CS 2010
Computer Science I

Instructor: Matthew Flatt
This Course is About...

Fundamentals of programming

- From specification to implementation
- Software engineering principles
This Course is...

Not about...

- A particular programming language (e.g., Java, C++, Scheme)
- A particular programming tool (e.g., gcc, DrScheme)
- Specific libraries or protocols (e.g., Gtk, XML, HTTP)
- How programs get translated into electronic signals
Book

How to Design Programs

HOW TO DESIGN PROGRAMS

An Introduction to Programming and Computing

Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi
Programming Environment

DrScheme

(define (square n)
  (* n n))

(define (hypotenuse ln ht)
  (sqrt (+ (square ln)
           (square ht))))

Welcome to DrScheme, version 205.
Language: Beginning Student.
> (square 5)
25
> (square 12)
144
> (hypotenuse 5 12)
13
>
What is Scheme?

- **Scheme** is a programming language
  - Used to implement DrScheme, for example
- The language for this course matches a subset of Scheme
- The course content is not Scheme-specific
Getting Started:
Arithmetic, Algebra, and Computing
Arithmetic is Computing

• Fixed, pre-defined rules for *primitive operators*:

\[ 2 + 3 = 5 \]
\[ 4 \times 2 = 8 \]
\[ \cos(0) = 1 \]
Arithmetic is Computing

- Fixed, pre-defined rules for *primitive operators*:

  \[ 2 + 3 \rightarrow 5 \]
  \[ 4 \times 2 \rightarrow 8 \]
  \[ \cos(0) \rightarrow 1 \]

- Rules for combining other rules:
  - Evaluate sub-expressions first
    \[ 4 \times (2 + 3) \rightarrow 4 \times 5 \rightarrow 20 \]
  - Precedence determines subexpressions:
    \[ 4 + 2 \times 3 \rightarrow 4 + 6 \rightarrow 10 \]
Algebra as Computing

- Definition:

  \[ f(x) = \cos(x) + 2 \]

- Expression:

  \[ f(0) \rightarrow \cos(0) + 2 \rightarrow 1 + 2 \rightarrow 3 \]

- First step uses the **substitution** rule for functions
Notation

• Why do some primitive operators go in the middle, like +, while others go at the front, like cos?
• What are the precedence rules?
• How do we know which arguments go with which operators?
• Which parentheses are redundant?
• When does = mean definition and when does it mean a computation step?
• ...
Simplified Expression Notation

- Put all operators at the front
- Start every operation with an open parenthesis
- Put a close parenthesis after the last argument
- Never add extra parentheses

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + 2$</td>
<td>$(+ \ 1 \ 2)$</td>
</tr>
<tr>
<td>$4 + 2 \times 3$</td>
<td>$(+ \ 4 \ (* \ 2 \ 3))$</td>
</tr>
<tr>
<td>$\cos(0) + 1$</td>
<td>$(+ \ (\cos \ 0) \ 1)$</td>
</tr>
</tbody>
</table>
Simplified Definition Notation

- Use the keyword **define** instead of =
- Put **define** at the front, and group with parentheses
- Move open parenthesis from after function name to before

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x) = cos(x) + 2</td>
<td>(define (f x) (+ (cos x) 2))</td>
</tr>
</tbody>
</table>

- Move open parenthesis in function calls

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(0)</td>
<td>(f 0)</td>
</tr>
<tr>
<td>f(2+3)</td>
<td>(f (+ 2 3))</td>
</tr>
</tbody>
</table>
Evaluation is the Same as Before

\[
\text{(define } (f \ x) (\text{(+ } (\cos \ x) \ 2) ))\\
\]

(\text{f } 0)
Evaluation is the Same as Before

(define (f x) (+ (cos x) 2))

(f 0)
→ (+ (cos 0) 2)
Evaluation is the Same as Before

\[
\text{(define } (f \ x) (\text{+ } (\cos \ x) \ 2))
\]

\[
(f \ 0)
\rightarrow (\text{+ } (\cos \ 0) \ 2)
\rightarrow (\text{+ } 1 \ 2)
\]
Evaluation is the Same as Before

\[
(\text{define } (f \ x) \ (\ + \ (\cos \ x) \ 2))
\]

\[
(f \ 0)
\]

\[
\rightarrow \ (\ + \ (\cos \ 0) \ 2)
\]

\[
\rightarrow \ (\ + \ 1 \ 2)
\]

\[
\rightarrow \ 3
\]
Beyond Numbers: Booleans

Numbers are not the only kind of values:

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 &lt; 2$ → true</td>
<td>$(&lt; 1 \ 2)$ → true</td>
</tr>
<tr>
<td>$1 &gt; 2$ → true</td>
<td>$(&gt; 1 \ 2)$ → false</td>
</tr>
<tr>
<td>$1 &gt; 2$ → true</td>
<td>$(&gt; 1 \ 2)$ → false</td>
</tr>
<tr>
<td>$2 \geq 2$ → true</td>
<td>$(\geq 1 \ 2)$ → true</td>
</tr>
</tbody>
</table>
Beyond Numbers: Booleans

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>true and false</td>
<td>(and true false)</td>
</tr>
<tr>
<td>true or false</td>
<td>(or true false)</td>
</tr>
<tr>
<td>$1 &lt; 2$ and $2 &gt; 3$</td>
<td>(and ($&lt; 1$ 2) ($&gt; 2$ 3))</td>
</tr>
<tr>
<td>$1 \leq 0$ and $1 = 1$</td>
<td>(or ($\leq 1$ 0) ($= 1$ 1))</td>
</tr>
<tr>
<td>$1 \neq 0$</td>
<td>(not ($= 1$ 0))</td>
</tr>
</tbody>
</table>
Beyond Numbers: Symbols

(symbol=? 'apple 'apple) → true
(symbol=? 'apple 'banana) → false
Beyond Numbers: Images

\((\text{filled-rect} \ 35 \ 35 \ '\text{red}) \rightarrow \square\) \\
\((\text{filled-circle} \ 25 \ 25 \ '\text{blue}) \rightarrow \bullet\) \\
\((\text{image+} \ \square \ \bullet) \rightarrow \square\bullet\) \\
\((\text{offset-image+} \ \square \ 5 \ 5 \ \bullet) \rightarrow \bullet\square\) \\
\((\text{image=}? \ (\text{image+} \ \square \ \bullet) \ \square\bullet) \rightarrow \) \\
\rightarrow (\text{image=}? \ \bullet\bullet \ \bullet\bullet) \rightarrow \text{true}\)


```
(define (anonymize i)
  (offset-image+ i i 0 0
  (filled-circle (image-width i) (image-height i) 'blue))

(anonymize ) → ... →
```

*Use the stepper to see all steps*