

Beginning Scheme

Summary: In this laboratory, you will continue to explore the syntax and capabilities of the Scheme programming language.

Note: You may want to keep the corresponding reading at hand.

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Simple Subtraction
 - Exercise 2: Simple Multiplication
 - Exercise 3: Extended Addition
 - Exercise 4: Fewer Arguments
 - Exercise 5: Absolute Value
 - Exercise 6: Exponentiation
 - Exercise 7: Other Notations
 - Exercise 8: Simple Definitions
 - Exercise 9: Square Roots Revisited
 - Exercise 10: Definitions, Revisited
 - Exercise 11: Grading
 - Exercise 12: Wrapup
- Notes
 - Notes on Exercise 7

Exercises

Exercise 0: Preparation

Start DrScheme and make sure that you're in Pretty Big Scheme mode. You may want to review the DrScheme lab to make sure you understand DrScheme.

Exercise 1: Simple Subtraction

- a. Ask DrScheme to subtract 68343 from 81722.
- b. Verify that the answer is correct.

Exercise 2: Simple Multiplication

- a. Ask DrScheme to multiply 162 by 1383.

b. How would you verify that the answer is correct?

Exercise 3: Extended Addition

a. Ask DrScheme to add 3 and 4.

b. Ask DrScheme to add 3 and 4 and then add 5 to the result. You'll need two calls to +.

c. Ask DrScheme to add 3, 4, and 5 using only one call to +.

d. What happens if you call the procedure + with no arguments? With only one? Why do you think it gives these results?

Exercise 4: Fewer Arguments

In the previous exercise, you explored what happens when you call the procedure + with zero and one arguments. Let us explore the same questions for other procedures.

a. What do you expect to happen if you call the procedure * with one argument?

b. Verify your answer experimentally. If the results differ, try to explain the result Scheme gives.

c. What do you expect to happen if you call the procedure * with no arguments?

d. Verify your answer experimentally. If the results differ, try to explain the result Scheme gives.

e. What do you expect to happen if you call the procedure - with one argument?

f. Verify your answer experimentally. If the results differ, try to explain the result Scheme gives.

g. What do you expect to happen if you call the procedure - with no arguments?

h. Verify your answer experimentally. If the results differ, try to explain the result Scheme gives.

Exercise 5: Absolute Value

Have DrScheme compute the absolute value of -197. You can use the abs procedure.

Exercise 6: Exponentiation

a. Ask DrScheme to compute the cube of 19 (that is, the result of raising 19 to the power 3). You can use `expt` to compute exponents.

b. Ask DrScheme to compute the nineteenth power of 3.

c. What do these results indicate about the relationship between procedures and arguments in Scheme?

Exercise 7. Other Notations

As you've learned, Scheme expects you to use parentheses and prefix notation when writing expressions. What happens if you use more traditional mathematical notation? Let's explore that question.

Type each of the following expressions at the Scheme prompt and see what reaction you get.

1. `(2 + 3)`
2. `7 * 9`
3. `sqrt(49)`

You may wish to read the notes on this problem for an explanation of the results that you get.

Exercise 8: Simple Definitions

- a. Write a definition that will cause Scheme to recognize `dozen` as a name for the number 12.
- b. Write a definition that will cause Scheme to recognize `raise-to-power` as a synonym for `expt`.
- c. Use both names in expressions to verify that Scheme has understood them.

Exercise 9: Square Roots Revisited

- a. What do you expect to happen when you ask DrScheme to compute the square root of -4?
- b. Verify your answer experimentally.
- c. What do your results suggest?

Exercise 10: Definitions, Revisited

As you observed in the DrScheme lab, you can use the definitions pane to name values that you expect to use again (or that you simply find it more convenient to refer to with a mnemonic). So far, all we've named is simple values. However, you can also name the results of expressions.

- a. In the definitions pane, write a definition that assigns the name `seconds-per-minute` to the value 60.
- b. In the definitions pane, write a definition that assigns the name `minutes-per-hour` to the value 60.
- c. In the definitions pane, write a definition that assigns the name `hours-per-day` to the value 24.
- d. In the definitions pane, write a definition that assigns the name `seconds-per-day` to the product of those three values. Note that you should use the following expression to express that product.

```
(* seconds-per-minutes minutes-per-hour hours-per-day)
```

e. Run your definitions and confirm in the interactions pane that `seconds-per-day` is defined correctly.

Exercise 11: Grading

Let's play for a bit with how one might use DrScheme to compute grades. (We teach you this, in part, so that you can figure out your estimated grade in this class and others.) Let's define five names, `grade1` through `grade5` that potentially represent grades on five homework assignments.

```
(define grade1 95)
(define grade2 93)
(define grade3 105)
(define grade4 30)
(define grade5 80)
```

Looking at those grades, you might observe that the student seems to have spent a bit of extra work on the third assignment, but that the extra work so disrupted the student's life that the next assignment was a disaster. (You may certainly analyze the grades differently.)

a. Write a definition that assigns the name `average-grade` to the average of the grades without dropping the highest and lowest grades.

Many faculty members discard these "outliers", with a grading policy of "I take the average of your grades after dropping the highest grade and the lowest grade".

b. Write a definition that assigns the name `highest-grade` to a *computed* highest grade. (That is, `highest-grade` should remain correct, even if I change the values associated with `grade1` through `grade5`.) You may find the `max` procedure useful.

c. Write a definition that assigns the name `lowest-grade` to a *computed* lowest grade. You may find the `min` procedure useful.

d. Write a definition that assigns the name `weighted-average` to the weighed average grade (that is, the grade that results from dropping the lowest and highest grades and then averaging the result).

Exercise 12: Wrapup

Quit DrScheme and log out of the workstation.

Notes

Notes on Exercise 7

```
(2 + 3)
```

When DrScheme sees the left parenthesis at the beginning of the expression `(2 + 3)`, it expects the expression to be a procedure call, and it expects the procedure to be identified right after the left parenthesis. But `2` does not identify a procedure; it stands for a number. (A "procedure application" is the same thing as a procedure call.)

7 * 9

In the absence of parentheses, DrScheme sees `7 * 9` as three separate and unrelated expressions -- the numeral `7`; `*`, a name for the primitive multiplication procedure; and `9`, another numeral. It interprets each of these as a command to evaluate an expression: “Compute the value of the numeral `7`! Find out what the name `*` stands for! Compute the value of the numeral `9`!” So it performs the first of these commands and displays `7`; then it carries out the second command, reporting that `*` is the name of the primitive procedure `*`; and finally it carries out the third command and displays the result, `9`. This behavior is confusing, but it’s strictly logical if you look at it from the computer’s point of view (remembering, of course, that the computer has absolutely no common sense).

```
sqrt(49)
```

As in the preceding case, DrScheme sees `sqrt(49)` as two separate commands: `sqrt` means “Find out what `sqrt` is!” and `(49)` means “Call the procedure `49`, with no arguments!” DrScheme responds to the first command by reporting that `sqrt` is the primitive procedure for computing square roots and to the second by pointing out that the number `49` is not a procedure.

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.