Higher-Order Programming

- A function is higher-order if its arguments or result is a function (otherwise, it is first-order).
- Higher-order functions permit greater reuse of programs.
- ie, instead of writing many specialized functions, we can write a single, generalized function.
- (Of course, we can already do this to some extent, but higher-order functions take the idea much farther.)

Example

Here are two specialized but similar First-order functions:

```
(define (remove-even L)
  (cond ((null? L) '())
        ((even? (car L)) (remove-even (cdr L)))
        (else (cons (car L) (remove-even (cdr L))))))
```

ie, Remove even elements from list L.

```
(define (remove-odd L)
  (cond ((null? L) '())
        ((odd? (car L)) (remove-odd (cdr L)))
        (else (cons (car L) (remove-odd (cdr L))))))
```

ie, Remove odd elements from list L.

Here is a more-general, higher-order function, but with very similar structure:

```
(define (remove-if P L)
  (cond ((null? L) '())
        ((P (car L)) (remove-if P (cdr L)))
        (else (cons (car L) (remove-if P (cdr L))))))
```

ie, Remove From L any element satisfying predicate P.

---

eg. (remove-if even? '(1 2 3)) ⇒ (1 3)

(remove-if odd? '(1 2 3)) ⇒ (2)

(remove-if number? '(1 a 2 b 3))
⇒ (a b)

Another Example : Mapping Functions

Here are two specialized but similar First-order Functions :

```
(define (square-all L)
  (cond ((null? L) '())
        (else (cons (square (car L))
                    (square-all (cdr L))))))
```

ie, Square each element of list L.

```
(define (add1-all L)
  (cond ((null? L) '())
        (else (cons (add1 (car L))
                    (add1-all (cdr L))))))
```

ie, Add 1 to each element of list L.

Map is a standard, higher-order function that is more general.

```
(define (map F L)
  (cond ((null? L) '())
        (else (cons (F (car L))
                    (map F (cdr L))))))
```

ie, Apply function F to every element of list L.

---

eg. (map square '(1 2 3)) ⟹ (1 4 9)

(map add1 '(1 2 3)) ⟹ (2 3 4)

(map zero? '(0 1 2)) ⟹ (#t #f #f)

(map car '((a b) (c d) (e f)))

⟹ (a c e)

Anonymous (Unnamed) Functions

- When using functions as arguments, it is often inconvenient to have to define them and give them a name.

---

eg. - (define (add1 N) (+ 1 N))

(map add1 '(1 2 3)) ⟹ (2 3 4)

- (define (add2 N) (+ 2 N))

(map add2 '(1 2 3)) ⟹ (3 4 5)

- (define (sq-add6 N) (square (+ 6 N)))

(map sq-add6 '(1 2 3)) ⟹ (49 64 81)

Solution: Lambda Expressions

eg. `(lambda (X Y) (square (+ X Y)))`

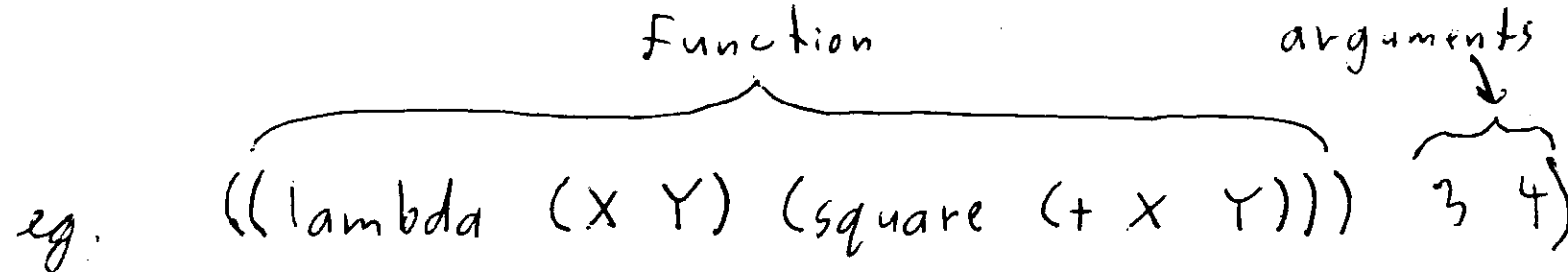
This lambda expression is a <u>function</u> with two arguments, X and Y. The function returns $(X+Y)^2$.

---

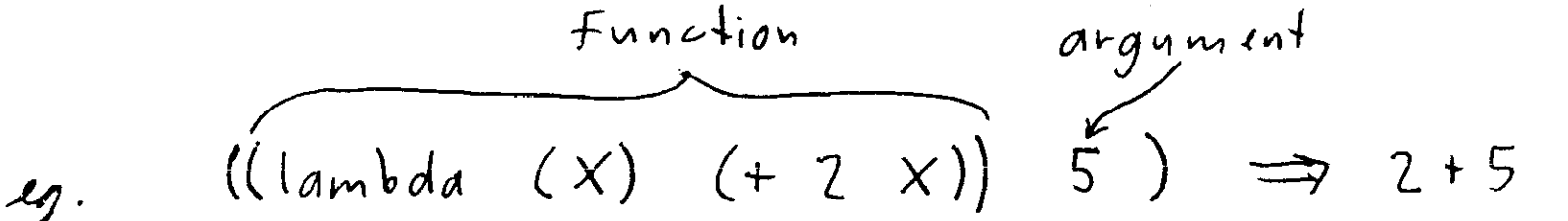
eg. `(lambda (X) (+ 1 X))`

This lambda expression is a function with one argument, X. The function returns $1+X$.

Lambda expressions can be used anywhere that named functions can be used.

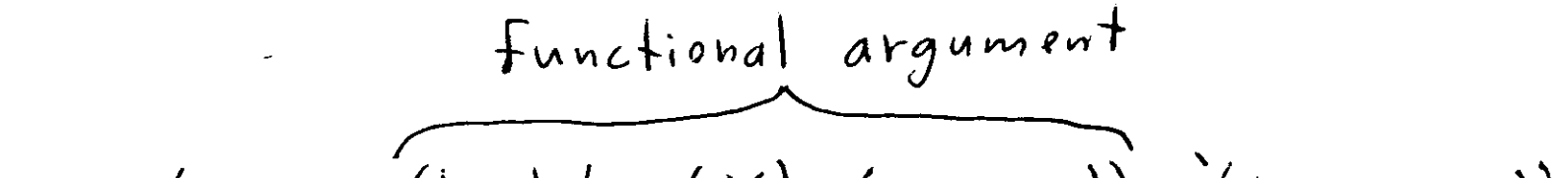


$\Rightarrow (3+4)^2 \Rightarrow 49$



$\Rightarrow 2+5$

$\Rightarrow 7$



$\Rightarrow$ (2+1 2+2 2+3 2+4)

$\Rightarrow$ (3 4 5 6)

Lambda expressions are often used as arguments to higher-order Functions.

---

eg.

```
(map (lambda (x) (+ 1 x)) '(1 2 3 4))
```

$\Rightarrow$ (2 3 4 5)

```
(map (lambda (x) (+ 3 x)) '(1 2 3 4))
```

$\Rightarrow$ (4 5 6 7)

```
(map (lambda (x) (* 2 x)) '(1 2 3 4))
```

$\Rightarrow$ (2 4 6 8)

```
(map (lambda (x) (* 2 (+ 3 x))) '(1 2 3 4))
```

$\Rightarrow$ (8 10 12 14)

Functions as Values

In Scheme, a function can return another function as its value.

Simple Example:

```
(define (F X) (if (< X 0) car cdr))
```

Note: F always returns car or cdr, i.e., a function.

eg. (F -1) ⇒ the function car

(F 1) ⇒ the function cdr

((F -1) '(a b c)) ⇒ a

((F 1) '(a b c)) ⇒ (b c)

Using lambda expressions, a function can build a new function, and return it.

eg. (define (compose F G)
(lambda (X) (F (G X))))

---

eg. (compose car cdr) ⇒

the function (lambda (X) (car (cdr X)))

∴ ((compose car cdr) '(a b c d))

⇒ (car (cdr '(a b c d)))

⇒ b

eg. (compose cdr cdr) ⇒

the function (lambda (x) (cdr (cdr x))

∴ ((compose cdr cdr) '(a b c d))

⇒ (cdr (cdr '(a b c d)))

⇒ (c d)

---

eg. (compose square add1) ⇒

the function (lambda (x) (square (add1 x)))

∴ ((compose square add1) 3) ⇒ 16

(map (compose square add1) '(1 2 3 4))

⇒ (4 9 16 25)

In general, (compose F G) returns the composition of functions F and G.

---

ie. (compose F G) $\Rightarrow$

the function (lambda (x) (F (G x)))

$\therefore$ ((compose F G) Y)

$\Rightarrow$ (((lambda (X) (F (G X))) Y)

Function — argument

$\Rightarrow$ (F (G Y))

Summary

- Higher-order functions can take other functions as arguments and return a function as a value.

- Higher-order functions allow code reuse, since a single, general function can replace many specialized functions.
  (eg. remove-if, map)

- Anonymous (or unnamed) functions can be defined with lambda expressions.

- Lambda expressions can be used to create new functions during execution.

- General form of lambda expressions:

  (lambda (x_1 x_2 ... x_n) Body)