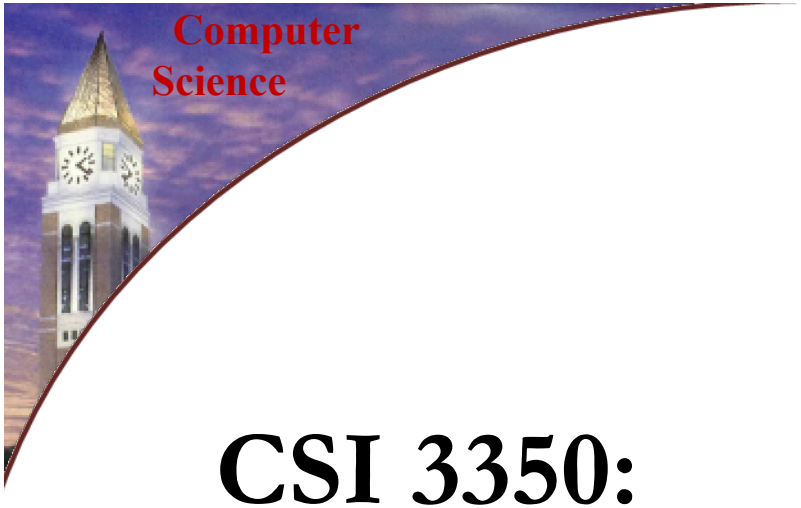


CSI 3350: PROGRAMMING LANGUAGES

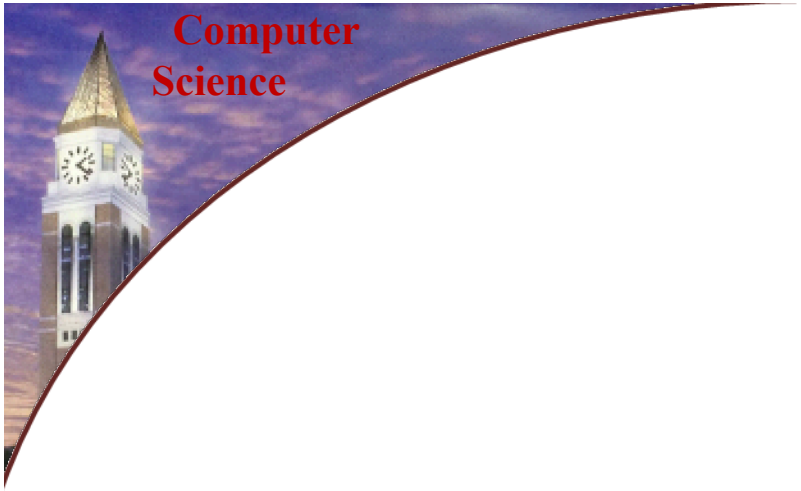
Department of Computer Science & Engineering

Oakland University

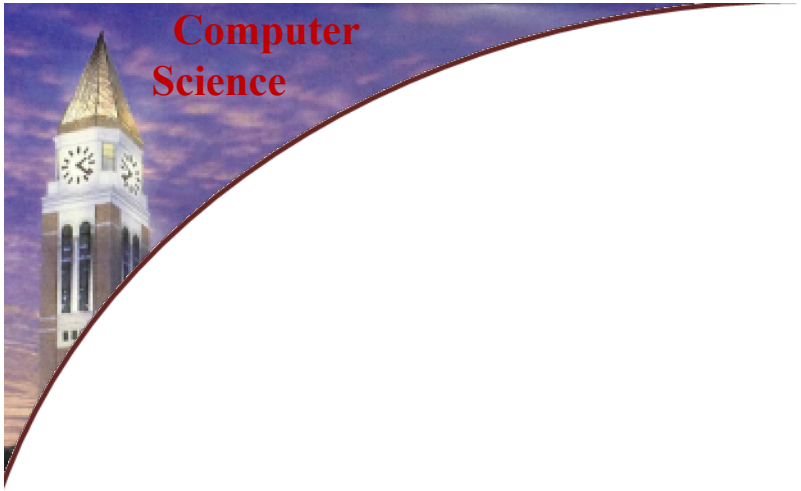
Making Use of Number Types

Factorial

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

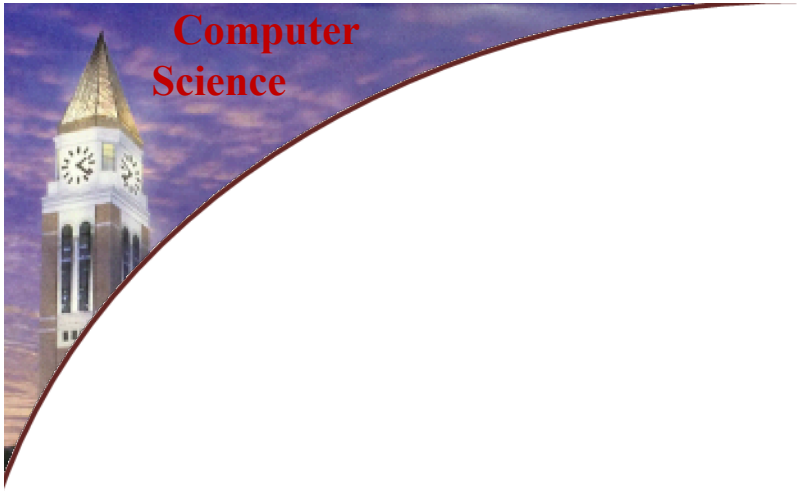



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```



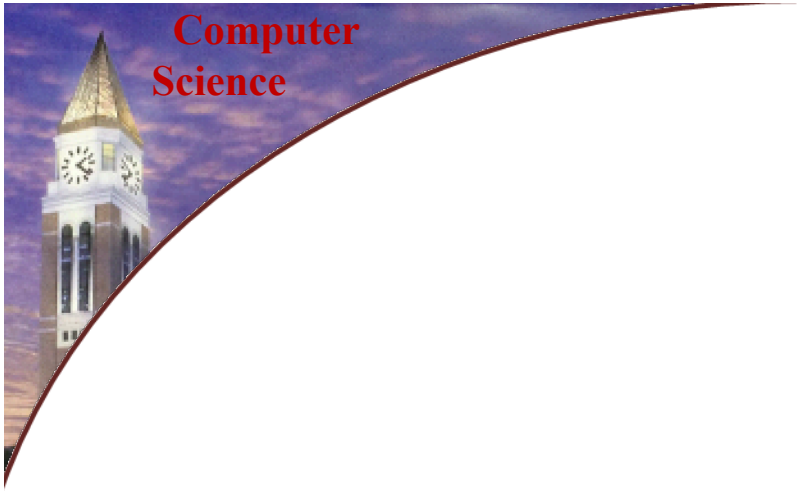

```
(fact 4)
```

```
(define      n = 4
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```




```
(fact 4)
```

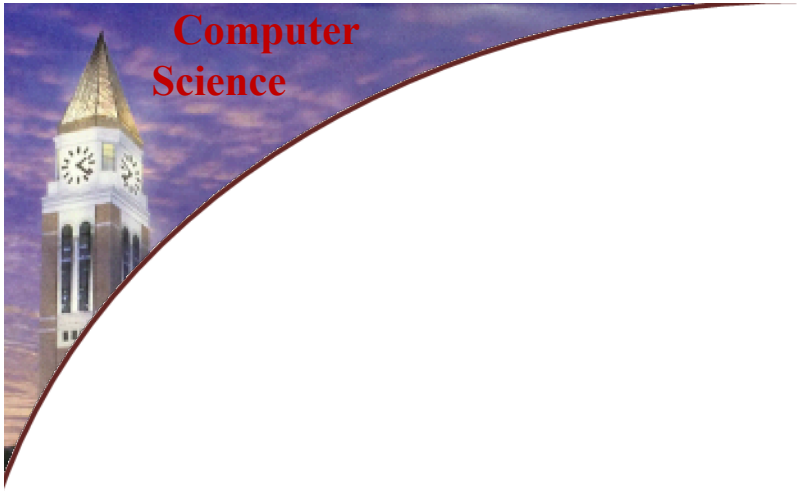
```
(define      n = 4
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```




```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 4
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4 n - 1 = 3

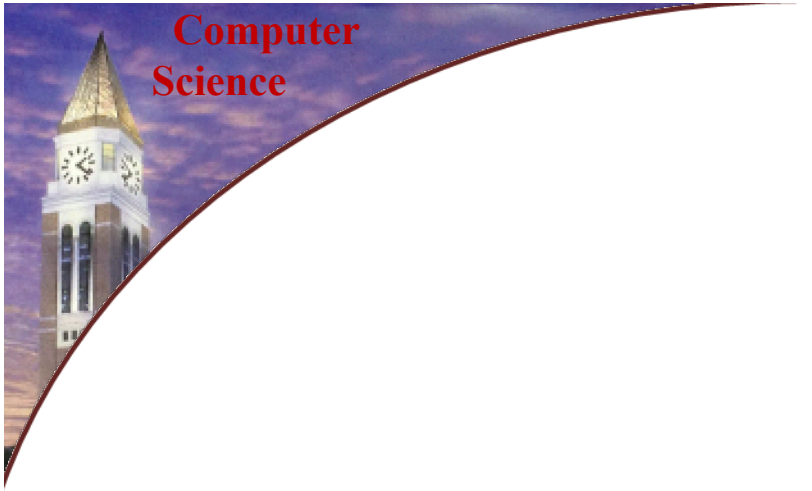



```
(fact 4)
= (* 4 (fact 3))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4

n - 1 = 3

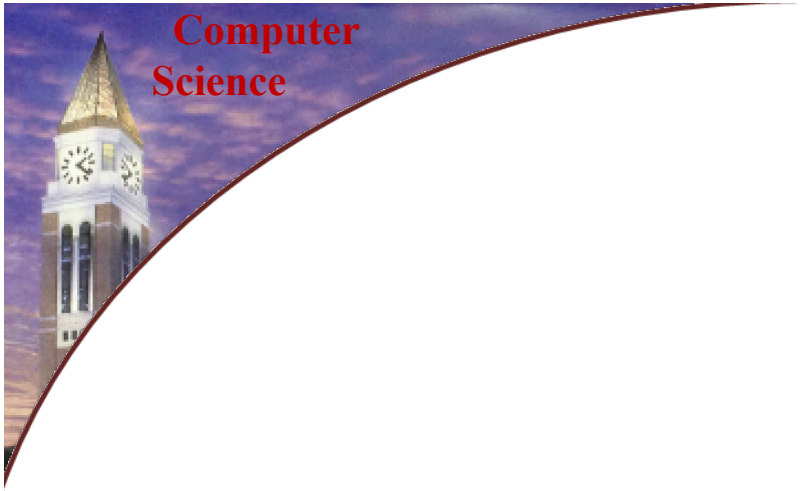



```
(fact 4)
= (* 4 (fact 3))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4

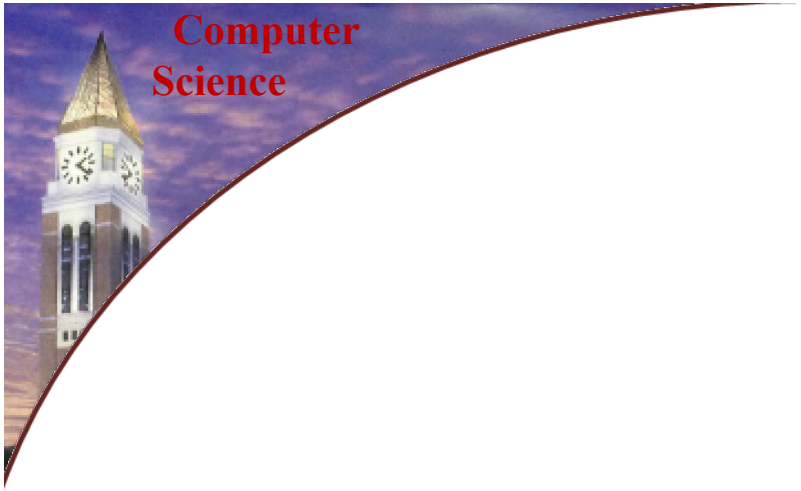
n - 1 = 3




```
(fact 4)
= (* 4 (fact 3))
```

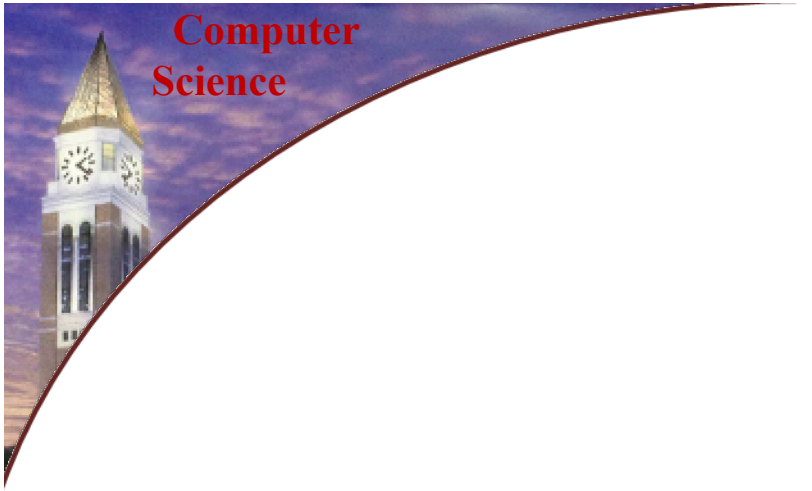
n = 3

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```



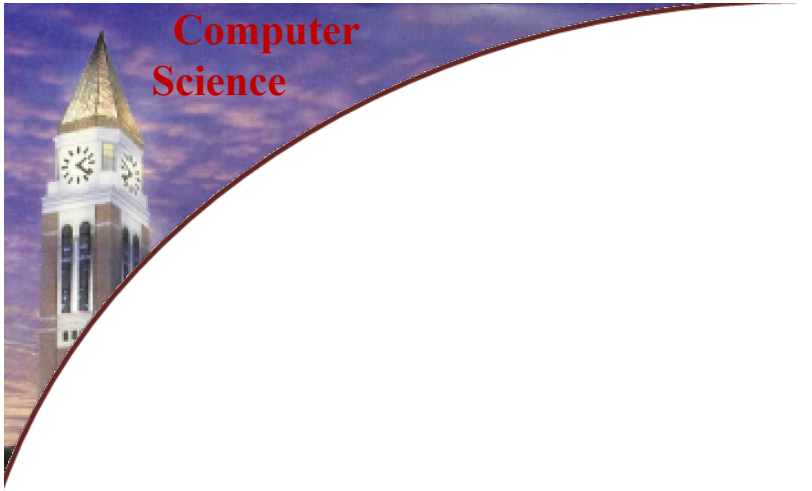

```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```




```
(fact 4)
= (* 4 (fact 3))
```

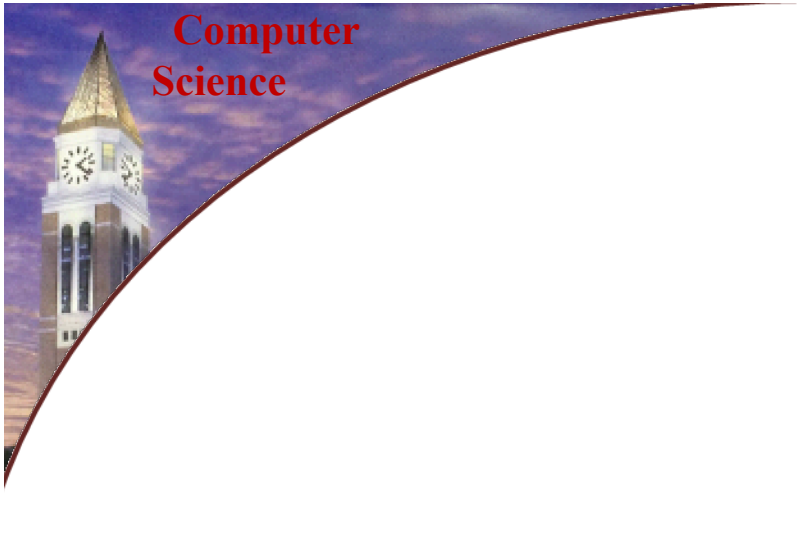
```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```




```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3 n - 1 = 2

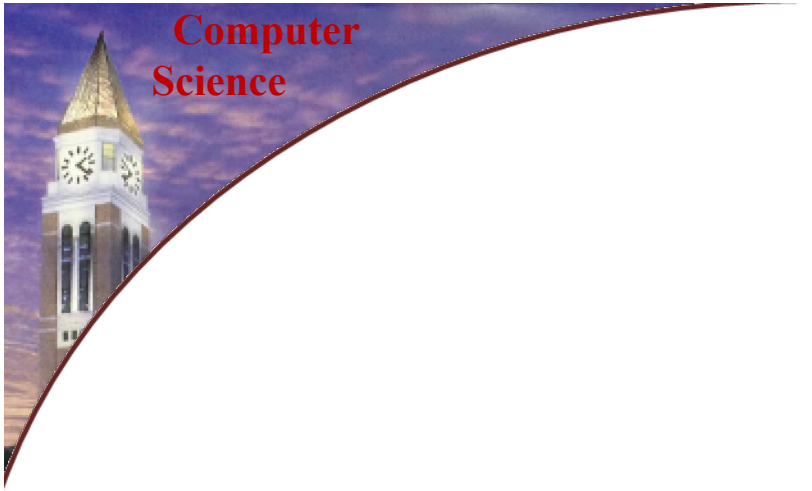



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

n = 3 **n - 1 = 2**

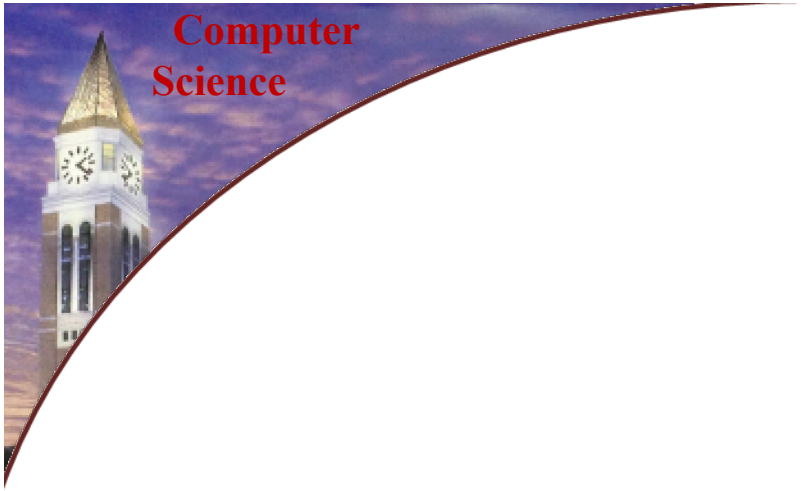
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```




```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3 n - 1 = 2

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

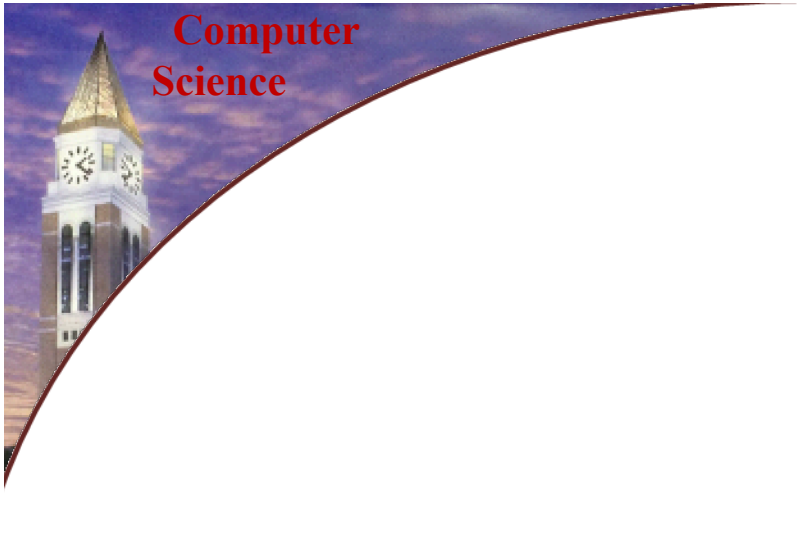



```
(fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

n - 1 = 2

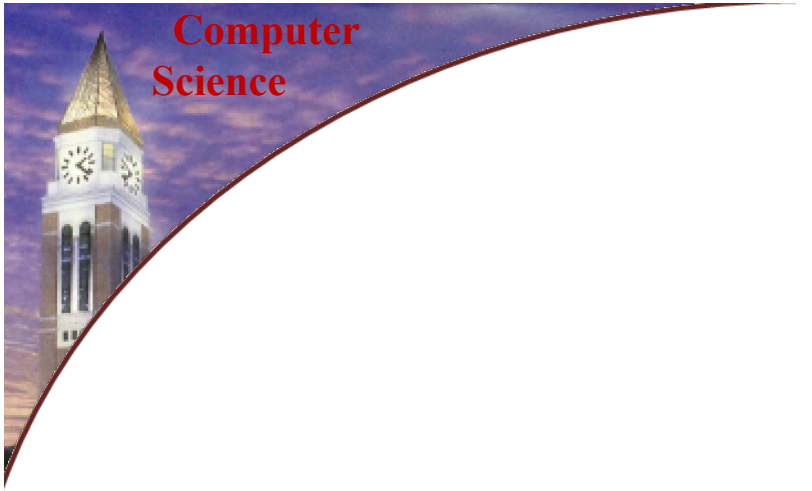



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

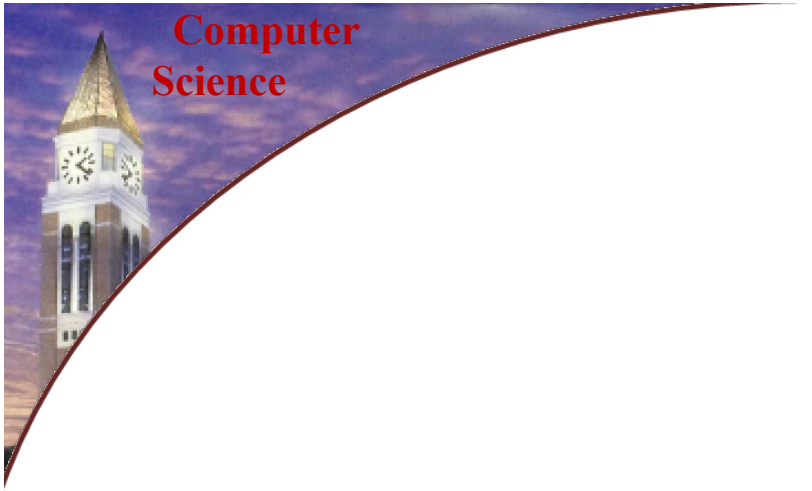
n - 1 = 2

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



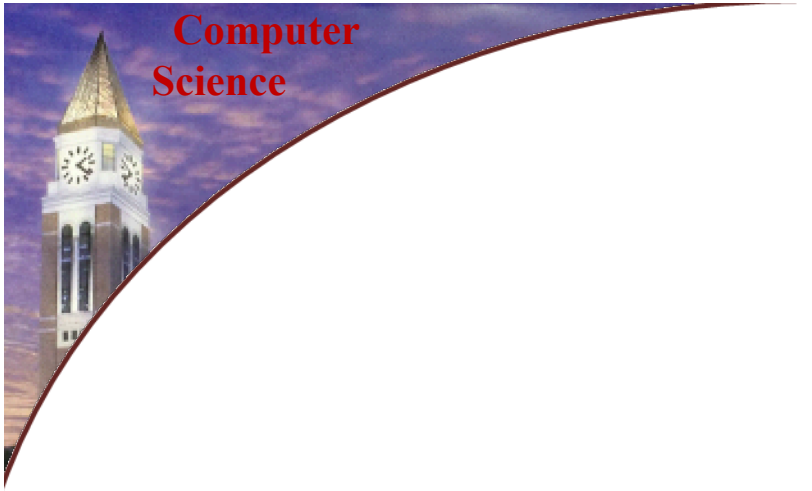

```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



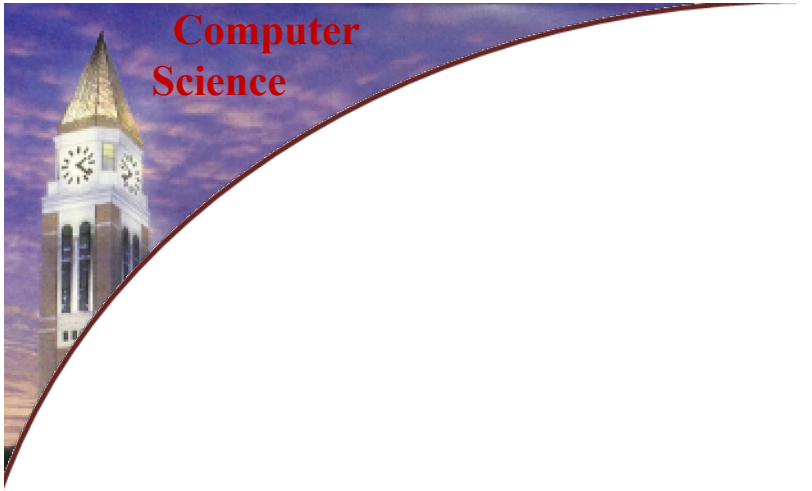

```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```




```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

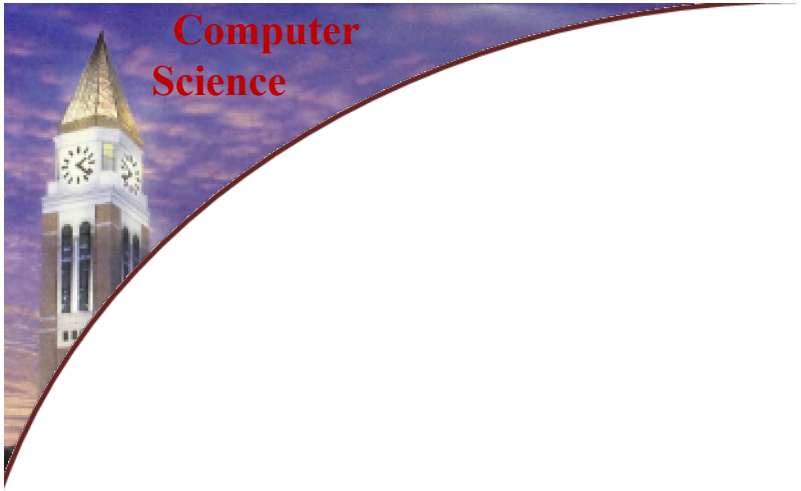
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```




```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2 n - 1 = 1

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

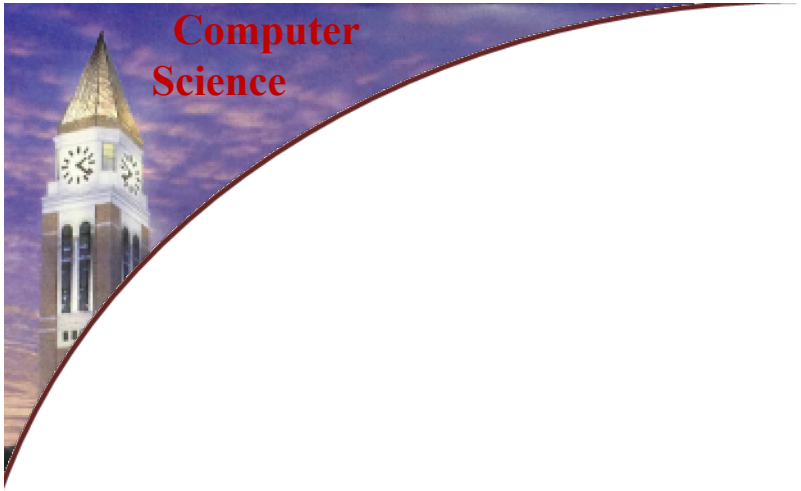



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n = 2 **n - 1 = 1**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

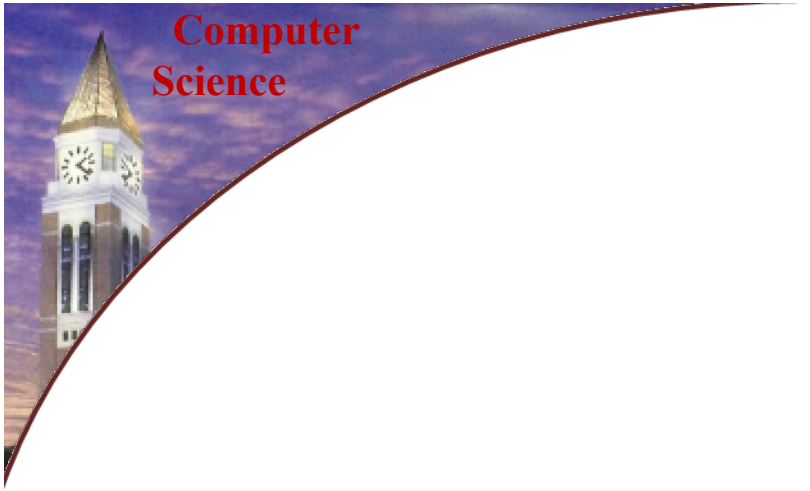



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n - 1 = 1

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

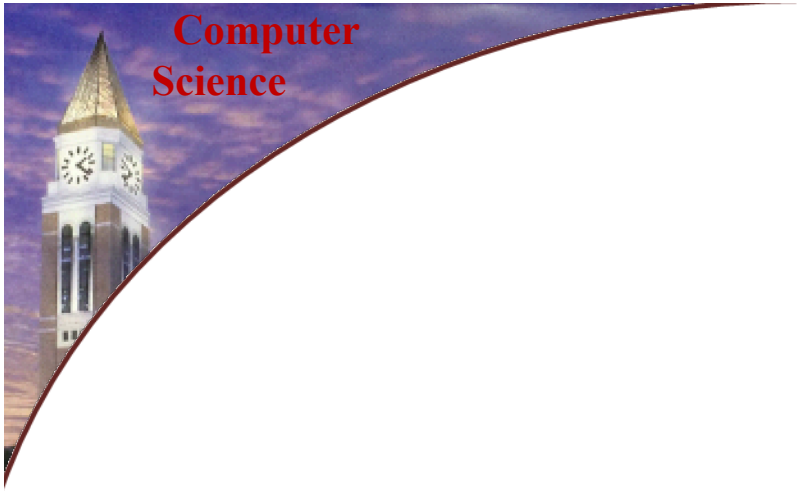



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n - 1 = 1

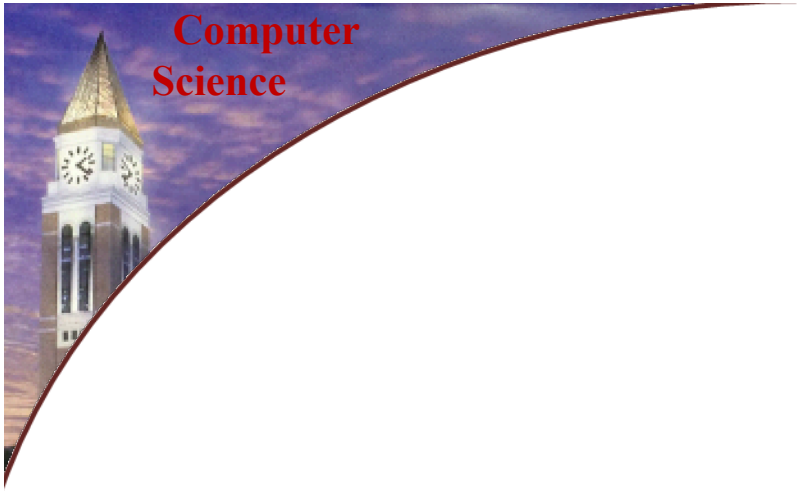
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

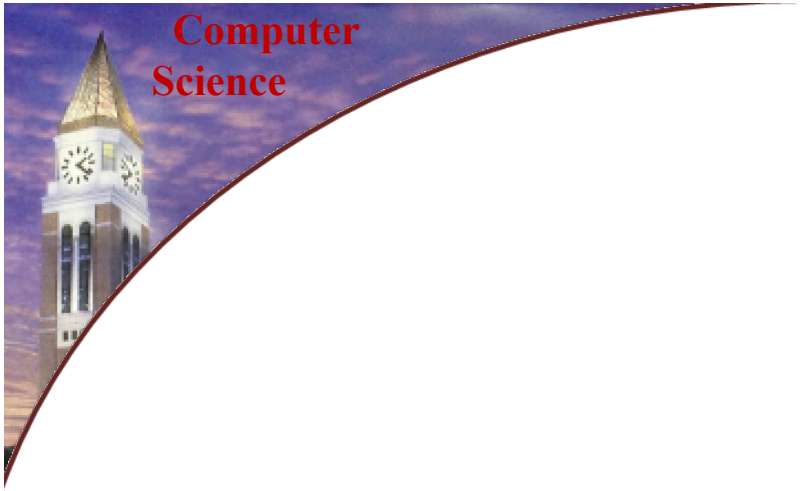
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

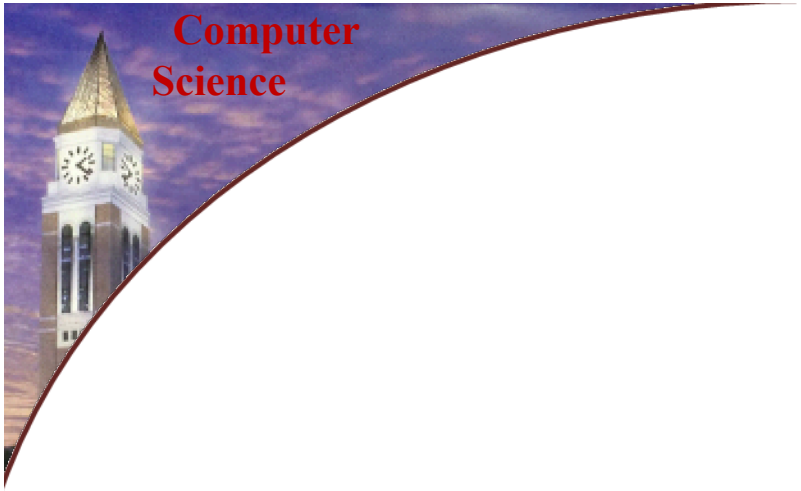



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n = 1 **n - 1 = 0**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n = 1 n - 1 = 0

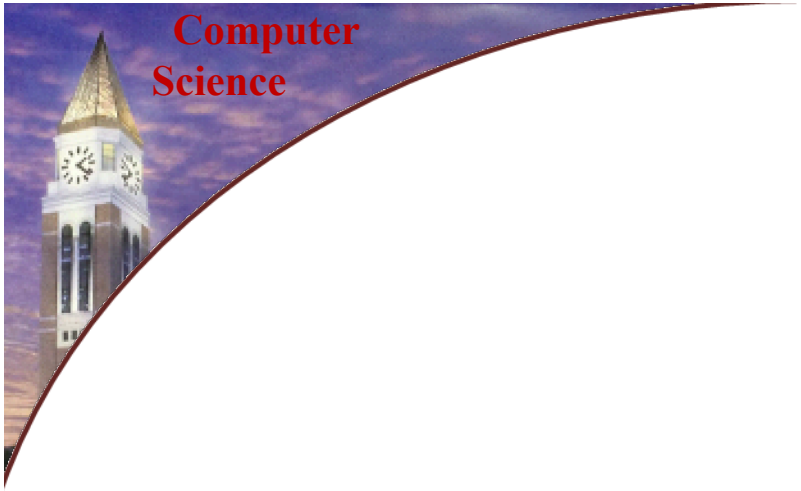
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n - 1 = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n - 1 = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define
    (fact n )
      (if
        (= n 0)
        1
        (* n (fact (- n 1)))
      )
)
```

n = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define      n = 0
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

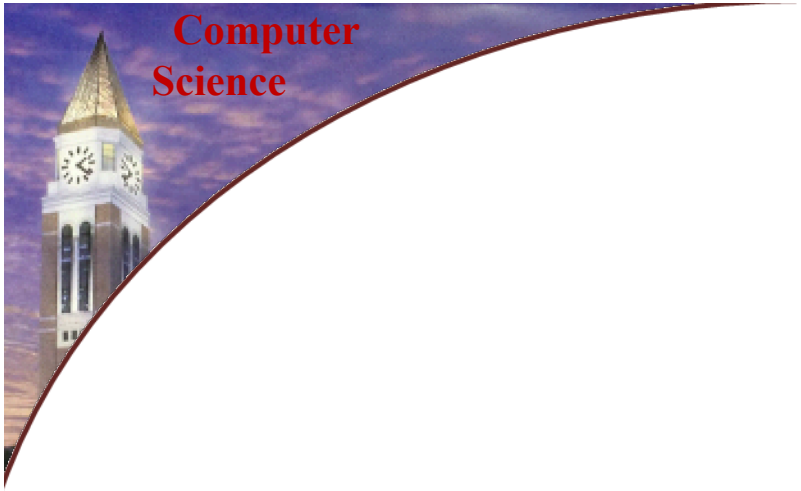
n = 0

base case is hit !

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

```
(define
    (fact n )
      (if
         (= n 0)
         1
         (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
```

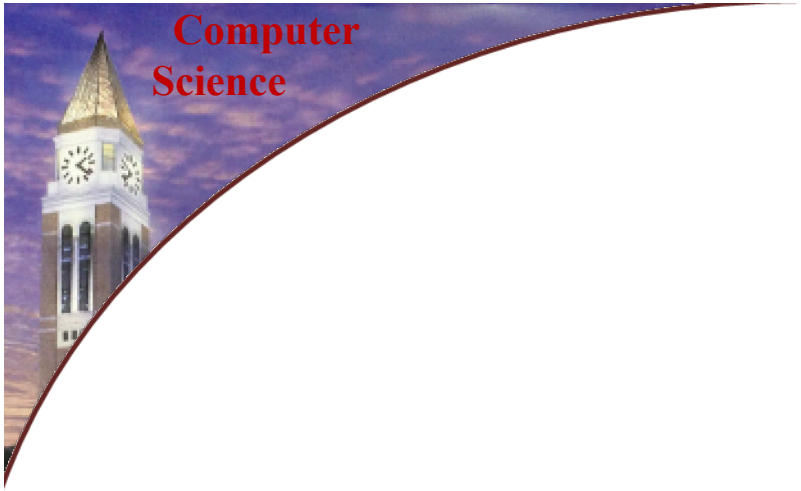



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

Is it a good solution?

Is there another solution?

```
(define
    (fact-new n )
    . . .
)
```

How many parameters do you need?

```
(define
    (fact-new n )
     (fact-tail n 1)

)
```

Use the parameter to store the partial result

```
(define
    (fact-new n )
    . . .
)
```

How many parameters do you need?

```
(define
    (fact-tail n prod)
      (if
         (= n 0)
         prod
         (fact-tail (- n 1) (* n prod) )
      )
)
```

```
(define
    (fact-new n )
    . . .
)
```

How many parameters do you need?

```
(define
    (fact-tail n prod)
      (if
         (= n 0)
         prod
         (fact-tail (- n 1) (* n prod) )
      )
)
```

2nd parameter stores the intermediate result!

```
(define
    (fact-new n )
    . . .
)
```

How many parameters do you need?

```
(define
    (fact-tail n prod)
      (if
         (= n 0)
         prod
         (fact-tail (- n 1) (* n prod) )
      )
)
```

2nd parameter stores the intermediate result!

```
(define
    (fact-new n )
    . . .
)
```

How many parameters do you need?

```
(define
    (fact-tail n prod)
      (if
         (= n 0)
         prod
         (fact-tail (- n 1) (* n prod) )
      )
)
```

2nd parameter stores the intermediate result!

```
(define
    (fact-tail n prod)
      (if
         (= n 0)
         prod
         (fact-tail (- n 1) (* n prod) )
      )
)
```

```
  (fact-new 4)
=(fact-tail 4 1)
=(fact-tail 3 4)
=(fact-tail 2 12)
=(fact-tail 1 24)
=(fact-tail 0 24)
= 24
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact-new n )
    (fact-tail n 1)
)


(define
    (fact-tail n prod)
      (if
          (= n 0)
          prod
          (fact-tail (- n 1) (* n prod) )
      )
)
```

tail recursion!

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1))))
)
```

Reversal of A List

- `(define (list-reversal lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(3 2 1)
'( 1 2 3 4)   => '(4 3 2 1)
```

Tail recursion?

Pairwise Reversal of A List

- `(define (pairwise-reversal lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(2 1 3)
'( 1 2 3 4)   => '(2 1 4 3)
```

Tail recursion?