Racket

```
; Function definition and application
(lambda (x y) (* x y))
(lambda () 16)

(+ 3 4 5)                  ; 12
(equal? 3 (- 4 25))        ; #f
((lambda (x) (+ 3 x)) 10)  ; 13

; Name bindings
(define x 10)
(define (f z) (first (rest z)))
(define f2 (lambda (z) (first (rest z))))
(let* ([y (+ 10 20)]
       [z (+ 10 y)])
  (* y z))

; Syntactic forms
(and #f (/ 1 0))      ; #f
(or #t (/ 1 0))       ; #t
(if #t
    42
    (/ 1 0))          ; 42
(cond [#f (/ 1 0)]
      [#t 42]
      [(/ 1 0) 25]
      [else 1])       ; 42

; Pattern matching
(define/match (comment x)
  [(7) "Lucky"]
  [(13) "Unlucky"]
  [(_) "Other"])

(define/match (f lst)
  [((list)) 100]
  [((cons x xs)) (+ x (length xs))])

; Lists
(cons 2 (cons 3 null))      ; '(2 3)
(list 1 2 20)               ; '(1 2 20)
(first (list 1 2 3))        ; 1
(rest (list 1 2 3))         ; '(2 3)
(null? null)                ; #t
(length (list 1 20 5))      ; 3
(append (list 1 2 3)
        (list 4 5 6))       ; '(1 2 3 4 5 6)
(member 2 (list 1 2 3))     ; '(2 3)
(member 4 (list 1 2 3))     ; #f
(list-ref (list 2 40 1) 1)  ; 40
(take (list 1 2 3 4) 2)     ; '(1 2)
(drop (list 1 2 3 4) 2)     ; '(3 4)

(map (lambda (x) (* 3 x))
     (list 1 2 3))                ; '(3 6 9)
(filter (lambda (x) (< 3 x))
        (list 10 -4 15))          ; '(10 15)
(foldl + 15 (list 1 2 3))         ; 21
(apply - (list 16 3))             ; 13
```

Haskell

```
-- Function definition and application
\x -> x + x
\x y -> x * y

max 3 4   -- 4
3 + 4     -- 7
(==) 3 4  -- False

-- Name bindings
x = 10
f z = z + 10
f2 = \z -> z + 10
let y = 20 + 10
    z = y + 10
in  y * z

-- If (note that && and || are just functions)
f x =
  if x == 10
  then 16
  else -20

-- Pattern matching
comment 7 = "Lucky"
comment 13 = "Unlucky"
comment _ = "Other"

f [] = 100
f (x:xs) = x + length xs

-- Lists
2:3:[]                          -- [2,3]
[1,2,20]                        -- [1,2,20]
head [1,2,3]                    -- 1
tail [1,2,3]                    -- [2,3]
null []                         -- True
length [1,2,20]                 -- 3
[1,2] ++ [4,6]                  -- [1,2,4,6]

elem 2 [1,2,3]                  -- True
elem 4 [1,2,3]                  -- False
[20,40,1] !! 1                  -- 40
take 2 [1,2,3,4]                -- [1,2]
drop 2 [1,2,3,4]                -- [3,4]

map (\x -> 3 * x) [1,2,3]       -- [3,6,9]

filter (\x -> 3 < x) [10,-4,15] -- [10,15]

foldl (+) 15 [1,2,3]            -- 21
```

```
; Class macro with `self`
(define-syntax my-class
  (syntax-rules (method)
    [(my-class <Class>
       (<attr> ...)
       (method (<method-name> <params> ...) <body>) ...)

     ; The template
     (begin
       (define class__dict__
         (make-immutable-hash (list
                               (cons (quote <method-name>)
                                     (lambda (<params> ...) <body>))
                               ...)))
       (define (<Class> <attr> ...)
         (letrec
             ([self__dict__
               (make-immutable-hash
                (list (cons (quote <attr>) <attr>) ...))]
              [me (lambda (attr)
                    (cond
                      [(hash-has-key? self__dict__ attr)
                       (hash-ref self__dict__ attr)]
                      [(hash-has-key? class__dict__ attr)
                       (fix-first me (hash-ref class__dict__ attr))]
                      [else
                       ((attribute-error (quote <Class>) attr))]))])

           me)))]))


; Sample usage
(my-class Point
  (x y)

  (method (size self)
          (sqrt (+ (* (self 'x) (self 'x))
                   (* (self 'y) (self 'y)))))

  (method (scale self n)
          (Point (* (self 'x) n) (* (self 'y) n)))

  (method (same-radius self other)
          (equal? ((self 'size)) ((other 'size)))))
```