LISP - STRINGS

Strings in Common Lisp are vectors, i.e., one-dimensional array of characters.

String literals are enclosed in double quotes. Any character supported by the character set can be enclosed within double quotes to make a string, except the double quote character " and the escape character (\). However, you can include these by escaping them with a backslash (\).

Example

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

```
(write-line "Hello World")
(write-line "Welcome to Tutorials Point")
;escaping the double quote character
(write-line "Welcome to \"Tutorials Point\"")
```

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

```
Hello World
Welcome to Tutorials Point
Welcome to "Tutorials Point"
```

String Comparison Functions

Numeric comparison functions and operators, like, < and > do not work on strings. Common LISP provides other two sets of functions for comparing strings in your code. One set is case-sensitive and the other case-insensitive.

The following table provides the functions:

Case Sensitive Functions	**Case-insensitive Functions**	**Description**
string=	string-equal	Checks if the values of the operands are all equal or not, if yes then condition becomes true.
string/=	string-not-equal	Checks if the values of the operands are all different or not, if values are not equal then condition becomes true.
string&#60;	string-lessp	Checks if the values of the operands are monotonically decreasing.
string>	string-greaterp	Checks if the values of the operands are monotonically increasing.
string&#60;=	string-not-greaterp	Checks if the value of any left operand is greater than or equal to the value of next right operand, if yes then condition becomes true.
string>=	string-not-lessp	Checks if the value of any left operand is less than or equal to the value of its right operand, if yes then condition becomes true.

Example

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

```
; case-sensitive comparison
(write (string= "this is test" "This is test"))
(terpri)
(write (string> "this is test" "This is test"))
(terpri)
(write (string< "this is test" "This is test"))
(terpri)

;case-insensitive comparision
(write (string-equal "this is test" "This is test"))
(terpri)
(write (string-greaterp "this is test" "This is test"))
(terpri)
(write (string-lessp "this is test" "This is test"))
(terpri)

;checking non-equal
(write (string/= "this is test" "this is Test"))
(terpri)
(write (string-not-equal "this is test" "This is test"))
(terpri)
(write (string/= "lisp" "lisping"))
(terpri)
(write (string/= "decent" "decency"))
```

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

```
NIL
0
NIL
T
NIL
NIL
8
NIL
4
5
```

Case Controlling Functions

The following table describes the case controlling functions:

Function	Description
string-upcase	Converts the string to upper case
string-downcase	Converts the string to lower case
string-capitalize	Capitalizes each word in the string

Example

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

```
(write-line (string-upcase "a big hello from tutorials point"))
(write-line (string-capitalize "a big hello from tutorials point"))
```

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

```
A BIG HELLO FROM TUTORIALS POINT
A Big Hello From Tutorials Point
```

Trimming Strings

The following table describes the string trimming functions:

Function	Description
string-trim	It takes a string of characters as first argument and a string as the second argument and returns a substring where all characters that are in the first argument are removed off the argument string.
String-left-trim	It takes a string of characters as first argument and a string as the second argument and returns a substring where all characters that are in the first argument are removed off the beginning of the argument string.
String-right-trim	It takes a string characters as first argument and a string as the second argument and returns a substring where all characters that are in the first argument are removed off the end of the argument string

Example

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

```
(write-line (string-trim " " "   a big hello from tutorials point   "))
(write-line (string-left-trim " " "   a big hello from tutorials point   "))
(write-line (string-right-trim " " "   a big hello from tutorials point   "))
(write-line (string-trim " a" "   a big hello from tutorials point   "))
```

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

```
a big hello from tutorials point
a big hello from tutorials point   
   a big hello from tutorials point
big hello from tutorials point
```

Other String Functions

Strings in LISP are arrays and thus also sequences. We will cover these data types in coming tutorials. All functions that are applicable to arrays and sequences also apply to strings. However, we will demonstrate some commonly used functions using various examples.

Calculating Length

The **length** function calculates the length of a string.

Extracting Sub-string

The **subseq** function returns a sub-string *asastringisalsoasequence* starting at a particular index and continuing to a particular ending index or the end of the string.

Accessing a Character in a String

The **char** function allows accessing individual characters of a string.

Example

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

```
(write (length "Hello World"))
(terpri)
(write-line (subseq "Hello World" 6))
(write (char "Hello World" 6))
```

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

```
11
World
#\W
```

Sorting and Merging of Strings

The **sort** function allows sorting a string. It takes a sequence *vectororstring* and a two-argument predicate and returns a sorted version of the sequence.

The **merge** function takes two sequences and a predicate and returns a sequence produced by merging the two sequences, according to the predicate.

Example

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

```
;sorting the strings
(write (sort (vector "Amal" "Akbar" "Anthony") #'string<))
(terpri)

;merging the strings
(write (merge 'vector (vector "Rishi" "Zara" "Priyanka") (vector "Anju" "Anuj" "Avni")
#'string<))
```

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

```
#("Akbar" "Amal" "Anthony")
#("Anju" "Anuj" "Avni" "Rishi" "Zara" "Priyanka")
```

Reversing a String

The **reverse** function reverses a string.

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

```
(write-line (reverse "Are we not drawn onward, we few, drawn onward to new era"))
```

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

```
are wen ot drawno nward ,wef ew ,drawno nward ton ew erA
```

Concatenating Strings

The concatenate function concatenates two strings. This is generic sequence function and you must provide the result type as the first argument.

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

```
(write-line (concatenate 'string "Are we not drawn onward, " "we few, drawn onward to new
era"))
```

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

```
Are we not drawn onward, we few, drawn onward to new era
```

Processing math: 100%