Index

accumulators, 101

booleans, 7

built-in functions
*: multiplication function, 19
+: addition function, 19
-: subtraction function, 19
/: division function, 19
<: less than, 43
<=: less than or equal to, 43
=: equal to, 43
>: greater than, 43
>=: greater than or equal to, 43
abs, 145
append, 147
apply, 176
boolean?: type checker, 41
cons: create a cons cell, 131, 133
eq?, 44
equal?, 205
eval: evaluation function, 21
even?: is its input even?, 145
even?even?, 145
first: access first element of a list, 131
format, 217
fourth, 133
integer?, 21, 111
length, 138
list, 135, 162
list-ref, 139
list?: type checker, 41
load: for loading files, 62
make-vector, 200
member, 140
not: flip boolean value, 86
null?: type checker, 41
number?: type checker, 41
printf, 22
procedure?: type checker, 41
quotient: integer division, 21, 30
random: pseudo-random number gen., 118
remainder: integer division, 21, 30
remove, 147
rest: access the rest of a list, 131
reverse, 153
second, 133
sort, 165
string-append, 217
string?: type checker, 41
symbol?: type checker, 41
third, 133
vector, 199
vector-length, 201
vector-ref, 200
vector-set!, 206
void, 22, 142
void?: type checker, 41, 60

character sequences, 7, 8
comments, 63
computational model, 2, 15, 30
cons cell, 165, see also pair
contracts, 20
as comments in a Scheme program, 64
data, 7
non-primitive, 10
primitive, 7
data structures, 165, 209
field-based, 209
fields, 165
implemented using vectors, 209, 210
index-based, 209
recursively defined, 209
data type, 7
booleans, 8
functions, 19
non-empty lists, 25
null, 9
numbers, 7
strings, 10
symbols, 9
void, 9, 22, 142
deep lists, see also hierarchical lists, nested lists
Default Rule for evaluating non-empty lists, 26, 29
applying functions to inputs, 27, 30
generating errors, 29
on lists of lists, 27
denotation
single arrow notation (→), 8, 16
destructive functions, 3
DrScheme, 15
  interactions window, 15, 16
  prompt, 16
  the Run button, 63
elements
  of non-empty lists, 25
empty list, 9
environments, 12
escape sequences, 59
evaluation, 11, 15
  double arrow notation (⇒), 12, 16
  of booleans, 11
  of functions, 21
  of non-empty lists, 26
  of numbers, 11
  of special forms, 33
  of strings, 11
  of symbols, 12
  of the empty list, 11
  of the void datum, 11
evaluation function, 11
  as a built-in function, 21
expressions, 7
  primitive data, 7
  primitive data expressions, 7
function-call expression, 75
functions, 2, 11, 47
  built-in, 9, 19
  defining vs. applying, 47
  factorial, 76
  function application using the Default Rule, 27
  recursive, 75
  tail-recursive, 99
Global Environment, 13, 16
  and built-in functions, 19
hierarchical lists, see also deep lists, nested lists
identity function, 11
insertion-sort algorithm, 156
keyword symbols, 33, see also special forms
  else, 83
  redefining, 36
Lambda Calculus, 2
lambda function
  applying to inputs, 52
lazy evaluation, 67
  of the and special form, 88, 94
  of the cond special form, 84
  of the if special form, 68
  of the or special form, 94
local environment, 52, 115
local variable, 115
local vs. global environments, 53, 118
nested lists, see also hierarchical lists, deep lists
  nested if expressions, 81
  equivalence in terms of cond, 82
non-destructive functions, 3
non-empty lists, 25
  syntax and semantics, 25
non-strict truth values
  with and, 88, 89
  with if, 69
  with not, 86
  with or, 89
numbers, 7
pair, 165, see also cons cell
predicates, 41
  defined using boolean operators, 91
  type-checker predicates, 41
program, 7
recursion, 75
  accumulators, 101
  base case, 75
  recursive case, 76
  tail recursion, 97
  wrapper functions, 109
sample data structures
  date, 213
  dice-dice, 221
sample functions
  add-squares, 138
  approx-e-acc, 108
  approx-e-wr, 110
  approx-pi-acc, 106
  approx-pi-wr, 110
  bad-reverse, 154
  cafe-open?, 91
  cafe-open?-alt, 92
  civil-to-mil-hours, 90
  compare-sorting-algs, 164
  conc, 147
  cond-effects, 85
  cone-volume, 50
  continue!, 223
  continue-wr!, 226
  convert-to-boolean, 72
  date-day, 211
cond, 81
define, 34
   semantics of, 34
   side effect of, 34
   syntax of, 34
define-struct, 213
   semantics of, 213
   syntax of, 213
dolist, 194
dotimes, 192
if, 67
lambda, 48
   semantics of, 51
   syntax of, 50
let, 115
   in terms of lambda, 123
   semantics of, 116
   syntax of, 115
let*, 115, 124
   equivalence to nested lets, 126
   semantics of, 125
   syntax of, 124
letrec, 115, 127
   semantics of, 127
   syntax of, 127
or, 89
quote, 37
   alternate syntax ('), 38
   semantics of, 37
   syntax of, 37
time, 163
when, 72
while, 183
syntax, 1

truth values, 7

#<undefined>, 127

variables, 12

vectors
   immutable, 198
   mutable, 199, 200
   the pound syntax, 198
   the #(...) syntax, 198
void datum, 9, 22, 34, 77
Appendices
Appendix A

Guide to Your CS Account

All of the programming work you do in this course will be done using your CS computer account which you can access from any of the classroom or lab computers in the CS Department. The name of your account is typically the same as the first part of your Vassar email, although there can be exceptions. For example, my CS account name is hunsberg, which harkens back to the days when account names were limited to eight characters! Every student in this course has his or her own CS account. In addition, the CMPU-101 course itself also has an account, called cs101. All of the computer files and directories (a.k.a. folders) for all of the CS account holders are organized into a single tree-like structure called a file system. All of the computer programs you write for this course will be computer files that are stored within your portion of the CS file system. Thus, it will be important to understand how to navigate through the file system, create new files and directories, start up the DrScheme software, and print out and electronically submit your program files. All of this will be enabled by simply opening up a Terminal window and entering the appropriate commands at the prompt. (Since the computers are running the Linux operating systems, we may refer to these commands as Linux commands.) The rest of this chapter describes the file system, how to explore the file system using the commands issued from a Terminal window, and how to format, submit and print out your assignment files.

A.1 The File System

The file system is organized into a tree-like hierarchy of computer files and directories. A directory (or folder) is a collection of computer files that typically have something in common. For example, a directory called lab1 might contain all of the program files associated with your first programming lab. A directory may also contain subsidiary directories (a.k.a. sub-directories or sub-folders), thereby enabling directories to be organized into a tree-like hierarchy.

At the root of the file system is a special directory, called the root directory, that is the topmost ancestor of every other file and directory in the entire file system. For convenience, the root directory is frequently denoted by a single forward slash: /. As indicated in Fig. A.1, the root directory typically contains lots of directories with strange names (e.g., bin, dev, etc and mnt). These directories are used by the Linux operating system to handle things that will not concern us. However, one of the directories in the root directory is relevant for us: the home directory. As its name suggests, the home directory contains the “home” directories of every CS account. For example, the home directory contains two directories, called hunsberg and cs101, which are the respective home directories of my CS account and that of the CMPU-101 course.

Full pathnames. Each file or directory can be referred to by an absolute address, called its full pathname. The full pathname for a file or directory, X, represents the unique path from the root directory to X in the file system’s hierarchy. For example, the full pathname for my home directory is /home/hunsberg, since the root directory contains the home directory, and the home directory contains the hunsberg directory. Similarly, the full pathname for the cs101 home directory is /home/cs101.
The Desktop directory. As illustrated in Fig. A.1, the home directory for each CS account contains a sub-directory called Desktop. Although my Desktop directory has the same name as your Desktop directory, they are in fact distinct directories. The operating system has no trouble distinguishing them because their full pathnames are unique. For example, the full pathname for my Desktop directory is /home/hunsberg/Desktop, while the full pathname for the Desktop directory belonging to the cs101 account is /home/cs101/Desktop.

* Most of the files and directories located within your Desktop directory will have a corresponding icon that is automatically displayed on your computer screen’s Desktop.

All of the files you create for your work in this course should be organized within your Desktop directory, as illustrated in Fig. A.2. Notice that this organization allows room for growth should you decide to take subsequent Computer Science courses (e.g., CMPU-102, CMPU-145, and so on).

A.2 Using Terminal to Explore and Augment the File System

The Linux operating system provides numerous commands that enable you to navigate through the file system. These commands are processed by a program called Terminal. When you start the Terminal program, it opens up a Terminal window. When a command is typed into the Terminal window, and the Enter key is tapped, the Terminal program will attempt to execute the command.
When using Linux commands in a Terminal window to navigate the file system, the Terminal program keeps track of your current location within the directory tree. That current location is called your working directory. The working directory is often automatically displayed as part of the prompt in the Terminal window. Below are listed some of the most useful Linux commands for navigating the file system and creating new directories. The use of these commands is covered by Lab 1.

- **pwd** – Print the Working Directory (i.e., display where you are in the tree of directories). When you first open the Terminal window, the working directory is typically set to be the home directory of your account. Thus, if I open up a terminal window in my account and immediately enter the `pwd` command, it will cause the following to be displayed: `/home/hunsberg`.

- **ls** – LiSt the contents (i.e., files and sub-directories) of the working directory.

- **cd** – Change Directory. If used by itself, this command returns you to your account’s home directory (i.e., it sets the working directory to be your home directory). If you give it an input (e.g., a full pathname), then the `cd` command will set the working directory to be whatever directory you specify.

- **mkdir** – MaKe (i.e., create) a new DIRectory. This command takes one input: either a full pathname for the new directory or just a simple name for it. For example, the following command would create a new directory named `tmp` within my Desktop directory:

  ```bash
  mkdir /home/hunsberg/Desktop/tmp
  ```

  Alternatively, if I was already in the Desktop directory (i.e., if the working directory was set to be my Desktop directory), then the following simpler command would have the same effect:

  ```bash
  mkdir tmp
  ```

As already mentioned, Lab 1 will demonstrate the use of these and other Linux commands in more detail.

### A.3 Submitting Programming Assignments

This section describes the process of submitting programming assignments. Typically, this will involve two steps: (1) printing out your definitions and interactions files; and (2) electronically submitting the directory that contains these two files.

* When doing any lab or assignment, be sure to save your definitions file periodically so that you don’t lose it should something go wrong! Give it a name such as `yourName-asmt3-defns.txt`.

#### Before Printing or Electronically Submitting your Files

Before printing or electronically submitting your files, you should carefully review the following guidelines.

- Your definitions and interactions must be saved as *plain-text* files! (If you are unsure about this, review the relevant portions of Lab 1.)

- Your *definitions window* should be nicely formatted. See the *code-from-class* postings on the course website for examples of nicely formatted code. Or look at the posted solutions to any lab or assignment. In particular:

  * Make sure that your definitions file begins with a block of comments like this:

    ```scheme
    ;;==============================================
    ;; CMPU-101, Spring 2018
    ;; Asmt. or Lab Info
    ;; Your Name
    ;;==============================================
    ```

    where *Asmt. or Lab Info* is replaced by the relevant assignment or lab number (e.g., Asmt. 3 or Lab 5), and *Your Name* is replaced by your name!
• Make sure that the first Scheme expression in your definitions file is: (load "asmt-helper.txt").
• Make sure that the second Scheme expression in your definitions file involves an application of the header function to appropriate inputs, for example, something having the form:
  (header "Your Name" "Asmt. 3").
When you hit the Run button, you should see a nicely displayed header at the top of your interactions.
• Make sure that each problem is introduced by an invocation of the problem function, surrounded by commented lines of dashes, as illustrated below:

  ;; -----------------------------------
  (problem "Description")
  ;; -----------------------------------

• Make sure that each function you define is preceded by a “contract” (i.e., a block of comments that specifies the name of the function, the names and descriptions of the input parameters, a brief description of the output, and, if your function has side effects, a brief description of those too. Make sure that your contract clearly distinguishes the output value of the function from any side effects it might have. The contract should have the following form:

  ;; FUNCTION-NAME
  ;; -----------------------------------
  ;; INPUTS: names and descriptions of inputs
  ;; OUTPUT: description of output value (or "none")
  ;; SIDE EFFECTS: description of side effects (if any)

• In your function definition, the names for your function and its inputs should match the names that appear in the contract!
• Make sure that your code is properly indented. This is easiest to do by selecting the DrScheme menu item and choosing Reindent All.
• Make sure that your code does not include long lines of text that wrap around to the next line! Instead, break up long lines by using the Enter key, and taking advantage of DrScheme’s automatic indentation!
• When needed, your code should be augmented with concise comments explaining (briefly) what your code does. (See code-from-class postings for examples.) For example, if your function uses the cond special form (cf. Chapter 11), then each case of your cond should be preceded by a brief comment describing that case.
• Make sure that you have thoroughly tested your functions to demonstrate that they work as desired. This is typically done by providing a bunch of tester expressions that test a variety of cases beyond those that are given in the lab or assignment instructions.
• Make sure that there are blank lines between the problem expression and the contract, between the contract and the function definition, between the function definition and the tester expressions, and between the tester expressions and the following problem (if any). Again, see code-from-class postings for examples.

• When you are confident that your definitions file adheres to the above guidelines, then do the following:

  • Save your definitions window one last time.
  • Hit the Run button one last time.
  • Save your interactions as plain text! (Use the Save Other and Save Interactions as Text... menu items in DrScheme.)
  • Double-check that your interactions begin with a nice block of text generated by the header function. The top of your interactions should have the following form:
Asmt. or Lab Info
Your Name

where “Asmt. or Lab Info” is replaced by the relevant information, and “Your Name” is replaced by your name. If this information does not appear at the top of your interactions, check that your definitions file includes a call to the header function as described earlier.

* The contents of your interactions should be laid out nicely using the problem and tester functions, as described earlier. If not, go back to your Definitions Window and make the needed changes.

* Double-check that each tester expression is properly displaying both the input(s) and output—and that each is generating the right answer! If you spot any errors, go back to your function definition and make needed changes. If you make any changes to your Definitions Window, you will need to save your definitions, hit the Run button again, and then save your interactions (as plain text) again.

Congratulations! You should now be ready to print out and electronically submit your work!

A.3.1 Printing Text Files

* Warning! The information in this section applies only to printing out files containing plain text! The commands given below should not be used to print out pdf, doc, jpg, or any other non-plain-text files.

For most programming assignments, you will need to print out only two files: your definitions file and your interactions file. (It is not necessary to print out anything for labs.) Both of these files should be plain-text files. If either appears with a bunch of gibberish then you should review the instructions for saving your definitions or interactions as plain-text files. You do not need to turn in printouts of the asmt-helper.txt file, since you are not expected to make changes to that file. In addition, you should not print out any file whose name ends with a ~ character (e.g., myfile.txt~); those files are automatically generated backup files that can be safely ignored.

---

Example A.3.1

Suppose that ~hunsberg/Desktop/my101/labs/lab2/hun-lab2.txt is the full pathname for a plain-text file called hun-lab2.txt. The following command can be used within a Terminal window to print out that file to the printer called Asprey, which is located in Room SP 307:

enscript -P Asprey ~hunsberg/Desktop/my101/labs/lab2/hun-lab2.txt

Since typing out full pathnames can be quite tedious, there’s an even easier way. First, cd into the desired directory—in this case, my lab2 directory; and then issue the following, simpler command:

enscript -P Asprey hun-lab2.txt

(See Section A.2 if you need a refresher on cd-ing into a desired directory.)

---

In general, if you are currently in a directory D that contains a plain-text file named myfile.txt, then you can print out that file using the following command:

enscript -P Asprey myfile.txt

If you have any trouble printing, ask a coach for help.

* After printing your definitions and interactions, make sure to staple them—with the definitions on top!

* The Asprey printer should only be used to print out Computer Science labs or assignments.
A.3.2 Submitting your Files Electronically

Assignment files must be electronically submitted using the submit101 command from a Terminal window. This command has the following syntax:

```
submit101  AsmtSubmissionName  YourAsmtDir
```

where AsmtSubmissionName is the name for this assignment for submission purposes (which is typically given to you as part of the assignment instructions) and YourAsmtDir is the name of your assignment directory. (That’s right: you must submit the entire directory; the submit101 command cannot be used to submit individual files.)

---

**Example A.3.2**

Suppose that the AsmtSubmissionName is h-asmt3 and your assignment directory is called asmt3. (We may also say that h-asmt3 is the name of the dropbox into which you are going to submit your assignment.) Suppose further that your asmt3 directory is contained within a directory called asmts. Then you would electronically submit your asmt3 directory by first cd-ing into your asmts directory, and then executing the following command:

```
submit101 h-asmt3 asmt3
```

Note that it is very important that you be in the parent directory of the directory that you want to submit! (The asmts directory is called the parent of the asmt3 directory because asmts contains asmt3.) If you are in the asmt3 directory, then you should execute the following command to cd into the parent asmts directory:

```
cd ..
```

The two periods denote the parent directory of the working directory.

If you have any trouble using the submit101 command, ask me or a coach during lab or office/coaching hours.
Appendix B

Labs
B.1 Lab 1: Your first CMPU-101 lab session!

The purpose of this lab is to demonstrate the basics of navigating your Computer Science account, creating files and directories, saving them, and so on.

• During this lab, you will see some Scheme expressions that you won’t fully understand until you read later chapters. For now, just think of them as fillers that illustrate where certain kinds of things go within a program file.

• If you get stuck anywhere along the line, please ask for help!

You will access your CS account through computers that are running the Linux operating system. The following instructions introduce the basic Linux commands that you will use from within your CS account to download files, create files, organize your files, and so on, for all future labs and programming assignments.

Part One: Logging into your CS account

Sit down at one of the computers. Log into your account using the following information:

Username: The same as the first part of your Vassar email address.

Password: Look at the whiteboard!

Once the “Desktop” appears on-screen, click on the System menu in the lower-left corner of the screen. Select System Tools and then one of the options for a Terminal window. A Terminal window should appear on-screen. (If you can’t find the appropriate menu item to open up a Terminal window, ask for help.)

The Terminal window acts a lot like DrScheme’s Interactions Window that you have seen in class. In the Interactions Window, you type a Scheme expression at the prompt, followed by hitting the Enter key. In response, the Scheme datum denoted by that Scheme expression is evaluated and, usually, some information is displayed.

In the Terminal window, you type Linux commands at the prompt. When you hit the Enter key, the Terminal program tries to execute the command you entered. Of course, if you enter something wrong, it may complain vigorously.

One of the main job of commands entered into the Terminal window is to enable you to navigate the files and directories not only in your account, but also the entire file system for all of the CS accounts.

* Before proceeding, be sure to read Chapter A through Section A.2.

Table B.1 lists a sequence of Linux commands, along with explanations for each. For each command shown, type the command into the Terminal window, and then hit the Enter key. You should enter the commands one at a time, in the order shown. If you get mixed up, just go back to the first command—or ask for help.

After entering the entire sequence of commands listed in Table B.1, you should end up in a newly created directory, called lab1, within your CS account. The full pathname for this lab1 directory should be displayed as ~Desktop/my101/labs/lab1 or /home/yourAcctName/Desktop/my101/labs/lab1. (The character ~ is frequently used as a convenient abbreviation for your home directory.)

* If you get stuck and want to return to your home directory, just type: cd ~. Alternatively, you could just type cd without any inputs because it assumes you want to go to your home directory by default.

* If you want to “back up” to the “parent” of your working directory, use the following command (with two periods): cd ..

Examples of using the cd command to navigate through a directory tree are shown in Fig. B.1. The figure presumes that the account name—and hence the name of the home directory—is hunsberg.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description of what it does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>Display the <em>working directory</em> (i.e., the directory you are in right now). You are probably in your account’s home directory, which may be displayed as: <code>/home/yourAcctName.</code></td>
</tr>
<tr>
<td><code>cd ~</code></td>
<td>Move (conceptually) to your “home” directory (i.e., set the working directory to be your home directory). (This is probably not necessary since you were probably already in your home directory.) Note: <code>cd</code> stands for “change directory”; and <code>~</code> is a convenient shorthand for your home directory.</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>List the contents of your working directory. It should contain a subdirectory called Desktop.</td>
</tr>
<tr>
<td><code>cd Desktop</code></td>
<td>Change the working directory to be your Desktop directory.</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>List the contents of the working directory—which should be the Desktop directory at this point. Note that most of the contents of the Desktop directory have a corresponding icon on the Desktop!</td>
</tr>
<tr>
<td><code>mkdir my101</code></td>
<td>Create a new directory called my101. Note that because my101 is <em>not</em> a full pathname, the new directory will created within the working directory—in this case, the Desktop directory. And since the new directory is located within the Desktop directory, an icon will automatically appear on the Desktop!</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>This should show you that the Desktop directory now contains a subdirectory called my101.</td>
</tr>
<tr>
<td><code>cd my101</code></td>
<td>Travel into the my101 directory.</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Show that you are now in the my101 sub-directory. It will probably be displayed before the prompt as <code>~/Desktop/my101</code> or <code>/home/yourAcctName/Desktop/my101.</code></td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Show the (non-existent) contents of your new my101 directory.</td>
</tr>
<tr>
<td><code>mkdir labs</code></td>
<td>Create a new directory called labs inside the my101 directory. (It is created within the my101 directory because that is the working directory—i.e., the directory where you are right now.)</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Show that you indeed have created labs.</td>
</tr>
<tr>
<td><code>cd labs</code></td>
<td>Move into the labs directory.</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Show that you are indeed there.</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Show that the labs directory is currently empty.</td>
</tr>
<tr>
<td><code>mkdir lab1</code></td>
<td>Create a sub-directory called lab1.</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>Show that lab1 is there.</td>
</tr>
<tr>
<td><code>cd lab1</code></td>
<td>Move into the lab1 directory.</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Show where you are.</td>
</tr>
</tbody>
</table>

Table B.1: A sequence of Linux commands to create the directory structure for Lab 1
Part Two: Downloading Files

Okay, you have now created a directory called `lab1`, which is where you will put all of the files needed for this lab. There are two ways to get the desired files into your `lab1` directory.

1. **Fast and easy.** In your Terminal window, type the following command, exactly as shown below—which assumes that you have created the folders/directories named `my101`, `labs` and `lab1`, as discussed above:

   ```
   ```

   This says to copy all of the files from the `lab1` directory at the specified location within the `cs101` course account into the `lab1` directory you recently created within your account. (The `cs101` account is the owner of the first `lab1` directory; you are the owner of the second `lab1` directory.) In the above command, `˜cs101` denotes the home directory of the `cs101` account, and `*` is used to specify that all files within the first `lab1` directory should be copied into your `lab1` directory.

   **Note.** If you are already in your `˜/Desktop/my101/labs/lab1/` directory, then you can get the same result using the following, shorter command:

   ```
   ```

   In this command, the period represents your working directory. So this command will copy all of the files from the `lab1` directory owned by the `cs101` account, into your working directory which, hopefully, is your `lab1` directory.

2. **Very slow—but perhaps more familiar.** Use a web browser to fetch the needed files from the course web site, and then move them into your newly created `lab1` directory, as follows. First, click on the GLOBE icon in your task bar to open up a web browser. Then enter the following URL into the address bar of the browser window:

   ```
   ```
On our course web page, scroll down slightly until you see the link for “Lab Files”. Click on it. Then click on the link for Lab 1. You should see the following list of Scheme files:

- lab1-defns.txt
- asmt-helper.txt

Download each of these files by taking the following steps:

1. Click on the file name.
2. Under the File menu of your browser, select Save Page As... (or Save File As...).

Tell the browser that (for fun) you want to save the file into your Desktop directory. Do this by first selecting your home directory (whose name is the same as your account name) and then your Desktop sub-directory. (If you don’t see how to do this, ask for help.) When successful, there should be a new icon on your Desktop for each file.

After downloading the files from the course’s Lab 1 page, issue the following Linux commands in the Terminal window:

```bash
$ cd ~/Desktop
$ ls
$ mv lab1-defns.txt asmt-helper.txt my101/labs/lab1/
```

The last command uses the MoVe command: `mv`. It moves the files `lab1-defns.txt` and `asmt-helper.txt`, from the working directory into the `my101/labs/lab1` directory. Note that `my101/labs/lab1` is not a full pathname (since it does not begin with a forward slash). Instead, it is a relative pathname, which specifies only the portion of the full pathname starting from the working directory. In this case:

- the working directory’s full pathname is: `/home/yourAcctName/Desktop`
- the relative pathname is: `my101/labs/lab1`
- and lab1’s full pathname is: `/home/yourAcctName/Desktop/my101/labs/lab1`

Notice that lab1’s full pathname is constructed by concatenating the working directory’s full pathname and the relative pathname. (Ask for help if you have trouble seeing this.) For the above `mv` command to work, your working directory must be your Desktop directory, which contains the my101 sub-directory. Because the my101 directory is contained within the Desktop directory, we may say that the my101 directory is visible from the Desktop directory.

Incidentally, since you just moved the two downloaded files from your Desktop directory into the lab1 directory, those files are no longer on your Desktop. However, their icons might still be there. To update your Desktop display, hit the F5 key on the keyboard. The icons should disappear.

Finally, use the `cd` command to move into the lab1 directory, and then the `ls` command to list its contents:

```bash
$ cd my101/labs/lab1
$ ls
```

If you don’t see the two downloaded files, ask for help.

*Phew!* The good news is: The above commands are almost all of the Linux commands we are going to need for the entire semester.
Part Three: Firing Up DrScheme

Use the `pwd` command to verify that you are currently in your lab1 directory. Use the `ls` command to verify that you have successfully downloaded the desired files and moved them into your lab1 directory. Then—while still in your lab1 directory—type the following command into the Terminal window to start up the DrScheme program:

```
drscheme&
```

If you forget to type the `&` character, then the Terminal window will freeze until the DrScheme program is closed/finished. If you include the `&` character, you can continue to use the Terminal window while the DrScheme program is running.

⇒ Since this is the first time that you have opened DrScheme, you will need to “choose” the “Full Swindle” language, as follows. First, in the DrScheme menu bar, click on the Language menu item, and then select Choose Language. Then, in the pop-up window, under Swindle, select Full Swindle.

Opening a file in DrScheme. Under the File menu of DrScheme, select the Open item. When prompted, select the file `lab1-defns.txt`. The contents of the file should appear in DrScheme’s Program Definitions window pane. Normally, the Program Definitions window pane occupies the top half of DrScheme’s main window, with the Interactions Window in the bottom half; however, until you click the Run button, the Program Definitions window may be all you see.

Under the File menu of DrScheme, select the Save Other menu item, and then, after that, choose Save Definitions As Text.... When prompted, type in: `yourName-lab1-defns.txt`. Make sure that it is saved within your lab1 directory. You can check this by using the `ls` command in the Terminal window.

Programming in DrScheme. Have a look at the contents of the `yourName-lab1-defns.txt` file. In addition to a variety of comments (the lines that start with semi-colons), it contains a bunch of (possibly strange-looking) Scheme expressions, such as:

```
(load "asmt-helper.txt")
(header "myName" "Lab 1")
(problem 0)
(problem 1)
```

As will be seen—particularly in Chapter 10—these expressions will enable us to generate nicely formatted text in the Interactions Window that is suitable for printing and submitting! For now, the following brief descriptions will suffice. (Remember, the point of this lab is to demonstrate the mechanics of using your CS account, not to delve into the meaning of these kinds of Scheme expressions.)

The `load` expression causes the expressions contained within the file `asmt-helper.txt` to be loaded into the Interactions Window, just as if you had typed those expressions, by hand, into the Interactions Window, one after the other. The `asmt-helper.txt` file defines several useful functions, including: `tester`, `header` and `problem`. The `tester` function can be used to facilitate testing Scheme functions; and the `header` and `problem` functions enable the display of nicely formatted headings within the Interactions Window.

The following sequence of actions will demonstrate what these functions do.

Click on the Run button at the top-right of DrScheme’s window. The Run button loads the contents of the Program Definitions Window into the Interactions Window, so that you don’t have to manually enter (and re-enter) them. You should see some stuff printed out in the Interactions Window. Scroll through the Interactions Window results to get an idea of which expressions in the Definitions Window gave rise to the expressions you see in the Interactions Window.

Find the expression, `(header "myName" "Lab 1")`, in the Program Definitions Window. Change the characters `myName` to something that more accurately reflects your name. (Keep the double-quotes.)
Find the expression, (problem 0), in the Program Definitions Window. Notice that, following that expression, there are several expressions that involve the tester function. You can see the corresponding results in the Interactions Window when you hit the Run button. Put in a few more tester expressions. Include expressions whose evaluation you are unsure of. Predict (to yourself) what the result will be, then hit the Run button to see what DrScheme does.

From time to time, be sure to hit the Save button, located near the top of the DrScheme window, so that your program file (i.e., the contents of the Definitions Window) is saved. (The Save button is only visible if you have made some changes to the contents of the Definitions Window since the last time the contents were saved.)

Below the expression, (problem 1), in the Definitions Window, you should see an expression indicating that you haven’t yet defined the distance-fallen function. Delete that expression and replace it with the following:

(define distance-fallen
  (lambda (num-seconds)
    (* 16 num-seconds num-seconds)))

This expression defines a function named distance-fallen that takes a single input, called num-seconds, which represents the number of seconds since an object was dropped. (Chapter 9 shows how to define functions in Scheme using the define and lambda special forms.) The output value generated by this function is the corresponding distance (in feet) that the object would have fallen in that number of seconds.

* After entering the above expression, be sure to hit the Save button. Then hit the Run button, which causes the contents of your Definitions Window to be loaded into DrScheme.

Enter expressions such as (distance-fallen 3) directly into the Interactions Window to see if your function is working properly. (Chapter 6 addresses the application of functions to inputs using the Default Rule for evaluating non-empty lists.) Since 16 · 3 · 3 = 144, that is the result that DrScheme should report.

Better yet, you can automate the process of testing by typing several tester expressions in the Definitions Window. (Make sure that you type the tester expressions after the function-definition expression.) For example, you could type expressions such as:

(tester '(distance-fallen 3))
(tester '(distance-fallen 8))

(The reason for the quote mark is discussed in Chapter 10.) After putting several such expressions into your Definitions Window, and hitting the Save button, hit the Run button to see the results.

**Saving your interactions!** When you are confident that everything is working properly, click the Run button one last time. The Interactions Window should now contain nicely formatted results.

* It is important to save the contents of your Interactions Window exactly as described below. Otherwise, you may end up with a file containing a bunch of garbled text.

Under the File menu, select Save Other, and then select Save Interactions as Text.... When prompted, enter a name for your file, such as: yourNameInteractions.txt. Be sure it is saved into your lab1 directory.

That’s it for this lab! The next lab will demonstrate how to print out your files and submit them electronically.