

# Learning to program with F#

Jon Sparring

Department of Computer Science,  
University of Copenhagen

July 19, 2017

# Contents

<b>1</b>	<b>Preface</b>	<b>4</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
2.1	How to learn to program . . . . .	5
2.2	How to solve problems . . . . .	6
2.3	Approaches to programming . . . . .	6
2.4	Why use F# . . . . .	7
2.5	How to read this book . . . . .	8
<b>3</b>	<b>Executing F# code</b>	<b>9</b>
3.1	Source code . . . . .	9
3.2	Executing programs . . . . .	9
<b>4</b>	<b>Quick-start guide</b>	<b>12</b>
<b>5</b>	<b>Using F# as a calculator</b>	<b>17</b>
5.1	Literals and basic types . . . . .	17
5.2	Operators on basic types . . . . .	22
5.3	Boolean arithmetic . . . . .	24
5.4	Integer arithmetic . . . . .	25
5.5	Floating point arithmetic . . . . .	27
5.6	Char and string arithmetic . . . . .	29
5.7	Programming intermezzo . . . . .	30
<b>6</b>	<b>Constants, functions, and variables</b>	<b>32</b>

6.1	Values . . . . .	35
6.2	Non-recursive functions . . . . .	40
6.3	User-defined operators . . . . .	44
6.4	The Printf function . . . . .	46
6.5	Variables . . . . .	49
<b>7</b>	<b>In-code documentation</b>	<b>55</b>
<b>8</b>	<b>Controlling program flow</b>	<b>60</b>
8.1	For and while loops . . . . .	60
8.2	Conditional expressions . . . . .	64
8.3	Recursive functions . . . . .	66
8.4	Programming intermezzo . . . . .	69
<b>9</b>	<b>Modules and Namespaces</b>	<b>73</b>
<b>10</b>	<b>Testing programs</b>	<b>82</b>
10.1	White-box testing . . . . .	85
10.2	Black-box testing . . . . .	88
10.3	Debugging by tracing . . . . .	91
<b>11</b>	<b>Ordered series of data</b>	<b>98</b>
11.1	Tuples . . . . .	99
11.2	Lists . . . . .	102
11.3	Arrays . . . . .	107
<b>12</b>	<b>Patterns</b>	<b>112</b>
12.1	Pattern matching . . . . .	112
<b>13</b>	<b>Handling Errors and Exceptions</b>	<b>115</b>
<b>14</b>	<b>Input and Output</b>	<b>123</b>
14.1	Interacting with the console . . . . .	124
14.2	Storing and retrieving data from a file . . . . .	125

14.3 Working with files and directories. . . . .	130
14.4 Reading from the internet . . . . .	130
14.5 Programming intermezzo . . . . .	131
<b>15 Graphical User Interfaces</b>	<b>134</b>
15.1 Drawing primitives in Windows . . . . .	134
15.2 Programming intermezzo . . . . .	147
15.3 Events, Controls, and Panels . . . . .	149
<b>16 Object-oriented programming</b>	<b>174</b>
<b>A Number systems on the computer</b>	<b>178</b>
A.1 Binary numbers . . . . .	180
A.2 IEEE 754 floating point standard . . . . .	180
<b>B Commonly used character sets</b>	<b>181</b>
B.1 ASCII . . . . .	181
B.2 ISO/IEC 8859 . . . . .	182
B.3 Unicode . . . . .	182
<b>C Common Language Infrastructure</b>	<b>186</b>
<b>D A brief introduction to Extended Backus-Naur Form</b>	<b>188</b>
<b>E Fb</b>	<b>192</b>
<b>F To Dos</b>	<b>197</b>
<b>Bibliography</b>	<b>198</b>
<b>Index</b>	<b>199</b>

## Chapter 16

# Object-oriented programming

*Object-oriented programming* is a programming paradigm that focusses on *objects* such as a person, place, thing, event, and concept relevant for the problem. Objects may contain data and code, which in the object-oriented paradigm are called *attributeds* and *methods*. Object-oriented programming is an extension of data types, in the sense that objects contains both data and functions in a similar manner as a module, but object-oriented programming emphasizes the semantic unity of the data and functions. Thus, objects are *models* of real world entities, and object-oriented programming leads to a particular style of programming analysis and design called *object-oriented analysis and design*.

An object is a variable of a class type. A class is defined using the “**type**” keyword, and there are allways parantheses after the class name. Consider the following problem.

### Problem 16.1:

A complex number is a pair of real numbers called the real and the imaginary part and a set of operators. In particular, addition of two complex numbers is the the addition of their real parts and of their imaginary parts. Define a class for complex numbers including the addition operator.

A solution to this problem is as follows.

### Listing 16.1, complex.fsx:

A class implementing complex numbers and the addition operator.

```
1 type complex(aReal, anImaginary) =
2     member this.re = aReal
3     member this.im = anImaginary
4     member this.add (a : complex) : complex =
5         new complex(this.re + a.re, this.im + a.im)
6
7 let x = new complex(1.0, 2.0)
8 let y = new complex(2.5, -1.2)
9 let z = x.add(y)
10 printfn "(%A, %A) + (%A, %A) = (%A, %A)" x.re x.im y.re y.im z.re z.im

```

---

```
1 (1.0, 2.0) + (2.5, -1.2) = (3.5, 0.8)

```

Things to remember:

- Object-oriented programming
- objects
- attributeds
- methods
- models
- object-oriented analysis
- object-oriented design

- upcast and downcast “`upcast`”, “`:`””, “`downcast`”, “`:`?””
- boxing `(box 5) :?> int;;`, see Spec-4.0 chapter 18.2.6.
- obj type Spec-4.0 chapter 18.1
- boxing Spec-4.0 Section 18.2.6

1

---

<sup>1</sup>Todo: In object oriented programming: functions and data are combined. Contrast the Anemic Domain Model (<https://www.martinfowler.com/bliki/AnemicDomainModel.html>)

# Appendix F

## To Dos

- Remove EBNF from main body of the text, possibly extend the appendix
- Add appendix on regular expressions
- Add Torben's notes on functional programming
- Rewrite list chapter (add sequences?)
- Add a chapter comparing the 3 paradigms
- Write structured programming part
- Write chapter on pattern matching (if not already in Torben's notes)
- Add something on piping (if not already in Torben's notes)
- Add abstraction of computer: places  $\leftrightarrow$  memory/disk. Mutable objects are abstractions of places <https://www.infoq.com/presentations/Value-Values>. Facts does not rime with set and get.
- Hickey: Difference between syntax and semantics. Values or locations, add a good figure. Functional programming: All values are freely shareable.
- something about organising stuff: <https://fsharpforfunandprofit.com/posts/organizing-functions/>, <https://fsharpforfunandprofit.com/posts/recipe-part3/>

# Bibliography

- [1] M. Auer, J. Poelz, A. Fuernweger, L. Meyer, and T. Tschurtschenthaler. Umlet, free uml tool for fast uml diagrams. <http://www.umlet.com>, Present version 14.2, version 1.0 was released June 21, 2002.
- [2] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.
- [3] European Computer Manufacturers Association (ECMA). Standard ecma-335, common language infrastructure (cli). <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [4] International Organization for Standardization. Iso/iec 23271:2012, common language infrastructure (cli). <https://www.iso.org/standard/58046.html>.
- [5] Object Management Group. Uml version 2.0. <http://www.omg.org/spec/UML/2.0/>.
- [6] Programming Research Group. Specifications for the ibm mathematical formula translating system, fortran. Technical report, Applied Science Division, International Business Machines Corporation, 1954.
- [7] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.
- [8] 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/>.
- [9] George Pólya. *How to solve it*. Princeton University Press, 1945.



# Index

#r directive, 78  
., 29  
Paint.Add, 138  
ReadKey, 124  
ReadLine, 124  
Read, 124  
System.Console.ReadKey, 124  
System.Console.ReadLine, 124  
System.Console.Read, 124  
System.Console.WriteLine, 124  
System.Console.Write, 124  
System.Drawing.Color, 136  
System.Drawing.Graphics.DrawLine, 138  
System.Drawing.PointToClient, 138  
System.Drawing.PointToScreen, 138  
System.Drawing, 134  
System.Windows.Forms.Button, 153  
System.Windows.Forms.FlowLayoutPanel, 154  
System.Windows.Forms.Label, 153  
System.Windows.Forms.Panel, 153  
System.Windows.Forms.TextBox, 153  
System.Windows.Forms, 134  
WriteLine, 124  
Write, 124  
bignum, 21  
bool, 17  
byte[], 21  
byte, 21  
char, 17  
decimal, 21  
double, 21  
eprintfn, 49  
eprintf, 49  
exn, 17  
failwithf, 49  
float32, 21  
float, 17  
fprintfn, 49  
fprintf, 49  
ignore, 49  
int16, 21  
int32, 21  
int64, 21  
int8, 21  
int, 17  
it, 17  
nativeint, 21  
obj, 17  
printfn, 49  
printf, 46, 49  
sbyte, 21  
single, 21  
sprintf, 49  
stderr, 49, 124  
stdin, 124  
stdout, 49, 124  
string, 17  
uint16, 21  
uint32, 21  
uint64, 21  
uint8, 21  
unativeint, 21  
unit, 17  
accessors, 136  
aliasing, 53  
American Standard Code for Information Inter-  
change, 181  
and, 24  
anonymous function, 43  
Array.toList, 108  
ASCII, 181  
ASCIIbetical order, 29, 182  
assembly, 186  
attributeds, 174  
base, 17, 180  
Basic Latin block, 182  
Basic Multilingual plane, 182  
basic types, 17  
binary, 180  
binary number, 19  
binary operator, 23  
binary64, 180  
binding, 12  
bit, 19, 180  
black-box testing, 83  
block, 38  
blocks, 182  
branches, 65  
branching coverage, 85

- bug, 82
- byte, 180
- call-back function, 138
- character, 19
- CIL, 186
- class, 22, 30, 175
- class diagram, 177
- CLI, 134, 186
- client coordinates, 138
- CLR, 186
- code point, 19, 182
- Command Line Interface, 186
- command-line interface, 134
- Common Intermediate Language, 186
- Common Language Infrastructure, 186
- Common Language Runtime, 186
- Common Type System, 186
- compiled, 9
- computation expressions, 102, 107
- conditions, 65
- Cons, 104
- console, 9
- Controls, 152
- coverage, 85
- CTS, 186
- currying, 44
- debugging, 11, 83, 91
- decimal number, 17, 180
- decimal point, 18, 180
- Declarative programming, 7
- digit, 18, 180
- dot notation, 30
- double, 180
- downcasting, 22
- EBNF, 18, 188
- efficiency, 83
- encapsulate code, 40
- encapsulation, 44, 51
- environment, 92
- event, 134
- event driven programming, 134
- event-loop, 135
- exception, 26
- exclusive or, 27
- executable file, 9
- expression, 12, 22
- expressions, 7
- Extended Backus-Naur Form, 18, 188
- Extensible Markup Language, 55
- file, 123
- floating point number, 18
- flushing, 127
- format string, 12
- forms, 134
- fractional part, 18, 22
- function, 15
- function coverage, 85
- Functional programming, 7
- functional programming, 7
- functionality, 82
- functions, 7
- GDI+, 134
- generic function, 42
- graphical user interface, 134
- GUI, 134
- hand tracing, 91
- Head, 104
- hexadecimal, 180
- hexadecimal number, 19
- how, 174
- HTML, 58
- Hyper Text Markup Language, 58
- IEEE 754 double precision floating-point format, 180
- Imperative programming, 7
- implementation file, 9
- implementation files, 79
- infix notation, 23
- infix operator, 22
- instance, 175
- integer, 18
- integer division, 26
- interactive, 9
- IsEmpty, 104
- Item, 104
- jagged arrays, 109
- just-in-time, 186
- keyword, 12
- Latin-1 Supplement block, 182
- Latin1, 182
- least significant bit, 180
- Length, 104
- length, 99
- lexeme, 15
- lexical scope, 14, 42
- lexically, 36
- library, 73
- lightweight syntax, 33, 36
- list, 102
- List.Empty, 104

- List.toArray, 104
- List.toList, 104
- literal, 17
- literal type, 21
- maintainability, 83
- manifest, 186
- member, 22, 99
- Metadata, 186
- method, 30
- methods, 135, 174
- mockup code, 91
- models, 174
- module, 73, 186
- modules, 9
- most significant bit, 180
- Mutable data, 49
- mutually recursive, 68
- namespace, 22, 76
- namespace pollution, 75
- NaN, 180
- nested scope, 38
- newline, 20
- not, 24
- not a number, 180
- nouns, 175
- obfuscation, 102
- object, 30, 175
- object-oriented analysis, 174
- object-oriented design, 174
- Object-oriented programming, 174
- Object-oriented programming, 7
- objects, 7, 174
- octal, 180
- octal number, 19
- operand, 41
- operands, 23
- operator, 23, 41
- option type, 121
- or, 24
- overflow, 25
- Panels, 153
- pattern matching, 112
- portability, 83
- precedence, 23
- prefix operator, 23
- problem statement, 174
- procedure, 44
- production rules, 188
- properties, 135
- raise an exception, 116
- range expression, 103
- reals, 180
- recursive function, 66
- reference cells, 52
- reliability, 82
- remainder, 26
- rounding, 22
- run-time error, 27
- scientific notation, 18
- scope, 38
- screen coordinates, 138
- script file, 9
- script files, 79
- script-fragment, 15
- script-fragments, 9
- side-effect, 108
- side-effects, 44, 52
- signature file, 9
- signature files, 79
- slicing, 108
- software testing, 83
- state, 7
- statement, 12
- statement coverage, 85
- statements, 7
- static, 175
- stopping criterium, 67
- stream, 124
- string, 12, 20
- Structured programming, 7, 141
- subnormals, 180
- Tail, 104
- tail-recursive, 67
- terminal symbols, 188
- tracing, 91
- truth table, 24
- tuple, 99
- type, 13, 17
- type declaration, 13
- type inference, 11, 13
- type safety, 41
- typecasting, 21
- UML, 177
- unary operator, 23
- underflow, 25
- Unicode, 19
- unicode general category, 182
- Unicode Standard, 182
- Uniform Resource Identifiers, 131
- Uniform Resource Locator, 130
- unit testing, 83
- ragged multidimensional list, 107

unit-testing, 11  
Universal Modelling Language 2, 177  
upcasting, 22  
URI, 131  
URL, 130  
usability, 83  
UTF-16, 184  
UTF-8, 184  
  
variable, 49  
verbatim, 21  
verbs, 176  
VES, 186  
Virtual Execution System, 186  
  
what, 174  
white-box testing, 83, 85  
whitespace, 20  
whole part, 18, 22  
wild card, 36  
Windows Graphics Device Interface, 134  
WinForms, 134  
word, 180  
  
XML, 55  
xor, 27