F# - BASIC I/O

Basic Input Output includes −

- Reading from and writing into console.
- Reading from and writing into file.

Core.Printf Module

We have used the *printf* and the *printfn* functions for writing into the console. In this section, we will look into the details of the **Printf** module of F#.

Apart from the above functions, the *Core.Printf* module of F# has various other methods for printing and formatting using % markers as placeholders. The following table shows the methods with brief description −

Value	Description
bprintf : StringBuilder → BuilderFormat<'T> → 'T	Prints to a StringBuilder.
eprintf : TextWriterFormat<'T> → 'T	Prints formatted output to stderr.
eprintfn : TextWriterFormat<'T> → 'T	Prints formatted output to stderr, adding a newline.
failwithf : StringFormat<'T,'Result> → 'T	Prints to a string buffer and raises an exception with the given result.
fprintf : TextWriter → TextWriterFormat<'T> → 'T	Prints to a text writer.
fprintfn : TextWriter → TextWriterFormat<'T> → 'T	Prints to a text writer, adding a newline.
kbprintf : $unit \rightarrow 'Result$ → StringBuilder → BuilderFormat<'T,'Result> → 'T	Like bprintf, but calls the specified function to generate the result.
kfprintf : $unit \rightarrow 'Result$ → TextWriter → TextWriterFormat<'T,'Result> → 'T	Like fprintf, but calls the specified function to generate the result.
kprintf : $string \rightarrow 'Result$ → StringFormat<'T,'Result> → 'T	Like printf, but calls the specified function to generate the result. For example, these let the printing force a flush after all output has been entered onto the channel, but not before.
ksprintf : $string \rightarrow 'Result$ → StringFormat<'T,'Result> → 'T	Like sprintf, but calls the specified function to generate the result.
printf : TextWriterFormat<'T> → 'T	Prints formatted output to stdout.
printfn : TextWriterFormat<'T> → 'T	Prints formatted output to stdout, adding a newline.
sprintf : StringFormat<'T> → 'T	Prints to a string by using an internal string buffer and returns the result as a string.

Format Specifications

Format specifications are used for formatting the input or output, according to the programmers'

need.

These are strings with % markers indicating format placeholders.

The syntax of a Format placeholders is −

```
%[flags][width][.precision][type]
```

The **type** is interpreted as −

Type	Description
%b	Formats a **bool,** formatted as **true** or **false**.
%c	Formats a character.
%s	Formats a **string,** formatted as its contents, without interpreting any escape characters.
%d, %i	Formats any basic integer type formatted as a decimal integer, signed if the basic integer type is signed.
%u	Formats any basic integer type formatted as an unsigned decimal integer.
%x	Formats any basic integer type formatted as an unsigned hexadecimal integer, using lowercase letters a through f.
%X	Formats any basic integer type formatted as an unsigned hexadecimal integer, using uppercase letters A through F.
%o	Formats any basic integer type formatted as an unsigned octal integer.
%e, %E, %f, %F, %g, %G	Formats any basic floating point type ***float, float32*** formatted using a C-style floating point format specifications.
%e, %E	Formats a signed value having the form [-]d.ddde[sign]ddd where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or -.
%f	Formats a signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
%g, %G	Formats a signed value printed in f or e format, whichever is more compact for the given value and precision.
%M	Formats a Decimal value.
%O	Formats any value, printed by boxing the object and using its **ToString** method.
%A, %+A	Formats any value, printed with the default layout settings. Use %+A to print the structure of discriminated unions with internal and private representations.
%a	A general format specifier, requires two arguments. The first argument is a function which accepts two arguments: first, a context parameter of the appropriate type for the given formatting function *forexample, aTextWriter*, and second, a value to print and which either outputs or returns appropriate text. The second argument is the particular value to print.

%t	A general format specifier, requires one argument: a function which accepts a context parameter of the appropriate type for the given formatting function *aTextWriter* and which either outputs or returns appropriate text. Basic integer types are **byte, sbyte, int16, uint16, int32, uint32, int64, uint64, nativeint,** and **unativeint.** Basic floating point types are **float** and **float32.**

The **width** is an optional parameter. It is an integer that indicates the minimal width of the result. For example, %5d prints an integer with at least spaces of 5 characters.

Valid **flags** are described in the following table −

Value	Description
0	Specifies to add zeros instead of spaces to make up the required width.
-	Specifies to left-justify the result within the width specified.
+	Specifies to add a + character if the number is positive *tomatcha − signfornegativenumbers.*
' ' *space*	Specifies to add an extra space if the number is positive *tomatcha − signfornegativenumbers.*
#	Invalid.

Example

```
printf "Hello "
printf "World"
printfn ""
printfn "Hello "
printfn "World"
printf "Hi, I'm %s and I'm a %s" "Rohit" "Medical Student"

printfn "d: %f" 212.098f
printfn "e: %f" 504.768f

printfn "x: %g" 212.098f
printfn "y: %g" 504.768f

printfn "x: %e" 212.098f
printfn "y: %e" 504.768f
printfn "True: %b" true
```

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

```
Hello World
Hello
World
Hi, I'm Rohit and I'm a Medical Studentd: 212.098000
e: 504.768000
x: 212.098
y: 504.768
x: 2.120980e+002
y: 5.047680e+002
True: true
```

The Console Class

This class is a part of the .NET framework. It represents the standard input, output, and error streams for console applications.

It provides various methods for reading from and writing into the console. The following table shows the methods −

Method	Description
Beep	Plays the sound of a beep through the console speaker.
Beep*Int*32, *Int*32	Plays the sound of a beep of a specified frequency and duration through the console speaker.
Clear	Clears the console buffer and corresponding console window of display information.
MoveBufferArea*Int*32, *Int*32, *Int*32, *Int*32, *Int*32, *Int*32	Copies a specified source area of the screen buffer to a specified destination area.
MoveBufferArea *Int*32, *Int*32, *Int*32, *Int*32, *Int*32, *Int*32, *Char*, *ConsoleColor*, *ConsoleColor*	Copies a specified source area of the screen buffer to a specified destination area.
OpenStandardError	Acquires the standard error stream.
OpenStandardError*Int*32	Acquires the standard error stream, which is set to a specified buffer size.
OpenStandardInput	Acquires the standard input stream.
OpenStandardInput*Int*32	Acquires the standard input stream, which is set to a specified buffer size.
OpenStandardOutput	Acquires the standard output stream.
OpenStandardOutput*Int*32	Acquires the standard output stream, which is set to a specified buffer size.
Read	Reads the next character from the standard input stream.
ReadKey	Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.
ReadKey*Boolean*	Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.
ReadLine	Reads the next line of characters from the standard input stream.
ResetColor	Sets the foreground and background console colors to their defaults.
SetBufferSize	Sets the height and width of the

	screen buffer area to the specified values.
SetCursorPosition	Sets the position of the cursor.
SetError	Sets the Error property to the specified TextWriter object.
SetIn	Sets the In property to the specified TextReader object.
SetOut	Sets the Out property to the specified TextWriter object.
SetWindowPosition	Sets the position of the console window relative to the screen buffer.
SetWindowSize	Sets the height and width of the console window to the specified values.
Write*Boolean*	Writes the text representation of the specified Boolean value to the standard output stream.
Write*Char*	Writes the specified Unicode character value to the standard output stream.
Write*Char*[]	Writes the specified array of Unicode characters to the standard output stream.
Write*Decimal*	Writes the text representation of the specified Decimal value to the standard output stream.
Write*Double*	Writes the text representation of the specified double-precision floating-point value to the standard output stream.
Write*Int*32	Writes the text representation of the specified 32-bit signed integer value to the standard output stream.
Write*Int*64	Writes the text representation of the specified 64-bit signed integer value to the standard output stream.
Write*Object*	Writes the text representation of the specified object to the standard output stream.
Write*Single*	Writes the text representation of the specified single-precision floating-point value to the standard output stream.
Write*String*	Writes the specified string value to the standard output stream.
Write*UInt*32	Writes the text representation of the specified 32-bit unsigned

	integer value to the standard output stream.
Write*UInt*64	Writes the text representation of the specified 64-bit unsigned integer value to the standard output stream.
Write*String*, *Object*	Writes the text representation of the specified object to the standard output stream using the specified format information.
Write*String*, *Object*[]	Writes the text representation of the specified array of objects to the standard output stream using the specified format information.
Write*Char*[], *Int*32, *Int*32	Writes the specified subarray of Unicode characters to the standard output stream.
Write*String*, *Object*, *Object*	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write*String*, *Object*, *Object*, *Object*	Writes the text representation of the specified objects to the standard output stream using the specified format information.
Write*String*, *Object*, *Object*, *Object*, *Object*	Writes the text representation of the specified objects and variable-length parameter list to the standard output stream using the specified format information.
WriteLine	Writes the current line terminator to the standard output stream.
WriteLine*Boolean*	Writes the text representation of the specified Boolean value, followed by the current line terminator, to the standard output stream.
WriteLine*Char*	Writes the specified Unicode character, followed by the current line terminator, value to the standard output stream.
WriteLine*Char*[]	Writes the specified array of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine*Decimal*	Writes the text representation of the specified Decimal value, followed by the current line terminator, to the standard output stream.
WriteLine*Double*	Writes the text representation of the specified double-precision floating-point value, followed by the current line terminator, to the

	standard output stream.
WriteLine*Int*32	Writes the text representation of the specified 32-bit signed integer value, followed by the current line terminator, to the standard output stream.
WriteLine*Int*64	Writes the text representation of the specified 64-bit signed integer value, followed by the current line terminator, to the standard output stream.
WriteLine*Object*	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream.
WriteLine*Single*	Writes the text representation of the specified single-precision floating-point value, followed by the current line terminator, to the standard output stream.
WriteLine*String*	Writes the specified string value, followed by the current line terminator, to the standard output stream.
WriteLine*UInt*32	Writes the text representation of the specified 32-bit unsigned integer value, followed by the current line terminator, to the standard output stream.
WriteLine*UInt*64	Writes the text representation of the specified 64-bit unsigned integer value, followed by the current line terminator, to the standard output stream.
WriteLine*String*, *Object*	Writes the text representation of the specified object, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine*String*, *Object*[]	Writes the text representation of the specified array of objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine*Char*[], *Int*32, *Int*32	Writes the specified subarray of Unicode characters, followed by the current line terminator, to the standard output stream.
WriteLine*String*, *Object*, *Object*	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.

WriteLine*String*, *Object*, *Object*, *Object*	Writes the text representation of the specified objects, followed by the current line terminator, to the standard output stream using the specified format information.
WriteLine*String*, *Object*, *Object*, *Object*, *Object*	Writes the text representation of the specified objects and variable-length parameter list, followed by the current line terminator, to the standard output stream using the specified format information.

The following example demonstrates reading from console and writing into it −

Example

```
open System
let main() =
   Console.Write("What's your name? ")
   let name = Console.ReadLine()
   Console.Write("Hello, {0}\n", name)
   Console.WriteLine(System.String.Format("Big Greetings from {0} and {1}",
"TutorialsPoint", "Absoulte Classes"))
   Console.WriteLine(System.String.Format("|{0:yyyy-MMM-dd}|", System.DateTime.Now))
main()
```

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

```
What's your name? Kabir
Hello, Kabir
Big Greetings from TutorialsPoint and Absoulte Classes
|2015-Jan-05|
```

The System.IO Namespace

The System.IO namespace contains a variety of useful classes for performing basic I/O.

It contains types or classes that allow reading and writing to files and data streams and types that provide basic file and directory support.

Classes useful for working with the file system −

- The System.IO.File class is used for creating, appending, and deleting files.
- System.IO.Directory class is used for creating, moving, and deleting directories.
- System.IO.Path class performs operations on strings, which represent file paths.
- System.IO.FileSystemWatcher class allows users to listen to a directory for changes.

Classes useful for working with the streams *sequenceofbytes* −

- System.IO.StreamReader class is used to read characters from a stream.
- System.IO.StreamWriter class is used to write characters to a stream.
- System.IO.MemoryStream class creates an in-memory stream of bytes.

The following table shows all the classes provided in the namespace along with a brief description −

Class	Description
BinaryReader	Reads primitive data types as binary values in a specific

	encoding.
BinaryWriter	Writes primitive types in binary to a stream and supports writing strings in a specific encoding.
BufferedStream	Adds a buffering layer to read and write operations on another stream.
Directory	Exposes static methods for creating, moving, and enumerating through directories and subdirectories.
DirectoryInfo	Exposes instance methods for creating, moving, and enumerating through directories and subdirectories.
DirectoryNotFoundException	The exception that is thrown when part of a file or directory cannot be found.
DriveInfo	Provides access to information on a drive.
DriveNotFoundException	The exception that is thrown when trying to access a drive or share that is not available.
EndOfStreamException	The exception that is thrown when reading is attempted past the end of a stream.
ErrorEventArgs	Provides data for the FileSystemWatcher.Error event.
File	Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of FileStream objects.
FileFormatException	The exception that is thrown when an input file or a data stream that is supposed to conform to a certain file format specification is malformed.
FileInfo	Provides properties and instance methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.
FileLoadException	The exception that is thrown when a managed assembly is found but cannot be loaded.
FileNotFoundException	The exception that is thrown when an attempt to access a file that does not exist on disk fails.
FileStream	Exposes a Stream around a file, supporting both synchronous and asynchronous read and write operations.
FileSystemEventArgs	Provides data for the directory events − Changed, Created, Deleted.
FileSystemInfo	Provides the base class for both FileInfo and DirectoryInfo objects.
FileSystemWatcher	Listens to the file system change notifications and raises events when a directory, or file in a directory, changes.
InternalBufferOverflowException	The exception thrown when the internal buffer overflows.
InvalidDataException	The exception that is thrown when a data stream is in an invalid format.
IODescriptionAttribute	Sets the description visual designers can display when referencing an event, extender, or property.
IOException	The exception that is thrown when an I/O error occurs.

MemoryStream	Creates a stream whose backing store is memory.
Path	Performs operations on String instances that contain file or directory path information. These operations are performed in a cross-platform manner.
PathTooLongException	The exception that is thrown when a path or file name is longer than the system-defined maximum length.
PipeException	Thrown when an error occurs within a named pipe.
RenamedEventArgs	Provides data for the Renamed event.
Stream	Provides a generic view of a sequence of bytes. This is an abstract class.
StreamReader	Implements a TextReader that reads characters from a byte stream in a particular encoding.
StreamWriter	Implements a TextWriter for writing characters to a stream in a particular encoding. To browse the .NET Framework source code for this type, see the Reference Source.
StringReader	Implements a TextReader that reads from a string.
StringWriter	Implements a TextWriter for writing information to a string. The information is stored in an underlying StringBuilder.
TextReader	Represents a reader that can read a sequential series of characters.
TextWriter	Represents a writer that can write a sequential series of characters. This class is abstract.
UnmanagedMemoryAccessor	Provides random access to unmanaged blocks of memory from managed code.
UnmanagedMemoryStream	Provides access to unmanaged blocks of memory from managed code.
WindowsRuntimeStorageExtensions	Contains extension methods for the IStorageFile and IStorageFolder interfaces in the Windows Runtime when developing Windows Store apps.
WindowsRuntimeStreamExtensions	Contains extension methods for converting between streams in the Windows Runtime and managed streams in the .NET for Windows Store apps.

Example

The following example creates a file called test.txt, writes a message there, reads the text from the file and prints it on the console.

Note − The amount of code needed to do this is surprisingly less!

```
open System.IO // Name spaces can be opened just as modules
File.WriteAllText("test.txt", "Hello There\n Welcome to:\n Tutorials Point")
let msg = File.ReadAllText("test.txt")
printfn "%s" msg
```

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

```
Hello There
```

```
Welcome to:
Tutorials Point
```