

## Higher-Order Procedures

- Exercises
  - Exercise 0: Preparation
  - Exercise 1: Using map
  - Exercise 2: Dot-Product
  - Exercise 3: Why Apply?
  - Exercise 4: Tallying
  - Exercise 5: Making Talliers

### Exercises

#### Exercise 0: Preparation

- If you have not done so already, please scan the corresponding reading on higher-order procedures.
- Start DrScheme

#### Exercise 1: Using map

- Use map to compute the successors to the squares of the integers between 1 and 10.
- Use map to turn a list into an association list by making each value in the first list into a key (you can also use the same value as the corresponding value).
- Use map to take the last element of each list in a list of lists. The result should be a list of the last elements.
- Use apply and map to sum the last elements of each list in a list of lists of numbers. The result should be a number. You should have written a similar procedure for another lab.

#### Exercise 2: Dot-Product

Use apply and map to concisely define a procedure, `(dot-product list1 list2)`, that takes as arguments two lists of numbers, equal in length, and returns the sum of the products of corresponding elements of the arguments:

```
> (dot-product (list 1 2 4 8) (list 11 5 7 3))
73
; ... because (1 x 11) + (2 x 5) + (4 x 7) + (8 x 3) = 11 + 10 + 28 + 24 = 73

> (dot-product null null)
0
; ... because in this case there are no products to add
```

### Exercise 3: Why Apply?

Sarah and Steven Schemer suggest that “`apply` is irrelevant. After all,” they say, “when you write

```
(apply prog (arg1 ... argn))
```

you’re just doing the same thing as

```
(proc arg1 arg2 ... argn)
```

```
.”
```

Given your experience in the previous exercise, are they correct? Why or why not?

### Exercise 4: Tallying

- Document and write a procedure, (`tally predicate list`), that counts the number of values in *list* for which *predicate* holds.
- Demonstrate the procedure by tallying the number of odd values in the list of the first twenty integers.
- Demonstrate the procedure by tallying the number of multiples of three in the list of the first twenty integers.

### Exercise 5: Making Talliers

Document and write a procedure, (`make-tallier predicate`), that builds a procedure that takes a list as a parameter and tallies the values in the list for which the predicate holds. For example

```
> (define count-odds (make-tallier odd?))  
> (count-odds (list 1 2 3 4 5))  
3
```

You can assume that `tally` already exists for the purpose of this problem.