Functions

Readings:

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

Topics:

- Programming language design
- The DrRacket environment
- Values, expressions, & functions
- Defining functions
- Programming in DrRacket

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, $4/9$, π.

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

These definitions consist of:

- the name of the function (e.g. g)
- its **parameters** (e.g. x, y)
- 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:

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

We'd like a canonical 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

Canonical substitutions

Two rules:

- Functions are applied to values
- 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: $(6 - 4)/(5 + 7)$.

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).

Evaluating a Racket expression

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

Each step is indicated using the 'yields' symbol ⇒ .

(∗ (− 6 4) (+ 3 2))

⇒ (∗ 2 (+ 3 2))

⇒ (∗ 2 5)

⇒ 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 **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: $+$, $-$, $*$, /
- Constants: e, pi
- Functions: (abs x), (max x y ...), (ceiling x) (expt x y), (exp x), (cos x), ...

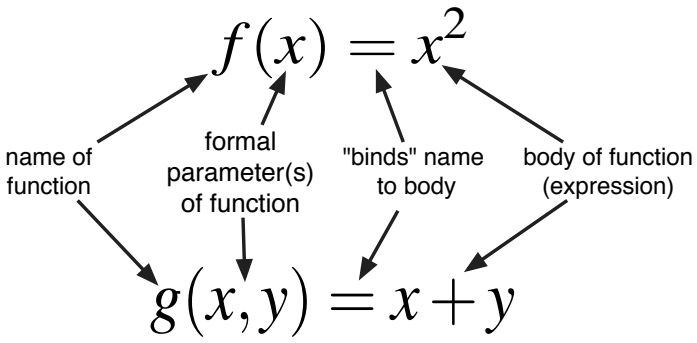
Look in DrRacket's "Help Desk". The web page that opens has many sections. The most helpful is under **Teaching**, then "*How to Design Programs* Languages", section 1.5.

Racket expressions causing errors

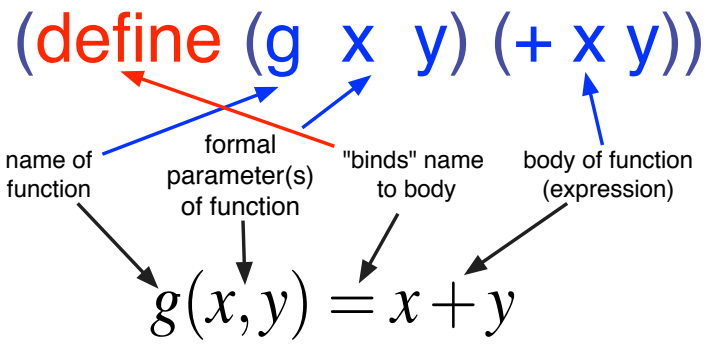
What is wrong with each of the following?

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

Defining functions (in mathematics)



Defining functions (in Racket)



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

```
(define (f x) (sqr 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.

Applying user-defined functions in Racket

An application of a user-defined function substitutes arguments for the corresponding parameters throughout the definition's expression.

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

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

When faced with choices of substitutions, we use the same rules defined earlier: apply functions only when all arguments are simple values; when you have a choice, take the leftmost one.

(g (g 1 3) (f 3))
⇒ (g (+ 1 3) (f 3))
⇒ (g 4 (f 3))
⇒ (g 4 (sqr 3))
⇒ (g 4 9)
⇒ (+ 4 9)
⇒ 13

$$
\begin{aligned}
& g(g(1,3), f(3)) \\
&= g(1+3, f(3)) \\
&= g(4, f(3)) \\
&= g(4, 3^2) \\
&= g(4, 9) \\
&= 4 + 9 \\
&= 13
\end{aligned}
$$

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

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

```
(define k 3)
(define p (sqr k))
```

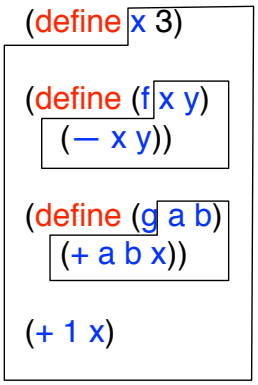
The effect of (define k 3) is to bind the name k to the value 3.

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

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 (incorrectly) called variables, but their values cannot be changed (until CS 136)

Scope: where an identifier has effect within the program.



- The smallest enclosing scope has priority
- Can't duplicate identifiers within the same scope

```
(define f 3)
(define (f x) (sqr x))
Racket Error: f: this name was defined...
```


Programming in DrRacket

Use the definitions window:

- Can save and restore your work to/from a file
- Can accumulate definitions and expressions
- Run button loads contents into Interactions window
- Provides a Stepper to let one evaluate expressions step-by-step
- Features: error highlighting, subexpression highlighting, syntax checking

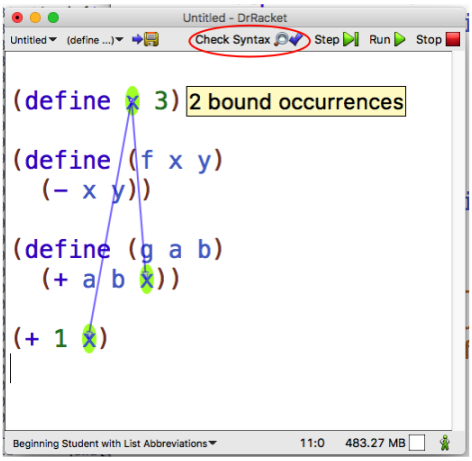
Programs in Racket

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.

Scope in DrRacket



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.