

# Learning to program with F#

Jon Sparring

August 5, 2016

# Contents

<b>1</b>	<b>Preface</b>	<b>4</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
<b>I</b>	<b>F# basics</b>	<b>7</b>
<b>3</b>	<b>Executing F# code</b>	<b>8</b>
3.1	Source code . . . . .	8
3.2	Executing programs . . . . .	8
<b>4</b>	<b>Quick-start guide</b>	<b>10</b>
<b>5</b>	<b>Using F# as a calculator</b>	<b>14</b>
5.1	Literals and basic types . . . . .	14
5.2	Operators on basic types . . . . .	19
5.3	Boolean arithmetic . . . . .	24
5.4	Integer arithmetic . . . . .	25
5.5	Floating point arithmetic . . . . .	26
5.6	Char and string arithmetic . . . . .	27
<b>6</b>	<b>Constants, functions, and variables</b>	<b>30</b>
6.1	Values . . . . .	32
6.2	Non-recursive functions . . . . .	35
6.3	User-defined operators . . . . .	38
6.4	The Printf function . . . . .	40
6.5	Variables . . . . .	42
<b>7</b>	<b>In-code documentation</b>	<b>46</b>
<b>8</b>	<b>Controlling program flow</b>	<b>50</b>
8.1	For and while loops . . . . .	50
8.2	Conditional expressions . . . . .	53
8.2.1	Programming intermezzo . . . . .	54
8.3	Pattern matching . . . . .	55
8.4	Recursive functions . . . . .	57
<b>9</b>	<b>Ordered series of data</b>	<b>59</b>
9.1	Tuples . . . . .	60
9.2	Lists . . . . .	62
9.3	Arrays . . . . .	65
9.4	Sequences . . . . .	70

<b>10 Testing programs</b>	<b>74</b>
10.1 White-box testing . . . . .	76
10.2 Back-box testing . . . . .	78
 <b>II Imperative programming</b>	 <b>81</b>
<b>11 Exceptions</b>	<b>83</b>
<b>12 Input/Output</b>	<b>84</b>
12.1 Console I/O . . . . .	84
12.2 File I/O . . . . .	84
<b>13 Graphical User Interfaces</b>	<b>86</b>
<b>14 Imperative programming</b>	<b>87</b>
14.1 Introduction . . . . .	87
14.2 Generating random texts . . . . .	87
14.2.1 0'th order statistics . . . . .	87
14.2.2 1'th order statistics . . . . .	89
 <b>III Declarative programming</b>	 <b>92</b>
<b>15 Types and measures</b>	<b>93</b>
15.1 Unit of Measure . . . . .	93
<b>16 Functional programming</b>	<b>96</b>
 <b>IV Structured programming</b>	 <b>97</b>
<b>17 Namespaces and Modules</b>	<b>98</b>
<b>18 Object-oriented programming</b>	<b>100</b>
 <b>V Appendix</b>	 <b>101</b>
<b>A Number systems on the computer</b>	<b>102</b>
A.1 Binary numbers . . . . .	102
A.2 IEEE 754 floating point standard . . . . .	102
<b>B Commonly used character sets</b>	<b>106</b>
B.1 ASCII . . . . .	106
B.2 ISO/IEC 8859 . . . . .	106
B.3 Unicode . . . . .	107
<b>C A brief introduction to Extended Backus-Naur Form</b>	<b>110</b>
<b>D Language Details</b>	<b>113</b>
<b>E The Collection</b>	<b>115</b>
E.1 <code>System.String</code> . . . . .	115
E.2 List, arrays, and sequences . . . . .	120
E.3 Mutable Collections . . . . .	122
E.3.1 Mutable lists . . . . .	122

E.3.2	Stacks . . . . .	122
E.3.3	Queues . . . . .	122
E.3.4	Sets and dictionaries . . . . .	123
<b>Bibliography</b>		<b>124</b>
<b>Index</b>		<b>125</b>

## Chapter 11

# Exceptions

Exceptions are runtime errors, which may be handled gracefully by F#. Exceptions are handled by the `try` keyword both in expressions and computation expressions,

```
expr = ...
| "try" expr "with" rules
| "try" expr "finally" expr
| ...
comp-expr = ...
| "try" comp-expr "with" comp-rules
| "try" comp-expr "finally" expr
| ...
```

As an example is integer division by zero,

```
let div enum denom =
    try
        enum / denom
    with
        | :? System.DivideByZeroException -> System.Int32.MaxValue

printfn "3 / 1 = %d" (div 3 1)
printfn "3 / 0 = %d" (div 3 0)
```

```
3 / 1 = 3
3 / 0 = 2147483647
```

Listing 11.1: exceptionDivByZero.fsx - A division by zero is caught and a default value is returned.

```
> let div enum denom =
-   try
-       Some (enum / denom)
-   with
-       | :? System.DivideByZeroException -> None;;

val div : enum:int -> denom:int -> int option

>
- let a = div 3 1;;

val a : int option = Some 3

> let b = div 3 0;;
```

Attribute	Description
<code>System.ArithmeticException</code>	Failed arithmetic operation.
<code>System.ArrayTypeMismatchException</code>	Failed attempt to store an element in an array failed because of type mismatch.
<code>System.DivideByZeroException</code>	Failed due to division by zero.
<code>System.IndexOutOfRangeException</code>	Failed to access an element in an array because the index is less than zero or equal or greater than the length of the array.
<code>System.InvalidCastException</code>	Failed to explicitly convert a base type or interface to a derived type at run time.
<code>System.NullReferenceException</code>	Failed use of a <code>null</code> reference was used, since it required the referenced object.
<code>System.OutOfMemoryException</code>	Failed to use <code>new</code> to allocate memory.
<code>System.OverflowException</code>	Failed arithmetic operation in a checked context which caused an overflow.
<code>System.StackOverflowException</code>	Failed use of the internal stack caused by too many pending method calls, e.g., from deep or unbounded recursion.
<code>System.TypeInitializationException</code>	Failed initialization of code for a type, which was not caught.

Table 11.1: Built-in exceptions.

```
val b : int option = None
```

**Listing 11.2:** `fsharp`, Option types can be used, when the value in case of exceptions is unclear.

Exceptions are a basic-type called `exn`, and F# has a number of built-in, see Table 11.1, and the user may construct new exceptions using the syntax,

```
exception-defn = [attributes] "exception" union-type-case-data [attributes]
               exception ident "=" long-ident
```

Exceptions are raised with the keywords `failwith`, `invalidArg`, `raise`, and `raise`

```
exception DontLikeFive of string

let div enum denom =
    if denom = 5 then
        raise (DontLikeFive "5 sucks")
    try
        Some (enum / denom)
    with
        | :? System.DivideByZeroException -> None

printfn "3 / 1 = %A" (div 3 1)
printfn "3 / 0 = %A" (div 3 0)
printfn "3 / 5 = %A" (div 3 5)
```

```
3 / 1 = Some 3
3 / 0 = <null>
FSI_0001+DontLikeFive: Exception of type 'FSI_0001+DontLikeFive' was thrown.
    at FSI_0001.div (Int32 enum, Int32 denom) <0x7063e60 + 0x0015f> in <filename
        unknown>:0
    at <StartupCode$FSI_0001>.$FSI_0001.main@ () <0x70632e0 + 0x001f7> in <
        filename unknown>:0
```

```
at (wrapper managed-to-native) System.Reflection.MonoMethod:InternalInvoke (
    System.Reflection.MonoMethod,object,object[],System.Exception&)
at System.Reflection.MonoMethod.Invoke (System.Object obj, BindingFlags
    invokeAttr, System.Reflection.Binder binder, System.Object[] parameters,
    System.Globalization.CultureInfo culture) <0x227c270 + 0x000a1> in <
    filename unknown>:0
Stopped due to error
```

Listing 11.3: exceptionDefinition.fsx - A user-defined exception is raised but not caught by outer construct.

Remember

- exn type Spec-4.0 Chapter 18.1
- Spec-4.0 Section 18.2.8

...

# 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

. [], 28  
abs, 20  
acos, 20  
asin, 20  
atan2, 20  
atan, 20  
bignum, 17  
byte[], 17  
byte, 17  
ceil, 20  
char, 14  
cosh, 20  
cos, 20  
decimal, 17  
double, 17  
eprintfn, 41  
eprintf, 41  
exn, 14  
exp, 20  
failwithf, 41  
float32, 17  
float, 14  
floor, 20  
fprintfn, 41  
fprintf, 41  
ignore, 41  
int16, 17  
int32, 17  
int64, 17  
int8, 17  
int, 14  
it, 14  
log10, 20  
log, 20  
max, 20  
min, 20  
nativeint, 17  
obj, 14  
pown, 20  
printfn, 41  
printf, 40, 41  
round, 20  
sbyte, 17  
sign, 20  
single, 17

sinh, 20  
sin, 20  
sprintf, 41  
sqrt, 20  
stderr, 41  
stdout, 41  
string, 14  
tanh, 20  
tan, 20  
uint16, 17  
uint32, 17  
uint64, 17  
uint8, 17  
unativeint, 17  
unit, 14

American Standard Code for Information Inter-  
change, 106

and, 24  
anonymous function, 37  
array sequence expressions, 73  
Array.toArray, 68  
Array.toList, 68  
ASCII, 106  
ASCIIbetical order, 27, 106

base, 14, 102  
Basic Latin block, 107  
Basic Multilingual plane, 107  
basic types, 14  
binary, 102  
binary number, 16  
binary operator, 20  
binary64, 102  
binding, 10  
bit, 16, 102  
black-box testing, 75  
block, 34  
blocks, 107  
boolean and, 23  
boolean or, 23  
branches, 54  
branching coverage, 76  
bug, 74  
byte, 102

- character, 16
- class, 19, 28
- code point, 16, 107
- compiled, 8
- computation expressions, 62, 65
- conditions, 54
- Cons, 65
- console, 8
- coverage, 76
- currying, 38
  
- debugging, 9, 75
- decimal number, 14, 102
- decimal point, 14, 102
- Declarative programming, 5
- digit, 14, 102
- dot notation, 28
- double, 102
- downcasting, 19
  
- EBNF, 14, 110
- efficiency, 74
- encapsulate code, 35
- encapsulation, 38, 43
- exception, 26
- exclusive or, 26
- executable file, 8
- expression, 10, 19
- expressions, 6
- Extended Backus-Naur Form, 14, 110
- Extensible Markup Language, 46
  
- floating point number, 14
- format string, 10
- fractional part, 14, 19
- function, 12
- function coverage, 76
- Functional programming, 6, 87
- functionality, 74
- functions, 6
  
- generic function, 36
  
- Head, 65
- hexadecimal, 102
- hexadecimal number, 16
- HTML, 48
- Hyper Text Markup Language, 48
  
- IEEE 754 double precision floating-point format, 102
- Imperativ programming, 87
- Imperative programming, 5
- implementation file, 8
- infix notation, 23
- infix operator, 19
- integer division, 25
- integer number, 14
- interactive, 8
- IsEmpty, 65
- Item, 65
  
- jagged arrays, 68
  
- keyword, 10
  
- Latin-1 Supplement block, 107
- Latin1, 106
- least significant bit, 102
- Length, 65
- length, 60
- lexeme, 12
- lexical scope, 12, 36
- lexically, 32
- lightweight syntax, 30, 32
- list, 62
- list sequence expression, 73
- List.Empty, 65
- List.toArray, 65
- List.toList, 65
- literal, 14
- literal type, 17
  
- machine code, 87
- maintainability, 75
- member, 19, 60
- method, 28
- module elements, 98
- modules, 8
- most significant bit, 102
- Mutable data, 42
  
- namespace, 19
- namespace pollution, 94
- NaN, 104
- nested scope, 12, 34
- newline, 17
- not, 24
- not a number, 104
  
- obfuscation, 62
- object, 28
- Object oriented programming, 87
- Object-oriented programming, 6
- objects, 6
- octal, 102
- octal number, 16
- operand, 35
- operands, 20
- operator, 20, 23, 35

- or, 24
- overflow, 25
- overshadow, 12
- overshadows, 34
- pattern matching, 55, 64
- portability, 75
- precedence, 23
- prefix operator, 20
- Procedural programming, 87
- procedure, 38
- production rules, 110
- ragged multidimensional list, 65
- range expression, 63
- reals, 102
- recursive function, 57
- reference cells, 44
- reliability, 74
- remainder, 25
- rounding, 19
- run-time error, 26
- scientific notation, 16
- scope, 12, 33
- script file, 8
- script-fragments, 8
- Seq.initInfinite, 73
- Seq.item, 71
- Seq.take, 71
- Seq.toArray, 73
- Seq.toList, 73
- side-effect, 67
- side-effects, 38, 44
- signature file, 8
- slicing, 68
- software testing, 75
- state, 5
- statement, 10
- statement coverage, 76
- statements, 5, 87
- states, 87
- stopping criterium, 57
- string, 10, 16
- Structured programming, 6
- subnormals, 104
- Tail, 65
- tail-recursive, 57
- terminal symbols, 110
- truth table, 24
- tuple, 60
- type, 10, 14
- type casting, 18
- type declaration, 10
- type inference, 9, 10
- type safety, 36
- unary operator, 20
- underflow, 25
- Unicode, 16
- unicode general category, 107
- Unicode Standard, 107
- unit of measure, 93
- unit testing, 75
- unit-less, 94
- unit-testing, 9
- upcasting, 19
- usability, 74
- UTF-16, 107
- UTF-8, 107
- variable, 42
- verbatim, 18
- white-box testing, 75, 76
- whitespace, 17
- whole part, 14, 19
- word, 102
- XML, 46
- xor, 26
- yield bang, 71