Scheme

✦ Dynamically typed

✦ Data types:
  - Booleans:
    - #t, #f, boolean?, not, and, or
    - Every Scheme value except #f is treated as true when used in a context where a boolean value is expected
  - Numbers:
    - +, -, *, /, =, >, <, >=, <=
    - number?, integer?, rational?, real?, exact?, complex?
  - Characters:
    - #\c, where c is the character (E.g. #\a represents the character “a”)
    - char?
  - Strings:
    - A fixed-length sequence of characters (E.g. “Hello”)
    - string?, string-length, string-ref, string
Expression

✦ Cambridge prefix notation for *all* Scheme expressions

  • Expressions are surrounded by parentheses
  
  • Operator or function precedes its operands
  
  • Allows arithmetic operators to take an arbitrary number of operands

✦ Comments begin with ;
Expression Example

✦ (+ 2 2) ; evaluates to 4
✦ (+ (* 5 4)(- 6 2)) ; means 5 * 4 + (6 - 2)
✦ (define f 120) ; define a global variable
Expression Evaluation

✧ Three major rules:

1. Replace names of symbols by their current bindings.

2. Evaluate lists as function calls in Cambridge prefix.

3. Constants evaluate to themselves.

✧ Prevent a symbol or list from being evaluated by quoting it using the quote function or the apostrophe (’).
Evaluation Examples

✦ Example 1:

```
(define f 120) ; define a variable f equal to 120
f ; evaluates to 120
(+ f 5) ; evaluates to 125, using the binding for +, f
f ; evaluates to 120, demonstrates Scheme has no notion of state
```

✦ Example 2:

```
(+ ) ; calls + with no arguments
(+ 5) ; calls + with 1 arguments
(+ 5 4 3 2 1) ; calls + with 5 arguments
(+ (5 4 3 2 1)) ; error, tries to evaluate 5 as a function
(f) ; error, f evaluates to 120, not a function
```

✦ Example 3:

```
5 ; evaluates to 5
#f ; is false, predefined
(define x 'f) ; defines x to be the symbol f
(define acolor red) ; an error, unbound variable red
```
List

✦ List is the fundamental data structure of Scheme.
✦ A list is a series of expressions enclosed in parentheses.

- Lists represent both functions and data.
- The empty list is written ().
List Examples

(define odds '(1 3 5 7 9)) ; define a list

(cons 2 (cons 4 (cons 6 ()()))) ; construct a list (2 4 6)
; second arguments should be a list

(car odds) ; evaluates to 1, the head of the list

(cdr odds) ; evaluates to (3 5 7 9), the tail of the list

(cddr oddr) ; evaluates to (5 7 9), the tail of the tail

(cadr oddr) ; evaluates to 3, the head of the tail

(list 1 2 3 4) ; evaluates to (1 2 3 4)
; construct a list consisting of N arguments

(list '(1 2 3) '(4 5) 6) ; evaluates to ((1 2 3) (4 5) 6)

(append '(1 2 3) '(4 5)) ; evaluates to (1 2 3 4 5 6)
; concatenates the second list onto the end of
; first list, both arguments must be list

(null? '()) ; evaluates to #t, test for an empty list

(equal? '(1 2 3) '(1 (2 3))) ; evaluates to #f, test if two objects have the
; same content and structure