LISP - INPUT & OUTPUT

Common LISP provides numerous input-output functions. We have already used the format function, and print function for output. In this section, we will look into some of the most commonly used input-output functions provided in LISP.

Input Functions

The following table provides the most commonly used input functions of LISP:

SL No.	Functions and Descriptions
1	**read** &#38; optional *input-stream eof-error-p eof-value recursive-p* It reads in the printed representation of a Lisp object from input-stream, builds a corresponding Lisp object, and returns the object.
2	**read-preserving-whitespace** &#38; optional *in-stream eof-error-p eof-value recursive-p* It is used in some specialized situations where it is desirable to determine precisely what character terminated the extended token.
3	**read-line** &#38; optional *input-stream eof-error-p eof-value recursive-p* It reads in a line of text terminated by a newline.
4	**read-char** &#38; optional *input-stream eof-error-p eof-value recursive-p* It takes one character from input-stream and returns it as a character object.
5	**unread-char** *character &#38; optional input-stream* It puts the character most recently read from the input-stream, onto the front of input-stream.
6	**peek-char** &#38; optional *peek-type input-stream eof-error-p eof-value recursive-p* It returns the next character to be read from input-stream, without actually removing it from the input stream.
7	**listen** &#38; optional *input-stream* The predicate **listen** is true if there is a character immediately available from input-stream, and is false if not.
8	**read-char-no-hang** &#38; optional *input-stream eof-error-p eof-value recursive-p* It is similar to **read-char**, but if it does not get a character, it does not wait for a character,

but returns nil immediately.

9

clear-input &#38; optional *input-stream*

It clears any buffered input associated with *input-stream.*

10

read-from-string *string* &#38; optional *eof-error-p eof-value &#38; key :start :end :preserve-whitespace*

It takes the characters of the string successively and builds a LISP object and returns the object. It also returns the index of the first character in the string not read, or the length of the string $or, length + 1$, as the case may be.

11

parse-integer *string &#38; key :start :end :radix :junk-allowed*

It examines the substring of string delimited by :start and :end $defaulttothebeginningandendofthestring$. It skips over whitespace characters and then attempts to parse an integer.

12

read-byte *binary-input-stream* &#38; optional *eof-error-p eof-value*

It reads one byte from the binary-input-stream and returns it in the form of an integer.

Reading Input from Keyboard

The **read** function is used for taking input from the keyboard. It may not take any argument.

For example, consider the code snippet:

```
(write ( + 15.0 (read)))
```

Assume the user enters 10.2 from the STDIN Input, it returns,

```
25.2
```

The read function reads characters from an input stream and interprets them by parsing as representations of Lisp objects.

Example

Create a new source code file named main.lisp and type the following code in it:

```
; the function AreaOfCircle
; calculates area of a circle
; when the radius is input from keyboard

(defun AreaOfCircle()
(terpri)
(princ "Enter Radius: ")
(setq radius (read))
(setq area (* 3.1416 radius radius))
(princ "Area: ")
(write area))
(AreaOfCircle)
```

When you execute the code, it returns the following result:

```
Enter Radius: 5 (STDIN Input)
Area: 78.53999
```

Example

Create a new source code file named main.lisp and type the following code in it.

```
(with-input-from-string (stream "Welcome to Tutorials Point!")
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (read-char stream))
   (print (peek-char nil stream nil 'the-end))
   (values)
)
```

When you execute the code, it returns the following result:

```
#\W
#\e
#\l
#\c
#\o
#\m
#\e
#\Space
#\t
#\o
#\Space
```

The Output Functions

All output functions in LISP take an optional argument called *output-stream*, where the output is sent. If not mentioned or *nil*, output-stream defaults to the value of the variable *standard-output*.

The following table provides the most commonly used output functions of LISP:

SL No.	Functions and Descriptions
1	**write** *object* &#38; key :stream :escape :radix :base :circle :pretty :level :length :case :gensym :array **write** *object* &#38; key :stream :escape :radix :base :circle :pretty :level :length :case :gensym :array :readably :right-margin :miser-width :lines :pprint-dispatch Both write the object to the output stream specified by :stream, which defaults to the value of *standard-output*. Other values default to the corresponding global variables set for printing.
2	**prin1** *object* &#38; optional *output-stream* **print** *object* &#38; optional *output-stream* **pprint** *object* &#38; optional *output-stream*

princ *object* &#38; optional *output-stream*

All these functions outputs the printed representation of object to *output-stream*. However, the following differences are there:

- prin1 returns the object as its value.
- print prints the object with a preceding newline and followed by a space. It returns object.
- pprint is just like print except that the trailing space is omitted.
- princ is just like prin1 except that the output has no escape character

3

write-to-string *object &#38; key* :escape :radix :base :circle :pretty :level :length :case :gensym :array

write-to-string *object* &#38; key :escape :radix :base :circle :pretty :level :length :case :gensym :array :readably :right-margin :miser-width :lines :pprint-dispatch

prin1-to-string *object*

princ-to-string *object*

The object is effectively printed and the output characters are made into a string, which is returned.

4

write-char *character* &#38; optional *output-stream*

It outputs the character to *output-stream,* and returns character.

5

write-string *string* &#38; optional *output-stream* &#38; key :start :end

It writes the characters of the specified substring of *string* to the *output-stream.*

6

write-line *string* &#38; optional *output-stream* &#38; key :start :end

It works the same way as write-string, but outputs a newline afterwards.

7

terpri &#38; optional *output-stream*

It outputs a newline to *output-stream.*

8

fresh-line &#38; optional *output-stream*

it outputs a newline only if the stream is not already at the start of a line.

9

finish-output &#38; optional *output-stream*

force-output &#38; optional *output-stream*

clear-output &#38; optional *output-stream*

- The function **finish-output** attempts to ensure that all output sent to output-stream has reached its destination, and only then returns nil.

- The function **force-output** initiates the emptying of any internal buffers but returns nil without waiting for completion or acknowledgment.
- The function **clear-output** attempts to abort any outstanding output operation in progress in order to allow as little output as possible to continue to the destination.

10
write-byte *integer binary-output-stream*

It writes one byte, the value of the *integer.*

Example

Create a new source code file named main.lisp and type the following code in it.

```
; this program inputs a numbers and doubles it
(defun DoubleNumber()
   (terpri)
   (princ "Enter Number : ")
   (setq n1 (read))
   (setq doubled (* 2.0 n1))
   (princ "The Number: ")
   (write n1)
   (terpri)
   (princ "The Number Doubled: ")
   (write doubled)
)
(DoubleNumber)
```

When you execute the code, it returns the following result:

```
Enter Number : 3456.78 (STDIN Input)
The Number: 3456.78
The Number Doubled: 6913.56
```

Formatted Output

The function **format** is used for producing nicely formatted text. It has the following syntax:

```
format destination control-string &rest arguments
```

where,

- destination is standard output
- control-string holds the characters to be output and the printing directive.

A **format directive** consists of a tilde , optional prefix parameters separated by commas, optional colon : and at-sign @ modifiers, and a single character indicating what kind of directive this is.

The prefix parameters are generally integers, notated as optionally signed decimal numbers.

The following table provides brief description of the commonly used directives:

Directive	Description
~A	Is followed by ASCII arguments
~S	Is followed by S-expressions
~D	For decimal arguments

~B	For binary arguments
~O	For octal arguments
~X	For hexadecimal arguments
~C	For character arguments
~F	For Fixed-format floating-point arguments.
~E	Exponential floating-point arguments
~$	Dollar and floating point arguments.
~%	A new line is printed
~*	Next argument is ignored
~?	Indirection. The next argument must be a string, and the one after it a list.

Example

Let us rewrite the program calculating a circle's area:

Create a new source code file named main.lisp and type the following code in it.

```
(defun AreaOfCircle()
   (terpri)
   (princ "Enter Radius: ")
   (setq radius (read))
   (setq area (* 3.1416 radius radius))
   (format t "Radius: = ~F~% Area = ~F" radius area)
)
(AreaOfCircle)
```

When you execute the code, it returns the following result:

```
Enter Radius: 10.234 (STDIN Input)
Radius: = 10.234
Area = 329.03473
```