**Scheme Quick Reference**

### Lists:
- `@` operator: `E.g. (cons '(a) '(b c d)) result: (a b c d)`
- `::` operator: `E.g. (list a b c d)) result: (a b c d)`
- `hd` operator: `E.g. (hd '(1 2 3)) result: 1`
- `tl` operator: `E.g. (tl (1 2 3)) result: (2 3)`

### Conditional & Selection Statements:
- `let` (let ( (<var1> <exp1>)
              ...) )
- `cond` ( ( <p1> <e1> )
            ( <p2> < e2>)
            ( else <en>) )
- `if` (<predicate> <consequent> <alternative>)

### Functional Abstraction
- `(lambda <expr-list> <body>)`

---

**ML Quick Reference**

### Lists:
- `[obj1, obj2,...]`
  - `@` operator: `E.g. [1.2]@[3] result: [1,2,3]`
  - `::` operator: `E.g. 1::[2] result: [1,2]`
  - `hd` operator: `E.g. hd[1,2,3] result: 1`
  - `tl` operator: `E.g. tl[1,2,3] result: [2,3]`

### Tuples:
- `(obj1, obj2,...)`
  - `#` operator: `E.g. #2(6,7,"abc") result: 7`
  - `=,<>` operators: `E.g. (3, "a", true) = (3, "a", (3>2)) result: true`

### Functions:
- `fun <func-name> <input-param> = <expression>;;`
- `fn <func-param> => <func-body>;;`
- `fun <func-name> <pattern1> = <expression1>
  |   <func-name> <pattern2> = <expression2>
  ...
  |   <func-name> <patternn> = <expressionn>;;`
  - `(fn x => <body>) lambda expression`

### Conditional & Selection Statements:
- `let` ( (<var1> <exp1>)
          ...
          ( <varn> <expn> ) )
  <body >
- `cond` ( ( <p1>  <e1>)
          ( <p2>  <e2>)
          ( else  <en>))
- `if` (<predicate> <consequent> <alternative>)

### Prolog Quick Reference

#### Horn clause:
- `<head> :- <body>`

#### Cut Operator:
- `!` operator

#### Lists:
- `[]` Empty list
- `|` operator: `E.g. [H|T]=[a,b,c] means H=a, T=[b,c]`

#### Function terms:
- `functor(parameter1,parameter2,...)`

#### Boolean Predicates:
- `X = Y` Succeeds if X and Y can be unified.
- `X # Y` Succeeds if X and Y cannot be unified.
- `X == Y` Succeeds if X and Y are already instantiated to the same object.
- `X #= Y` Succeeds if X and Y are not already instantiated to the same object.
- `X =:= Y` Succeeds if X and Y are identical after evaluating both terms.
- `X is Expr` Evaluate Expr and unify with X.

#### Logical Operators:
- `/\` not provable
- `,` logical conjunction (AND)
- `;` logical disjunction (OR)