

Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

- I. Language Concepts
- II. Tools and Techniques
- III. The Runtime System
- Index

Login with GitHub to view
and add comments

Table of Contents

Prologue

- Why OCaml?
 - A Brief History
 - The Core Standard Library
 - The OCaml Platform

About This Book

- What to Expect
- Installation Instructions
- Code Examples

Safari® Books Online

How to Contact Us

Contributors

I. Language Concepts

1. A Guided Tour

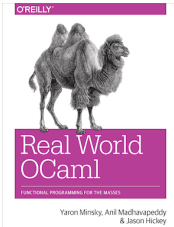
- OCaml as a Calculator
- Functions and Type Inference
 - Type Inference
 - Inferring Generic Types
- Tuples, Lists, Options, and Pattern Matching
 - Tuples
 - Lists
 - The List module
 - Constructing lists with ::
 - List patterns using match
 - Recursive list functions
 - Options
- Records and Variants
- Imperative Programming
 - Arrays
 - Mutable Record Fields
 - Refs
 - For and While Loops
- A Complete Program
 - Compiling and Running
- Where to Go from Here

2. Variables and Functions

- Variables
 - Pattern Matching and let
- Functions
 - Anonymous Functions
 - Multiargument functions
 - Recursive Functions
 - Prefix and Infix Operators
 - Declaring Functions with Function
 - Labeled Arguments
 - Higher-order functions and labels
 - Optional Arguments
 - Explicit passing of an optional argument
 - Inference of labeled and optional arguments
 - Optional arguments and partial application

3. Lists and Patterns

- List Basics



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

Table of Contents / Real World OCaml

Using Patterns to Extract Data from a List

Limitations (and Blessings) of Pattern Matching

Performance

Detecting Errors

Using the List Module Effectively

More Useful List Functions

Combining list elements with `List.reduce`

Filtering with `List.filter` and `List.filter_map`

Partitioning with `List.partition_tf`

Combining lists

Tail Recursion

Terser and Faster Patterns

4. Files, Modules, and Programs

Single-File Programs

Multifile Programs and Modules

Signatures and Abstract Types

Concrete Types in Signatures

Nested Modules

Opening Modules

Including Modules

Common Errors with Modules

Type Mismatches

Missing Definitions

Type Definition Mismatches

Cyclic Dependencies

Designing with Modules

Expose Concrete Types Rarely

Design for the Call Site

Create Uniform Interfaces

Interfaces before implementations

5. Records

Patterns and Exhaustiveness

Field Punning

Reusing Field Names

Functional Updates

Mutable Fields

First-Class Fields

6. Variants

Catch-All Cases and Refactoring

Combining Records and Variants

Variants and Recursive Data Structures

Polymorphic Variants

Example: Terminal Colors Redux

When to Use Polymorphic Variants

7. Error Handling

Error-Aware Return Types

Encoding Errors with `Result`

Error and `Or_error`

`bind` and Other Error Handling Idioms

Exceptions

Helper Functions for Throwing Exceptions

Exception Handlers

Cleaning Up in the Presence of Exceptions

Catching Specific Exceptions

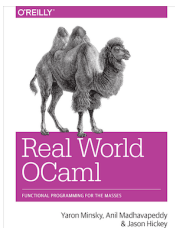
Backtraces

From Exceptions to Error-Aware Types and Back Again

Choosing an Error-Handling Strategy

8. Imperative Programming

Example: Imperative Dictionaries



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

Table of Contents / Real World OCaml

Primitive Mutable Data

Array-Like Data

Ordinary arrays

Strings

Bigarrays

Mutable Record and Object Fields and Ref Cells

Ref cells

Foreign Functions

for and while Loops

Example: Doubly Linked Lists

Modifying the List

Iteration Functions

Laziness and Other Benign Effects

Memoization and Dynamic Programming

Input and Output

Terminal I/O

Formatted Output with printf

File I/O

Order of Evaluation

Side Effects and Weak Polymorphism

The Value Restriction

Partial Application and the Value Restriction

Relaxing the Value Restriction

Summary

9. Functors

A Trivial Example

A Bigger Example: Computing with Intervals

Making the Functor Abstract

Sharing Constraints

Destructive Substitution

Using Multiple Interfaces

Extending Modules

10. First-Class Modules

Working with First-Class Modules

Example: A Query-Handling Framework

Implementing a Query Handler

Dispatching to Multiple Query Handlers

Loading and Unloading Query Handlers

Living Without First-Class Modules

11. Objects

OCaml Objects

Object Polymorphism

Immutable Objects

When to Use Objects

Subtyping

Width Subtyping

Depth Subtyping

Variance

Narrowing

Subtyping Versus Row Polymorphism

12. Classes

OCaml Classes

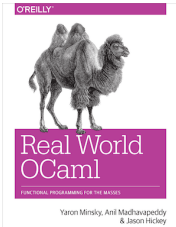
Class Parameters and Polymorphism

Object Types as Interfaces

Functional Iterators

Inheritance

Class Types



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

Open Recursion
Private Methods
Binary Methods
Virtual Classes and Methods
Create Some Simple Shapes

Initializers
Multiple Inheritance
How Names Are Resolved
Mixins
Displaying the Animated Shapes

II. Tools and Techniques

13. Maps and Hash Tables

Maps
Creating Maps with Comparators
Trees
The Polymorphic Comparator
Sets
Satisfying the Comparable.S Interface

Hash Tables
Satisfying the Hashable.S Interface

Choosing Between Maps and Hash Tables

14. Command-Line Parsing

Basic Command-Line Parsing
Anonymous Arguments
Defining Basic Commands
Running Basic Commands

Argument Types
Defining Custom Argument Types
Optional and Default Arguments
Sequences of Arguments

Adding Labeled Flags to the Command Line
Grouping Subcommands Together
Advanced Control over Parsing
The Types Behind Command.Spec
Composing Specification Fragments Together
Prompting for Interactive Input
Adding Labeled Arguments to Callbacks

Command-Line Autocompletion with bash
Generating Completion Fragments from Command
Installing the Completion Fragment

Alternative Command-Line Parsers

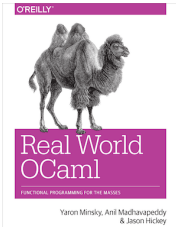
15. Handling JSON Data

JSON Basics
Parsing JSON with Yojson
Selecting Values from JSON Structures
Constructing JSON Values
Using Nonstandard JSON Extensions
Automatically Mapping JSON to OCaml Types
ATD Basics
ATD Annotations
Compiling ATD Specifications to OCaml
Example: Querying GitHub Organization Information

16. Parsing with OCamllex and Menhir

Lexing and Parsing
Defining a Parser
Describing the Grammar
Parsing Sequences

Defining a Lexer



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

Table of Contents / Real World OCaml

OCaml Prelude

Regular Expressions

Lexing Rules

Recursive Rules

Bringing It All Together

17. Data Serialization with S-Expressions

Basic Usage

Generating S-Expressions from OCaml Types

The Sexp Format

Preserving Invariants

Getting Good Error Messages

Sexp-Conversion Directives

sexp_opaque

sexp_list

sexp_option

Specifying Defaults

18. Concurrent Programming with Async

Async Basics

Ivars and Upon

Examples: An Echo Server

Improving the Echo Server

Example: Searching Definitions with DuckDuckGo

URI Handling

Parsing JSON Strings

Executing an HTTP Client Query

Exception Handling

Monitors

Example: Handling Exceptions with DuckDuckGo

Timeouts, Cancellation, and Choices

Working with System Threads

Thread-Safety and Locking

III. The Runtime System

19. Foreign Function Interface

Example: A Terminal Interface

Basic Scalar C Types

Pointers and Arrays

Allocating Typed Memory for Pointers

Using Views to Map Complex Values

Structs and Unions

Defining a Structure

Adding Fields to Structures

Incomplete Structure Definitions

Recap: A time-printing command

Defining Arrays

Passing Functions to C

Example: A Command-Line Quicksort

Learning More About C Bindings

Struct Memory Layout

20. Memory Representation of Values

OCaml Blocks and Values

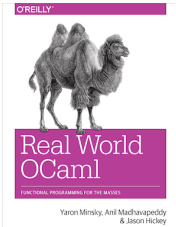
Distinguishing Integers and Pointers at Runtime

Blocks and Values

Integers, Characters, and Other Basic Types

Tuples, Records, and Arrays

Floating-Point Numbers and Arrays



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

Variants and Lists

Polymorphic Variants

String Values

Custom Heap Blocks

Managing External Memory with Bigarray

21. Understanding the Garbage Collector

Mark and Sweep Garbage Collection

Generational Garbage Collection

The Fast Minor Heap

Allocating on the Minor Heap

The Long-Lived Major Heap

Allocating on the Major Heap

Memory Allocation Strategies

Next-fit allocation

First-fit allocation

Marking and Scanning the Heap

Heap Compaction

Intergenerational Pointers

The mutable write barrier

Attaching Finalizer Functions to Values

22. The Compiler Frontend: Parsing and Type Checking

An Overview of the Toolchain

Parsing Source Code

Syntax Errors

Automatically Indenting Source Code

Generating Documentation from Interfaces

Preprocessing Source Code

Using Camlp4 Interactively

Running Camlp4 from the Command Line

Preprocessing Module Signatures

Further Reading on Camlp4

Static Type Checking

Displaying Inferred Types from the Compiler

Type Inference

Adding type annotations to find errors

Enforcing principal typing

Modules and Separate Compilation

The mapping between files and modules

Defining a module search path

Packing Modules Together

Shorter Module Paths in Type Errors

The Typed Syntax Tree

Using ocp-index for Autocompletion

Examining the Typed Syntax Tree Directly

23. The Compiler Backend: Bytecode and Native code

The Untyped Lambda Form

Pattern Matching Optimization

Benchmarking Pattern Matching

Generating Portable Bytecode

Compiling and Linking Bytecode

Executing Bytecode

Embedding OCaml Bytecode in C

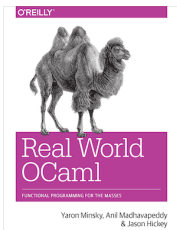
Compiling Fast Native Code

Inspecting Assembly Output

The impact of polymorphic comparison

Benchmarking polymorphic comparison

Debugging Native Code Binaries



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

Index

Copyright 2012-2013, Jason Hickey, Anil Madhavapeddy and Yaron Minsky. Licensed under [CC BY-NC-ND 3.0 US](#).

Login with GitHub to view
and add comments

Understanding name mangling

Interactive breakpoints with the GNU debugger

Profiling Native Code

Gprof

Perf

Embedding Native Code in C

Summarizing the File Extensions

Index

Next >