F# - BASIC SYNTAX

You have seen the basic structure of an F# program, so it will be easy to understand other basic building blocks of the F# programming language.

Tokens in F#

An F# program consists of various tokens. A token could be a keyword, an identifier, a constant, a string literal, or a symbol. We can categorize F# tokens into two types −

- Keywords
- Symbol and Operators

F# Keywords

The following table shows the keywords and brief descriptions of the keywords. We will discuss the use of these keywords in subsequent chapters.

Keyword	Description
abstract	Indicates a method that either has no implementation in the type in which it is declared or that is virtual and has a default implementation.
and	Used in mutually recursive bindings, in property declarations, and with multiple constraints on generic parameters.
as	Used to give the current class object an object name. Also used to give a name to a whole pattern within a pattern match.
assert	Used to verify code during debugging.
base	Used as the name of the base class object.
begin	In verbose syntax, indicates the start of a code block.
class	In verbose syntax, indicates the start of a class definition.
default	Indicates an implementation of an abstract method; used together with an abstract method declaration to create a virtual method.
delegate	Used to declare a delegate.
do	Used in looping constructs or to execute imperative code.
done	In verbose syntax, indicates the end of a block of code in a looping expression.
downcast	Used to convert to a type that is lower in the inheritance chain.
downto	In a **for** expression, used when counting in reverse.
elif	Used in conditional branching. A short form of else if.
else	Used in conditional branching.
end	In type definitions and type extensions, indicates the end of a section of member definitions. In verbose syntax, used to specify the end of a code block that starts with the begin keyword.

exception	Used to declare an exception type.
extern	Indicates that a declared program element is defined in another binary or assembly.
false	Used as a Boolean literal.
finally	Used together with try to introduce a block of code that executes regardless of whether an exception occurs.
for	Used in looping constructs.
fun	Used in lambda expressions, also known as anonymous functions.
function	Used as a shorter alternative to the fun keyword and a match expression in a lambda expression that has pattern matching on a single argument.
global	Used to reference the top-level .NET namespace.
if	Used in conditional branching constructs.
in	Used for sequence expressions and, in verbose syntax, to separate expressions from bindings.
inherit	Used to specify a base class or base interface.
inline	Used to indicate a function that should be integrated directly into the caller's code.
interface	Used to declare and implement interfaces.
internal	Used to specify that a member is visible inside an assembly but not outside it.
lazy	Used to specify a computation that is to be performed only when a result is needed.
let	Used to associate, or bind, a name to a value or function.
let!	Used in asynchronous workflows to bind a name to the result of an asynchronous computation, or, in other computation expressions, used to bind a name to a result, which is of the computation type.
match	Used to branch by comparing a value to a pattern.
member	Used to declare a property or method in an object type.
module	Used to associate a name with a group of related types, values, and functions, to logically separate it from other code.
mutable	Used to declare a variable, that is, a value that can be changed.
namespace	Used to associate a name with a group of related types and modules, to logically separate it from other code.
new	Used to declare, define, or invoke a constructor that creates or that can create an object. Also used in generic parameter constraints to indicate that a type must have a certain constructor.
not	Not actually a keyword. However, not struct in combination is used as a generic parameter constraint.
null	Indicates the absence of an object. Also used in generic parameter constraints.

of	Used in discriminated unions to indicate the type of categories of values, and in delegate and exception declarations.
open	Used to make the contents of a namespace or module available without qualification.
or	Used with Boolean conditions as a Boolean or operator. Equivalent to ||. Also used in member constraints.
override	Used to implement a version of an abstract or virtual method that differs from the base version.
private	Restricts access to a member to code in the same type or module.
public	Allows access to a member from outside the type.
rec	Used to indicate that a function is recursive.
return	Used to indicate a value to provide as the result of a computation expression.
return!	Used to indicate a computation expression that, when evaluated, provides the result of the containing computation expression.
select	Used in query expressions to specify what fields or columns to extract. Note that this is a contextual keyword, which means that it is not actually a reserved word and it only acts like a keyword in appropriate context.
static	Used to indicate a method or property that can be called without an instance of a type, or a value member that is shared among all instances of a type.
struct	Used to declare a structure type. Also used in generic parameter constraints. Used for OCaml compatibility in module definitions.
then	Used in conditional expressions. Also used to perform side effects after object construction.
to	Used in for loops to indicate a range.
true	Used as a Boolean literal.
try	Used to introduce a block of code that might generate an exception. Used together with *with* or finally.
type	Used to declare a class, record, structure, discriminated union, enumeration type, unit of measure, or type abbreviation.
upcast	Used to convert to a type that is higher in the inheritance chain.
use	Used instead of let for values that require Dispose to be called to free resources.
use!	Used instead of let! in asynchronous workflows and other computation expressions for values that require Dispose to be called to free resources.
val	Used in a signature to indicate a value, or in a type to declare a member, in limited situations.

void	Indicates the .NET void type. Used when interoperating with other .NET languages.
when	Used for Boolean conditions $whenguards$ on pattern matches and to introduce a constraint clause for a generic type parameter.
while	Introduces a looping construct.
with	Used together with the match keyword in pattern matching expressions. Also used in object expressions, record copying expressions, and type extensions to introduce member definitions, and to introduce exception handlers.
yield	Used in a sequence expression to produce a value for a sequence.
yield!	Used in a computation expression to append the result of a given computation expression to a collection of results for the containing computation expression.

Some reserved keywords came from the OCaml language −

asr	land	lor	lsl	lsr	lxor	mod	sig

Some other reserved keywords are kept for future expansion of F#.

atomic	break	checked	component	const	constraint	constructor
continue	eager	event	external	fixed	functor	include
method	mixin	object	parallel	process	protected	pure
sealed	tailcall	trait	virtual	volatile		

Comments in F#

F# provides two types of comments −

- One line comment starts with // symbol.
- Multi line comment starts with $\ast andendswith \ast$.

A Basic Program and Application Entry Point in F#

Generally, you don't have any explicit entry point for F# programs. When you compile an F# application, the last file provided to the compiler becomes the entry point and all top level statements in that file are executed from top to bottom.

A well-written program should have a single top-level statement that would call the main loop of the program.

A very minimalistic F# program that would display 'Hello World' on the screen −

```
(* This is a comment *)
(* Sample Hello World program using F# *)
printfn "Hello World!"
```

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

```
Hello World!
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js