Scheme Programming Language Cheatsheet

A concise cheat sheet covering the core syntax, data types, procedures, and control structures of the Scheme programming language, useful for quick reference and learning.

Basic Syntax and Data Types

Core Syntax

`()`	All Scheme code is enclosed in parentheses. This signifies a function call or a special form.
`define`	Used to define variables and procedures. **Example:** `(define x 10)`
`lambda`	Creates anonymous functions (procedures). **Example:** `(lambda (x) (+ x 1))`

Data Types

Numbers	Integers, decimals, fractions. **Example:** `10`, `3.14`, `1/2`
Booleans	`#t` (true) and `#f` (false).
Characters	Represented with `#\`. **Example:** `#\a`, `#\space`
Strings	Sequences of characters enclosed in double quotes. **Example:** `"hello"`
Symbols	Unique identifiers, often used as keys. **Example:** `'symbol`
Lists	Ordered collections of data. **Example:** `'(1 2 3)`

Basic Procedures

`+`, `-`, `*`, `/`	Arithmetic operators. **Example:** `(+ 1 2)` (evaluates to 3)
`=`	Numerical equality comparison. **Example:** `(= 1 1)` (evaluates to `#t`)
`cons`	Constructs a new list by adding an element to the beginning of an existing list. **Example:** `(cons 1 '(2 3))` (evaluates to `'(1 2 3)`)
`car`	Returns the first element of a list. **Example:** `(car '(1 2 3))` (evaluates to `1`)
`cdr`	Returns the rest of the list after the first element. **Example:** `(cdr '(1 2 3))` (evaluates to `'(2 3)`)

Control Structures

Conditional Execution

`if`	Basic conditional statement. `(if condition then-expression else-expression)` **Example:** `(if (= x 10) "x is 10" "x is not 10")`
`cond`	Multi-way conditional. `(cond (condition1 expression1) (condition2 expression2) ... (else expression))` **Example:** `(cond ((> x 0) "positive") ((< x 0) "negative") (else "zero"))`

Iteration and Recursion

Recursion	Scheme primarily uses recursion for iteration. A function calls itself until a base case is reached. **Example:** `(define (factorial n) (if (= n 0) 1 (* n (factorial (- n 1)))))`
`do`	The `do` form provides a looping construct. `(do ((variable initial update) ...) (termination-condition result) body ...)` **Example:** `(do ((i 0 (+ i 1)) (sum 0 (+ sum i))) ((> i 10) sum) (display i) (newline))`

Boolean Operations

`and`	Returns `#t` if all expressions are true. **Example:** `(and (> 5 3) (< 10 20))`
`or`	Returns `#t` if at least one expression is true. **Example:** `(or (> 5 3) (> 10 20))`
`not`	Negates a boolean value. **Example:** `(not (= 5 3))`

Working with Lists

List Manipulation

`list`	Creates a list from given elements. **Example:** `(list 1 2 3)` (evaluates to `'(1 2 3)`)
`append`	Concatenates lists. **Example:** `(append '(1 2) '(3 4))` (evaluates to `'(1 2 3 4)`)
`reverse`	Reverses the order of elements in a list. **Example:** `(reverse '(1 2 3))` (evaluates to `'(3 2 1)`)
`length`	Returns the number of elements in a list. **Example:** `(length '(1 2 3))` (evaluates to `3`)

Mapping and Filtering

`map`	Applies a function to each element of a list and returns a new list with the results. **Example:** `(map (lambda (x) (* x 2)) '(1 2 3))` (evaluates to `'(2 4 6)`)
`filter`	Creates a new list containing only the elements that satisfy a given predicate (a function that returns `#t` or `#f`). **Example:** `(filter (lambda (x) (> x 1)) '(1 2 3))` (evaluates to `'(2 3)`)

List Predicates

`null?`	Checks if a list is empty. **Example:** `(null? '())` (evaluates to `#t`)
`list?`	Checks if a value is a list. **Example:** `(list? '(1 2 3))` (evaluates to `#t`)
`member`	Checks if an element is a member of a list. **Example:** `(member 2 '(1 2 3))` (evaluates to `'(2 3)`)

Input and Output

Basic I/O

`display`	Prints a value to the console. **Example:** `(display "Hello, world!")`
`newline`	Prints a newline character to the console. **Example:** `(newline)`
`read`	Reads a Scheme expression from the input. **Example:** `(read)`

Formatted Output

`format`	Formatted output (implementation-dependent, check your Scheme system's documentation). **Example (Racket):** `(format #t "The value of x is ~a" x)`

File I/O

`open-input-file`	Opens a file for input. **Example:** `(define input-port (open-input-file "data.txt"))`
`open-output-file`	Opens a file for output. **Example:** `(define output-port (open-output-file "output.txt"))`
`read-char`	Reads a single character from an input port. **Example:** `(read-char input-port)`
`write-char`	Writes a single character to an output port. **Example:** `(write-char #\a output-port)`
`close-input-port`	Closes an input port. **Example:** `(close-input-port input-port)`
`close-output-port`	Closes an output port. **Example:** `(close-output-port output-port)`