Real-World

Functional Programming

With examples in F# and C#

Tomas Petricek
with Jon Skeet

Foreword by Mads Torgersen

MANNING

contents

foreword *xvii*
preface *xix*
acknowledgments *xxi*
about this book *xxiv*
about the cover illustration *xxx*

PART 1 LEARNING TO THINK FUNCTIONALLY 1

1 ***Thinking differently 3***

1.1 What is functional programming? 4

1.2 The path to real-world functional programming 6

Functional languages 6 ▪ Functional programming on the .NET platform 7

1.3 Being productive with functional programming 7

The functional paradigm 7 ▪ Declarative programming style 8 Understanding what a program does 9 ▪ Concurrency-friendly application design 10 ▪ How functional style shapes your code 11

1.4 Functional programming by example 12

Expressing intentions using declarative style 13 ▪ Understanding code using immutability 17 ▪ Writing efficient parallel programs 20

1.5 Introducing F# 21

Hello world in F# 22 ▪ From simplicity to the real world 24

1.6 Summary 27

2 ***Core concepts in functional programming 29***

2.1 The foundation of functional programming 31

2.2 Evaluation of functional programs 32

Working with immutable values 33 ▪ Using immutable data structures 33 ▪ Changing program state using recursion 34 Using expressions instead of statements 35 ▪ Computation by calculation 37

2.3 Writing declarative code 39

Functions as values 39 ▪ Higher-order functions 41

2.4 Functional types and values 45

Type inference in C# and F# 46 ▪ Introducing the discriminated union type 47 ▪ Pattern matching 48 ▪ Compile-time program checking 51

2.5 Summary 52

3 ***Meet tuples, lists, and functions in F# and C# 54***

3.1 Value and function declarations 55

Value declarations and scope 55 ▪ Function declarations 57 Declaring mutable values 59

3.2 Using immutable data structures 60

Introducing tuple type 60 ▪ Implementing a tuple type in C# 63 Calculating with tuples 64 ▪ Pattern matching with tuples 66

3.3 Lists and recursion 68

Recursive computations 68 ▪ Introducing functional lists 69 Functional lists in C# 72 ▪ Functional list processing 74

3.4 Using functions as values 75

Processing lists of numbers 76 ▪ Benefits of parameterized functions 79

3.5 Summary 79

4 ***Exploring F# and .NET libraries by example 81***

4.1 Drawing pie charts in F# 82

4.2 Writing and testing code in FSI 83

Loading and parsing data 83 ▪ Calculating with the data 86

4.3 Creating a console application 89

4.4 Creating a Windows Forms application 92

Creating the user interface 92 ▪ Drawing graphics 95 Creating the Windows application 101

4.5 Summary 102

PART 2 FUNDAMENTAL FUNCTIONAL TECHNIQUES 105

5 ***Using functional values locally 107***

5.1 What are values? 108

Primitive types, value types, and objects 108 ▪ Recognizing values and data 109

5.2 Multiple values 109

Multiple values in F# and C# 109 ▪ Tuple type and value constructors 111 ▪ Using tuples compositionally 112

5.3 Alternative values 114

Discriminated unions in F# 115 ▪ Working with alternatives 116 Adding types vs. functions 118 ▪ Using the option type in F# 120

5.4 Generic values 122

Implementing the option type in C# 122 ▪ Generic option type in F# 125 ▪ Type inference for values 127 ▪ Writing generic functions 129

5.5 Function values 130

Lambda functions 132 ▪ The function type 135 ▪ Functions of multiple arguments 137

5.6 Summary 140

6 ***Processing values using higher-order functions 142***

6.1 Generic higher-order functions 143

Writing generic functions in F# 144 ▪ Custom operators 145

6.2 Working with tuples 147

Working with tuples using functions 147 ▪ Methods for working with tuples in C# 150

6.3 Working with schedules 151

Processing a list of schedules 152 ▪ Processing schedules in C# 153

6.4 Working with the option type 154

Using the map function 155 ▪ Using the bind function 155 Evaluating the example step-by-step 156 ▪ Implementing operations for the option type 158

6.5 Working with functions 160

Function composition 160 ▪ Function composition in C# 162

6.6 Type inference 163

Type inference for function calls in F# 163 ▪ Automatic generalization 164

6.7 Working with lists 165

Implementing list in F# 165 ▪ Understanding type signatures of list functions 166 ▪ Implementing list functions 170

6.8 Common processing language 173

Mapping, filtering, and folding 173 ▪ The bind operation for lists 174

6.9 Summary 175

7 ***Designing data-centric programs 177***

7.1 Functional data structures 178

Using the F# record type 179 ▪ Functional data structures in C# 181

7.2 Flat document representation 182

Drawing elements 183 ▪ Displaying a drawing on a form 184

7.3 Structured document representation 187

Converting representations 188 ▪ XML document representation 191

7.4 Writing operations 194

Updating using a map operation 195 ▪ Calculating using an aggregate operation 198

7.5 Object-oriented representations 199

Representing data with structural patterns 200 ▪ Adding functions using the visitor pattern 202

7.6 Summary 204

8 ***Designing behavior-centric programs 205***

8.1 Using collections of behaviors 206

Representing behaviors as objects 206 ▪ Representing behaviors as functions in C# 207 ▪ Using collections of functions in C# 208 ▪ Using lists of functions in F# 209

8.2 Idioms for working with functions 211

The strategy design pattern 212 ▪ The command design pattern 213 ▪ Capturing state using closures in F# 215

8.3 Working with composed behaviors 219

Records of functions 219 ▪ Building composed behaviors 221 Further evolution of F# code 222

8.4 Combining data and behaviors 223

Decision trees 223 ▪ Decision trees in F# 224 ▪ Decision trees in C# 227

8.5 Summary 229

PART 3 ADVANCED F# PROGRAMMING TECHNIQUES 231

9 ***Turning values into F# object types with members 233***

9.1 Improving data-centric applications 234

Adding members to F# types 235 ▪ Appending members using type extensions 238

9.2 Improving behavior-centric applications 240

Using records of functions 240 ▪ Using interface object types 241

9.3 Working with .NET interfaces 243

Using .NET collections 244 ▪ Cleaning resources using IDisposable 245

9.4 Concrete object types 248

Functional and imperative classes 249 ▪ Implementing interfaces and casting 251

9.5 Using F# libraries from C# 255

Working with records and members 256 ▪ Working with values and delegates 258

9.6 Summary 259

10 ***Efficiency of data structures 260***

10.1 Optimizing functions 261

Avoiding stack overflows with tail recursion 261 ▪ Caching results using memoization 266

10.2 Working with large collections 271

Avoiding stack overflows with tail recursion (again!) 271 Processing lists efficiently 273 ▪ Working with arrays 275

10.3 Introducing continuations 279

What makes tree processing tricky? 279 ▪ Writing code using continuations 281

10.4 Summary 283

11 *Refactoring and testing functional programs 285*

11.1 Refactoring functional programs 286

Reusing common code blocks 287 ▪ Tracking dependencies and side effects 289

11.2 Testing functional code 292

From the interactive shell to unit tests 293 ▪ Writing tests using structural equality 296 ▪ Testing composed functionality 299

11.3 Refactoring the evaluation order 300

Different evaluation strategies 301 ▪ Comparing evaluation strategies 302 ▪ Simulating lazy evaluation using functions 303 Lazy values in F# 304 ▪ Implementing lazy values for C# 306

11.4 Using lazy values in practice 307

Introducing infinite lists 308 ▪ Caching values in a photo browser 310

11.5 Summary 313

12 *Sequence expressions and alternative workflows 314*

12.1 Generating sequences 315

Using higher-order functions 316 ▪ Using iterators in C# 316 Using F# sequence expressions 317

12.2 Mastering sequence expressions 320

Recursive sequence expressions 320 ▪ Using infinite sequences 322

12.3 Processing sequences 325

Transforming sequences with iterators 326 ▪ Filtering and projection 327 ▪ Flattening projections 329

12.4 Introducing alternative workflows 334

Customizing query expressions 335 ▪ Customizing the F# language 336

12.5 First steps in custom computations 338

Declaring the computation type 338 ▪ Writing the computations 339 ▪ Implementing a computation builder in F# 340 ▪ Implementing query operators in C# 342

12.6 Implementing computation expressions for options 343

12.7 Augmenting computations with logging 346

Creating the logging computation 346 ▪ Creating the logging computation 347 ▪ Refactoring using computation expressions 349

12.8 Summary 350

PART 4 APPLIED FUNCTIONAL PROGRAMMING.................351

13 *Asynchronous and data-driven programming 353*

13.1 Asynchronous workflows 354

Why do asynchronous workflows matter? 354 ▪ Downloading web pages asynchronously 355 ▪ Understanding how workflows work 358 ▪ Creating primitive workflows 361

13.2 Connecting to the World Bank 362

Accessing the World Bank data 363 ▪ Recovering from failures 365

13.3 Exploring and obtaining the data 366

Implementing XML helper functions 366 ▪ Extracting region codes 368 ▪ Obtaining the indicators 369

13.4 Gathering information from the data 372

Reading values 372 ▪ Formatting data using units of measure 374 ▪ Gathering statistics about regions 377

13.5 Visualizing data using Excel 378

Writing data to Excel 378 ▪ Displaying data in an Excel chart 381

13.6 Summary 382

14 *Writing parallel functional programs 383*

14.1 Understanding different parallelization techniques 384

Parallelizing islands of imperative code 385 ▪ Declarative data parallelism 386 ▪ Task-based parallelism 390

14.2 Running graphical effects in parallel 395

Calculating with colors in F# 395 ▪ Implementing and running color filters 396 ▪ Designing the main application 399 Creating and running effects 401 ▪ Parallelizing the application 404 ▪ Implementing a blur effect 406

14.3 Creating a parallel simulation 408

Representing the simulated world 409 ▪ Designing simulation operations 411 ▪ Implementing helper functions 413 Implementing smart animals and predators 415 ▪ Running the simulation in parallel 417

14.4 Summary 419

15 ***Creating composable functional libraries 420***

15.1 Approaches for composable design 421

Composing animations from symbols 421 ▪ Giving meaning to symbols 422 ▪ Composing values 424 ▪ Composing functions and objects 425

15.2 Creating animated values 428

Introducing functional animations 428 ▪ Introducing behaviors 429 ▪ Creating simple behaviors in C# 431 Creating simple behaviors in F# 433

15.3 Writing computations with behaviors 434

Reading values 434 ▪ Applying a function to a behavior 435 Turning functions into "behavior functions" 436 Implementing lifting and map in C# 438

15.4 Working with drawings 440

Representing drawings 440 ▪ Creating and composing drawings 442

15.5 Creating animations 445

Implementing the animation form in F# 446 ▪ Creating animations using behaviors 448 ▪ Adding animation primitives 449 ▪ Creating a solar system animation 452

15.6 Developing financial modeling language 454

Modeling financial contracts 455 ▪ Defining the primitives 455 ▪ Using the modeling language 457

15.7 Summary 459

16 ***Developing reactive functional programs 460***

16.1 Reactive programming using events 461

Introducing event functions 462 ▪ Using events and observables 463 ▪ Creating a simple reactive application 466 Declarative event processing using LINQ 467 ▪ Declaring events in F# 470

16.2 Creating reactive animations 471

Using the switch function 472 ▪ Implementing the switch function 473

16.3 Programming UIs using workflows 474

Waiting for events asynchronously 474 ▪ Drawing rectangles 477

16.4 Storing state in reactive applications 480

Working with state safely 480 ▪ Creating a mailbox processor 481 Communicating using messages 483 ▪ Encapsulating mailbox processors 487 ▪ Waiting for multiple events 488

16.5 Message passing concurrency 490

Creating a state machine processor 491 ▪ Accessing mailbox concurrently 492

16.6 Summary 493

appendix Looking ahead 495

resources 498

index 501