Unless otherwise specified

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

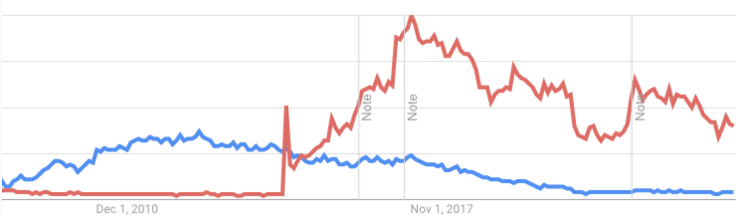
- Don’t know what you’ll use in industry in 10 years
    - If you know how languages work, you can learn *any* language quickly

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

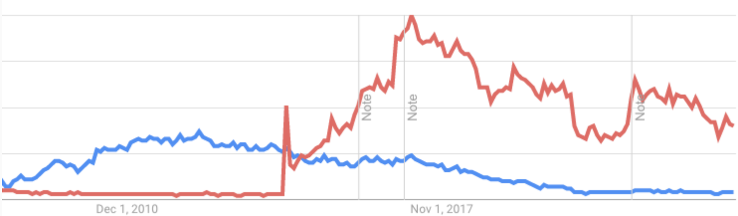
Language Trends (from Google Trends)

Objective C vs Swift

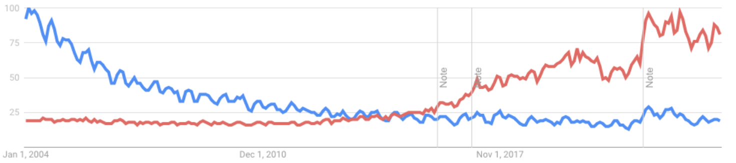


Language Trends (from Google Trends)

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:

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

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:

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

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:

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

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:

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

```
    (define (pow x y)
```

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:

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

```
    (define (pow x y)
      (if
```

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:

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

```
    (define (pow x y)
      (if
        (<= y 0)
```

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:

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

```
(define (pow x y)
  (if
    (<= y 0)
    0
```

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:

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

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

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
    - Added to C++11

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11
  - Added in Java 8

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
    - Added to C++11
    - Added in Java 8
    - Most language have some form of `map` to apply a function to each element of a list

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11
  - Added in Java 8
  - Most language have some form of `map` to apply a function to each element of a list
- Sum types

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
    - Added to C++11
    - Added in Java 8
    - Most language have some form of `map` to apply a function to each element of a list
- Sum types
    - Also called variants, algebraic datatypes

Functional Programming Going Mainstream?

- We're seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
    - Added to C++11
    - Added in Java 8
    - Most language have some form of `map` to apply a function to each element of a list
- Sum types
    - Also called variants, algebraic datatypes
    - Perfect for syntax trees

Functional Programming Going Mainstream?

- We’re seeing more languages adopt functional features
- Anonymous functions (lambda)
  - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
  - Added to C++11
  - Added in Java 8
  - Most language have some form of `map` to apply a function to each element of a list
- Sum types
  - Also called variants, algebraic datatypes
  - Perfect for syntax trees
  - Now in Python, Typescript, C++ (`std::variant`), Java (sealed interfaces), Rust (enums)

Functional Programming Going Mainstream?

- We're seeing more languages adopt functional features
- Anonymous functions (lambda)
    - Python, Ruby, JS, PHP, Swift, Go, Rust, etc.
    - Added to C++11
    - Added in Java 8
    - Most language have some form of `map` to apply a function to each element of a list
- Sum types
    - Also called variants, algebraic datatypes
    - Perfect for syntax trees
    - Now in Python, Typescript, C++ (`std::variant`), Java (sealed interfaces), Rust (enums)
- Learning these features in Racket will help if/when they show up in other languages in the future

Motivation: Interpreters

Importance of Programming Languages

- Interpreter: Code + input $\hookrightarrow$ Output + effects

Importance of Programming Languages

- Interpreter: Code + input $\hookrightarrow$ Output + effects
  - Effects: write to disk, display pixels, etc

Importance of Programming Languages

- Interpreter: Code + input $\hookrightarrow$ Output + effects
  - Effects: write to disk, display pixels, etc
- You interact with a compiler or interpreter every time you:

Importance of Programming Languages

- Interpreter: Code + input $\hookrightarrow$ Output + effects
  - Effects: write to disk, display pixels, etc
- You interact with a compiler or interpreter every time you:
  - Write a program

Importance of Programming Languages

- Interpreter: Code + input ↪ Output + effects
    - Effects: write to disk, display pixels, etc
- You interact with a compiler or interpreter every time you:
    - Write a program
    - Run a program

Importance of Programming Languages

- Interpreter: Code + input $\hookrightarrow$ Output + effects
    - Effects: write to disk, display pixels, etc
- You interact with a compiler or interpreter every time you:
    - Write a program
    - Run a program
        - Python, JavaScript, JVM all use some kind of interpreter

Importance of Programming Languages

- Interpreter: Code + input ↪ Output + effects
  - Effects: write to disk, display pixels, etc
- You interact with a compiler or interpreter every time you:
  - Write a program
  - Run a program
    - Python, JavaScript, JVM all use some kind of interpreter
    - The CPU is just an interpreter for machine code

Programming Languages Aren’t Magic

- Understanding how languages are implemented can help you understand your code

Programming Languages Aren’t Magic

- Understanding how languages are implemented can help you understand your code
    - Why is it slow/fast

Programming Languages Aren’t Magic

- Understanding how languages are implemented can help you understand your code
    - Why is it slow/fast
    - How to prevent/properly handle errors

Programming Languages Aren't Magic

- Understanding how languages are implemented can help you understand your code
    - Why is it slow/fast
    - How to prevent/properly handle errors
    - How to know that it's doing what you think it does

Is this a hard course?

Why interpreters are hard

Is this a hard course?

Why interpreters are 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 interpreters are 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 interpreters are easy

Is this a hard course?

Why interpreters are 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 interpreters are easy

- It’s just a bunch of tree traversals