

## Summary of Scheme features

## Types:

numbers: integers (3, 802), reals (3.4), rationals (3/4), complex (2+3.4i)  
 symbols: x, y, hello, r2d2  
 booleans: #t, #f  
 strings: "hello", "how are you?"  
 lists: (3 4 5) (98.5 "hello" (3 82.9) 73)

## Constructs:

function call: (f arg1 arg2 arg3 ... arg-n)  
 variable binding: (define sym expr)  
 function binding: (define (f p1 p2 ... pn) expr)  
 function binding with helper functions:  
   (define (f p1 p2 ... pn) (define ...) (define ...) expr)  
 let binding: (let ([sym1 e1] [sym2 e2] ... [sym-n en]) expr)  
 let\* binding: (let\* ([sym1 e1] [sym2 e2] ... [sym-n en]) expr)  
 set! assignment: (set! sym expr)  
 if expression: (if test e1 e2)  
 cond expression: (cond (test1 e1) (test2 e2) ... (test-n e-n))  
                   (cond (test1 e1) (test2 e2) ... (else e-n))

## Useful functions

arithmetic: +, -, \*, /, modulo, quotient, remainder  
 mathematical: abs, sin, cos, max, min, expt (exponent), sqrt, floor,  
                 ceiling, truncate, round  
 relational (for numbers): =, <, >, <=, >=  
 equality (for other structures): eq? (pointer), eqv? (value), equal? (deep)  
 logical: and, or, not  
 type predicates: number? integer? real? symbol? boolean? string? list?  
 higher-order: map, filter, foldl, foldr, sort, andmap, ormap  
 list operations:  
   length -- length of a list  
   car -- first element of a list  
   cdr -- rest of the list  
   cons -- takes a value and a list and constructs a new list with the  
           value at the front and the list after  
   append -- joins lists together  
   list -- forms a list from a sequence of values  
   member -- whether a value is in a list  
   remove -- removes one occurrence of a value from a list  
   null? -- is something an empty list?  
   pair? -- is something a nonempty list?

## Summary of Ruby features

numbers: Integer (3, 802), Float (3.4)  
 booleans: true, false  
 strings: "hello", "how are you?"  
 arrays: [3, 15, "hello", [7, "howdy"], 308.4, true]  
 ranges: 1..10, 'a'..'z'

```
class Foo [< <superclass>]
  <definitions>
end
def f(p1, p2, ..., pn)
  <body>
end
def f(p1 = v1, p2 = v2, ..., pn = vn)
  <body>
end
```

## Syntactic sugar for defining getters and setters:

```
attr_reader symbol1, symbol2, ..., symbol-n
attr_writer symbol1, symbol2, ..., symbol-n
```

The constructor for a class always calls a method called initialize.  
public/protected/private keywords used inside a class to mean "starting here, constructs have this kind of access"  
instance-variables start with @  
class-variables (static variables) start with @@  
symbols start with :

```
if <bool-expr> [then]
  <body>
elsif <bool-expr> [then]
  <body>
else
  <body>
end
while <bool-expr> [do]
  <body>
end
for <variable> in <collection> [do]
  <body>
end
loop do
  <statements>
  break if <bool-expr>
  <statements>
end
<variable>.<method>( <params> ) { <block> }
<variable>.<method>( <params> ) do
  <block>
end
<block> can begin with |params| and has a sequence of statements
```

embed an expression to be evaluated in a string:  
"text before #{expression to evaluate} text after"  
a method takes a block if it includes calls on yield  
nil is a special value that represents "no object"

#### Useful methods

arithmetic: +, -, \*, /, %, \*\* (exponentiation), +=, -=, \*=

mathematical: Math.sqrt, Math.sin, Math.cos

relational: <, >, <=, >=, !=, ==

logical: &&, ||, !

conversion: <Integer>.to\_f, <Float>.to\_i, <Float>.round, <Object>.to\_s

random: rand, rand(<integer>)

#### Array/String:

x.length	# of elements
x[i]	element i (0-based indexing)
x[i, n]	slice with n elements starting at i
x[i..j]	slice with values at index i through j
x << v	append v to x
x.sort	sort values
x.member? v	does it contain v?
x.index v	index of v in x
push v	appends v to end of array (Arrays only)
pop	remove and return top value (Arrays only)
chomp	eliminate trailing newline from string (string only)
upcase	uppercase version of string (string only)

#### Input/Output

puts x	like Java println
print x	like Java print (stay on current output line)
print x, y	print multiple values
gets	reads a line of input
File.open(s)	returns a file object for given file
<file>.gets	read a line from file object