CS 3520/6520
Programming Languages

Fall 2020

Instructor: **Matthew Flatt**

TAs: **Brandon Mouser**
Kyle Price

CS 3520/6520 Programming Languages

a survey course:

an object-oriented language

a functional language

a logic language

CS 3520/6520 Programming Languages

Not a survey course:

an object-oriented language

a functional language

a logic language

CS 3520/6520 Programming Languages

This course is about programming language **concepts**

lexical scope	closures	recursion
λ-calculus	objects	classes
continuations	eager and lazy evaluation	
state	type checking	polymorphism
soundness	type inference	subtyping
compilation	garbage collection	

... especially **functional programming** concepts

use one language, implement many languages

CS 3520/6520 Programming Languages

This course is about programming language **concepts**

- To help you understand new programming languages
- To make you a better programmer in any language

Course Details

`http://www.eng.utah.edu/~cs3520/`

Formal prerequisite: CS 3500

Informal prerequisite: more programming experience than that

- Homework is weekly, roughly
- Two mid-term exams
- Extended homework assignment place of a final exam
- Late policy: up to 48 hours, two automatic "free lates"

Lectures are Online

All slide presentations are online

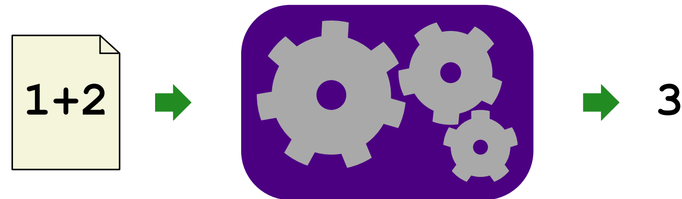
- **Watch the videos before class**

- **Meet as a class for more examples and homework solutions**
  - a.k.a. "recitation"
  - guideline: no new material introduced in class
  - need volunteers to sign up

- **Volunteer for in-class participation**
  - be prepared to share your screen, write code with class help
  - see sign-up in Canvas
  - presenters get one extra "free late"

Interpreters

- **Learn concepts by implementing interpreters**



new concept ⇒ new interpreter

We'll always call the language that we implement **Curly**, even though the language keeps changing

Racket and Plait

- **Implement interpreters using Plait, a variant of Racket**

  Historically: **Lisp** ⇒ **Scheme** ⇒ **Racket** ⇒ **Plait**

Racket and Plait

- **Implement interpreters using Plait, a variant of Racket**

Historically: **Lisp** ⇒ **Scheme** ⇒ **Racket** ⇒ **Plait** ⇐ **ML**

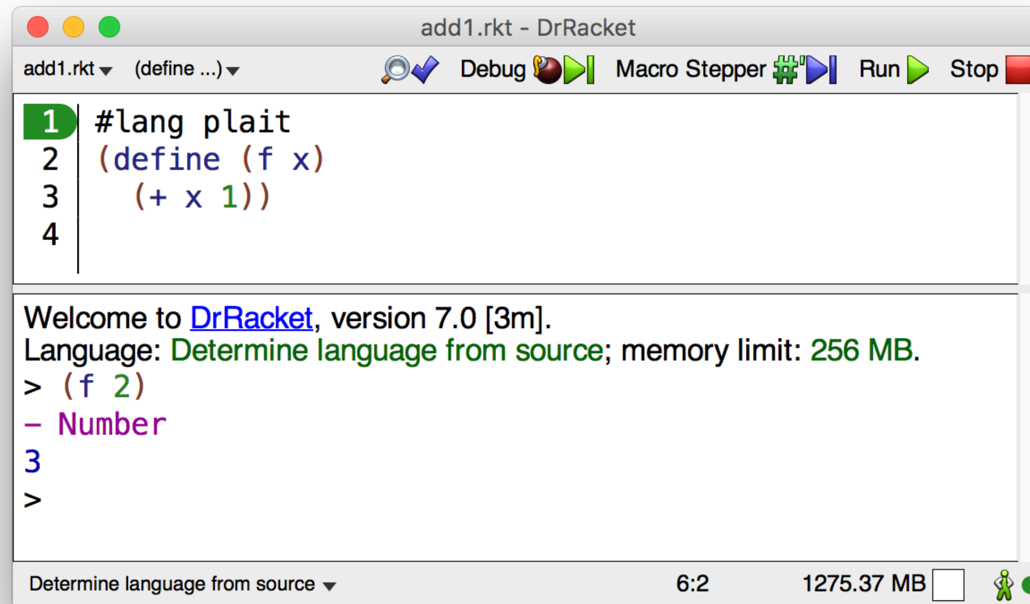
Racket is

- a programming language
- a family of programming languages
- a language for creating programming languages

... including **Plait**

PLAI = *Programming Languages: Application and Interpretation*, a textbook

DrRacket



Preview: Plait Tutorial

http://docs.racket-lang.org/plait/index.html

v.7.0

Plait Language

```
#lang plait                                              package: plait
```

The Plait language syntactically resembles the `plai` language, which is based on `racket`, but the type system is close to that of ML.

1 Tutorial

Preview: Plait's Parenthesized Prefix Notation

`f(x)`	`(f x)`
`1+2`	`(+ 1 2)`
`1+2*3`	`(+ 1 (* 2 3))`
`s=6`	`(define s 6)`
`f(x)=x+1`	`(define (f x)` `  (+ x 1))`

$\begin{cases} x<0 & -1 \\ x=0 & 0 \\ x>0 & 1 \end{cases}$	`(cond`
	`  [(< x 0) -1]`
	`  [(= x 0) 0]`
	`  [(> x 0) 1])`

Preview: Plait Data

- Numbers and strings — obvious

```
1    3.4    "Hello, World!"
```

- Booleans — straightforward

```
#t    #f
```

- Symbols and quoted lists — unusual

```
'apple    'define    '+

'(1 2 3)    '(f x)
```

Preview: Plait S-Expressions

- Backquote ` instead of regular quote '　　　　convenient

```
`x
```

```
`{+ x 1}
```

```
`{define {f x}
   {+ x 1}}
```

Preview: Plait Datatypes

```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
```

Preview: Interpreters

See **lambda.rkt**

Example **Plait** program:

```
(define-type Value
  (numV [n : Number])
  (closV [arg : Symbol]
         [body : Exp]
         [env : Env]))
```

Example **Curly** program:

```
{+ {* 3 4} 8}
```

Example **Curly** program as a **Plait** value:

```
`{+ {* 3 4} 8}
```

Datatype and Function Shapes Match

```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

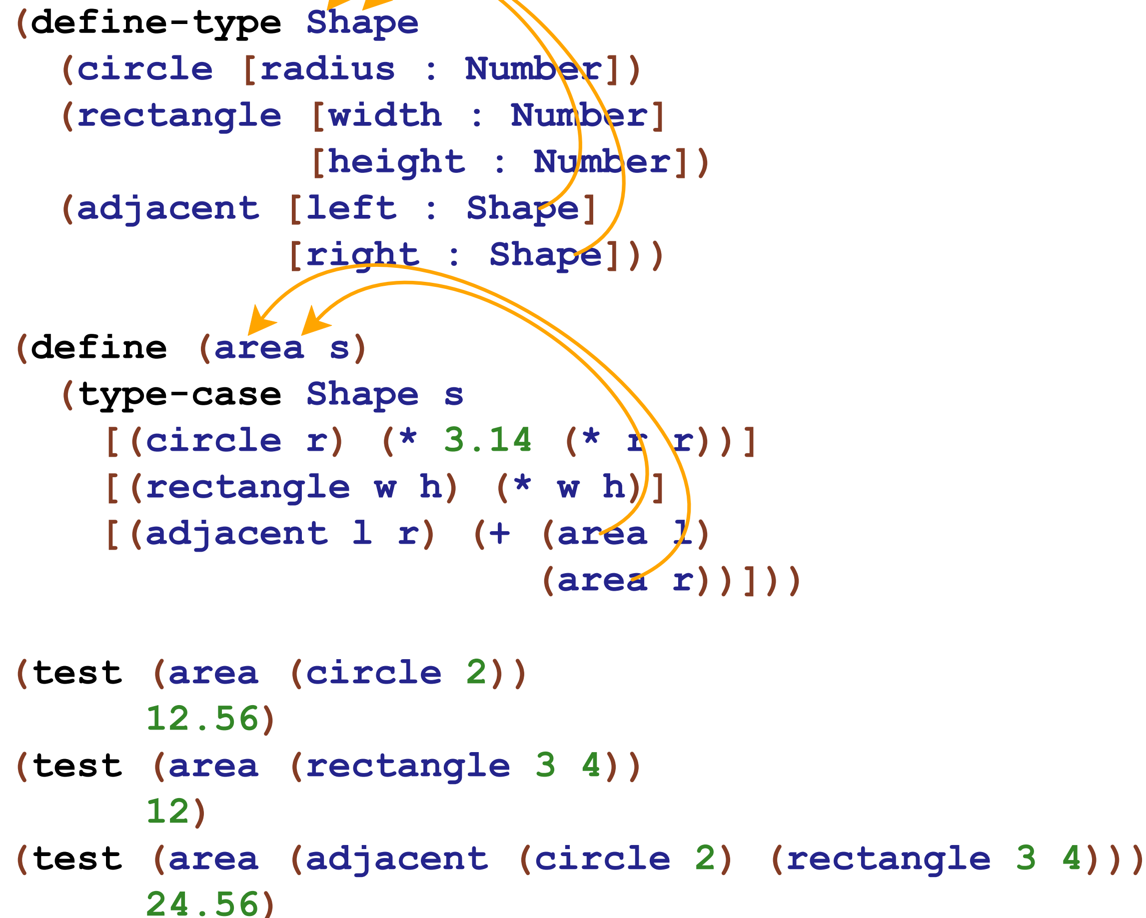
Datatype and Function Shapes Match

```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

Datatype and Function Shapes Match



```
(define-type Shape
  (circle [radius : Number])
  (rectangle [width : Number]
             [height : Number])
  (adjacent [left : Shape]
            [right : Shape]))

(define (area s)
  (type-case Shape s
    [(circle r) (* 3.14 (* r r))]
    [(rectangle w h) (* w h)]
    [(adjacent l r) (+ (area l)
                       (area r))]))

(test (area (circle 2))
      12.56)
(test (area (rectangle 3 4))
      12)
(test (area (adjacent (circle 2) (rectangle 3 4)))
      24.56)
```

Course Outline

- Functional programming
- Interpreters
- State
- Control
- Compilation and GC
- Objects and classes
- Types
- Macros

Homework 0

- Create handin account
- Plait warm-up exercises

Due Sunday, August 30