Functions

Readings:

• HTDP, sections 1-3
• Survival and Style guides

Topics:

• Programming language design [1-4]
• The DrRacket environment [5-6]
• Values, expressions, & functions [7-21]
• Defining functions [22-30]
• Scope [31]
• Programming in DrRacket [32-34]
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
- graduated set of teaching languages are a subset of Racket
Functional vs. imperative

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.
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 (now) and Definitions (later)

• 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.
Values, expressions, & functions

Values are *numbers or other mathematical objects*. Examples: $5, \frac{4}{9}, \pi$.

Expressions *combine values with operators and functions*. Examples: $5 + 2, \sin(2\pi), \frac{\sqrt{2}}{100\pi}$.

Functions *generalize similar expressions*. Example...
Values, expressions, & functions (cont)

Values are numbers or other mathematical objects.

Expressions combine values with operators and functions.

Functions generalize similar expressions.

Example:

\[ 3^2 + 4(3) + 2 \]
\[ 6^2 + 4(6) + 2 \]
\[ 7^2 + 4(7) + 2 \]

are generalized by the function

\[ f(x) = x^2 + 4x + 2. \]
Functions in mathematics

Definitions: \( f(x) = x^2, \ g(x, y) = x + y \)

Function definitions in mathematics consist of three components:

1. the name of the function (e.g. \( g \))
2. its parameters (e.g. \( x, y \))
3. an algebraic expression using the parameters as placeholders for values to be supplied in the future
Function application

Definitions: \( f(x) = x^2 \), \( g(x, y) = x + y \)

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

An example: \( g(1, 3) = 1 + 3 = 4 \)

The arguments supplied may themselves be applications.

Example: \( g(g(1, 3), f(3)) \)
Function application (cont)

Definitions: \( f(x) = x^2 \), \( g(x, y) = x + y \)

We evaluate each of the arguments to yield values.

Evaluation by substitution:

\[
g(g(1, 3), f(3)) = \\
g(1 + 3, f(3)) = \\
g(4, f(3)) = \\
g(4, 3^2) = \\
g(4, 9) = 4 + 9 = 13
\]
Many possible substitutions

Definitions: \( f(x) = x^2 \), \( g(x, y) = x + y \)

There are many mathematically valid substitutions:

\[
g(g(1, 3), f(3)) = g(1 + 3, f(3))\ldots
\]

\[
g(g(1, 3), f(3)) = g(g(1, 3), 3^2)\ldots
\]

\[
g(g(1, 3), f(3)) = g(1, 3) + f(3)\ldots
\]

We’d like a **canonical form** (**a standard form**) for two reasons:

- Easier for us to think about
- When we extend this idea to programming, we’ll find cases where different orderings result in different values
Canonically substituted

There are two rules for canonical substitutions

1. *Functions are applied to values*

2. When there is a choice of possible substitutions, *always take the leftmost choice.*

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: $(6 - 4)/(5 + 7)$.
Infix vs. Prefix Notation

- **Infix notation** *operators go between values*: 1 + 2.
  - need precedence rules to determine order of evaluation
  - **BEDMAS**: brackets, exponents, division/multiplication, addition/subtraction
  - if operators have the same precedence (e.g. addition/subtraction) then evaluation left to right

- **Prefix notation**: *operators go before values*: (+ 1 2)
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 \((\ast (- 6 4) (+ 3 2))\)

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).
Evaluating a Racket expression

*We use a process of substitution*, just as with our mathematical expressions.

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

$$(\ast (\neg 6 4) (\oplus 3 2))$$

$\Rightarrow (\ast 2 (\oplus 3 2))$

$\Rightarrow (\ast 2 5)$

$\Rightarrow 10$
Numbers in Racket

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

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

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

Racket has many built-in functions which can be used in expressions:

- Arithmetic operators: $+, -, \ast, /$
- Constants: $e, \pi$
- Functions: $(\text{abs } x), (\text{max } x \; y \; \ldots), (\text{ceiling } x) \; (\text{expt } x \; y), (\text{exp } x), \; (\text{cos } x), \ldots$

Look in DrRacket’s “Help Desk”. The web page that opens has many sections. The most helpful is under Teaching, then “How to Design Programs Languages”, section 1.5.
Racket expressions causing errors

What is wrong with each of the following?

• \((5 \times 14)\)

• \((\times (5) 3)\)

• \((+ (\times 2 4)\)

• \((\times + 3 5 2)\)

• \((/ 25 0)\)
Defining functions (in mathematics)

\[ f(x) = x^2 \]

\[ g(x, y) = x + y \]
Defining functions (in Racket)

\[ g(x, y) = x + y \]

\[
\text{define} \quad (g \ x \ y) \ (\ + \ x \ y )
\]

- **name of function**
- **formal parameter(s) of function**
- "binds" name to body
- **body of function (expression)"**

CS 135 Fall 2019 02: Functions 23