

Characters and Strings

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Collating Sequences
 - Exercise 2: A Control Predicate
 - Exercise 3: String Basics
 - Exercise 4: Creating Questions
 - Exercise 5: Referencing Lengths
 - Exercise 6: Generating Headings
 - Exercise 7: Marking Text
 - Exercise 8: Other Markup
 - Exercise 9: Build a Page

Exercises

Exercise 0: Preparation

If you have not done so already, you may want to skim the reading on characters and strings.

Exercise 1: Collating Sequences

- a. Determine the ASCII collating-sequence numbers for the capital letter A and for the lower-case letter a.
- b. Find out what ASCII character is in position 38 in the collating sequence.
- c. Do the digit characters *precede* or *follow* the capital letters in the ASCII collating sequence?
- d. If you were designing a character set, where in the collating sequence would you place the space character? Why?
- e. What position does the space character occupy in ASCII?

Exercise 2: A Control Predicate

In ASCII, the collating-sequence numbers of the control characters are 0 through 31 and 127. Define a predicate `char-control?` that returns `#t` if its argument is a control character, `#f` otherwise.

Exercise 3: String Basics

- Is the symbol `hyperbola` a string?
- Is the character `#\A` a string?
- Does the empty string count as a string?

Exercise 4: Creating Questions

Suggest three ways of constructing the string `???` -- one using a call to `make-string`, one a call to `string`, and one a call to `list->string`.

Exercise 5: Referencing Lengths

Here are two opposing views about the relationship between `string-length` and `string-ref`:

- “No matter what string `str` is, provided that it’s not the empty string, `(string-ref str (string-length str))` will return the last character in the string.”
- “No matter what string `str` is, `(string-ref str (string-length str))` is an error.”

Which, if either, of these views is correct? Why?

Exercise 6: Generating Headings

Write a procedure, `(heading level text)` that generates a string that contains HTML heading of the appropriate level. For example,

```
> (heading 2 "Exercise 6")
"<h2>Exercise 6</h2>"
> (heading 4 "History")
"<h4>History</h4>"
```

You may find it useful to use the procedure `number->string`.

Exercise 7: Marking Text

- Write a procedure, `(markup tag text)` that surrounds text with the given tag. For example.

```
> (markup "p" "Hi There")
"<p>Hi There</p>"
> (markup "strong" "Wicked Neat!")
"<strong>Wicked Neat!</strong>"
```

- Use `markup`, `string-append`, and any other procedures you deem appropriate to generate the following HTML:

```
<p>
Sam says <q>Scheme is <strong>Wicked Neat!</strong></q>
</p>
```

c. What are the advantages of using markup rather than marking your code directly?

Note that you may want to use the character `#\newline` for new lines.

Exercise 8: Other Markup

Use markup to implement the following procedures, each of which takes one argument (some text) and generates HTML for appropriately formatted text.

a. `bold`

b. `strong`

c. `paragraph`

d. `emphasize`

Exercise 9: Build a Page

a. Using the previous procedures, write a procedure, `page`, of no arguments that builds a simple HTML page of your choice. Your procedure will begin

```
(define page
  (lambda ()
    instructions-for-building-the-page)))
```

You can call the procedure with `(page)`.

b. Why might you use Scheme rather than a text editor to build a Web page?