`if`-Expressions

`(if` *x y z*`)`

is evaluated by

`if`-Expressions

`(if` ***x*** *y z*`)`

is evaluated by

- First evaluating expression *x*

`if`-Expressions

`(if` ***x*** ***y*** *z*`)`

is evaluated by

- First evaluating expression *x*
- If the value of *x* is non-`#f`, then evaluate expression *y* and return the value of *y* as the value of the `if`-expression

`if`-Expressions

`(if` ***x*** *y* ***z*** `)`

is evaluated by

- First evaluating expression *x*
- If the value of *x* is non-`#f`, then evaluate expression *y* and return the value of *y* as the value of the `if`-expression
- However, if the value of *x* is `#f`, then evaluate expression *z* and return the value of *z* as the value of the `if`-expression

`cond`-Expressions

(cond
  (x_1 y_1)
  (x_2 y_2)
  ...
  (x_{n-1} y_{n-1})
  (else y_n))

is evaluated by

`cond`-Expressions

(cond
  (**x_1** y_1)
  (x_2 y_2)
  ...
  (x_{n-1} y_{n-1})
  (else y_n))

is evaluated by

- First evaluating x_1

`cond`-Expressions

(cond
  (x_1 y_1)
  (x_2 y_2)
  ...
  (x_{n-1} y_{n-1})
  (else y_n))

is evaluated by

- First evaluating x_1
- If the value of x_1 is non-`#f`, then evaluate y_1 and return the value of y_1 as the value of the `cond`-expression

`cond`-Expressions

(cond
  (x_1 y_1)
  ($\mathbf{x_2}$ y_2)
  ...
  (x_{n-1} y_{n-1})
  (else y_n))

- Otherwise, then evaluate x_2

`cond`-Expressions

```
(cond
   (x1  y1)
   (x2  y2)
   ...
   (xn-1  yn-1)
   (else  yn))
```

- Otherwise, then evaluate x_2
- If the value of x_2 is non-`#f`, then evaluate y_2 and return the value of y_2 as the value of the `cond`-expression

`cond`-Expressions

```
(cond
   (x1  y1)
   (x2  y2)
   ...
   (xn-1  yn-1)
   (else yn))
```

- Otherwise, keep evaluating x_j's in order, until you find the first one whose value is non-`#f`

`cond`-Expressions

```
(cond
   (x1  y1)
   (x2  y2)
   ...
   (xn-1  yn-1)
   (else yn))
```

- Otherwise, keep evaluating x_j's in order, until you find the first one whose value is non-`#f`
- Then evaluate the associated y_j and return that as the value of the `cond`-expression

`cond`-Expressions

```
(cond
   (x1  y1)
   (x2  y2)
   ...
   (xn-1  yn-1)
   (else yn))
```

- If <u>all</u> of the x_j's have value `#f`, then evaluate y_n and return that as the value of the `cond`-expression

`and`-Expressions

- The value of the expression

  (`and` x_1 x_2 ... x_n)

  is determined by evaluating the x_j's in order
- If each of the x_j's has a non-`#f` value, return the value of x_n as the value of the `and`-expression
- Otherwise, return `#f` as the value of the `and`-expression
- In the same spirit as the `&&` operator in Java and C++ and the `and` operator in Python

`or`-Expressions

- The value of the expression

  (`or` x_1 x_2 ... x_n)

  is determined by evaluating the x_j's in order
- If each of the x_j's has a `#f` value, return the value `#f` as the value of the entire `or`-expression
- Otherwise, return the value of the <u>first</u> non-`#f` x_j as the value of the `or`-expression
- In the same spirit as the `||` operator in Java and C++ and the `or` operator in Python

Typical Procedure Definition

(define *name*
  (lambda (*formal*$_1$ ... *formal*$_n$)
    (if
      test
      consequent
      alternate)))

Typical Procedure Definition

```
(define name
  (lambda (formal_1 ... formal_n)
    (cond
      (test_1  consequent_1)
      (test_2  consequent_2)
      ...
      (test_m-1  consequent_m-1)
      (else alternate_m))))
```