Contents

1 Prologue

1.1 Why OCaml?
  1.1.1 A Brief History  1
  1.1.2 The Base Standard Library  2
  1.1.3 The OCaml Platform  3
1.2 About This Book
  1.2.1 What to Expect  4
  1.2.2 Installation Instructions  5
  1.2.3 Code Examples  6
1.3 Contributors  6

Part I Language Concepts

2 A Guided Tour

2.1 OCaml as a Calculator  9
2.2 Functions and Type Inference
  2.2.1 Type Inference  10
  2.2.2 Inferring Generic Types  11
2.3 Tuples, Lists, Options, and Pattern Matching
  2.3.1 Tuples  12
  2.3.2 Lists  13
  2.3.3 Options  14
2.4 Records and Variants  15
2.5 Imperative Programming
  2.5.1 Arrays  16
  2.5.2 Mutatable Record Fields  17
  2.5.3 Refs  18
  2.5.4 For and While Loops  19
2.6 A Complete Program
  2.6.1 Compiling and Running  20
2.7 Where to Go from Here  21

page 1
# Contents

3  **Variables and Functions**  
3.1  Variables  
3.1.1  Pattern Matching and Let  
3.2  Functions  
3.2.1  Anonymous Functions  
3.2.2  Multiargument Functions  
3.2.3  Recursive Functions  
3.2.4  Prefix and Infix Operators  
3.2.5  Declaring Functions with function  
3.2.6  Labeled Arguments  
3.2.7  Optional Arguments

4  **Lists and Patterns**  
4.1  List Basics  
4.2  Using Patterns to Extract Data from a List  
4.3  Limitations (and Blessings) of Pattern Matching  
4.3.1  Performance  
4.3.2  Detecting Errors  
4.4  Using the List Module Effectively  
4.4.1  More Useful List Functions  
4.5  Tail Recursion  
4.6  Terser and Faster Patterns

5  **Files, Modules, and Programs**  
5.1  Single-File Programs  
5.2  Multifile Programs and Modules  
5.3  Signatures and Abstract Types  
5.4  Concrete Types in Signatures  
5.5  Nested Modules  
5.6  Opening Modules  
5.6.1  Open Modules Rarely  
5.6.2  Prefer Local Opens  
5.6.3  Using Module Shortcuts Instead  
5.7  Including Modules  
5.8  Common Errors with Modules  
5.8.1  Type Mismatches  
5.8.2  Missing Definitions  
5.8.3  Type Definition Mismatches  
5.8.4  Cyclic Dependencies  
5.9  Designing with Modules  
5.9.1  Expose Concrete Types Rarely  
5.9.2  Design for the Call Site  
5.9.3  Create Uniform Interfaces
6 Records

6.1 Patterns and Exhaustiveness 87
6.2 Field Punning 89
6.3 Reusing Field Names 90
6.4 Functional Updates 93
6.5 Mutable Fields 94
6.6 First-Class Fields 95

7 Variants

7.1 Catch-All Cases and Refactoring 102
7.2 Combining Records and Variants 104
7.2.1 Embedded Records 107
7.3 Variants and Recursive Data Structures 108
7.4 Polymorphic Variants 111
7.4.1 Example: Terminal Colors Redux 113
7.4.2 When to Use Polymorphic Variants 117

8 Error Handling

8.1 Error-Aware Return Types 119
8.1.1 Encoding Errors with Result 120
8.1.2 Error and Or_error 121
8.1.3 bind and Other Error Handling Idioms 122
8.2 Exceptions 124
8.2.1 Helper Functions for Throwing Exceptions 126
8.2.2 Exception Handlers 127
8.2.3 Cleaning Up in the Presence of Exceptions 128
8.2.4 Catching Specific Exceptions 129
8.2.5 Backtraces 130
8.2.6 From Exceptions to Error-Aware Types and Back Again 132
8.3 Choosing an Error-Handling Strategy 133

9 Imperative Programming

9.1 Example: Imperative Dictionaries 134
9.2 Primitive Mutable Data 138
9.2.1 Array-Like Data 138
9.2.2 Mutable Record and Object Fields and Ref Cells 139
9.2.3 Foreign Functions 140
9.3 For and While Loops 140
9.4 Example: Doubly Linked Lists 141
9.4.1 Modifying the List 143
9.4.2 Iteration Functions 144
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.5</td>
<td>Laziness and Other Benign Effects</td>
<td>145</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Memoization and Dynamic Programming</td>
<td>146</td>
</tr>
<tr>
<td>9.6</td>
<td>Input and Output</td>
<td>152</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Terminal I/O</td>
<td>153</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Formatted Output with printf</td>
<td>154</td>
</tr>
<tr>
<td>9.6.3</td>
<td>File I/O</td>
<td>156</td>
</tr>
<tr>
<td>9.7</td>
<td>Order of Evaluation</td>
<td>158</td>
</tr>
<tr>
<td>9.8</td>
<td>Side Effects and Weak Polymorphism</td>
<td>159</td>
</tr>
<tr>
<td>9.8.1</td>
<td>The Value Restriction</td>
<td>160</td>
</tr>
<tr>
<td>9.8.2</td>
<td>Partial Application and the Value Restriction</td>
<td>161</td>
</tr>
<tr>
<td>9.8.3</td>
<td>Relaxing the Value Restriction</td>
<td>162</td>
</tr>
<tr>
<td>9.9</td>
<td>Summary</td>
<td>164</td>
</tr>
<tr>
<td>10</td>
<td>GADTs</td>
<td>166</td>
</tr>
<tr>
<td>10.1</td>
<td>A Little Language</td>
<td>166</td>
</tr>
<tr>
<td>10.1.1</td>
<td>Making the Language Type-Safe</td>
<td>168</td>
</tr>
<tr>
<td>10.1.2</td>
<td>Trying to Do Better with Ordinary Variants</td>
<td>169</td>
</tr>
<tr>
<td>10.1.3</td>
<td>GADTs to the Rescue</td>
<td>170</td>
</tr>
<tr>
<td>10.1.4</td>
<td>GADTs, Locally Abstract Types, and Polymorphic Recursion</td>
<td>172</td>
</tr>
<tr>
<td>10.2</td>
<td>When Are GADTs Useful?</td>
<td>173</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Varying Your Return Type</td>
<td>173</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Capturing the Unknown</td>
<td>176</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Abstracting Computational Machines</td>
<td>177</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Narrowing the Possibilities</td>
<td>180</td>
</tr>
<tr>
<td>10.3</td>
<td>Limitations of GADTs</td>
<td>187</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Or-Patterns</td>
<td>188</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Deriving Serializers</td>
<td>188</td>
</tr>
<tr>
<td>11</td>
<td>Functors</td>
<td>191</td>
</tr>
<tr>
<td>11.1</td>
<td>A Trivial Example</td>
<td>191</td>
</tr>
<tr>
<td>11.2</td>
<td>A Bigger Example: Computing with Intervals</td>
<td>193</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Making the Functor Abstract</td>
<td>196</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Sharing Constraints</td>
<td>197</td>
</tr>
<tr>
<td>11.2.3</td>
<td>Destructive Substitution</td>
<td>199</td>
</tr>
<tr>
<td>11.2.4</td>
<td>Using Multiple Interfaces</td>
<td>201</td>
</tr>
<tr>
<td>11.3</td>
<td>Extending Modules</td>
<td>205</td>
</tr>
<tr>
<td>12</td>
<td>First-Class Modules</td>
<td>209</td>
</tr>
<tr>
<td>12.1</td>
<td>Working with First-Class Modules</td>
<td>209</td>
</tr>
<tr>
<td>12.1.1</td>
<td>Creating First-Class Modules</td>
<td>209</td>
</tr>
<tr>
<td>12.1.2</td>
<td>Inference and Anonymous Modules</td>
<td>210</td>
</tr>
<tr>
<td>12.1.3</td>
<td>Unpacking First-Class Modules</td>
<td>210</td>
</tr>
<tr>
<td>12.1.4</td>
<td>Functions for Manipulating First-Class Modules</td>
<td>210</td>
</tr>
<tr>
<td>12.1.5</td>
<td>Richer First-Class Modules</td>
<td>211</td>
</tr>
</tbody>
</table>
12.1.6 Exposing types 211
12.2 Example: A Query-Handling Framework 213
12.2.1 Implementing a Query Handler 215
12.2.2 Dispatching to Multiple Query Handlers 216
12.2.3 Loading and Unloading Query Handlers 219
12.3 Living Without First-Class Modules 222

13 Objects 223
13.1 OCaml Objects 224
13.2 Object Polymorphism 225
13.3 Immutable Objects 227
13.4 When to Use Objects 228
13.5 Subtyping 228
13.5.1 Width Subtyping 229
13.5.2 Depth Subtyping 229
13.5.3 Variance 230
13.5.4 Narrowing 234
13.5.5 Subtyping Versus Row Polymorphism 235

14 Classes 237
14.1 OCaml Classes 237
14.2 Class Parameters and Polymorphism 238
14.3 Object Types as Interfaces 239
14.3.1 Functional Iterators 242
14.4 Inheritance 243
14.5 Class Types 244
14.6 Open Recursion 245
14.7 Private Methods 246
14.8 Binary Methods 247
14.9 Virtual Classes and Methods 251
14.9.1 Create Some Simple Shapes 251
14.10 Initializers 254
14.11 Multiple Inheritance 254
14.11.1 How Names Are Resolved 254
14.11.2 Mixins 255
14.11.3 Displaying the Animated Shapes 258

Part II Tools and Techniques 261

15 Maps and Hash Tables 263
15.1 Maps 263
15.1.1 Sets 265
15.1.2 Modules and Comparators 265
15.1.3 Why Do We Need Comparator Witnesses? 267
15.1.4 The Polymorphic Comparator 269
15.1.5 Satisfying Comparator. S with [@deriving] 270
15.1.6 Applying [@deriving] to Maps and Sets 272
15.1.7 Trees 273
15.2 Hash Tables 274
15.2.1 Time Complexity of Hash Tables 274
15.2.2 Collisions with the Polymorphic Hash Function 275
15.3 Choosing Between Maps and Hash Tables 276

16 Command-Line Parsing 280
16.1 Basic Command-Line Parsing 280
16.1.1 Defining an Anonymous Argument 281
16.1.2 Defining Basic Commands 281
16.1.3 Running Commands 282
16.1.4 Multi-Argument Commands 284
16.2 Argument Types 285
16.2.1 Defining Custom Argument Types 286
16.2.2 Optional and Default Arguments 286
16.2.3 Sequences of Arguments 288
16.3 Adding Labeled Flags 289
16.4 Grouping Subcommands Together 291
16.5 Prompting for Interactive Input 293
16.6 Command-Line Autocompletion with bash 294
16.6.1 Generating Completion Fragments from Command 295
16.6.2 Installing the Completion Fragment 295
16.7 Alternative Command-Line Parsers 296

17 Concurrent Programming with Async 297
17.1 Async Basics 297
17.1.1 Using Let Syntax 300
17.1.2 Ivars and Upon 301
17.2 Example: An Echo Server 303
17.2.1 Improving the Echo Server 306
17.3 Example: Searching Definitions with DuckDuckGo 309
17.3.1 URI Handling 309
17.3.2 Parsing JSON Strings 310
17.3.3 Executing an HTTP Client Query 310
17.4 Exception Handling 312
17.4.1 Monitors 314
17.4.2 Example: Handling Exceptions with DuckDuckGo 316
17.5 Timeouts, Cancellation, and Choices 318
17.6 Working with System Threads 320
18 Testing

18.1 Inline Tests
   18.1.1 More Readable Errors with `test_eq` 327
   18.1.2 Where Should Tests Go? 328

18.2 Expect Tests
   18.2.1 Basic Mechanics 329
   18.2.2 What Are Expect Tests Good For? 330
   18.2.3 Exploratory Programming 331
   18.2.4 Visualizing Complex Behavior 333
   18.2.5 End-to-End Tests 336
   18.2.6 How to Make a Good Expect Test 339

18.3 Property Testing with Quickcheck
   18.3.1 Handling Complex Types 341
   18.3.2 More Control with Let-Syntax 342

18.4 Other Testing Tools
   18.4.1 Other Tools to Do (Mostly) the Same Things 343
   18.4.2 Fuzzing 344

19 Handling JSON Data

19.1 JSON Basics 345
19.2 Parsing JSON with Yojson 346
19.3 Selecting Values from JSON Structures 348
19.4 Constructing JSON Values 351
19.5 Using Nonstandard JSON Extensions 353
19.6 Automatically Mapping JSON to OCaml Types 354
   19.6.1 ATD Basics 354
   19.6.2 ATD Annotations 355
   19.6.3 Compiling ATD Specifications to OCaml 355
   19.6.4 Example: Querying GitHub Organization Information 357

20 Parsing with OCamllex and Menhir

20.1 Lexing and Parsing 362
20.2 Defining a Parser
   20.2.1 Describing the Grammar 364
   20.2.2 Parsing Sequences 365
20.3 Defining a Lexer
   20.3.1 OCaml Prelude 367
   20.3.2 Regular Expressions 368
   20.3.3 Lexing Rules 368
   20.3.4 Recursive Rules 370
20.4 Bringing It All Together 371
21 Data Serialization with S-Expressions

21.1 Basic Usage
   21.1.1 S-Expression Converters for New Types

21.2 The Sexp Format

21.3 Preserving Invariants

21.4 Getting Good Error Messages

21.5 Sexp-Conversion Directives
   21.5.1 @sexp.opaque
   21.5.2 @sexp.list
   21.5.3 @sexp.option
   21.5.4 Specifying Defaults

22 The OCaml Platform

22.1 A Hello World OCaml Project
   22.1.1 Setting Up an Opam Local Switch
   22.1.2 Choosing an OCaml Compiler Version
   22.1.3 Structure of an OCaml Project
   22.1.4 Defining Module Names
   22.1.5 Defining Libraries as Collections of Modules
   22.1.6 Writing Test Cases for a Library
   22.1.7 Building an Executable Program

22.2 Setting Up an Integrated Development Environment
   22.2.1 Using Visual Studio Code
   22.2.2 Browsing Interface Documentation
   22.2.3 Autoformatting Your Source Code

22.3 Publishing Your Code Online
   22.3.1 Defining Opam Packages
   22.3.2 Generating Project Metadata from Dune
   22.3.3 Setting up Continuous Integration
   22.3.4 Other Conventions
   22.3.5 Releasing Your Code into the Opam Repository

22.4 Learning More from Real Projects

Part III The Compiler and Runtime System

23 Foreign Function Interface

23.1 Example: A Terminal Interface

23.2 Basic Scalar C Types

23.3 Pointers and Arrays
   23.3.1 Allocating Typed Memory for Pointers
   23.3.2 Using Views to Map Complex Values

23.4 Structs and Unions
   23.4.1 Defining a Structure
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.4.2</td>
<td>Adding Fields to Structures</td>
<td>415</td>
</tr>
<tr>
<td>23.4.3</td>
<td>Incomplete Structure Definitions</td>
<td>415</td>
</tr>
<tr>
<td>23.4.4</td>
<td>Defining Arrays</td>
<td>418</td>
</tr>
<tr>
<td>23.5</td>
<td>Passing Functions to C</td>
<td>419</td>
</tr>
<tr>
<td>23.5.1</td>
<td>Example: A Command-Line Quicksort</td>
<td>420</td>
</tr>
<tr>
<td>23.6</td>
<td>Learning More About C Bindings</td>
<td>422</td>
</tr>
<tr>
<td>23.6.1</td>
<td>Struct Memory Layout</td>
<td>423</td>
</tr>
<tr>
<td>24</td>
<td>Memory Representation of Values</td>
<td>424</td>
</tr>
<tr>
<td>24.1</td>
<td>OCaml Blocks and Values</td>
<td>425</td>
</tr>
<tr>
<td>24.1.1</td>
<td>Distinguishing Integers and Pointers at Runtime</td>
<td>425</td>
</tr>
<tr>
<td>24.2</td>
<td>Blocks and Values</td>
<td>427</td>
</tr>
<tr>
<td>24.2.1</td>
<td>Integers, Characters, and Other Basic Types</td>
<td>428</td>
</tr>
<tr>
<td>24.3</td>
<td>Tuples, Records, and Arrays</td>
<td>428</td>
</tr>
<tr>
<td>24.3.1</td>
<td>Floating-Point Numbers and Arrays</td>
<td>429</td>
</tr>
<tr>
<td>24.4</td>
<td>Variants and Lists</td>
<td>430</td>
</tr>
<tr>
<td>24.5</td>
<td>Polymorphic Variants</td>
<td>431</td>
</tr>
<tr>
<td>24.6</td>
<td>String Values</td>
<td>432</td>
</tr>
<tr>
<td>24.7</td>
<td>Custom Heap Blocks</td>
<td>432</td>
</tr>
<tr>
<td>24.7.1</td>
<td>Managing External Memory with Bigarray</td>
<td>433</td>
</tr>
<tr>
<td>25</td>
<td>Understanding the Garbage Collector</td>
<td>435</td>
</tr>
<tr>
<td>25.1</td>
<td>Mark and Sweep Garbage Collection</td>
<td>435</td>
</tr>
<tr>
<td>25.2</td>
<td>Generational Garbage Collection</td>
<td>436</td>
</tr>
<tr>
<td>25.3</td>
<td>The Fast Minor Heap</td>
<td>436</td>
</tr>
<tr>
<td>25.3.1</td>
<td>Allocating on the Minor Heap</td>
<td>437</td>
</tr>
<tr>
<td>25.4</td>
<td>The Long-Lived Major Heap</td>
<td>438</td>
</tr>
<tr>
<td>25.4.1</td>
<td>Allocating on the Major Heap</td>
<td>439</td>
</tr>
<tr>
<td>25.4.2</td>
<td>Memory Allocation Strategies</td>
<td>439</td>
</tr>
<tr>
<td>25.4.3</td>
<td>Marking and Scanning the Heap</td>
<td>441</td>
</tr>
<tr>
<td>25.4.4</td>
<td>Heap Compaction</td>
<td>442</td>
</tr>
<tr>
<td>25.4.5</td>
<td>Intergenerational Pointers</td>
<td>443</td>
</tr>
<tr>
<td>25.5</td>
<td>Attaching Finalizer Functions to Values</td>
<td>445</td>
</tr>
<tr>
<td>26</td>
<td>The Compiler Frontend: Parsing and Type Checking</td>
<td>447</td>
</tr>
<tr>
<td>26.1</td>
<td>An Overview of the Toolchain</td>
<td>447</td>
</tr>
<tr>
<td>26.1.1</td>
<td>Obtaining the Compiler Source Code</td>
<td>448</td>
</tr>
<tr>
<td>26.2</td>
<td>Parsing Source Code</td>
<td>449</td>
</tr>
<tr>
<td>26.2.1</td>
<td>Syntax Errors</td>
<td>449</td>
</tr>
<tr>
<td>26.2.2</td>
<td>Generating Documentation from Interfaces</td>
<td>450</td>
</tr>
<tr>
<td>26.3</td>
<td>Preprocessing with ppx</td>
<td>451</td>
</tr>
<tr>
<td>26.3.1</td>
<td>Extension Attributes</td>
<td>452</td>
</tr>
<tr>
<td>26.3.2</td>
<td>Commonly Used Extension Attributes</td>
<td>453</td>
</tr>
<tr>
<td>26.3.3</td>
<td>Extension Nodes</td>
<td>454</td>
</tr>
</tbody>
</table>
26.4 Static Type Checking 454
  26.4.1 Displaying Inferred Types from the Compiler 455
  26.4.2 Type Inference 456
  26.4.3 Modules and Separate Compilation 460
  26.4.4 Wrapping Libraries with Module Aliases 462
  26.4.5 Shorter Module Paths in Type Errors 464
26.5 The Typed Syntax Tree 465
  26.5.1 Examining the Typed Syntax Tree Directly 465

27 The Compiler Backend: Bytecode and Native code 468
  27.1 The Untyped Lambda Form 468
    27.1.1 Pattern Matching Optimization 468
    27.1.2 Benchmarking Pattern Matching 470
  27.2 Generating Portable Bytecode 472
    27.2.1 Compiling and Linking Bytecode 474
    27.2.2 Executing Bytecode 474
    27.2.3 Embedding OCaml Bytecode in C 475
  27.3 Compiling Fast Native Code 476
    27.3.1 Inspecting Assembly Output 477
    27.3.2 Debugging Native Code Binaries 480
    27.3.3 Profiling Native Code 483
    27.3.4 Embedding Native Code in C 485
  27.4 Summarizing the File Extensions 486

Index 487