**11. Characters and Strings**

**1. Introduction**

I have only explained about lists and numbers, because they are most frequently used in Scheme. Scheme has, however, other data types such as characters, strings, symbols, vectors so on, which I am going to explain in chapters 11—14.

First, I will explain about characters and strings in this chapter.

**2. Characters**

Adding **#\** before the character indicates that it is a character. For instance, #\a means the character a. Characters #\Space, #\Tab, #\Linefeed, and #\Returnrepresent space, tab, linefeed, and return, respectively.

Following functions about characters are defined in the R5RS.

**(char? obj)**

It returns #t if **obj** is a character.

**(char=? c1 c2)**

It returns #t if **c1** and **c2** are the same character.

**(char->integer c)**

It converts **c** to the corresponding integer (character code).  
Example: (char->integer #\a) ⇒ 97

**(integer->char n)**

It converts an integer to the corresponding character.

**(char<? c1 c2),   
(char<=? c1, c2),   
(char> c1 c2),   
(char>= c1 c2)**

These functions compare characters. Actually, the functions compare the size of the character codes.  
For instance, (char<? **c1** **c2**) is equal to (< (char->integer **c1**) (char->integer **c2**)) .

**(char-ci=? c1 c2),   
(char-ci<? c1 c2),   
(char-ci<=? c1 c2),   
(char-ci>? c1 c2),   
(char-ci>=? c1 c2)**

These functions compare characters without case sensitivity.

**(char-alphabetic? c),   
(char-numeric? c),   
(char-whitespace? c),   
(char-upper-case? c),   
(char-lower-case? c)**

These functions return #t if **c** is alphabetic, numerical, blank, cap, and lower-case, respectively

**(char-upcase c),   
(char-downcase c)**

These functions returns corresponding cap/lower if **c** is lower/cap. If not they returns **c** itself.

**3. Strings**

Strings are enclosed by double quotation marks. For instance, "abc" represents the string abc.

Following functions are about strings defined in the R5RS.

**(string? s)**

It returns #t if **s** is a string.

**(make-string n c)**

It returns a string consisting of **n** of characters **c**. The character **c** can be omitted.

**(string-length s)**

It returns the length of a string **s**.

**(string=? s1 s2)**

It returns #t if strings **s1** and **s2** are the same.

**(string-ref s idx)**

It returns the **idx**-th character (counting from 0) of a string **s**.

**(string-set! s idx c)**

It sets the **idx**-th character of a string **s** to **c**.

**(substring s start end)**

It returns a substring of **s** consisting of characters from **start** to **(end**-1).   
(substring "abcdefg" 1 4) ⇒ "bcd"

**(string-append s1 s2 ...)**

It connects strings **s1, s2 ...**.

**(string->list s)**

It converts a string **s** to a list of characters.

**(list->string ls)**

It converts a list of characters (**ls**) to a string.

**(string-copy s)**

It copies a string **s**.

**Exercise 1**

Write a function (title-style) that capitalizes the first character of words.

(title-style "the cathedral and the bazaar")

⇒ "The Cathedral And The Bazaar"

**4. Summary**

I have explained about characters and strings in this chapter.

I will explain about symbol in the next chapter. Symbol is a characteristic data type of Lisp/Scheme. Fast text manipulation is possible using this data type.

**Answer 1**

Convert a string to a list and capitalize the characters before spaces and convert it again to the string.

(define (identity x) x)

(define (title-style str)

(let loop ((ls (string->list str))

(w #t)

(acc '()))

(if (null? ls)

(list->string (reverse acc))

(let ((c (car ls)))

(loop (cdr ls)

(char-whitespace? c)

(cons ((if w char-upcase identity) c) acc))))))

;;; Another answer, You can assign caps to the string.

(define (title-style str)

(let ((n (string-length str)))

(let loop ((w #t) (i 0))

(if (= i n)

str

(let ((c (string-ref str i)))

(if w (string-set! str i (char-upcase c)))

(loop (char-whitespace? c) (1+ i)))))))

(title-style "the cathedral and the bazaar")

⇒ "The Cathedral And The Bazaar"