Parsing

On slides,

```
(+ 1 2)
```

means

```
(make-plus 1 2)
```

Parsing

On slides,

$$\boxed{\texttt{x}}$$

means

```
'x
```

Parsing

On slides,

```
(lambda (x) (+ 1 x))
```

means

```
(make-lambda 'x (make-plus 1 'x))
```

Parsing

On slides,

```
((lambda (g) (g 10))
 (lambda (x) (+ 1 x)))
```

means

```
(make-app
 (make-lambda 'g (make-app 'g 10))
 (make-lambda 'x (make-plus 1 'x)))
```

Cost of Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
            1)                                   )
```

Cost of Substitution

```
(evaluate ((lambda (x)
              ((lambda (y)
                 (+ 100 (+ 99 (+ 98 ... (+ y x)))))
               2))
            1))
```

⇒

```
(evaluate ((lambda (y)
              (+ 100 (+ 99 (+ 98 ... (+ y 1)))))
            2))
```

Cost of Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          )
```

⇒

```
(evaluate ((lambda (y)
             (+ 100 (+ 99 (+ 98 ... (+ y 1)))))
           2)
          )
```

⇒

```
(evaluate (+ 100 (+ 99 (+ 98 ... (+ 2 1)))) )
```

Cost of Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1))
```

⇒

```
(evaluate ((lambda (y)
             (+ 100 (+ 99 (+ 98 ... (+ y 1)))))
           2))
```

⇒

```
(evaluate (+ 100 (+ 99 (+ 98 ... (+ 2 1))))))
```

With **n** variables, evaluation will take O(**n**^{2}) time!

Deferring Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)                                          )
```

Deferring Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1))
```

⇒

```
                                         x = 1
(evaluate ((lambda (y)
             (+ 100 (+ 99 (+ 98 ... (+ y x)))))
           2))
```

Deferring Substitution

```
(evaluate ((lambda (x)
              ((lambda (y)
                  (+ 100 (+ 99 (+ 98 ... (+ y x)))))
               2))
            1)                                          )
```

⇒

x = 1

```
(evaluate ((lambda (y)
              (+ 100 (+ 99 (+ 98 ... (+ y x)))))
            2)                                   )
```

⇒

y = 2   x = 1

```
(evaluate (+ 100 (+ 99 (+ 98 ... (+ y x)))))
```

Deferring Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
            1))
```

⇒

```
                                          x = 1
(evaluate ((lambda (y)
             (+ 100 (+ 99 (+ 98 ... (+ y x)))))
            2))
```

⇒

```
                                    y = 2    x = 1
(evaluate (+ 100 (+ 99 (+ 98 ... (+ y x)))))
```

⇒ . . . ⇒

```
             y = 2    x = 1
(evaluate y)
```

Deferring Substitution with the Same Identifier

```
(evaluate  ((lambda (x)
              ((lambda (x)
                 x)
               2))
            1)                )
```

Deferring Substitution with the Same Identifier

```
(evaluate ((lambda (x)
              ((lambda (x)
                  x)
               2))
           1)          )
```

⇒

```
                                  x = 1
(evaluate ((lambda (x)
              x)
           2)          )
```

Deferring Substitution with the Same Identifier

```
(evaluate ((lambda (x)
              ((lambda (x)
                  x)
                2))
            1)          )
```

⇒

```
                                   x = 1
(evaluate ((lambda (x)
              x)
            2)          )
```

⇒

```
            x = 2    x = 1
(evaluate x )
```

Deferring Substitution with the Same Identifier

```
(evaluate ((lambda (x)
              ((lambda (x)
                  x)
                2))
            1)          )
```

⇒

```
                                x = 1
(evaluate ((lambda (x)
              x)
            2)      )
```

⇒

```
           x = 2    x = 1
(evaluate  x )
```

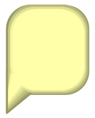
Always add to start, then always check from start

Environment

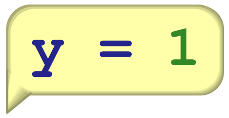
```
; An env is either
;   - empty
;   - (make-sub sym val env)
(define-struct sub (id val))
```

Environment

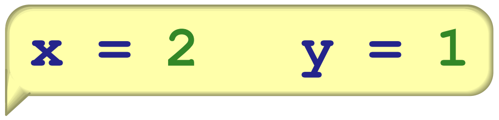
```
; An env is either
;   - empty
;   - (make-sub sym val env)
(define-struct sub (id val))
```



= `empty`



= `(make-sub 'y 1 empty)`



= `(make-sub 'x 2 (make-sub 'y 1 empty))`

Evaluation with an Environment

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          empty)
```

Evaluation with an Environment

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          empty)

⇒ (evaluate ((lambda (y)
               (+ 100 (+ 99 (+ 98 ... (+ y x)))))
             2)
            (make-sub 'x 1 empty))
```

Evaluation with an Environment

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          empty)

⇒ (evaluate ((lambda (y)
               (+ 100 (+ 99 (+ 98 ... (+ y x)))))
             2)
            (make-sub 'x 1 empty))

⇒ (evaluate (+ 100 (+ 99 (+ 98 ... (+ y x))))
            (make-sub 'y 2 (make-sub 'x 1 empty)))
```

Evaluation with an Environment

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          empty)

⇒ (evaluate ((lambda (y)
               (+ 100 (+ 99 (+ 98 ... (+ y x)))))
             2)
            (make-sub 'x 1 empty))

⇒ (evaluate (+ 100 (+ 99 (+ 98 ... (+ y x))))
            (make-sub 'y 2 (make-sub 'x 1 empty)))

⇒ ...

⇒ (evaluate y (aSub 'y 2 (aSub 'x 1 (mtSub))))
```

Functions and Environments

```
(evaluate ((lambda (y) (lambda (x) (+ y x)))
           10)                                )
```

Functions and Environments

```
(evaluate ((lambda (y) (lambda (x) (+ y x)))
            10))
```

⇒

```
                                y = 10
(evaluate (lambda (x) (+ y x)))
```

Function Calls with Environments

```
(evaluate (((lambda (y) (lambda (x) (+ y x))) 10)
           ((lambda (y) y) 7))                   )
```

Function Calls with Environments

```
(evaluate (((lambda (y) (lambda (x) (+ y x))) 10)
           ((lambda (y) y) 7)))
```

Argument expression:

```
(evaluate ((lambda (y) y) 7))
```

⇒

```
           y = 7
(evaluate y) ⇒ 7
```

Function Calls with Environments

```
(evaluate (((lambda (y) (lambda (x) (+ y x))) 10)
           ((lambda (y) y) 7)))
```

Argument expression:

```
(evaluate ((lambda (y) y) 7))
```

⇒

```
           y = 7
(evaluate y) ⇒ 7
```

Function expression:

```
(evaluate ((lambda (y) (lambda (x) (+ y x))) 10))
```

⇒

```
                             y = 10
(evaluate (lambda (x) (+ y x))) ⇒ ?
```

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))
```

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))


(test (evaluate ((lambda (y) (lambda (x) (+ y x))) 10)
                empty)
      ...)
```

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))
```

```
(test (evaluate ((lambda (y) (lambda (x) (+ y x))) 10)
                empty)
      (make-function ... ... ...))
```

$\boxed{\texttt{((lambda (y) (lambda (x) (+ y x))) 10)}}$

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))
```

```
(test (evaluate ((lambda (y) (lambda (x) (+ y x))) 10)
                empty)
      (make-function 'x (+ y x)
                     ...))
```

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))
```

```
(test (evaluate ((lambda (y) (lambda (x) (+ y x))) 10)
                empty)
      (make-function 'x (+ y x)
                        (make-sub 'y 10 empty)))
```

Continuing Evaluation

`y = 10`

Function:
```
(lambda (x) (+ y x))
```

Argument: 7

Continuing Evaluation

`y = 10`

Function: `(lambda (x) (+ y x))`

Argument: 7

To apply, interpret the function body with the given argument:

```
(evaluate ...)
```

Continuing Evaluation

`y = 10`

Function: $\boxed{\texttt{(lambda (x) (+ y x))}}$

Argument: `7`

To apply, interpret the function body with the given argument:

`. . .`

`(evaluate` $\boxed{\texttt{(+ y x)}}$`)`

Continuing Evaluation

`y = 10`

Function: `(lambda (x) (+ y x))`

Argument: `7`

To apply, interpret the function body with the given argument:

`x = 7    y = 10`

```
(evaluate (+ y x))
```

Continuing Evaluation

y = 10

Function: $\boxed{\texttt{(lambda (x) (+ y x))}}$

Argument: 7

To apply, interpret the function body with the given argument:

```
(evaluate (+ y x)
          (make-sub 'x 7
                    (make-sub 'y 10 empty)))
```