

Learning to program with F#

Jon Spurring

November 12, 2016

Contents

1	Preface	5
2	Introduction	6
2.1	How to learn to program	6
2.2	How to solve problems	7
2.3	Approaches to programming	7
2.4	Why use F#	8
2.5	How to read this book	9
I	F# basics	10
3	Executing F# code	11
3.1	Source code	11
3.2	Executing programs	11
4	Quick-start guide	14
5	Using F# as a calculator	19
5.1	Literals and basic types	19
5.2	Operators on basic types	24
5.3	Boolean arithmetic	26
5.4	Integer arithmetic	27
5.5	Floating point arithmetic	29
5.6	Char and string arithmetic	31
5.7	Programming intermezzo	32

6	Constants, functions, and variables	34
6.1	Values	37
6.2	Non-recursive functions	42
6.3	User-defined operators	46
6.4	The Printf function	48
6.5	Variables	51
7	In-code documentation	57
8	Controlling program flow	62
8.1	For and while loops	62
8.2	Conditional expressions	66
8.3	Recursive functions	68
8.4	Programming intermezzo	71
9	Ordered series of data	75
9.1	Tuples	76
9.2	Lists	79
9.3	Arrays	84
10	Testing programs	89
10.1	White-box testing	92
10.2	Black-box testing	95
10.3	Debugging by tracing	98
11	Exceptions	105
12	Input and Output	113
12.1	Interacting with the console	114
12.2	Storing and retrieving data from a file	115
12.3	Working with files and directories.	118
12.4	Reading from the internet	119
12.5	Programming intermezzo	120

II	Imperative programming	124
13	Graphical User Interfaces	126
13.1	Drawing primitives in Windows	126
14	Imperative programming	127
14.1	Introduction	127
14.2	Generating random texts	128
14.2.1	0'th order statistics	128
14.2.2	1'th order statistics	128
III	Declarative programming	129
15	Sequences and computation expressions	130
15.1	Sequences	130
16	Patterns	136
16.1	Pattern matching	136
17	Types and measures	139
17.1	Unit of Measure	139
18	Functional programming	143
IV	Structured programming	146
19	Namespaces and Modules	147
20	Object-oriented programming	149
V	Appendix	150
A	Number systems on the computer	151
A.1	Binary numbers	153
A.2	IEEE 754 floating point standard	153

B	Commonly used character sets	154
B.1	ASCII	154
B.2	ISO/IEC 8859	155
B.3	Unicode	155
C	A brief introduction to Extended Backus-Naur Form	159
D	F_b	163
E	Language Details	168
E.1	Arithmetic operators on basic types	168
E.2	Basic arithmetic functions	171
E.3	Precedence and associativity	172
E.4	Lightweight Syntax	174
F	The Some Basic Libraries	175
F.1	System.String	176
F.2	List, arrays, and sequences	176
F.3	Mutable Collections	179
F.3.1	Mutable lists	179
F.3.2	Stacks	179
F.3.3	Queues	179
F.3.4	Sets and dictionaries	179
	Bibliography	180
	Index	181

Chapter 13

Graphical User Interfaces

A *command-line interface (CLI)* is a method for communicating with the user through text. In contrast, a *graphical user interface (GUI)* extends the ways of communicating with the user to also include organising the screen space in windows, icons, and other visual elements, and a typical way to activate these elements are through a pointing device such as the mouse or by touch. Some of these elements may themselves be textual, and thus most operating systems offers access to a command-line interface in a window alongside other interface types.

- command-line interface
- CLI
- graphical user interface
- GUI

Fsharp includes a number of implementations of graphical user interfaces, but at time of writing only *WinForms* is supported on both the Microsoft .Net and the Mono platform, and hence, WinForms will be the subject of the following chapter.

- WinForms

WinForms is designed for *event driven programs*, which spends most time waiting for the user to perform an action, called an event, and for each event has a set of predefined responses to be performed by the program. For example, Figure 13.1 shows the program Safari, which is a graphical user interface for accessing web-servers. The program present information to the user in terms of text and images, and has areas that when activated by clicking with a mouse or similar allows the user to, e.g., go to other web-pages by type URL, to follow hyperlinks, and to generate new pages by entering search queries.

- event driven programs

13.1 Drawing primitives in Windows

WinForms is based on two namespaces: `System.Windows.Forms` and `System.Drawing`. To start making a graphical display on the screen, the first thing to do is open a window, which acts as a reserved screen space for our output. With WinForms, this may be done as shown in Listing 13.1, and the result is shown in Figure 13.3.

Listing 13.1, winforms/openWindow.fsx:

Create the window and turn over control to the operating system. Use `win.Show()` on Microsoft Windows instead.

```
// Create a window
let win = new System.Windows.Forms.Form ()
// Start the event-loop. Use "win.Show()" on Microsoft Windows instead.
System.Windows.Forms.Application.Run win
```

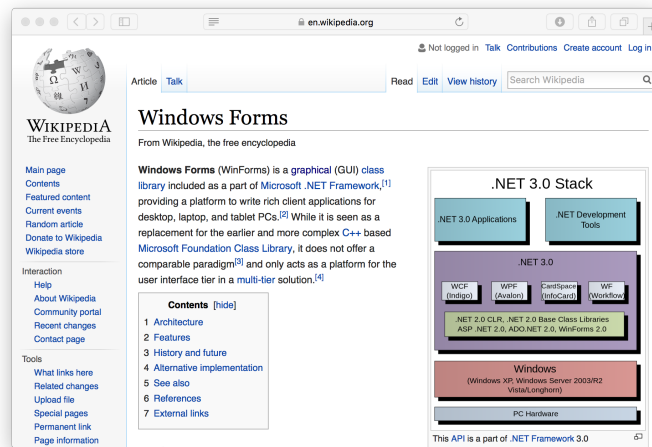


Figure 13.1: A web-browser is a graphical user interface for accessing a web-server and interacting with its services. Here the browser is showing the page https://en.wikipedia.org/wiki/Windows_Forms at time of writing.

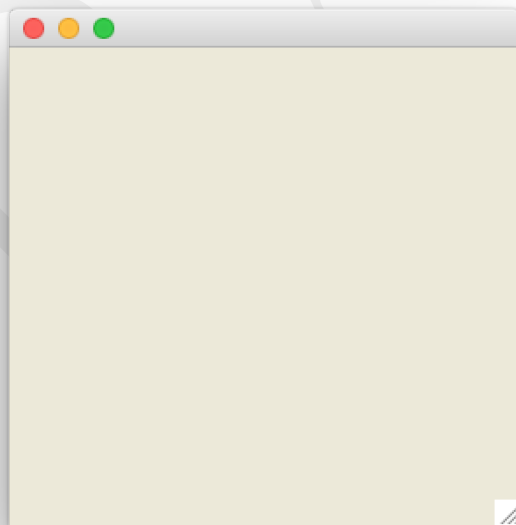


Figure 13.2: Result of running listing Listing 13.1.

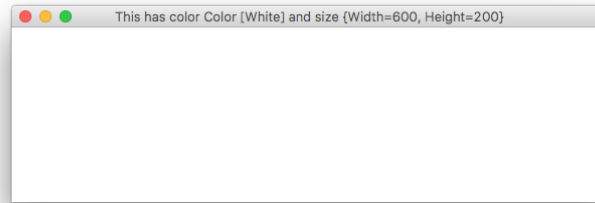


Figure 13.3: Result of running listing Listing 13.2.

The `new System.Windows.Forms.Form ()` creates an object (See Chapter 20), but does not display the window on the screen. When the function `System.Windows.Forms.Application.Run` is applied to the object, then the control is handed over to the WinForm's *event-loop*, which continues until the window is closed by, e.g., pressing the icon designated by the operating system. On the mac OSX that is the red button in the top left corner of the window frame, and on Windows it is the cross on the top right corner of the window frame.

The window, which WinForms calls a form, has a long list of *methods* and *properties*. E.g., the background color may be set by `BackColor`, the title of the window may be set by `Text`, and you may get and set the size of the window with the `Size`. This is demonstrated in Listing

**Listing 13.2, winforms/windowAttributes.fsx:
Create the window and changing its properties.**

```
// Create a window
let win = new System.Windows.Forms.Form ()
// Set some properties
win.BackColor <- System.Drawing.Color.White
win.Size <- System.Drawing.Size (600, 200)
win.Text <- sprintf "This has color %A and size %A" win.BackColor win.Size
// Start the event-loop. Use "win.Show()" on Microsoft Windows instead.
System.Windows.Forms.Application.Run win
```

These properties have been programmed as *accessors* implying that they may be used as mutable variables. The `System.Drawing.Color` is a general structure for specifying colors as 4 channels: alpha, red, green, blue, where each channel is an 8 bit unsigned integer, where the alpha channel specifies the transparency of a color, where values 0–255 denotes the range of fully transparent to fully opaque, and the remaining channels denote the amount of red, green, and blue where 0 is none and 255 is full intensity. Any color may be created using the `FromArgb` method, e.g., an opaque red is given by `System.Drawing.Color.FromArgb (255, 255, 0, 0)`. There are also many build-in colors, e.g., the same red color is also a known color and may be obtained as `System.Drawing.Color.Red`. For a given color, then the 4 alpha, red, green, and blue channel's values may be obtained as the A, R, G, B, see Listing 13.3

Listing 13.3, drawingColors.fsx:
Defining colors and accessing their values.

```
// open namespace for brevity
open System.Drawing
// Define a color from ARGB
let c = Color.FromArgb (0xFF, 0x7F, 0xFF, 0xD4) //Aquamarine
printfn "The color %A is (%x, %x, %x, %x)" c c.A c.R c.G c.B
// Define a list of named colors
let colors = [Color.Red; Color.Green; Color.Blue; Color.Black; Color.Gray;
              Color.White]
for col in colors do
    printfn "The color %A is (%x, %x, %x, %x)" col col.A col.R col.G col.B
```

```
The color Color [A=255, R=127, G=255, B=212] is (ff, 7f, ff, d4)
The color Color [Red] is (ff, ff, 0, 0)
The color Color [Green] is (ff, 0, 80, 0)
The color Color [Blue] is (ff, 0, 0, ff)
The color Color [Black] is (ff, 0, 0, 0)
The color Color [Gray] is (ff, 80, 80, 80)
The color Color [White] is (ff, ff, ff, ff)
```

The `System.Drawing.Size` is a general structure for specifying sizes as height and width pair of integers.

WinForms supports drawing of simple graphics primitives. Simple examples are `System.Drawing.Pen` to specify the color to be drawn, `System.Drawing.Point` to specify a pair of coordinates, and `System.Drawing.Graphics.DrawLine`. `DrawLine` is different than the previous examples since it must be related to a specific device, and it is typically accessed as an event. Displaying graphics in WinForms is performed as the reaction to an event. E.g., windows are created by the program, moved, minimized, occluded by other windows, resized, etc., by the user or the program, and each action may require that the content of the window is refreshed. Thus, we must create a function that WinForms can call, when it determines that the content needs to be redrawn. This is known as a *call-back function*, and it is added to an existing form using the `Paint.Add` function. As an example, consider the problem of draw a triangle in a window. For this we need to make a function that can draw a triangle not once, but any time WinForms determines it necessary to draw and redraw the triangle. This is done in Listing 13.4.

· call-back
function

Listing 13.4, winforms/Window.fsx:
Create the window and changing its properties.

```
// Choose some points and a color
let Points =
    [|System.Drawing.Point (0,0);
     System.Drawing.Point (10,170);
     System.Drawing.Point (320,20);
     System.Drawing.Point (0,0)|]
let penColor = System.Drawing.Color.Black
// Create window and setup drawing function
let pen = new System.Drawing.Pen (penColor)
let win = new System.Windows.Forms.Form ()
win.Paint.Add (fun e -> e.Graphics.DrawLine (pen, Points))
// Start the event-loop. Use "win.Show()" on Microsoft Windows instead.
System.Windows.Forms.Application.Run win
```

A walk-through of the code is as follows: First we create an array of points and a pen color, then we create a pen and a window. The method for drawing the triangle is added as an anonymous function using the created window's `Paint.Add` method. This function is to be called as a response to a paint event, and takes a `PaintEventArgs` object, which includes the `System.Drawing.Graphics` object. Since this object will be related to a specific device, when `Paint` is called then we may call the `DrawLine` function to sequentially draw lines between our array of points. Finally, we hand the form to the event-loop, which as one of the earliest events will open the window and call the `Paint` function we have associated with the form.

...

Bibliography

- [1] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.
- [2] Programming Research Group. Specifications for the ibm mathematical formula translating system, fortran. Technical report, Applied Science Division, International Business Machines Corporation, 1954.
- [3] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.
- [4] X3: ASA Sectional Committee on Computers and Information Processing. American standard code for information interchange. Technical Report ASA X3.4-1963, American Standards Association (ASA), 1963. <http://worldpowersystems.com/projects/codes/X3.4-1963/>.
- [5] George Pólya. *How to solve it*. Princeton University Press, 1945.

Index

. [], 31
ReadKey, 114
ReadLine, 114
Read, 114
System.Console.ReadKey, 114
System.Console.ReadLine, 114
System.Console.Read, 114
System.Console.WriteLine, 114
System.Console.Write, 114
WriteLine, 114
Write, 114
abs, 171
acos, 171
asin, 171
atan2, 171
atan, 171
bignum, 23
bool, 19
byte[], 23
byte, 23
ceil, 171
char, 19
cosh, 171
cos, 171
decimal, 23
double, 23
eprintfn, 51
eprintf, 51
exn, 19
exp, 171
failwithf, 51
float32, 23
float, 19
floor, 171
fprintfn, 51
fprintf, 51
ignore, 51
int16, 23
int32, 23
int64, 23
int8, 23
int, 19
it, 19
log10, 171
log, 171
max, 171
min, 171
nativeint, 23
obj, 19
pown, 171
printfn, 51
printf, 48, 51
round, 171
sbyte, 23
sign, 171
single, 23
sinh, 171
sin, 171
sprintf, 51
sqrt, 171
stderr, 51, 114
stdin, 114
stdout, 51, 114
string, 19
tanh, 171
tan, 171
uint16, 23
uint32, 23
uint64, 23
uint8, 23
unativeint, 23
unit, 19

aliasing, 55
American Standard Code for Information Inter-
change, 154
and, 26
anonymous function, 45
array sequence expressions, 134
Array.toList, 85
ASCII, 154
ASCIIbetical order, 31, 155

base, 19, 153
Basic Latin block, 155
Basic Multilingual plane, 155
basic types, 19
binary, 153
binary number, 21
binary operator, 25
binary64, 153

- binding, 14
- bit, 21, 153
- black-box testing, 90
- block, 40
- blocks, 155
- boolean and, 172
- boolean or, 172
- branches, 67
- branching coverage, 92
- bug, 89
- byte, 153
- character, 21
- class, 24, 32
- code point, 21, 155
- compiled, 11
- computation expressions, 79, 84
- conditions, 67
- Cons, 81
- console, 11
- coverage, 92
- currying, 46
- debugging, 13, 90, 98
- decimal number, 19, 153
- decimal point, 20, 153
- Declarative programming, 8
- digit, 20, 153
- dot notation, 32
- double, 153
- downcasting, 24
- EBNF, 20, 159
- efficiency, 90
- encapsulate code, 42
- encapsulation, 46, 53
- environment, 99
- exception, 28
- exclusive or, 29
- executable file, 11
- expression, 14, 24
- expressions, 8
- Extended Backus-Naur Form, 20, 159
- Extensible Markup Language, 57
- file, 113
- floating point number, 20
- flushing, 117
- format string, 14
- fractional part, 20, 24
- function, 17
- function coverage, 92
- Functional programming, 8, 128
- functional programming, 8
- functionality, 89

- functions, 8
- generic function, 44
- hand tracing, 98
- Head, 81
- hexadecimal, 153
- hexadecimal number, 21
- HTML, 60
- Hyper Text Markup Language, 60
- IEEE 754 double precision floating-point format, 153
- Imperativ programming, 127
- Imperative programming, 8
- implementation file, 11
- infix notation, 25
- infix operator, 24
- integer, 20
- integer division, 28
- interactive, 11
- IsEmpty, 81
- Item, 81
- jagged arrays, 86
- keyword, 14
- Latin-1 Supplement block, 155
- Latin1, 155
- least significant bit, 153
- Length, 81
- length, 76
- lexeme, 17
- lexical scope, 16, 44
- lexically, 38
- lightweight syntax, 35, 38
- list, 79
- list sequence expression, 134
- List.Empty, 81
- List.toArray, 81
- List.toList, 81
- literal, 19
- literal type, 23
- machine code, 127
- maintainability, 90
- member, 24, 76
- method, 32
- mockup code, 98
- module elements, 147
- modules, 11
- most significant bit, 153
- Mutable data, 51
- mutually recursive, 70

- namespace, 24
- namespace pollution, 142
- NaN, 153
- nested scope, 40
- newline, 22
- not, 26
- not a number, 153

- obfuscation, 79
- object, 32
- Object oriented programming, 127
- Object-oriented programming, 8
- objects, 8
- octal, 153
- octal number, 21
- operand, 43
- operands, 25
- operator, 25, 43
- option type, 111
- or, 26
- overflow, 27

- pattern matching, 136, 143
- portability, 90
- precedence, 25
- prefix operator, 25
- Procedural programming, 127
- procedure, 46
- production rules, 159

- ragged multidimensional list, 84
- raise an exception, 106
- range expression, 80
- reals, 153
- recursive function, 68
- reference cells, 54
- reliability, 89
- remainder, 28
- rounding, 24
- run-time error, 29

- scientific notation, 20
- scope, 40
- script file, 11
- script-fragment, 17
- script-fragments, 11
- Seq.initInfinite, 134
- Seq.item, 131
- Seq.take, 131
- Seq.toArray, 134
- Seq.toList, 134
- side-effect, 85
- side-effects, 46, 54
- signature file, 11
- slicing, 85

- software testing, 90
- state, 8
- statement, 14
- statement coverage, 92
- statements, 8, 127
- states, 127
- stopping criterium, 69
- stream, 114
- string, 14, 22
- Structured programming, 8
- subnormals, 153

- Tail, 81
- tail-recursive, 69
- terminal symbols, 159
- tracing, 98
- truth table, 26
- tuple, 76
- type, 15, 19
- type declaration, 15
- type inference, 13, 15
- type safety, 43
- typecasting, 23

- unary operator, 25
- underflow, 27
- Unicode, 21
- unicode general category, 155
- Unicode Standard, 155
- Uniform Resource Identifiers, 121
- Uniform Resource Locator, 120
- unit of measure, 139
- unit testing, 90
- unit-less, 140
- unit-testing, 13
- upcasting, 24
- URI, 121
- URL, 120
- usability, 90
- UTF-16, 157
- UTF-8, 157

- variable, 51
- verbatim, 23

- white-box testing, 90, 92
- whitespace, 22
- whole part, 20, 24
- wild card, 38
- word, 153

- XML, 57
- xor, 29

- yield bang, 131