

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

Index

Symbols

`%.byte` rule, [Executing Bytecode](#)

A

abstract types, [Signatures and Abstract Types](#), [Nested Modules](#), [Working with First-Class Modules](#)

algebraic data types, [Combining Records and Variants](#)

alignment, formatting with `printf`, [Formatted Output with `printf`](#)

animation

creating with mixins, [Mixins](#)

displaying animated shapes, [Displaying the Animated Shapes](#)

annotations, for type checking, [Adding type annotations to find errors](#)

anonymous arguments, [Anonymous Arguments](#)

anonymous functions, [Records and Variants](#), [Anonymous Functions](#)

Arg module, [Alternative Command-Line Parsers](#)

Argot HTML generator, [Generating Documentation from Interfaces](#)

arguments

anonymous arguments, [Anonymous Arguments](#)

argument types, [Argument Types](#)

default arguments, [Optional and Default Arguments](#)

defining custom types, [Defining Custom Argument Types](#)

inference of, [Inference of labeled and optional arguments](#)

labeled arguments, [The List module](#), [Labeled Arguments](#), [Adding Labeled Arguments to Callbacks](#)

optional arguments, [Optional Arguments-Optional arguments and partial application](#), [Optional and Default Arguments](#)

sequences of, [Sequences of Arguments](#)

unit argument to callbacks, [Defining Basic Commands](#)

Array module

`Array.blit`, [Ordinary arrays](#)

`Array.set`, [Ordinary arrays](#)

array-like data, [Primitive Mutable Data](#)

arrays

definition of, [Defining Arrays](#)

imperative programming and, [Arrays](#)

memory representation of, [Tuples, Records, and Arrays](#)

pointers and, [Pointers and Arrays](#)

`assert` directive, [Helper Functions for Throwing Exceptions](#)

association lists, [Maps and Hash Tables](#)

AST (abstract syntax-tree), [Lexing and Parsing](#), [Memory Representation of Values](#), [Parsing Source Code](#), [The Typed Syntax Tree](#)

Async library

basics of, [Async Basics-Ivars and Upon](#)

benefits of, [Concurrent Programming with Async](#)

DuckDuckGo searching example, [Example: Searching Definitions with DuckDuckGo-Executing an HTTP Client Query](#)

echo server example, [Examples: An Echo Server-Improving the Echo Server](#)

exception handling in, [Exception Handling-Example: Handling Exceptions with DuckDuckGo](#)

ivars, [Ivars and Upon](#)

system threads and, [Working with System Threads-Thread-Safety and Locking](#)

timeouts and cancellations, [Timeouts, Cancellation, and Choices](#)

ATDgen Library

annotations in, [ATD Annotations](#)

basics of, [ATD Basics](#)

compiling specifications to OCaml, [Compiling ATD Specifications to OCaml](#)

example of, [Example: Querying GitHub Organization Information](#)

installation of, [Automatically Mapping JSON to OCaml Types](#)

autocompletion, [Command-Line Autocompletion with bash](#), [Using ocp-index for Autocompletion](#)

automatic type inference, [Static Type Checking](#)

(see also type inference)

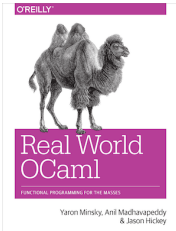
B

backtraces, [Backtraces](#), [The impact of polymorphic comparison](#)

bash autocompletion, [Command-Line Autocompletion with bash](#)

benign effects

laziness, [Laziness and Other Benign Effects](#)



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

memoization, [Memoization and Dynamic Programming-Memoization and Dynamic Programming](#)

Bibtex, [Generating Documentation from Interfaces](#)

bigarrays, [Bigarrays](#), [Managing External Memory with Bigarray](#)

Bigstring module, [Managing External Memory with Bigarray](#)

binary methods, [Binary Methods](#)

binary numbers, formatting with printf, [Formatted Output with printf](#)

bind function, [bind and Other Error Handling Idioms](#), [Async Basics](#)

bindings

scope of, [Variables](#)

top-level, [Variables](#)

wildcards in let bindings, [Running Camlp4 from the Command Line](#)

Bin_prot library, [Preprocessing Source Code](#)

BLAS mathematical library, [Managing External Memory with Bigarray](#)

block values, [Generating Portable Bytecode](#)

blocking, [Async Basics](#)

blocks (of memory), [OCaml Blocks and Values-Blocks and Values](#)

boxing (of values), [Distinguishing Integers and Pointers at Runtime](#)

byte arrays, [Strings](#)

bytecode compiler

compiling and linking code, [Compiling and Linking Bytecode](#)

instruction set for, [Generating Portable Bytecode](#)

vs. native-code compiler, [Single-File Programs](#)

tools used, [Generating Portable Bytecode](#)

values stored by, [Generating Portable Bytecode](#)

C

C object files, [Embedding OCaml Bytecode in C](#)

C99 scalar types, [Basic Scalar C Types](#)

callback function, [Defining Basic Commands](#), [The Types Behind Command.Spec](#), [Adding Labeled Arguments to Callbacks](#)

Camlimages library, [When to Use Objects](#)

Camlp4 syntax extension mechanism, [Generating S-Expressions from OCaml Types](#), [Preprocessing Source Code-Further Reading on Camlp4](#)

Camomile unicode parser, [Recursive Rules](#)

cancellations, [Timeouts](#), [Cancellation](#), and [Choices](#)

catch-all cases, [Example: Terminal Colors Redux](#)

classes

basic syntax for, [OCaml Classes](#)

benefits of, [When to Use Objects](#)

binary methods for, [Binary Methods](#)

class parameters and polymorphism, [Class Parameters and Polymorphism](#)

class types, [Class Types](#)

inheritance in, [Inheritance](#)

initializers for, [Initializers](#)

multiple inheritance in, [Multiple Inheritance](#)

object types as interfaces, [Object Types as Interfaces](#)

open recursion in, [Open Recursion](#)

private methods for, [Private Methods](#)

virtual classes, [Virtual Classes and Methods](#)

client queries, [Executing an HTTP Client Query](#)

Cmdliner, [Alternative Command-Line Parsers](#)

cmi files, [Defining a module search path](#), [Compiling Fast Native Code](#)

cmt files, [The Typed Syntax Tree](#)

cmti files, [The Typed Syntax Tree](#)

cmx files, [Compiling Fast Native Code](#)

code compilers

bytecode vs. native code, [Single-File Programs](#), [An Overview of the Toolchain](#)

order of code, [Displaying Inferred Types from the Compiler](#)

(see also [compilation process](#))

warning enable/disable, [Patterns and Exhaustiveness](#)

code offset values, [Generating Portable Bytecode](#)

cohttp library, [Example: Searching Definitions with DuckDuckGo](#)

combinators

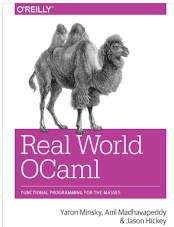
functional combinators, [Command-Line Parsing](#), [Selecting Values from JSON Structures](#)

in Yojson library, [Selecting Values from JSON Structures](#)

Command module, [The Types Behind Command.Spec](#)

command-line parsing

advanced control over, [Advanced Control over Parsing-Adding Labeled Arguments to Callbacks](#)



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

Table of Contents

Prologue

I. Language Concepts

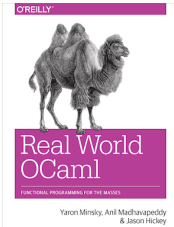
II. Tools and Techniques

III. The Runtime System

Index

Login with GitHub to view
and add comments

alternatives to Command library, [Alternative Command-Line Parsers](#)
 argument types, [Argument Types](#)
 autocompletion with bash, [Command-Line Autocompletion with bash](#)
 basic approach to, [Basic Command-Line Parsing](#)
 Command library for, [Command-Line Parsing](#)
 labeled flags and, [Adding Labeled Flags to the Command Line](#)
 subcommand grouping, [Grouping Subcommands Together](#)
 with Camlp4, [Running Camlp4 from the Command Line](#)
 Command.basic module, [Defining Basic Commands](#)
 Command.group, [Grouping Subcommands Together](#)
 commas vs. semicolons, [Constructing lists with ::](#)
 compaction, [Heap Compaction](#)
 Comparable module
 Comparable.Make, [Extending Modules](#), [Satisfying the Comparable.S Interface](#)
 Comparable.S, [Satisfying the Comparable.S Interface](#)
 Comparator.Poly module, [The Polymorphic Comparator](#)
 comparators, creating maps with, [Creating Maps with Comparators](#)
 compilation process
 compiler source code, [An Overview of the Toolchain](#)
 diagram of, [An Overview of the Toolchain](#)
 fast native code, [Compiling Fast Native Code-Embedding Native Code in C](#)
 file extensions, [Summarizing the File Extensions](#)
 parsing source code, [Parsing Source Code-Generating Documentation from Interfaces](#)
 phases of, [Memory Representation of Values](#)
 portable bytecode, [Generating Portable Bytecode-Embedding OCaml Bytecode in C](#)
 preprocessing source code, [Preprocessing Source Code-Further Reading on Camlp4](#)
 static type checking, [Static Type Checking-Shorter Module Paths in Type Errors](#)
 toolchain for, [The Compiler Frontend: Parsing and Type Checking](#)
 typed syntax tree, [The Typed Syntax Tree-Examining the Typed Syntax Tree Directly](#)
 untyped lambda form, [The Untyped Lambda Form-Benchmarking Pattern Matching](#)
 compilation units, [An Overview of the Toolchain](#), [Modules and Separate Compilation](#)
 compile-time static checking, [JSON Basics](#), [Memory Representation of Values](#), [The Compiler Frontend: Parsing and Type Checking](#), [Adding type annotations to find errors](#)
 completion handlers, [Installing the Completion Fragment](#)
 concrete types, [Concrete Types in Signatures](#)
 concurrent programming, [Concurrent Programming with Async](#), [Exception Handling](#)
 conditional compilation, [Preprocessing Module Signatures](#)
 config file formats, [Specifying Defaults](#)
 Container.Make, [Extending Modules](#)
 context-free grammars, [Describing the Grammar](#)
 contravariance, [Variance](#)
 copying collection, [The Fast Minor Heap](#)
 Core standard library
 development of, [The Core Standard Library](#)
 finding with ocamlfind, [Single-File Programs](#)
 imperative dictionaries in, [Example: Imperative Dictionaries](#)
 opening, [OCaml as a Calculator](#)
 corebuild, [Compiling and Running](#), [Single-File Programs](#)
 covariance, [Variance](#)
 Cryptokit library, [When to Use Objects](#)
 Ctypes library
 build directives for, [Example: A Terminal Interface](#)
 installation of, [Foreign Function Interface](#)
 lifetime of allocated Ctypes, [Example: A Command-Line Quicksort](#)
 terminal interface example, [Example: A Terminal Interface](#)
 curly braces ({}), [Describing the Grammar](#)
 curried functions, [Multiargument functions](#)
 custom heap blocks, [Custom Heap Blocks](#)
 custom runtime mode, [Executing Bytecode](#)
 cyclic data structures, [Example: Doubly Linked Lists](#)
 cyclic dependencies, [Cyclic Dependencies](#)
D
 data serialization
 with s-expressions, [Data Serialization with S-Expressions](#)
 with JSON, [Handling JSON Data-Example: Querying GitHub Organization Information](#)
 data structures
 arrays, [Arrays](#)
 cyclic, [Example: Doubly Linked Lists](#)



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

Table of Contents

Prologue

I. Language Concepts

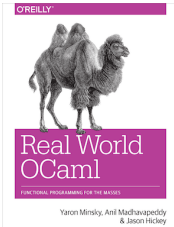
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

immutable, [Imperative Programming](#)
 key/value pairs, [Maps and Hash Tables](#)
 lists, [Lists-Recursive list functions](#)
 mutable record fields, [Mutable Record Fields](#)
 options, [Options](#)
 pattern matching and, [Limitations \(and Blessings\) of Pattern Matching](#)
 primitive mutable data, [Primitive Mutable Data](#)
 recursive, [Variants and Recursive Data Structures](#)
 tuples, [Tuples](#)
 datatypes
 algebraic types, [Combining Records and Variants](#)
 covariant, [Relaxing the Value Restriction](#)
 defining new, [Records and Variants](#)
 fixed vs. variable structure of, [Patterns and Exhaustiveness](#)
 locally abstract types, [Working with First-Class Modules](#)
 nullable, [Options](#)
 record types, [Records and Variants](#)
 variant types, [Records and Variants](#), [Variants-When to Use Polymorphic Variants](#)
 debugging
 activating debug runtime, [Embedding Native Code in C](#)
 Command mode, [The Types Behind Command.Spec](#)
 interactive debuggers, [Understanding name mangling](#)
 native code binaries, [Debugging Native Code Binaries](#)
 Obj module warning, [Variants and Lists](#)
 s-expressions, [Getting Good Error Messages](#)
 single errant characters, [Automatically Indenting Source Code](#)
 stack backtraces, [Backtraces](#)
 decimals, formatting with printf, [Formatted Output with printf](#)
 default arguments, [Optional and Default Arguments](#)
 Deferred.bind, [Async Basics](#)
 Deferred.both, [Timeouts, Cancellation, and Choices](#)
 Deferred.never, [Improving the Echo Server](#)
 Deferred.peek, [Async Basics](#)
 Deferred.t, [Async Basics](#)
 denial-of-service attacks, avoiding, [Hash Tables](#)
 dependencies, cyclic, [Cyclic Dependencies](#)
 depth subtyping, [Depth Subtyping](#)
 destructive substitution, [Destructive Substitution](#)
 destutter function, [Terser and Faster Patterns](#)
 dictionaries, imperative, [Example: Imperative Dictionaries-Example: Imperative Dictionaries](#)
 dispatching, dynamic vs. static, [Virtual Classes and Methods](#)
 documentation, generating from interfaces, [Generating Documentation from Interfaces](#)
 Domain Specific Language, [Automatically Mapping JSON to OCaml Types](#)
 doubly-linked lists, [Example: Doubly Linked Lists](#)
 Doubly-linked module, [Modifying the List](#)
 down casting, [Narrowing](#)
 DuckDuckGo search engine
 additional libraries needed, [Example: Searching Definitions with DuckDuckGo](#)
 exception handling in, [Example: Handling Exceptions with DuckDuckGo](#)
 executing an HTTP client query in, [Executing an HTTP Client Query](#)
 parsing JSON strings in, [Parsing JSON Strings](#)
 URI handling in, [URI Handling](#)
 duplicates, removing, [Prefix and Infix Operators](#), [Filtering with List.filter and List.filter_map](#),
[Terser and Faster Patterns](#)
 dynamic dispatch, [Virtual Classes and Methods](#)
 dynamic programming, [Memoization and Dynamic Programming](#), [Preprocessing Source Code](#)
 dynamic type checking, [Narrowing](#), [Binary Methods](#), [Memory Representation of Values](#)
E
 echo servers, [Examples: An Echo Server-Improving the Echo Server](#)
 edit distance, [Memoization and Dynamic Programming](#)
 elements
 combining with List.reduce, [Combining list elements with List.reduce](#)
 defining new, [Modifying the List](#)
 inserting in lists, [Modifying the List](#)
 partitioning with List.partition_tf, [Partitioning with List.partition_tf](#)
 setting with Array.set, [Ordinary arrays](#)
 traversing with iterator objects, [Object Types as Interfaces](#)
 ellipses (..), [Object Polymorphism](#)



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

Table of Contents

Prologue

I. Language Concepts

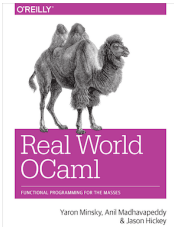
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

end-of-file condition, [Examples: An Echo Server](#)
 eprintf function, [Formatted Output with printf](#)
 equal (=) operator, [Choosing Between Maps and Hash Tables](#)
 equal equal (= =) operator, [Choosing Between Maps and Hash Tables](#)
 equality, tests of, [Choosing Between Maps and Hash Tables](#)
 error handling
 combining exceptions and error-aware types, [From Exceptions to Error-Aware Types and Back Again](#)
 error-aware return types, [Error Handling-bind and Other Error Handling Idioms](#)
 exception backtracing, [Backtraces](#)
 exception clean up, [Cleaning Up in the Presence of Exceptions](#)
 exception detection, [Catching Specific Exceptions](#)
 exception handlers, [Exception Handlers](#)
 exception helper functions, [Helper Functions for Throwing Exceptions](#)
 exceptions, [Exceptions](#)
 and imperative data structures, [Modifying the List](#)
 strategy choice, [Choosing an Error-Handling Strategy](#)
 (see also errors)
 error-aware return types, [Error Handling-bind and Other Error Handling Idioms, From Exceptions to Error-Aware Types and Back Again](#)
 Error.of_list, [Error and Or_error](#)
 Error.t type, [Error and Or_error](#)
 Error.tag, [Error and Or_error](#)
 errors
 catch-all cases and refactoring, [Catch-All Cases and Refactoring](#)
 compiler warnings, [Patterns and Exhaustiveness](#)
 cyclic dependencies, [Cyclic Dependencies](#)
 detecting with match statements, [Detecting Errors](#)
 detecting with type annotations, [Adding type annotations to find errors](#)
 error messages with s-expressions, [Getting Good Error Messages](#)
 "give up on first error" approach, [Bringing It All Together](#)
 missing field warnings, [Patterns and Exhaustiveness](#)
 missing module definitions, [Missing Definitions](#)
 module type definition mismatches, [Type Definition Mismatches](#)
 module type mismatches, [Type Mismatches](#)
 reducing verbosity in, [Shorter Module Paths in Type Errors](#)
 runtime vs. compile time, [Inferring Generic Types](#)
 syntax errors, [Syntax Errors](#)
 timeouts and cancellations, [Timeouts, Cancellation, and Choices](#)
 transformation of, [Error and Or_error](#)
 type errors, [Constructing JSON Values](#)
 (see also error handling)
 evaluation, order of, [Order of Evaluation](#)
 event loops, [Concurrent Programming with Async](#)
 exceptions
 asynchronous errors, [Exception Handling](#)
 benefits and drawbacks of, [Choosing an Error-Handling Strategy](#)
 catching specific, [Catching Specific Exceptions](#)
 and error-aware types, [From Exceptions to Error-Aware Types and Back Again](#)
 exception clean up, [Cleaning Up in the Presence of Exceptions](#)
 exception handlers, [Exception Handlers](#)
 helper functions for, [Helper Functions for Throwing Exceptions](#)
 in concurrent programming, [Exception Handling](#)
 search engine example, [Example: Handling Exceptions with DuckDuckGo](#)
 stack backtraces for, [Backtraces](#)
 textual representation of, [Exceptions](#)
 usefulness of, [Exceptions](#)
 vs. type errors, [Inferring Generic Types](#)
 exhaustion checks, [Catch-All Cases and Refactoring](#)
 Exn module
 Backtrace.Exn.set_recording false, [Backtraces](#)
 Exn.backtrace, [Backtraces](#)
 exn type, [Exceptions](#)
 explicit subtyping, [Static Type Checking](#)
 (see also subtyping)
 expressions, order of evaluation, [Order of Evaluation](#)
 extensible parsers, [Preprocessing Source Code](#)
 extensions (see syntax extensions)



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

external libraries

Camlimages, [When to Use Objects](#)

Cryptokit, [When to Use Objects](#)

for graphics, [Displaying the Animated Shapes](#)

interfacing with, [Foreign Functions](#)

external memory, [Managing External Memory with Bigarray](#)

F

Field module

Field.fset, [First-Class Fields](#)

Field.get, [First-Class Fields](#)

Field.name, [First-Class Fields](#)

Field.setter, [First-Class Fields](#)

fields

adding to structures, [Adding Fields to Structures](#)

field punning, [Records and Variants](#), [Field Punning](#)

first-class fields, [First-Class Fields-First-Class Fields](#)

mutability of, [Example: Imperative Dictionaries](#), [Mutable Record and Object Fields and Ref Cells](#)

reusing field names, [Reusing Field Names-Reusing Field Names](#)

fieldslib, [First-Class Fields](#), [Preprocessing Source Code](#)

FIFO (first-in, first-out) queue, [Extending Modules](#)

files

chart of file extensions, [Summarizing the File Extensions](#)

cmi files, [Defining a module search path](#), [Compiling Fast Native Code](#)

cmt files, [The Typed Syntax Tree](#)

cmtii files, [The Typed Syntax Tree](#)

cmx files, [Compiling Fast Native Code](#)

config files, [Specifying Defaults](#)

file I/O, [File I/O](#)

ml files, [Displaying Inferred Types from the Compiler](#)

mli files, [Signatures and Abstract Types](#), [Displaying Inferred Types from the Compiler](#)

mll files, [Defining a Lexer](#)

mly files, [Defining a Parser](#)

multi-file programs, [Multifile Programs and Modules](#)

o files, [Compiling Fast Native Code](#)

relationship with modules, [The mapping between files and modules](#)

single-file programs, [Single-File Programs-Single-File Programs](#)

filter_string function, [Selecting Values from JSON Structures](#)

finalizers

for C cleanup functions, [Custom Heap Blocks](#)

in garbage collection, [Attaching Finalizer Functions to Values](#)

find_exn function, [Exceptions](#)

first-class fields, [First-Class Fields-First-Class Fields](#)

first-class modules

alternatives to, [Living Without First-Class Modules](#)

vs. objects, [When to Use Objects](#)

polymorphism in, [Working with First-Class Modules](#)

query-handling framework, [Example: A Query-Handling Framework-Loading and Unloading Query Handlers](#)

type equality in, [Working with First-Class Modules](#)

working with, [First-Class Modules-Working with First-Class Modules](#)

first-fit allocation, [Memory Allocation Strategies](#)

flag functions, [Adding Labeled Flags to the Command Line](#)

flags, [Adding Labeled Flags to the Command Line](#), [Compiling ATD Specifications to OCaml](#), [Examining the Typed Syntax Tree Directly](#)

floating-point values, [Floating-Point Numbers and Arrays](#)

for loops, [For and While Loops](#), [for and while Loops](#)

foreign function interface (FFI)

basic scalar C types, [Basic Scalar C Types](#)

basics of, [Foreign Function Interface](#)

C bindings, [Learning More About C Bindings](#)

imperative operations and, [Foreign Functions](#)

passing functions to C, [Passing Functions to C](#)

pointers and arrays, [Pointers and Arrays](#)

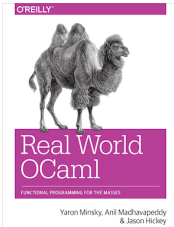
structs and unions, [Structs and Unions](#)

terminal interface example, [Example: A Terminal Interface-Example: A Terminal Interface](#)

format strings, [Formatted Output with printf](#)

Fortran libraries, [Managing External Memory with Bigarray](#)

fprintf function, [Formatted Output with printf](#)



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

frame pointers, [Perf](#)

fun keyword

anonymous functions, [Anonymous Functions](#)

currying syntax, [Multiargument functions](#)

multi-argument functions, [Multiargument functions](#)

function keyword, [Declaring Functions with Function](#), [Terse and Faster Patterns](#)

functional code, [Imperative Programming](#)

functional combinators, [Selecting Values from JSON Structures](#)

functional iterators, [Functional Iterators](#)

functional updates, [Functional Updates](#), [Catch-All Cases and Refactoring functions](#)

anonymous functions, [Records and Variants](#), [Anonymous Functions](#)

argument inference, [Inference of labeled and optional arguments](#)

curried functions, [Multiargument functions](#)

declaring with function keyword, [Declaring Functions with Function](#)

defining, [Functions and Type Inference](#), [Declaring Functions with Function](#)

exception warnings for, [Exceptions](#)

hash functions, [Hash Tables](#)

higher-order and labels, [Higher-order functions and labels](#)

iteration functions, [Iteration Functions](#)

labeled arguments, [Labeled Arguments](#)

member functions, [Selecting Values from JSON Structures](#)

multi-argument functions, [Multiargument functions](#)

name mangling of, [Understanding name mangling](#)

non-returning, [Examples: An Echo Server](#)

optional arguments, [Optional Arguments-Optional arguments and partial application](#)

passing to C, [Passing Functions to C](#)

prefix and infix operators, [Prefix and Infix Operators-Prefix and Infix Operators](#)

recursive functions, [Recursive Functions](#)

to_init function, [Selecting Values from JSON Structures](#)

to_string function, [Selecting Values from JSON Structures](#)

with multiple arguments, [Functions and Type Inference](#), [Labeled Arguments](#)

functors

basic mechanics of, [A Trivial Example](#)

benefits of, [Functors](#)

interval computation with, [A Bigger Example: Computing with Intervals-Using Multiple Interfaces](#)

module extension with, [Extending Modules](#)

G

garbage collection

and boxed values, [Distinguishing Integers and Pointers at Runtime](#)

finalizer functions, [Attaching Finalizer Functions to Values](#)

generational collection, [Generational Garbage Collection](#)

of longer-lived values, [The Long-Lived Major Heap](#)

mark and sweep collection, [Mark and Sweep Garbage Collection](#), [The Long-Lived Major Heap](#)

of allocated Ctypes, [Example: A Command-Line Quicksort](#)

opaque bytes and, [Blocks and Values](#)

of short-lived values, [The Fast Minor Heap](#)

Gc module, [Generational Garbage Collection](#)

(see also garbage collection)

gdb debugger, [Understanding name mangling](#)

generational garbage collection, [Generational Garbage Collection](#)

generational hypothesis, [Generational Garbage Collection](#)

geometric shapes, [Width Subtyping](#), [Create Some Simple Shapes](#)

GitHub API, [ATD Basics](#), [Example: Querying GitHub Organization Information](#)

GNU debugger, [Interactive breakpoints with the GNU debugger](#)

gprof code profiler, [Gprof](#)

grammars

avoiding grammar clashes, [Preprocessing Module Signatures](#)

context-free, [Describing the Grammar](#)

extension of standard language, [Preprocessing Source Code](#)

graphics libraries, [Displaying the Animated Shapes](#)

gray values, [Marking and Scanning the Heap](#)

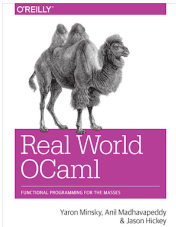
H

hash tables

basics of, [Maps and Hash Tables](#), [Hash Tables](#)

vs. maps, [Choosing Between Maps and Hash Tables](#)

polymorphic hash function, [Hash Tables](#)



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

Table of Contents

Prologue

I. Language Concepts

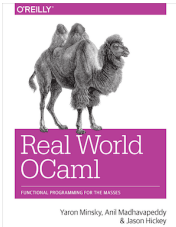
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

satisfying Hashable.S interface, [Satisfying the Hashable.S Interface](#)
time complexity of, [Hash Tables](#)
Hashable.Make, [Extending Modules](#), [Satisfying the Hashable.S Interface](#)
Hashable.S interface, [Satisfying the Hashable.S Interface](#)
Hashtbl module, [Hash Tables](#)
heaps
definition of, [Understanding the Garbage Collector](#)
heap blocks, [Understanding the Garbage Collector](#)
Heap_block module, [Attaching Finalizer Functions to Values](#)
major heaps, [The Long-Lived Major Heap-The mutable write barrier](#)
minor heaps, [The Fast Minor Heap](#)
regions of, [Generational Garbage Collection](#)
hex numbers, formatting with printf, [Formatted Output with printf](#)
higher-order functions, and labels, [Higher-order functions and labels](#)
Hindley-Milner algorithm, [Type Inference](#)
HTML generators, [Generating Documentation from Interfaces](#)
HTTP client queries, [Executing an HTTP Client Query](#)
I
I.Query_handler module, [Dispatching to Multiple Query Handlers](#)
I/O (input/output) operations
copying data, [Examples: An Echo Server](#)
file I/O, [File I/O](#)
formatted output, [Formatted Output with printf](#)
terminal I/O, [Input and Output](#)
identifiers
adding to modules, [Including Modules](#)
dealing with multiple, [Nested Modules](#)
open modules and, [Opening Modules](#)
imperative programming
arrays, [Arrays](#)
benefits of, [Imperative Programming](#)
benign effects and, [Laziness and Other Benign Effects](#)
doubly-linked lists, [Example: Doubly Linked Lists-Iteration Functions](#)
drawbacks of, [Modifying the List](#)
for and while loops, [For and While Loops](#)
imperative dictionaries, [Example: Imperative Dictionaries-Example: Imperative Dictionaries](#)
input and output, [Input and Output-File I/O](#)
mutable record fields, [Mutable Record Fields](#)
order of evaluation, [Order of Evaluation](#)
overview of, [Summary](#)
primitive mutable data, [Primitive Mutable Data](#)
ref type, [Refs](#)
side effects/weak polymorphism, [Side Effects and Weak Polymorphism-Relaxing the Value Restriction](#)
impure heaps, [Marking and Scanning the Heap](#)
infix operators, [Prefix and Infix Operators](#)
inheritance, [Inheritance](#), [Multiple Inheritance](#)
initializers, [Initializers](#)
install keyword, [Grouping Subcommands Together](#)
installation instructions, [Installation Instructions](#)
integers, [OCaml Blocks and Values](#), [Integers](#), [Characters](#), and [Other Basic Types](#)
interactive input
with camlp4, [Using Camlp4 Interactively](#)
concurrent programming for, [Concurrent Programming with Async](#)
prompts for, [Prompting for Interactive Input](#)
interfaces
Comparable.S, [Satisfying the Comparable.S Interface](#)
foreign function interface (FFI), [Foreign Function Interface-Struct Memory Layout](#)
generating documentation from, [Generating Documentation from Interfaces](#)
Hashable.S, [Satisfying the Hashable.S Interface](#)
hiding implementation details with, [Signatures and Abstract Types](#)
object types as, [Object Types as Interfaces](#)
with OCaml binaries, [Memory Representation of Values](#)
synonyms for, [Signatures and Abstract Types](#)
intergenerational pointers, [Intergenerational Pointers](#)
interval computation
abstract functor for, [Making the Functor Abstract](#)
comparison function for, [A Bigger Example: Computing with Intervals](#)



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

destructive substitution, [Destructive Substitution](#)

generic library for, [A Bigger Example: Computing with Intervals](#)

multiple interfaces and, [Using Multiple Interfaces](#)

sharing constraints, [Sharing Constraints](#)

invariance, [Variance](#)

invariant checks, [Preserving Invariants](#)

In_channel module, [Terminal I/O](#)

In_thread module, [Working with System Threads](#)

Iobuf module, [Managing External Memory with Bigarray](#)

irrefutable patterns, [Pattern Matching and let](#), [Patterns and Exhaustiveness](#)

iteration functions, [Iteration Functions](#)

iterators, [Object Types as Interfaces](#)

ivars, [Ivars and Upon](#)

J

-j-custom-fields FUNCTION, [Compiling ATD Specifications to OCaml](#)

-j-defaults, [Compiling ATD Specifications to OCaml](#)

-j-std flag, [Compiling ATD Specifications to OCaml](#)

JSON data

automatic mapping of, [Automatically Mapping JSON to OCaml Types-Example: Querying GitHub Organization Information](#)

basics of, [Handling JSON Data](#)

benefits and drawbacks of, [JSON Basics](#)

constructing values, [Constructing JSON Values](#)

nonstandard extensions for, [Using Nonstandard JSON Extensions](#)

parsing with Yojson, [Parsing JSON with Yojson](#)

selecting values from, [Selecting Values from JSON Structures-Selecting Values from JSON Structures](#)

Xen custom generator for, [Generating Documentation from Interfaces](#)

js_of_ocaml library, [Displaying the Animated Shapes](#)

"Just-in-Time" dynamic patching, [Memory Representation of Values](#), [Preprocessing Source Code](#)

K

kernel-level threads, [Working with System Threads](#)

key/value pairs, [Maps and Hash Tables](#), [JSON Basics](#)

L

label punning, [Labeled Arguments](#), [Field Punning](#)

labeled arguments, [The List module](#), [Labeled Arguments](#), [Inference of labeled and optional arguments](#), [Adding Labeled Arguments to Callbacks](#)

LablGL library, [Displaying the Animated Shapes](#)

Lablgtk library, [Displaying the Animated Shapes](#)

Lacaml library, [Managing External Memory with Bigarray](#)

lambda form code

basics of, [The Untyped Lambda Form](#)

pattern matching benchmarking, [Benchmarking Pattern Matching](#)

pattern matching optimization, [Pattern Matching Optimization](#)

LAPACK bindings, [Foreign Functions](#)

LAPACK mathematical library, [Managing External Memory with Bigarray](#)

late binding, [When to Use Objects](#)

laziness, [Laziness and Other Benign Effects](#)

lazy keyword, [Laziness and Other Benign Effects](#)

let () declaration, [Single-File Programs](#)

let rec, [Example: Doubly Linked Lists](#), [Memoization and Dynamic Programming](#)

let syntax

function definition with, [Functions and Type Inference](#)

functions and, [Anonymous Functions](#)

nested bindings, [Variables](#)

nested let binding, [Options](#)

nonrecursive vs. recursive functions, [Recursive Functions](#)

pattern matching, [Pattern Matching and let](#)

top-level bindings, [Variables](#)

wildcards in bindings, [Running Camlp4 from the Command Line](#)

Levenshtein distance, [Memoization and Dynamic Programming](#)

lexers

optional OCaml code for, [OCaml Prelude](#)

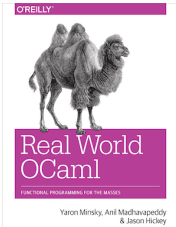
recursive rules, [Recursive Rules](#)

regular expressions collection, [Regular Expressions](#)

rules for, [Lexing Rules](#)

specification of, [Defining a Lexer](#)

Unicode parsing, [Recursive Rules](#)



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

Table of Contents

Prologue

I. Language Concepts

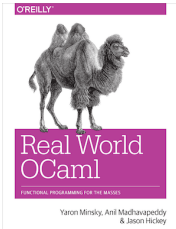
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

lexical analysis (lexing), [Lexing and Parsing](#)
 libasmrun.a library, [Embedding Native Code in C](#)
 libffi library, [Foreign Function Interface](#)
 libraries
 Camlimages, [When to Use Objects](#)
 Cryptokit, [When to Use Objects](#)
 for graphics, [Displaying the Animated Shapes](#)
 interfacing with external, [Foreign Functions](#)
 linear algebra, [Managing External Memory with Bigarray](#)
 linear mixins, [Mixins](#)
 -linkpkg, [Single-File Programs](#)
 List module
 creating tables with, [Using the List Module Effectively](#)
 List.append, [Combining lists](#)
 List.filter, [Filtering with List.filter and List.filter_map](#)
 List.fold, [Using the List Module Effectively](#)
 List.init, [Tail Recursion](#)
 List.map, [Using the List Module Effectively](#)
 List.map2_exn, [Using the List Module Effectively](#)
 List.partition_tf, [Partitioning with List.partition_tf](#)
 List.reduce, [Combining list elements with List.reduce](#)
 String.concat and, [Using the List Module Effectively](#)
 List.Assoc module
 List.Assoc.add, [Single-File Programs](#)
 List.Assoc.find, [Single-File Programs](#)
 List.dedup, [Prefix and Infix Operators](#)
 lists
 adding new bindings in, [Single-File Programs](#)
 association lists, [Maps and Hash Tables](#)
 combining, [Combining lists](#)
 combining elements in, [Combining list elements with List.reduce](#)
 computing length of, [Tail Recursion](#)
 doubly-linked lists, [Example: Doubly Linked Lists](#)
 duplicate removal, [Prefix and Infix Operators](#), [Filtering with List.filter and List.filter_map](#), [Terse and Faster Patterns](#)
 extension of, [List Basics](#)
 extracting data from, [Using Patterns to Extract Data from a List](#)
 filtering values in, [Filtering with List.filter and List.filter_map](#)
 finding key associations in, [Single-File Programs](#)
 generation of, [List Basics](#)
 List module, [Using the List Module Effectively-Combining lists](#)
 memory representation of, [Variants and Lists](#)
 operator : :, [Constructing lists with ::](#), [List Basics](#)
 partitioning elements in, [Partitioning with List.partition_tf](#)
 pattern matching, [List patterns using match](#)
 recursive list functions, [Recursive list functions](#)
 structure of, [List Basics](#)
 lit suffix, [Using Nonstandard JSON Extensions](#)
 local opens, [Opening Modules](#)
 locally abstract types, [Working with First-Class Modules](#)
 long values, [Generating Portable Bytecode](#)
 looping constructs, [for and while Loops](#)
 loop_forever, [Examples: An Echo Server](#)
 LR(1) grammars, [Describing the Grammar](#)
M
 macros, [Preprocessing Module Signatures](#)
 main function, [Single-File Programs](#)
 major heaps
 allocating on, [Allocating on the Major Heap](#)
 controlling collections, [Marking and Scanning the Heap](#)
 controlling growth of, [Allocating on the Major Heap](#)
 garbage collection in, [The Long-Lived Major Heap](#)
 heap compaction, [Heap Compaction](#)
 intergenerational pointers in, [Intergenerational Pointers](#)
 marking and scanning, [Marking and Scanning the Heap](#)
 memory allocation strategies, [Memory Allocation Strategies](#)
 malloc(3), [Allocating on the Major Heap](#)
 Map module



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

Map.of_alist_exn, [Creating Maps with Comparators](#)

Map.to_tree, [Trees](#)

mapping

complex values with views, [Using Views to Map Complex Values](#)

of JSON to OCaml types, [Automatically Mapping JSON to OCaml Types-Example: Querying GitHub Organization Information](#)

of OCaml types to runtime values, [Memory Representation of Values-Managing External Memory with Bigarray](#)

maps

basics of, [Maps and Hash Tables](#)

comparable.S interface, [Satisfying the Comparable.S Interface](#)

creating with comparators, [Creating Maps with Comparators](#)

vs. hashtables, [Choosing Between Maps and Hash Tables](#)

polymorphic comparison in, [The Polymorphic Comparator](#)

tree structure, [Trees](#)

mark and sweep garbage collection, [Mark and Sweep Garbage Collection](#), [The Long-Lived Major Heap](#)

match statements, [Using Patterns to Extract Data from a List](#), [Detecting Errors](#)

MD5 one-way cryptographic hash function, [Basic Command-Line Parsing](#)

member function, [Selecting Values from JSON Structures](#)

memoization

benefits and drawbacks of, [Memoization and Dynamic Programming](#)

example of, [Memoization and Dynamic Programming](#)

function of, [Memoization and Dynamic Programming](#)

memory

and allocated Ctypes, [Example: A Command-Line Quicksort](#)

allocation for pointers, [Allocating Typed Memory for Pointers](#)

layout for structs, [Struct Memory Layout](#)

major heap allocation strategies, [Memory Allocation Strategies](#)

managing external, [Managing External Memory with Bigarray](#)

memory management, [Understanding the Garbage Collector](#)

reducing fragmentation of, [First-fit allocation](#), [Heap Compaction](#)

strings vs. C character buffers, [Using Views to Map Complex Values](#)

(see also runtime memory representation)

Menhir parser generator

built-in rules of, [Parsing Sequences](#)

context-free grammars in, [Describing the Grammar](#)

error handling in, [Bringing It All Together](#)

invoking, [Parsing Sequences](#)

left-recursive definitions, [Parsing Sequences](#)

vs. ocaml yacc, [Parsing with OCamllex and Menhir](#)

methods

binary methods, [Binary Methods](#)

method overloading, [Binary Methods](#)

private methods, [Private Methods](#)

and virtual classes, [Virtual Classes and Methods](#)

minor heaps

allocating on, [Allocating on the Minor Heap](#)

garbage collection in, [The Fast Minor Heap](#)

setting size of, [Allocating on the Minor Heap](#)

mixin patterns, [Mixins](#)

ml files, [Displaying Inferred Types from the Compiler](#)

mli files, [Displaying Inferred Types from the Compiler](#)

mll files, [Defining a Lexer](#)

module keyword, [Working with First-Class Modules](#)

modules

basics of, [Multifile Programs and Modules](#)

benefits of, [Static Type Checking](#)

cyclic dependencies, [Cyclic Dependencies](#)

defining search paths, [Defining a module search path](#)

first-class modules, [First-Class Modules-Living Without First-Class Modules](#)

hiding implementation details, [Signatures and Abstract Types](#)

including, [Including Modules](#)

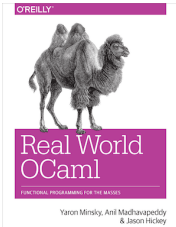
missing definitions in, [Missing Definitions](#)

module type, [Signatures and Abstract Types](#)

naming of, [Multifile Programs and Modules](#)

nested modules, [Nested Modules](#)

opening, [Opening Modules](#)



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

Table of Contents

Prologue

I. Language Concepts

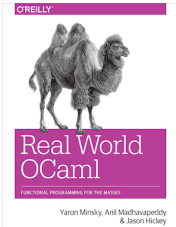
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

packing together, [Packing Modules Together](#)
 preprocessing signatures of, [Preprocessing Module Signatures](#)
 separate compilation in, [Modules and Separate Compilation](#)
 in static type checking, [Static Type Checking](#)
 type definition mismatches, [Type Definition Mismatches](#)
 type mismatches in, [Type Mismatches](#)
 type-specific functionality in, [Extending Modules](#)
 Monad.Make, [Extending Modules](#)
 monads, [Async Basics](#)
 monitors, [Monitors](#)
 multi-argument functions, [Functions and Type Inference](#), [Multiargument functions](#), [Labeled Arguments](#)
 multiple inheritance
 displaying animated shapes with, [Displaying the Animated Shapes](#)
 mixin pattern and, [Mixins](#)
 name resolution in, [Multiple Inheritance](#)
 mutable data, [Primitive Mutable Data](#)
 mutable record fields, [Mutable Record Fields](#), [Mutable Fields](#)
 mutexes, [Thread-Safety and Locking](#)
N
 name mangling, [Understanding name mangling](#)
 narrowing, [Narrowing](#), [Binary Methods](#)
 native-code compiler
 benefits of, [Compiling Fast Native Code](#)
 vs. bytecode compiler, [Single-File Programs](#)
 debugging binaries, [Debugging Native Code Binaries](#)
 embedding code in C, [Embedding Native Code in C](#)
 files generated by, [Summarizing the File Extensions](#)
 inspecting assembly output, [Inspecting Assembly Output](#)
 performance profiling, [Profiling Native Code](#)
 Ncurses terminal toolkit, [Foreign Function Interface](#)
 negation operators, [Prefix and Infix Operators](#)
 nested let binding, [Variables](#)
 nested modules, [Nested Modules](#)
 never_returns, [Examples: An Echo Server](#)
 next-fit allocation, [Memory Allocation Strategies](#)
 non-terminal symbols, [Describing the Grammar](#)
 numerical calculations, [OCaml as a Calculator](#)
O
 o files, [Compiling Fast Native Code](#)
 OAuth web protocol, [ATD Basics](#)
 Obj module, [Variants and Lists](#)
 object-oriented programming (OOP), [Objects](#)
 objects
 benefits and drawbacks of, [When to Use Objects](#)
 immutable, [Immutable Objects](#)
 narrowing and, [Narrowing](#)
 object types as interfaces, [Object Types as Interfaces](#)
 in object-oriented programming, [Objects](#)
 in OCaml, [OCaml Objects](#)
 polymorphism of, [Object Polymorphism](#)
 subtyping and, [Subtyping-Subtyping Versus Row Polymorphism](#)
 object_fields, [Parsing Sequences](#)
 OCaml
 benefits of, [Why OCaml?](#)
 code examples for, [Code Examples](#)
 Core standard library, [The Core Standard Library](#)
 history of, [A Brief History](#)
 installation instructions, [Installation Instructions](#)
 key features of, [Why OCaml?](#)
 numerical calculations in, [OCaml as a Calculator](#)
 operating system support, [Installation Instructions](#)
 third-party libraries for, [The OCaml Platform](#)
 OCaml toolchain
 benefits of, [Memory Representation of Values](#)
 ocaml-getopt, [Alternative Command-Line Parsers](#)
 ocamlbuild, [Single-File Programs](#)



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

ocamlc, [Single-File Programs](#), [The Compiler Frontend: Parsing and Type Checking](#), [Generating Portable Bytecode](#)

ocamldoc, [Generating Documentation from Interfaces](#)

ocamldoc-generators, [Generating Documentation from Interfaces](#)

ocamlfind, [Single-File Programs](#)

ocamllex, [Defining a Lexer](#), [Recursive Rules](#)

ocamlloginfo, [Defining a module search path](#)

ocamlpt, [Single-File Programs](#), [The Compiler Frontend: Parsing and Type Checking](#), [Compiling Fast Native Code](#)

ocamlrun, [Generating Portable Bytecode](#)

overview of, [An Overview of the Toolchain](#)

OCAMLRUNPARAM, [Generational Garbage Collection](#)

ocamlyacc parser generator, [Parsing with OCamllex and Menhir](#)

ocp-index, [Using ocp-index for Autocompletion](#)

OPAM package manager, [Grouping Subcommands Together](#), [Alternative Command-Line Parsers](#)

opaque bytes, [Blocks and Values](#)

open hashing, [Example: Imperative Dictionaries](#)

open object types, [Object Polymorphism](#)

open recursion, [When to Use Objects](#), [Open Recursion](#)

operators

: : operator, [Constructing lists with ::](#), [List Basics](#)

controlling pointers, [Defining Arrays](#)

negation operators, [Prefix and Infix Operators](#)

polymorphic comparison operators, [Terser and Faster Patterns](#)

prefix and infix operators, [Prefix and Infix Operators](#)

sequencing operators, [Prefix and Infix Operators](#)

subtraction operators, [Prefix and Infix Operators](#)

optional arguments

and default arguments, [Optional and Default Arguments](#)

explicit passing of, [Explicit passing of an optional argument](#)

inference of, [Inference of labeled and optional arguments](#)

options, [Options](#)

or patterns, [Recursive Functions](#)

order of evaluation, [Order of Evaluation](#)

Out_channel module

Out_channel.stderr, [Terminal I/O](#)

Out_channel.stdout, [Terminal I/O](#)

P

parallelism, [Working with System Threads](#)

parametric polymorphism, [Inferring Generic Types](#), [Refs](#)

parsing

extensible parsers, [Preprocessing Source Code](#)

lexer and parser composition, [Bringing It All Together](#)

lexer definition, [Defining a Lexer-Recursive Rules](#)

of source code, [Parsing Source Code-Generating Documentation from Interfaces](#)

parser definition, [Defining a Parser-Parsing Sequences](#)

parser generators, [Parsing with OCamllex and Menhir](#)

partial application, [Multiargument functions](#), [Prefix and Infix Operators](#), [Partial Application and the Value Restriction](#)

pattern matching

benchmarking of, [Benchmarking Pattern Matching](#)

catch-all cases, [Catch-All Cases and Refactoring](#), [Example: Terminal Colors Redux](#)

and exhaustiveness, [Patterns and Exhaustiveness](#)

extracting data with, [Using Patterns to Extract Data from a List-Detecting Errors](#)

fundamental algorithms in, [Pattern Matching Optimization](#)

and let, [Pattern Matching and let](#)

in lists, [List patterns using match](#)

optimization in lambda form code, [Pattern Matching Optimization](#)

terser and faster patterns, [Terser and Faster Patterns-Terser and Faster Patterns](#)

vs. lexing rules, [Lexing Rules](#)

performance profiling, [Profiling Native Code](#)

physical equality, [Choosing Between Maps and Hash Tables](#)

phys_equal function, [Choosing Between Maps and Hash Tables](#)

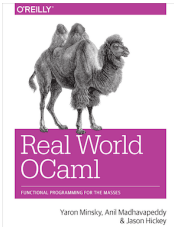
pipes, [Improving the Echo Server](#)

pointers

allocating typed memory for, [Allocating Typed Memory for Pointers](#)

intergenerational pointers, [Intergenerational Pointers](#)

operators controlling, [Defining Arrays](#)



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

Table of Contents

Prologue

I. Language Concepts

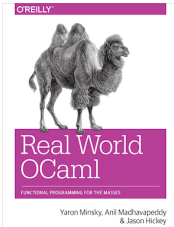
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

support for in Ctypes, [Pointers and Arrays](#)
 values for, [OCaml Blocks and Values](#)
 word-aligned, [Distinguishing Integers and Pointers at Runtime](#)
 polymorphic comparisons, [Terse and Faster Patterns](#), [The Polymorphic Comparator](#), [Using Camlp4 Interactively](#), [The impact of polymorphic comparison](#)
 polymorphic variant subtyping, [Depth Subtyping](#)
 polymorphic variant types
 automatic inference of, [Polymorphic Variants](#)
 basic syntax of, [Polymorphic Variants](#)
 and catch-all cases, [Example: Terminal Colors Redux](#)
 drawbacks of, [When to Use Polymorphic Variants](#)
 in JSON data, [Parsing JSON with Yojson](#)
 memory representation of, [Polymorphic Variants](#)
 vs. ordinary variants, [Example: Terminal Colors Redux](#), [When to Use Polymorphic Variants](#)
 type checking and, [Constructing JSON Values](#), [Adding type annotations to find errors](#)
 upper/lower bounds of, [Polymorphic Variants](#)
 polymorphism
 class parameters and, [Class Parameters and Polymorphism](#)
 in hash functions, [Hash Tables](#)
 in first-class modules, [Working with First-Class Modules](#)
 in objects, [Object Polymorphism](#)
 in locally abstract types, [Working with First-Class Modules](#)
 polymorphic equality, [Binary Methods](#)
 row polymorphism, [Object Polymorphism](#), [Subtyping Versus Row Polymorphism](#)
 weak polymorphism, [Side Effects and Weak Polymorphism](#)
 POSIX functions, [Pointers and Arrays](#)
 prefix operators, [Prefix and Infix Operators](#)
 pretty printers, [Basic Usage](#)
 primitive mutable data
 array-like data, [Primitive Mutable Data](#)
 foreign functions, [Foreign Functions](#)
 record/object fields and ref cells, [Mutable Record and Object Fields and Ref Cells](#)
 principal type checking, [Enforcing principal typing](#)
 printf function, [Formatted Output with printf](#)
 private methods, [Private Methods](#)
 product types, [Combining Records and Variants](#)
 profiling, [Profiling Native Code](#)
 programming
 concurrent programming with Async, [Concurrent Programming with Async](#)
 dynamic programming, [Memoization and Dynamic Programming](#), [Preprocessing Source Code](#)
 immutable vs. imperative, [Imperative Programming](#), [Imperative Programming](#)
 imperative programming, [Imperative Programming-Summary](#)
 language interfaces, [Foreign Function Interface](#)
 object-oriented programming (OOP), [Objects](#), [Classes](#)
 simple standalone example, [A Complete Program](#)
 programs
 multi-file programs, [Multifile Programs and Modules](#)
 single-file programs, [Single-File Programs-Single-File Programs](#)
 protected methods, [Private Methods](#)
 punning, [Labeled Arguments](#)
 pure code, [Imperative Programming](#), [Imperative Programming](#)
 pushback, [Examples: An Echo Server](#)
Q
 qsort binding, [Example: A Command-Line Quicksort](#)
 query-handlers
 dispatching to multiple, [Dispatching to Multiple Query Handlers](#)
 executing an HTTP client query, [Executing an HTTP Client Query](#)
 and first-class modules, [Example: A Query-Handling Framework](#)
 implementation of, [Implementing a Query Handler](#)
 loading/unloading of, [Loading and Unloading Query Handlers](#)
R
 Random module, [For and While Loops](#)
 Reader module, [Examples: An Echo Server](#)
 rec keyword, [Recursive Functions](#)
 record field accessor functions, [First-Class Fields](#)
 records
 basic syntax for, [Records](#)
 construction of, [Field Punning](#)



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

Table of Contents

Prologue

I. Language Concepts

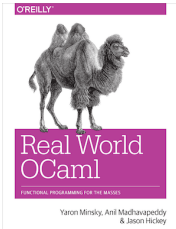
II. Tools and Techniques

III. The Runtime System

[Index](#)

Login with GitHub to view
and add comments

field punning in, [Field Punning](#)
 first-class fields in, [First-Class Fields-First-Class Fields](#)
 functional updates to, [Functional Updates](#)
 label punning in, [Field Punning](#)
 memory representation of, [Tuples, Records, and Arrays](#)
 missing field warnings, [Patterns and Exhaustiveness](#)
 mutable fields in, [Mutable Fields](#)
 patterns and exhaustiveness in, [Patterns and Exhaustiveness](#)
 record types, [Records and Variants](#), [Refs](#)
 reusing field names, [Reusing Field Names-Reusing Field Names](#)
 and variant types, [Combining Records and Variants-Combining Records and Variants](#)
 recursion
 in lexers, [Recursive Rules](#)
 in json types, [Parsing JSON with Yojson](#)
 open recursion, [When to Use Objects, Open Recursion](#)
 tail recursion, [Tail Recursion](#)
 recursive data structures, [Variants and Recursive Data Structures](#)
 recursive functions
 definition of, [Recursive Functions](#)
 list functions, [Recursive list functions](#)
 ref cells, [Ref cells](#)
 refactoring, [Catch-All Cases and Refactoring](#), [Composing Specification Fragments Together](#)
 regular expressions, [Regular Expressions](#)
 remembered sets, [Intergenerational Pointers](#)
 remote keyword, [Grouping Subcommands Together](#)
 representation types, [Binary Methods](#)
 Result.t option, [Encoding Errors with Result](#)
 return function, [Async Basics](#)
 returning function, [Recap: A time-printing command](#)
 rev_object_fields, [Parsing Sequences](#)
 RFC3986, [URI Handling](#)
 RFC4627, [JSON Basics](#), [Parsing JSON Strings](#)
 risky type, [Enforcing principal typing](#)
 root values, [Marking and Scanning the Heap](#)
 row polymorphism, [Object Polymorphism](#), [Subtyping Versus Row Polymorphism](#), [Adding type annotations to find errors](#)
 runtime exceptions vs. type errors, [Inferring Generic Types](#)
 runtime memory representation
 blocks and values, [Blocks and Values](#)
 custom heap blocks, [Custom Heap Blocks](#)
 importance of, [Memory Representation of Values](#)
 polymorphic variants, [Polymorphic Variants](#)
 string values, [String Values](#)
 tuples, records, and arrays, [Tuples, Records, and Arrays](#)
 variants and lists, [Variants and Lists](#)
S
 s-expressions
 basic usage of, [Basic Usage](#)
 deserializing a type from, [Getting Good Error Messages](#)
 example of, [Error and Or_error](#), [Using Multiple Interfaces](#)
 format of, [The Sexp Format](#)
 generating from OCaml types, [Generating S-Expressions from OCaml Types](#)
 modifying default behavior of, [Sexp-Conversion Directives](#)
 preserving invariants in, [Preserving Invariants](#)
 in queries and responses, [Example: A Query-Handling Framework](#)
 specifying defaults in, [Specifying Defaults](#)
 uses for, [Data Serialization with S-Expressions](#)
 scalar C types, [Basic Scalar C Types](#)
 Scheduler.go, [Examples: An Echo Server](#)
 scope, [Variables](#)
 search engines, [Example: Searching Definitions with DuckDuckGo](#)
 security issues
 denial-of-service attacks, [Hash Tables](#)
 Obj module warning, [Variants and Lists](#)
 segfaults, [Thread-Safety and Locking](#)
 semantic actions, [Describing the Grammar](#)
 semicolons vs. commas, [Constructing lists with ::](#)
 serialization formats



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

JSON, [Handling JSON Data-Example: Querying GitHub Organization Information](#)

s-expressions, [Data Serialization with S-Expressions-Specifying Defaults](#)

set types, [Sets](#)

sexp declaration, [Exceptions](#), [Using Multiple Interfaces](#), [Example: A Query-Handling Framework](#)
Sexplib package

sexp converter, [Error and Or_error](#), [Creating Maps with Comparators](#), [Preprocessing Source Code](#)

sexp_list, [sexp_list](#)

sexp_opaque, [sexp_opaque](#)

sexp_option, [sexp_option](#)

syntax extension in, [Generating S-Expressions from OCaml Types](#)

Type_conv library and, [Generating S-Expressions from OCaml Types](#)

shadowing, [Variables](#)

sharing constraint, [Sharing Constraints](#)

short paths heuristic, [Shorter Module Paths in Type Errors](#)

side effects, [Laziness and Other Benign Effects](#), [Side Effects and Weak Polymorphism](#)

signatures

abstract types, [Signatures and Abstract Types](#)

concrete types, [Concrete Types in Signatures](#)

preprocessing module signatures, [Preprocessing Module Signatures](#)

source code

automatically indenting, [Automatically Indenting Source Code](#)

parsing of, [Parsing Source Code-Generating Documentation from Interfaces](#)

preprocessing of, [Preprocessing Source Code-Further Reading on Camlp4](#)

sprintf function, [Formatted Output with printf](#)

stack backtraces, [Backtraces](#)

stack frames, [Tail Recursion](#)

start symbols, [Describing the Grammar](#)

static checking, [JSON Basics](#), [The Compiler Frontend: Parsing and Type Checking](#)

static dispatch, [Virtual Classes and Methods](#)

strict evaluation, [Order of Evaluation](#)

string matching, [Memoization and Dynamic Programming](#)

String.concat, [Using the List Module Effectively](#)

strings

vs. C character buffers, [Using Views to Map Complex Values](#)

concatenation of, [Using the List Module Effectively](#)

format strings, [Formatted Output with printf](#)

memory representation of, [String Values](#)

padding of, [Using the List Module Effectively](#)

vs. Char.t arrays, [Strings](#)

structs and unions

array definition, [Defining Arrays](#)

field addition, [Adding Fields to Structures](#)

incomplete structure definitions, [Incomplete Structure Definitions](#)

memory layout of, [Struct Memory Layout](#)

structure definition, [Structs and Unions](#)

time-printing command, [Recap: A time-printing command](#)

structural equality, [Choosing Between Maps and Hash Tables](#)

subcommands, grouping of, [Grouping Subcommands Together](#)

subtraction operators, [Prefix and Infix Operators](#)

subtyping

basics of, [Subtyping](#)

depth subtyping, [Depth Subtyping](#)

vs. row polymorphism, [Subtyping Versus Row Polymorphism](#)

in static type checking, [Static Type Checking](#)

variance and, [Variance-Variance](#)

width subtyping, [Width Subtyping](#)

sum types, [Combining Records and Variants](#)

syntax errors, [Syntax Errors](#)

syntax extension

building new, [Further Reading on Camlp4](#)

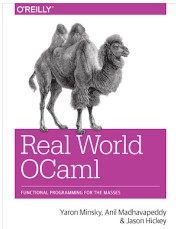
in Camlp4, [Generating S-Expressions from OCaml Types](#), [Preprocessing Source Code-Further Reading on Camlp4](#)

in Sexplib package, [Generating S-Expressions from OCaml Types](#)

potential overuse of, [Preprocessing Module Signatures](#)

system threads, [Concurrent Programming with Async](#), [Working with System Threads-Thread-Safety and Locking](#)

(see also threads)



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

T

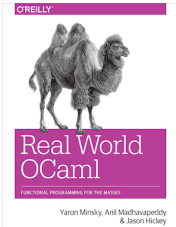
tab-autocompletion, [Command-Line Autocompletion with bash](#)
 tables, creating with List module, [Using the List Module Effectively](#)
 tail calls, [Tail Recursion](#), [Examples: An Echo Server](#)
 tail recursion, [Tail Recursion](#)
 TCP clients/servers, [Examples: An Echo Server](#)
 textwrap library, [Example: Searching Definitions with DuckDuckGo](#)
 threads
 benefits of, [Working with System Threads](#)
 kernel-level threads, [Working with System Threads](#)
 locking and, [Thread-Safety and Locking](#)
 preemptive vs. single-threaded programs, [Concurrent Programming with Async](#)
 thread-safety, [Thread-Safety and Locking](#)
 turning on with -thread, [Single-File Programs](#)
 thunks, [Error and Or_error](#), [Laziness and Other Benign Effects](#), [Ivars and Upon](#)
 timeouts, [Timeouts](#), [Cancellation](#), and [Choices](#)
 tokens, declaration of, [Defining a Parser](#)
 top-level bindings, [Variables](#)
 top-level printers, [Basic Usage](#)
 to_init function, [Selecting Values from JSON Structures](#)
 to_string function, [Selecting Values from JSON Structures](#)
 tuples, [Tuples](#), [Using Nonstandard JSON Extensions](#), [Tuples, Records, and Arrays](#)
 type annotations, [Inference of labeled and optional arguments](#), [Adding type annotations to find errors](#)
 type checking, [Constructing JSON Values](#), [Memory Representation of Values](#)
 type definition mismatches, [Type Definition Mismatches](#)
 type errors vs. exceptions, [Inferring Generic Types](#)
 type inference
 algorithm basis of, [Type Inference](#)
 benefits of, [Constructing JSON Values](#)
 drawbacks of, [Static Type Checking](#)
 error detection with, [Adding type annotations to find errors](#)
 generic types, [Inferring Generic Types](#)
 principality checks, [Enforcing principal typing](#)
 process of, [Type Inference](#)
 in static type checking, [Static Type Checking](#)
 type mismatches, [Type Mismatches](#)
 type variables, [Inferring Generic Types](#), [Side Effects and Weak Polymorphism](#), [Object Polymorphism](#)
 typed syntax tree, [The Typed Syntax Tree-Examining the Typed Syntax Tree Directly](#)
 Type_conv library, [Generating S-Expressions from OCaml Types](#)

U

Ulex lexer generator, [Recursive Rules](#)
 unaligned memory access, [Distinguishing Integers and Pointers at Runtime](#)
 unboxed integers, [Integers, Characters, and Other Basic Types](#)
 Unicode, parsing solutions for, [Recursive Rules](#)
 Uniform Resource Identifiers (URIs), [URI Handling](#)
 unions (see structs and unions)
 unit argument, [Defining Basic Commands](#)
 unit tests, [JSON Basics](#)
 uri library, [Example: Searching Definitions with DuckDuckGo](#)
 -use-menhir flag, [Parsing Sequences](#)
 Utf Unicode codec, [Recursive Rules](#)

V

value restriction, [The Value Restriction](#)
 values
 allocation requests and, [Understanding the Garbage Collector](#)
 boxing of, [Distinguishing Integers and Pointers at Runtime](#)
 copying with Array.blit, [Ordinary arrays](#)
 filtering with List.filter, [Filtering with List.filter and List.filter_map](#)
 finalizer functions for, [Attaching Finalizer Functions to Values](#)
 integer vs. pointer, [OCaml Blocks and Values](#)
 in JSON data, [JSON Basics](#), [Constructing JSON Values](#), [Describing the Grammar](#)
 mapping complex with views, [Using Views to Map Complex Values](#)
 memory representation of, [Memory Representation of Values-Managing External Memory with Bigarray](#)
 selecting from JSON structures, [Selecting Values from JSON Structures-Selecting Values from JSON Structures](#)



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

stored by bytecode compiler, [Generating Portable Bytecode](#)

variables

immutability of, [Variables](#)

pattern matching in, [Pattern Matching and let](#)

scope of, [Variables](#)

shadowing of, [Variables](#)

variance, [Variance-Variance](#)

variant types

basic syntax of, [Variants](#)

catch-all cases and refactoring, [Catch-All Cases and Refactoring](#)

combining multiple object types with, [Records and Variants](#)

memory representation of, [Variants and Lists](#)

polymorphic, [Polymorphic Variants-When to Use Polymorphic Variants](#)

and records, [Combining Records and Variants-Combining Records and Variants](#)

and recursive data structures, [Variants and Recursive Data Structures](#)

usefulness of, [Variants](#)

Yojson support for, [Using Nonstandard JSON Extensions](#)

virtual classes, [Virtual Classes and Methods](#)

virtual methods, [Virtual Classes and Methods](#)

W

weak polymorphism, [Side Effects and Weak Polymorphism](#)

while loops, [For and While Loops](#), [for and while Loops](#)

whitespace-sensitive indentation, [Preprocessing Module Signatures](#)

width subtyping, [Width Subtyping](#)

wildcards, [Running Camlp4 from the Command Line](#)

word-aligned pointers, [Distinguishing Integers and Pointers at Runtime](#)

write barriers, [Intergenerational Pointers](#)

Writer module, [Examples: An Echo Server](#)

X

Xen, [Generating Documentation from Interfaces](#)

Y

Yojson library

combinators in, [Selecting Values from JSON Structures](#)

extended JSON format support, [Using Nonstandard JSON Extensions](#)

installation of, [JSON Basics](#)

parsing JSON with, [Parsing JSON with Yojson](#), [Parsing JSON Strings](#)

[< Previous](#)

[Next >](#)