F# - STRINGS

In F#, the string type represents immutable text as a sequence of Unicode characters.

String Literals

String literals are delimited by the quotation mark " character.

Some special characters are there for special uses like newline, tab, etc. They are encoded using backslash (\) character. The backslash character and the related character make the escape sequence. The following table shows the escape sequence supported by F#.

Character	Escape sequence
Backspace	\b
Newline	\n
Carriage return	\r
Tab	\t
Backslash	\\
Quotation mark	\"
Apostrophe	\'
Unicode character	\uXXXX or \UXXXXXXXX *whereXindicatesahexadecimaldigit*

Ways of Ignoring the Escape Sequence

The following two ways makes the compiler ignore the escape sequence −

- Using the @ symbol.
- Enclosing the string in triple quotes.

When a string literal is preceded by the @ symbol, it is called a **verbatim string.** In that way, all escape sequences in the string are ignored, except that two quotation mark characters are interpreted as one quotation mark character.

When a string is enclosed by triple quotes, then also all escape sequences are ignored, including double quotation mark characters.

Example

The following example demonstrates this technique showing how to work with XML or other structures that include embedded quotation marks −

```
// Using a verbatim string
let xmldata = @"<book author=""Lewis, C.S"" title=""Narnia"">"
printfn "%s" xmldata
```

When you compile and execute the program, it yields the following output −

```
<book author="Lewis, C.S" title="Narnia">
```

Basic Operators on Strings

The following table shows the basic operations on strings −

Value	Description
collect : *char → string* → string → string	Creates a new string whose characters are the results of applying a specified function to each of the characters of the input string and concatenating the resulting strings.
concat : string → seq<string> → string	Returns a new string made by concatenating the given strings with a separator.
exists : *char → bool* → string → bool	Tests if any character of the string satisfies the given predicate.
forall : *char → bool* → string → bool	Tests if all characters in the string satisfy the given predicate.
init : int → *int → string* → string	Creates a new string whose characters are the results of applying a specified function to each index and concatenating the resulting strings.
iter : *char → unit* → string → unit	Applies a specified function to each character in the string.
iteri : *int → char → unit* → string → unit	Applies a specified function to the index of each character in the string and the character itself.
length : string → int	Returns the length of the string.
map : *char → char* → string → string	Creates a new string whose characters are the results of applying a specified function to each of the characters of the input string.
mapi : *int → char → char* → string → string	Creates a new string whose characters are the results of applying a specified function to each character and index of the input string.
replicate : int → string → string	Returns a string by concatenating a specified number of instances of a string.

The following examples demonstrate the uses of some of the above functionalities −

Example 1

The String.collect function builds a new string whose characters are the results of applying a specified function to each of the characters of the input string and concatenating the resulting strings.

```
let collectTesting inputS =
   String.collect (fun c -> sprintf "%c " c) inputS
printfn "%s" (collectTesting "Happy New Year!")
```

When you compile and execute the program, it yields the following output −

```
H a p p y N e w Y e a r !
```

Example 2

The String.concat function concatenates a given sequence of strings with a separator and returns a new string.

```
let strings = [ "Tutorials Point"; "Coding Ground"; "Absolute Classes" ]
let ourProducts = String.concat "\n" strings
printfn "%s" ourProducts
```

When you compile and execute the program, it yields the following output −

```
Tutorials Point
Coding Ground
Absolute Classes
```

Example 3

The String.replicate method returns a string by concatenating a specified number of instances of a string.

```
printfn "%s" <| String.replicate 10 "*! "
```

When you compile and execute the program, it yields the following output −

```
*! *! *! *! *! *! *! *! *! *!
```