Contents

1 Lecture 01 2
   1.1 Administrivia . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.2 Introduction to CS 115 - Course Website and Slides 1-9 . . 2
   1.3 Introduction to DrRacket - Slides 10-20 . . . . . . . . . . . 2

2 Lecture 02 4
   2.1 Administrivia . . . . . . . . . . . . . . . . . . . . . . . . . 4
   2.2 Mathematical Functions in DrRacket . . . . . . . . . . . . . 4
   2.3 The DrRacket Environment . . . . . . . . . . . . . . . . . . 4
   2.4 Defining New Functions in DrRacket . . . . . . . . . . . . . 5

3 Lecture 03 6
   3.1 Administrivia . . . . . . . . . . . . . . . . . . . . . . . . . 6
   3.2 Identifiers and Binding . . . . . . . . . . . . . . . . . . . . 7
   3.3 Semantics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   3.4 Values and Substitution . . . . . . . . . . . . . . . . . . . . 8
   3.5 Goals of Module 1 . . . . . . . . . . . . . . . . . . . . . . . 9

4 Lecture 04 9
   4.1 Administrivia . . . . . . . . . . . . . . . . . . . . . . . . . 9
   4.2 The Design Recipe - Examples . . . . . . . . . . . . . . . . 9
   4.3 Tests/Examples - Check-Expect/Check-Within . . . . . . . . 10
1 Lecture 01

Outline

1. Administrivia
2. Introduction to CS 115 - Course Website and Slides 1-9
3. Introduction to DrRacket - Slides 10-20

1.1 Administrivia

1. Labs start this week.
2. I will lecture on the blackboard using the course slides as a resource. I will announce pre-reading for each future lecture.
3. Register your iClicker exactly once. The course staff will contact you at the end of January if your iClicker is not yet correctly registered.

1.2 Introduction to CS 115 - Course Website and Slides 1-9

Refer to the slides.

1.3 Introduction to DrRacket - Slides 10-20

Programming language design

- Imperative:
  - frequent changes to data
  - examples: machine language, Java, C++
- Functional:
examples: Excel formulas, LISP, ML, Haskell, Mathematica, XSLT, R (used in STAT 231)
more closely connected to mathematics
easier to design and reason about programs

We use DrRacket, a functional program, in CS 115. DrRacket is great for teaching, although I have yet to see a real computer system developed in it. DrRacket is Turing Complete, so in theory any computer system could be developed using it. DrRacket provides an easy entry point into coding.

You will work in the imperative language, Python, in CS 116. Real computer systems are written in Python.

Themes of the course

• design (the art of creation)
• abstraction (finding commonality, not worrying about details)
• refinement (revisiting and improving initial ideas)
• syntax (how to say it), expressiveness (how easy it is to say and understand), and semantics (the meaning of what’s being said)
• communication (in general)

Functions: A mathematical function definition consists of

• the name of the function,
• its parameters (aka argument) (what the function consumes), and
• a mathematical expression using the parameters, to define what the function produces. The mathematical expression is evaluated by substitution.

Functions in DrRacket:

• As in the slides, we start with \( f(x) = x^2 \) and \( g(x, y) = x - y \).
• Include exactly one set of parentheses for each function call.
• Write the function first inside the open parenthesis, followed by the arguments.
• Observe that following this syntax removes any ambiguity about the order of operations.
• Using this setup, try some examples of your own in DrRacket.
2 Lecture 02

Outline

1. Administrivia
2. Mathematical Functions in DrRacket
3. The DrRacket Environment
4. Defining New Functions in DrRacket

2.1 Administrivia

1. Email Barb Daly (contact information in Personnel page) with your issues with iClicker registration or Markus setup.

2.2 Mathematical Functions in DrRacket

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>(- 3 2)</td>
</tr>
<tr>
<td>3 - 2 + 4/5</td>
<td>(+ (- 3 2) (/ 4 5))</td>
</tr>
<tr>
<td>(6 - 4)(3 + 2)</td>
<td>(* (- 6 4) (+ 3 2))</td>
</tr>
</tbody>
</table>

- CQ 2
- CQ 3

2.3 The DrRacket Environment

Built-in Functions - Examples

Integer division (in the positive integers to keep things simple for a start) produces a quotient and remainder, for example

- Dividing 17 by 5 produces the quotient 3 and the remainder 2 because $17 = 3 \cdot 5 + 2$.
- Dividing 15 by 5 produces the quotient 3 and the remainder 0 because $15 = 3 \cdot 5 + 0$.
- If dividing $a$ by $b$ produces the quotient $q$ and the remainder $r$, so that $a = q \cdot b + r$, then requiring $0 \leq r < b$ makes the choice of $q$ and $r$ unique, so that quotient and remainder are actually functions.

- $(\text{quotient} \ 75 \ 7)$
Bad Syntax / Semantics - Examples

- (* (5) 3)
- (+ (* 2 4))
- (5 * 14)
- (* + 3 5 2)
- (/ 25 0)

CQ 4

2.4 Defining New Functions in DrRacket

- DrRacket enforces correct syntax at all times.
- If you enter an expression which is not syntactically correct, then you will get an error message.
- Every syntactically correct DrRacket function call has the form (functionname exp1 . . . expk), for some function functionname and expressions exp1 ... expk.

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x) = x^2 )</td>
<td>(define (f x) (* x x))</td>
</tr>
<tr>
<td>( g(x, y) = x - y )</td>
<td>(define (g x y) (- x y))</td>
</tr>
</tbody>
</table>

- `define` is a special form; it looks like a DrRacket function, but not all of its arguments are evaluated.
- It binds a name to an expression (which uses the parameters that follow the name).

Identifiers

- To give names to the function and parameters, we use identifiers.
- Syntax rule: an **identifier** starts with a letter, and can include letters, numbers, hyphens, underscores, and a few other punctuation marks.
- It cannot contain spaces or any of ( ), { }, [ ], ’ ’ “ ”.
- Syntax rule: **function definition** is of the form (define (id1 . . . idk) exp), where exp is an expression and each id is an identifier.
• CQ 5

DrRacket Definitions Window

• can accumulate definitions and expressions
• Run button loads contents into Interactions window
• can save and restore Definitions window
• provides a Stepper to let one evaluate expressions step-by-step
• features include: error highlighting, subexpression highlighting, syntax checking, bracket matching

DrRacket Constants

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 3$</td>
<td>(define k 3)</td>
</tr>
<tr>
<td>$p = k^2$</td>
<td>(define p (* k k))</td>
</tr>
</tbody>
</table>

DrRacket Programs A DrRacket program is a sequence of definitions and expressions.

• The definitions are of functions and constants.
• The expressions typically involve both user-defined and built-in functions and constants.

Programs may also make use of special forms (which look like functions, but don’t necessarily evaluate all their arguments).

• CQ 6

3 Lecture 03

Outline

1. Administrivia
2. Identifiers and Binding
3. Semantics
4. Values and Substitution

3.1 Administrivia

Anything?
3.2 Identifiers and Binding

1. Two functions can use the same parameter name (e.g. our definitions of $f$ and $g$ from last lecture both used $x$, with no conflicts).

2. A constant and a parameter can share the same name, for example

   (define x 3)
   (define (f x) (* x x))

   but this is not very readable and hence is not recommended.

3. Two constants cannot share the same name, for example

   (define x 4)
   (define x 5)

4. A constant and a function cannot share the same name, for example

   (define x 4)
   (define (x y) (* 5 y))

5. The name of a function can be the same as a parameter name of another function, for example

   (define (x y) (* 5 y))
   (define (z x) (* 3 x))

   but this is not very readable and hence is not recommended.

6. The name of a function can be the same as the name of one of its parameters, for example

   (define (x x) (+ 1 x))

   but this is not very readable and hence is not recommended.

Remarks:

1. **define** binds an identifier to

   (a) an expression in the case of a constant.
   (b) a function declaration (arguments plus expression) in the case of a function.
3.3 Semantics

- **Syntax** is concerned with deciding which expressions are written correctly for DrRacket to process them.
- **Semantics** is concerned with assigning a value to each syntactically correct DrRacket expression.
- We must make sure that our DrRacket programs are unambiguous (i.e. have exactly one interpretation).
- A DrRacket program is a sequence of definitions and expressions.
- Our model involves the simplification of the program using a sequence of steps (i.e. substitution rules).
- Each step yields a valid DrRacket program.
- A fully-simplified program is a sequence of definitions which ends in a value (see Definition 3.4.1).

3.4 Values and Substitution

**Definition 3.4.1.** A value is an expression which cannot be further simplified.

- For example, 3 is a value, but (+ 3 5) and (f 3 2) are not.
- To ensure that we all agree on how to simplify any expression, we adopt the convention that we always simplify the leftmost expression requiring simplification first.
- As in the slides, we use $\Rightarrow$ to separate steps in a simplification.

**Examples:**

1. (define (f x) (* x x))
   (define (g x y) (- x y))
   (g (f 2) (g 3 1))
   ⇒ (g (* 2 2) (g 3 1))
   ⇒ (g 4 (g 3 1))
   ⇒ (g 4 (- 3 1))
   ⇒ (g 4 2)
   ⇒ (- 4 2)
   ⇒ 2
2. (define (term x y) (* x (sqr y)))
   (term (- 5 3) (+ 1 2))
   ⇒ (term 2 (+ 1 2))
   ⇒ (term 2 3)
   ⇒ (* 2 (sqr 3))
   ⇒ (* 2 9)
   ⇒ 18

- **Exercise:** Run the Stepper on these two examples yourself.
- Be prepared to demonstrate tracing on exams.
- The Stepper may not use the same rules as our convention. Write your own trace to be sure you understand your code.

- **CQ 7**

3.5 Goals of Module 1

Refer to Slides 52-53.

4 Lecture 04

Outline

1. Administrivia
2. The Design Recipe - Examples (M2:1-20)
3. Tests/Examples - Check-Expect/Check-Within (M2:21-26)

4.1 Administrivia

1. Your submission for a01 need **not** follow the design recipe.
2. In DrRacket, #i indicates an **inexact** value.

4.2 The Design Recipe - Examples

**Background:** Recall that the roots (i.e. the values for \( x \) which make the polynomial equal 0) of the quadratic polynomial \( ax^2 + bx + c \) are given by the **quadratic formula**

\[
\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\]
1. Solve, using the design recipe:

**Problem:** Write a DrRacket function `quadratic-root-right` which consumes real numbers `a > 0`, `b` and `c` and produces the the right-hand root \(-\frac{b+\sqrt{b^2-4ac}}{2a}\) of the quadratic polynomial \(ax^2+bx+c\). For now you may assume that \(b^2-4ac \geq 0\).

**Solution:** Solve it from scratch with examples / tests developed by the class. Refer to the associated DrRacket file for the finished version.

2. **Exercise:** Re-do the example for `quadratic-root-left`, and the root \(-\frac{b-\sqrt{b^2-4ac}}{2a}\).

3. **Exercise:** Re-do the example with \(b^2-4ac < 0\).

**Examples of Functions that can return inexact values:**

1. `/`, division
2. `sqrt`, square root
3. `expt`, exponentiation, like on a01.
4. later, `sin`, `cos`, etc, trig functions.

- **CQ 1**
- **CQ 2**

4.3 **Tests/Examples - Check-Expect/Check-Within**

1. Use `check-expect` if DrRacket will produce the expected value exactly.
2. Use `check-within` if DrRacket will produce an inexact value.

We saw both of these in the preceding example!

- **CQ 3**

5 **Lecture 05**

Outline

1. Administrivia
2. String Functions (M2:28-31)
3. Helper Functions (M2:32-36)
4. Using Constants (M2:37-41)
5. Summary of Module 2 (M2:42-43)
5.1 Administrivia

1. In the example from last time, we can enforce $a > 0$, instead of $a \neq 0$. Then there is no ambiguity about which root is the right hand root.
2. I will post all my .rkt example files on my Instructor Specific page soon.

5.2 String Functions

Examples:

1. Problem: Develop a function start-end-middle, which will consume a string s and construct a new string from it, composed of
   
   (a) the first character,  
   (b) the last character and  
   (c) the middle, i.e. the substring lying strictly between the first and the last characters.

   For example, (start-end-middle "angle") ⇒ "aengl".

   Solution: Solve with the class; post the DrRacket file on your Instructor Specific Page afterward.

• CQ 4

5.3 Helper Functions

Remarks:

1. We already used helper functions in the preceding example.

5.4 Using Constants

Remarks:

1. In CS 115, the first natural number is 0, not 1. You may have used the convention that the first natural number is 1 in previous courses.

Examples:
1. **Problem:** Develop a function `electricity-charge`, which will consume natural numbers `total-kwh` and `high-rate-kwh` and produce the total charge, assuming that high rate kWh are charged at $0.15 and low rate kWh are charged at $0.06. Assume that `total-kwh` ≥ `high-rate-kwh`. For example, `(electricity-charge 10 5) ⇒ 1.05`.

**Solution:** Solve with the class; post the DrRacket file on your Instructor Specific Page afterward.

- CQ 5

5.5 **Summary of Module 2**

Refer to slides 42-43.