Welcome to CS 135 (Winter 2017)

Instructor: Victoria Sakhnini

Web page (main information source):

`http://www.student.cs.uwaterloo.ca/~cs135/`

Other course personnel: ISAs (Instructional Support Assistants), IAs (Instructional Apprentices), ISC (Instructional Support Coordinator): see website for details

Lectures: 2 sections on Tuesdays and Thursdays

Tutorials: 3 sections on Fridays

Computer labs: MC 3003, 3004, 3005, 3027, 2062, 2063

Textbook: "How to Design Programs" (HtDP) by Felleisen, Flatt, Findler, Krishnamurthi (`http://www.htdp.org`)

Presentation handouts: available on Web page **and** as printed coursepack from media.doc (MC 2018)

Marking Scheme: 20% assignments (roughly weekly), 25% midterm, 5% participation, 50% final

$\Rightarrow$ **You must pass both assignments and weighted exams in order to pass the course.**

Software: DrRacket v6.7 (`http://racket-lang.org`)

Class participation mark

- Based on use of “clickers” (purchase at Bookstore, register as part of Assignment 0)
- Purpose: to encourage active learning and provide real-time feedback
- Several multiple-choice questions during each lecture
- Marks for answering (more for correct answer)
- Best 75% over whole term used for 5% of final grade

CS 135 Survival Guide

- Available on the course Web page
- Read it as soon as possible
- You must do your own work in this course.
- Completing assignments is the key to success.
- Policy 71 - Student Discipline: plagiarism, sharing assignments, etc.

Intellectual Property

The teaching material used is CS 135 are the property of its authors. This includes:

- Lecture slides and instructor written notes
- Assignment specifications and solutions
- Tutorial slides and notes
- Examinations and solutions

Sharing this material without the IP owner's permission is a violation of IP rights.

More suggestions for success

- Keep up with the readings (keep ahead if possible).
- Take notes in lecture.
- Start assignments early.
- Get help early.
- Follow our advice on approaches to writing programs (e.g. design recipe, templates).

- Keep on top of your workload.
- Visit office hours to get help.
- Integrate exam study into your weekly routine.
- Go beyond the minimum required (e.g. do extra exercises).
- Maintain a “big picture” perspective: look beyond the immediate task or topic.
- Go over your assignments and exams; learn from your mistakes.
- Read your mail sent to your UW email account.

Programming language design

Imperative: based on frequent changes to data

- Examples: machine language, Java, C++, Turing, VB

Functional: based on the computation of new values rather than the transformation of old ones.

- Examples: Excel formulas, LISP, ML, Haskell, Erlang, F#, Mathematica, XSLT, Clojure.
- More closely connected to mathematics
- Easier to design and reason about programs

Racket

- a functional programming language
- minimal but powerful syntax
- small toolbox with ability to construct additional required tools
- interactive evaluator
- used in education and research since 1975
- a dialect of Scheme

Functional and imperative programming share many concepts.

However, they require you to think differently about your programs.

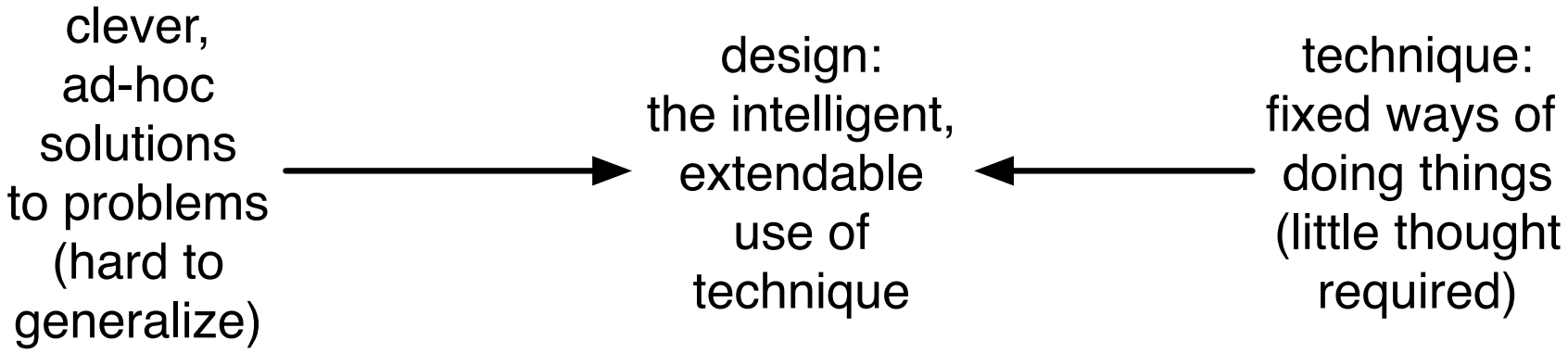
If you have had experience with imperative programming, you may find it difficult to adjust initially.

By the end of CS 136, you will be able to express computations in both these styles, and understand their advantages and disadvantages.

Ideas behind the course

- An introduction to basic concepts in computer science
- Not a course in programming in Racket
- Teaching languages used are a small subset of standard Racket (plus some additional features)
- CS 136 continues with Racket and introduces the C language (imperative, industrially-relevant)

We will cover the whole process of designing programs.



Careful use of design processes can save time and reduce frustration, even with the fairly small programs written in this course.

Themes of the course

- Design (the art of creation)
- Abstraction (finding commonality, neglecting 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 in mathematics

A function generalizes similar expressions.

$3^2 + 4(3) + 2$

$6^2 + 4(6) + 2$

$7^2 + 4(7) + 2$

These are generalized by the function
$f(x) = x^2 + 4x + 2$.

Functions in mathematics

Definition: $f(x) = x^2$, $g(x, y) = x + y$

These definitions consist of:

- the name of the function (e.g. g)
- its **parameters** (e.g. x, y)
- an algebraic expression using the parameters

An **application** of a function: $g(1, 3)$

An application supplies **arguments** for the parameters, which are substituted into the algebraic expression.

Example: $g(1, 3) = 1 + 3 = 4$

The arguments supplied may themselves be applications.

Example: $g(g(1, 3), f(2))$

We **evaluate** each of the arguments to yield values.

Evaluation by **substitution:**

$$g(g(1, 3), f(2)) = g(4, f(2)) = g(4, 4) = 8$$

Note the left-to-right, innermost ordering chosen.

We now enforce this.

$$g(g(1,3),f(2)) = g(g(1,3),4)$$

Mathematically, this is a valid substitution.

We now disallow this substitution.

We insist that when we have a choice of possible substitutions, we take the **leftmost** one.

$$g(g(1,3),f(2)) = g(1,3) + f(2)$$

Mathematically, this is a valid substitution.

We now disallow this substitution.

We insist that we only apply a function when all of its arguments are values (not general expressions).

Now, for any expression:

- there is at most one choice of substitution;
- the computed final result is the same as for other choices.

The use of parentheses: ordering

In arithmetic expressions, we often place operators between their operands.

Example: $3 - 2 + 4/5$.

We have some rules (division before addition, left to right) to specify order of operation.

Sometimes these do not suffice, and parentheses are required.

Example: $(6 - 4)/(5 + 7)$.

The use of parentheses: functions

If we treat infix operators ($+$, $-$, etc.) like functions, we don't need parentheses to specify order of operations:

Example: $3 - 2$ becomes $-(3, 2)$

Example: $(6 - 4)/(5 + 7)$ becomes $/(-(6, 4), +(5, 7))$

The substitution rules we developed for functions now work uniformly for functions and operators.

Parentheses now have only one use: function application.

The use of parentheses: functions

Racket writes its functions slightly differently: the function name moves *inside* the parentheses, and the commas are changed to spaces.

Example: $g(1, 3)$ becomes (g 1 3)

Example: $(6 - 4)/(5 + 7)$ becomes (/ (− 6 4) (+ 5 7))

These are valid Racket expressions (once g is defined).

Functions and mathematical operations are treated exactly the same way in Racket.

Expressions in Racket

$3 - 2 + 4/5$ becomes (+ (− 3 2) (/ 4 5))

$(6 - 4)(3 + 2)$ becomes (∗ (− 6 4) (+ 3 2))

Extra parentheses are harmless in arithmetic expressions.

They are harmful in Racket.

Only use parentheses when necessary (to signal a function application or some other Racket syntax).

The DrRacket environment

- Designed for education, powerful enough for “real” use
- Sequence of language levels keyed to textbook
- Includes good development tools
- Two windows: Interactions and Definitions
- Interactions window: a read-evaluate-print loop (REPL)

Setting the Language in DrRacket

CS 135 will progress through the Teaching Languages starting with *Beginning Student*.

- 1. Under the *Language* tab, select *Choose Language ...*
- 2. Select *Beginning Student* under *Teaching Languages*
- 3. Click the *Show Details* button in the bottom left
- 4. Under *Constant Style*, select *true false empty*

Remember to follow steps 3 and 4 each time you change the language.

Note about Constant Style

In the DrRacket documentation, you will see #true, #false, '() instead of true, false, empty, respectively.

In CS 135 exams and stepper questions you must use true, false, empty.

Numbers in Racket

- Integers in Racket are unbounded.
- Rational numbers are represented exactly.
- Expressions whose values are not rational numbers are flagged as being **inexact**.

  Example: (sqrt 2) evaluates to #i1.414213562370951.

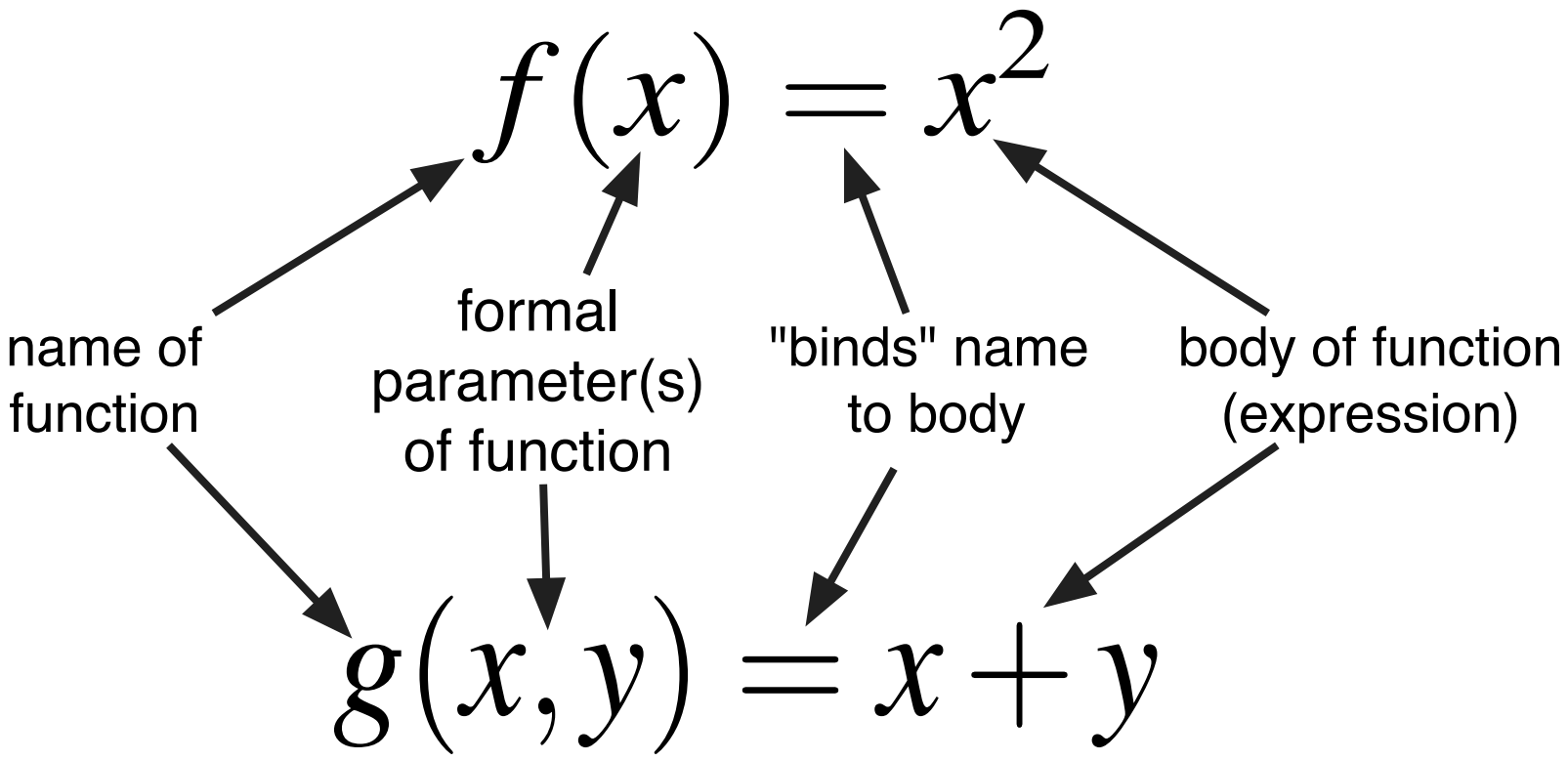
  We will not use inexact numbers much (if at all).

Racket expressions causing errors

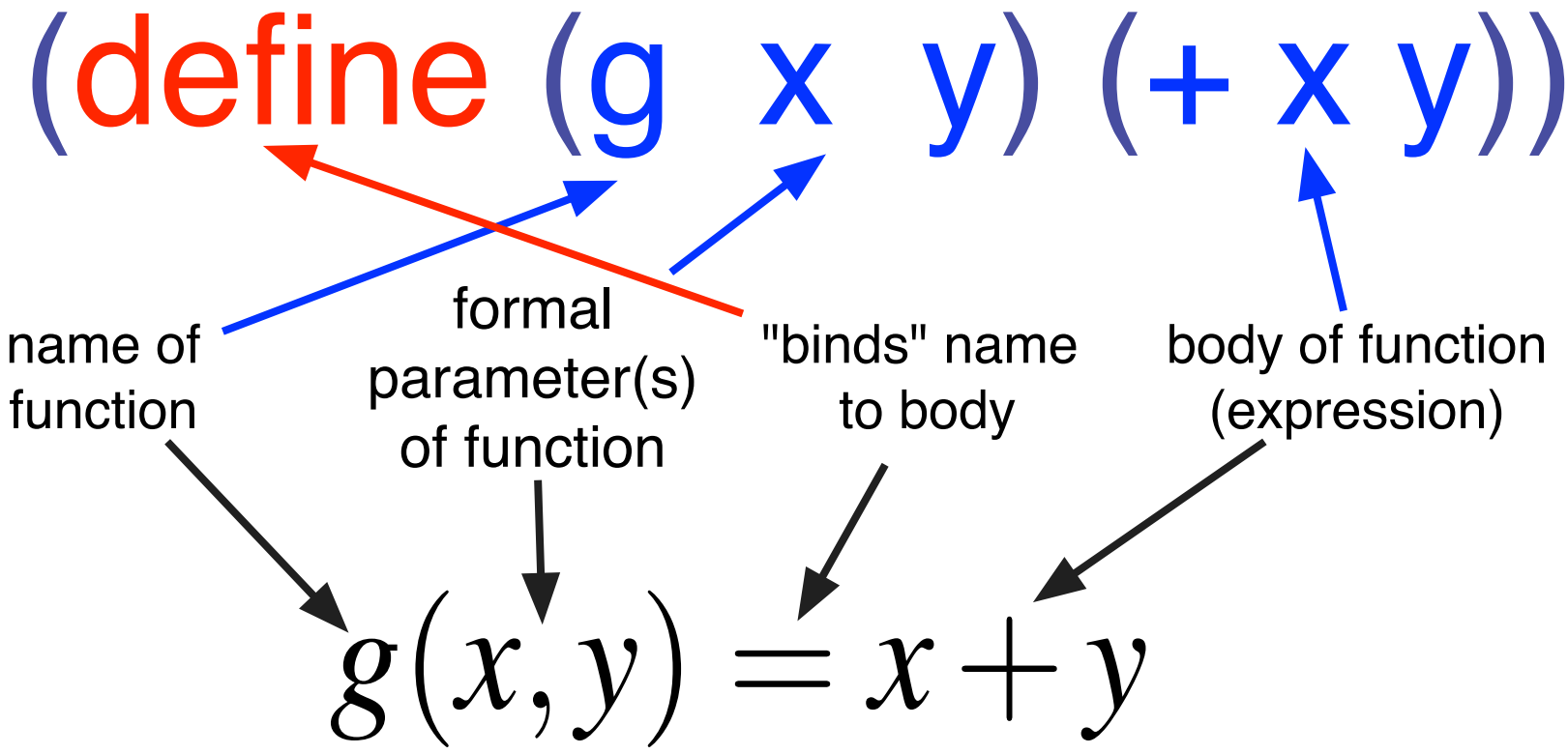
What is wrong with each of the following?

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

Defining functions in mathematics



Defining functions in Racket



Defining functions in Racket

Our definitions $f(x) = x^2$, $g(x, y) = x + y$ become

```
(define (f x) (* x x))
(define (g x y) (+ x y))
```

define is a **special form** (looks like a Racket function, but not all of its arguments are evaluated).

It **binds** a name to an expression (which uses the parameters that follow the name).

A function definition consists of:

- a name for the function,
- a list of parameters,
- a single “body” expression.

The body expression typically uses the parameters together with other built-in and user-defined functions.

An application of a user-defined function substitutes some values for the corresponding parameters in the definition's expression.

```
(define (g x y) (+ x y))
```

The substitution for (g 3 5) would be (+ 3 5).

Evaluating a Racket expression

We use a process of substitution.

Each step is indicated using the 'yields' symbol $\Rightarrow$.

$(* \ (- \ 6 \ 4) \ (+ \ 3 \ 2)) \Rightarrow (* \ 2 \ (+ \ 3 \ 2)) \Rightarrow (* \ 2 \ 5) \Rightarrow 10$

This mirrors how we work with mathematical expressions.

The same process works with user-defined functions, using the substitution into the body of the function we described earlier.

(g (g 1 3) (f 2))

$\Rightarrow$ (g (+ 1 3) (f 2))

$\Rightarrow$ (g 4 (f 2))

$\Rightarrow$ (g 4 (* 2 2))

$\Rightarrow$ (g 4 4)

$\Rightarrow$ (+ 4 4)

$\Rightarrow$ 8

Each parameter name has meaning only within the body of its function.

```
(define (f x y) (+ x y))

(define (g x z) (* x z))
```

The two uses of x are independent.

Additionally, the following two function definitions define the **same** function:

```
(define (f x y) (+ x y))

(define (f a b) (+ a b))
```

Defining constants in Racket

The definitions $k = 3, p = k^2$ become

```
(define k 3)
(define p (* k k))
```

The effect of (define k 3) is to bind the name k to the value 3.

In (define p (* k k)), the expression (* k k) is first evaluated to give 9, and then p is bound to that value.

In the body of a function, a parameter name "shadows" a constant of the same name.

```
(define x 3)

(define (f x y)
   (- x y))
```

(+ x x) ⇒ (+ 3 x) ⇒ (+ 3 3) ⇒ 6

(f 7 6) ⇒ (− 7 6) ⇒ 1

(f 5 x) ⇒ (f 5 3) ⇒ (− 5 3) ⇒ 2

Advantages of constants

- Can give meaningful names to useful values (e.g. interest-rate, passing-grade, and starting-salary).
- Reduces typing and errors when such values need to be changed
- Makes programs easier to understand
- Constants can be used in any expression, including the body of function definitions
- Sometimes called variables, but their values cannot be changed (until CS 136)

DrRacket's 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: error highlighting, subexpression highlighting, syntax checking

Programs in Racket

A Racket program is a sequence of definitions and expressions.

The expressions are evaluated, using substitution, to produce values.

Expressions may also make use of **special forms** (e.g. define), which look like functions, but don't necessarily evaluate all their arguments.

Goals of this module

You should understand the basic syntax of Racket, how to form expressions properly, and what DrRacket might do when given an expression causing an error.

You should be comfortable with these terms: function, parameter, application, argument, constant, expression.

You should be able to define and use simple arithmetic functions.

You should understand the purposes and uses of the Definitions and Interactions windows in DrRacket.