CS135 L02 — Functions, Substitution, Inexact Numbers

Luke Lu • 2025-12-17

Function Definitions (Math → Racket)

In CS 135, a function definition specifies a **name**, **parameters**, and one **body expression**.

Math form → Racket (prefix)

```
; f(x) = x^2 + 3x + 4
(define (f x) (+ (* x x) (* 3 x) 4))

; g(x, y) = x^2 + 6xy + y^2 + 9x - 3y - 100
(define (g x y) (+ (sqr x) (* 6 x y) (sqr y) (* 9 x) (- (* 3 y)) -100))
```

> 💡 **Tip** — Parameters are **local**; renaming them consistently doesn't change behavior. The **order** of parameters **does** matter.

Substitution Model — How Evaluation Proceeds

Evaluate the **innermost, leftmost** reducible part each step.

Example 1 — Built-ins first

```
(+ (* 3 2) 5)
; step 1
(+ 6 5)
; step 2
11
```

Example 2 — User-defined call

```
(define (f x) (+ (* x x) (* 3 x) 4))
(f 2)
; substitute x = 2 everywhere in f's body
(+ (* 2 2) (* 3 2) 4)
(+ 4 6 4)
14
```

Example 3 — Nested call (argument first)

```
(define (h x) (+ (sqr x) (* 2 x) 1))
(h (+ 1 2))
; evaluate argument first
(h 3)
(+ (sqr 3) (* 2 3) 1)
(+ 9 6 1)
16
```

> ⚠️ **Warning** — Do **not** partially substitute. Replace **all** occurrences of a parameter in one step for that call for functions.
> Subsitute paritially for predefined constants.

Boolean Operators & Short-Circuit

```
(and p q r) ; false if any argument is false (stops early)
(or  p q r) ; true  if any argument is true  (stops early)
(not p)     ; flips truth
```

Quick check

```
(and (> 3 1) (= 2 2) (< 5 4)) ; #false, last term is false
```

Identifier Rules (Legal Names)

- Letters, digits, and `- _ . ? =` allowed; must contain ≥**1 non-digit**.
- No spaces, quotes, or brackets.

Good: `cool?`, `is-even?`, `sum3` **Bad:** `3to1` (all digits), `my var` (space), `"name"` (quotes)

Inexact Numbers (Floating Point)

Real values not exactly representable are shown with `#i` and stored **approximately**.

Classic surprise

```
(+ 0.1 0.2)            ; -> #i0.30000000000000004 (approx)
(= (+ 0.1 0.2) 0.3)    ; -> #false
```

Use closeness instead of =

```
(define (close? x y)
  (< (abs (- x y)) 1e-9))

(close? (+ 0.1 0.2) 0.3) ; #true
```

> 🐋 **Info** — 16 decimal digits precision (double). Special values: `+inf.0`, `-inf.0`, `+nan.0`.

Built-ins & Number Kinds

- Math: `abs sqrt log exp sin cos tan asin acos atan`
- Constants: `pi`, `e`
- Predicates: `number? inexact? integer? rational?`
- Course kinds: **Nat, Int, Rat, Num, Bool, Sym**

Comments & Submission Style

- Line comments with `;` (use `;;` for prominent notes).
- Block comments:

```
#| ... multi-line comment ... |#
```

- Handin hygiene:
  - Header block (name, ID, course/term).
  - Lines ≤ 102 chars.
  - Concise **purpose** comments above each function.

Worked Example + Tests

```
;; quad: Num Num Num Num -> Num
;; Purpose: Evaluate ax^2 + bx + c at x.
(define (quad a b c x)
  (+ (* a x x) (* b x) c))

(require rackunit)
(check-expect (quad 1 0 0 5) 25)
(check-expect (quad 1 2 1 3) 16)
```

Practice (do by substitution rules)

```
(+ 3 (* 5 2))
(/ (- 10 4) 2)
(* 7 (+ 3 1))
(- (- (- 10 100)) (- -10 -15) (- -20))
```

Try writing & testing your own:

```
;; square-plus: Num -> Num
;; Purpose: return x^2 + x.
(define (square-plus x) (+ (* x x) x))
(require rackunit)
(check-expect (square-plus 3) 12)
```