

---

## Hands-on Scala Programming

This repository is the online hub for the book *Hands-on Scala Programming*:

- Chapter Notes, Errata, and Discussion: go here if you want to leave comments or ask questions about individual chapters
- Code Snippets: copy-paste friendly versions of every code snippet in the book
- Chapter Resource Files: files used as part of the book's programming exercises
- Executable Code Examples: self-contained executable programs for the topics being presented in each chapter.

### Executable Code Examples

The executable code examples from *Hands-on Scala* are freely available online, and open source under an MIT license. These examples are meant to be useful to anyone who is interested in learning Scala, whether or not they are working through *Hands-on Scala Programming*, though following along with the book will give you the best experience and help you get the most out of them.

Each example is:

- **Self-contained:** to be run either using the Ammonite script runner or Mill build tool, with no other setup
- **Tested:** with simple test suites provided and instructions on how to run each example in that folder's [readme.md](#) file
- **Executable:** you can download the folder and run the example yourself to see it in action

Many of the examples are related, with only small code changes between them to illustrate a new feature or technique. The [readme.md](#) file of such downstream examples will show a diff from the upstream example that it was based upon, so you can focus your attention on the important code changes that are happening, with a link back up to the upstream example.

The executable code examples below are organized by each part and each chapter of the book. ## Part I Introduction to Scala ### Chapter 1: Hands-on Scala ### Chapter 2: Setting Up ### Chapter 3: Basic Scala - 3.1 - Values: Using Numbers, Strings, Options and Arrays - 3.2 - LoopsConditionals: Using Loops, If-Else statements and expressions, and List Comprehensions - 3.3 - MethodsFunctions: Basic Usage of Methods and Function Values - 3.4 - ClassesTraits: Basic usage of Classes and Traits - 3.5 - FlexibleFizzBuzz: Implementation of FizzBuzz that takes a callback used to handle the generated strings - 3.6 - PrintMessages: A method to render a flat array of `Msg` instances into a “threaded”

---

conversation with child messages printed nested under their parents - 3.7 - ContextManagers: Methods that act as Python “context managers”, opening a file reader/writer, passing the reader/writer to a callback, and closing the file after. ### Chapter 4: Scala Collections - 4.1 - BuildersFactories: Builders and Factory Methods to efficiently construct collections - 4.2 - Transforms: Various ways of transforming collections - 4.3 - QueriesAggregations: Querying collections to find elements within, and aggregating data across an entire collection - 4.4 - Combining: Combining different collection operations together - 4.5 - ConvertersViews: Converting between collections, and using views to avoid allocating intermediate collections - 4.6 - ImmutableVectors: Immutable Vectors, convenient ordered indexed collections - 4.7 - ImmutableSets: Immutable Sets, unordered collections with uniqueness of each item - 4.8 - ImmutableMaps: Immutable Maps, unordered collections of key-value pairs - 4.9 - ImmutableList: Immutable Lists, ordered collections with fast operations at the front - 4.10 - MutableArrayDeque: Mutable Array Deques, fast mutable ordered indexed collection with adding and removal of elements at both ends - 4.11 - MutableSets: Mutable Sets, mutable unordered collections with uniqueness of elements - 4.12 - MutableMaps: Mutable Maps, mutable unordered collections of key-value pairs - 4.13 - InPlaceOperations: In-place operations for conveniently and efficiently modifying mutable collections - 4.14 - CommonInterfaces: Using common interfaces to write code that can work across multiple collections - 4.15 - PartialValidSudoku: Checks whether a partially-filled Sudoku grid is valid so far, with 0s representing empty squares - 4.16 - RenderSudoku: Pretty-prints a Sudoku grid ### Chapter 5: Notable Scala Features - 5.1 - CaseClass: Basic usage of **case class** features - 5.2 - SealedTrait: Basic usage of **sealed traits** - 5.3 - PatternMatching: Demonstration of pattern matching - 5.4 - ByName: Use cases for by-name method parameters - 5.5 - ImplicitParameters: Implicit parameters for dependency injection - 5.6 - TypeclassInference: Use case for typeclass inference and recursive typeclass inference - 5.7 - Simplify: Using pattern matching to perform algebraic simplifications on simple equations - 5.8 - Backoff: A version of our by-name **def retry** method with configurable exponential backoff - 5.9 - Deserialize: Using recursive typeclass inference to parse and de-serialize arbitrarily-deep data structures from a JSON-like syntax - 5.10 - Serialize: Using recursive typeclass inference to parse and de-serialize arbitrarily-deep data structures from a JSON-like syntax ## Part II Local Development ### Chapter 6: Implementing Algorithms in Scala - 6.1 - MergeSort: Simple merge-sort implementation, hard-coded to only work on **Array[Int]** - 6.2 - GenericMergeSort: Generic merge sort implementation, that can sort any **IndexedSeq[T]** with an **Ordering** - 6.3 - Trie: A simple mutable Trie implementation - 6.4 - Search: Breadth-first-search implementation - 6.5 - SearchPaths: Breadth-first-search implementation that keeps track of the shortest paths to every node - 6.6 - BinarySearch: Binary search implementation - 6.7 - ImmutableTrie: Immutable trie implementation - 6.8 - DepthSearchPaths: Depth-first search implementation that keeps track of shortest paths - 6.9 - Sudoku: Sudoku solver implemented via depth-first search - 6.10 - TrieMap: Using a trie to implement a **Map[String, T]**-like data structure - 6.11 - FloodFill: Breadth-first implementation of a floodfill algorithm, taking a callback that allows the user to specify what color pairs are similar enough to traverse ### Chapter 7: Files and Subprocesses - 7.1 - LargestFiles: Script to

---

---

finds the largest 5 files in the current folder tree - 7.2 - FileSync: Method to batch synchronize files between two local folders - 7.3 - RemoveBranches: Script to remove all non-active branches from the local Git repository - 7.4 - InteractivePython: Example of how to interact with a running Python process from Scala - 7.5 - StreamingDownloadProcessReupload1: Three-stage streaming subprocess pipeline - 7.6 - StreamingDownloadProcessReupload2: Four-stage streaming subprocess pipeline - 7.7 - FileSyncDelete: Method to synchronize files between two folders, with support for deletion - 7.8 - StreamingDownloadProcessReuploadTee: Five stage subprocess pipeline, [teed](#) streaming data to a file

### Chapter 8: JSON and Binary Data Serialization - 8.1 - Create: Parsing JSON structures from strings and constructing it directly - 8.2 - Manipulate: Querying and modifying JSON structures in-memory - 8.3 - Traverse: Recursively traversing a JSON structure - 8.4 - SerializationBuiltins: Serializing built-in Scala data types to JSON - 8.5 - SerializationCaseClass: Serializing Scala case classes to JSON - 8.6 - SerializationBinary: Serializing Scala data types to binary MessagePack blobs - 8.7 - BiMap-Class: Using [bimap](#) to make a non-case class serializable - 8.8 - TraverseFilter: Recursively traversing a JSON structure

### Chapter 9: Self-Contained Scala Scripts - 9.1 - Printing: Listing blog-post-like files in a folder and printing them out - 9.2 - Index: Rendering an [index.html](#) for our static blog using Scalatags - 9.3 - Markdown: Rendering individual blog posts using Atlassian's Commonmark-Java library - 9.4 - Links: Adding links between our [index.html](#) and the individual blog posts - 9.5 - Bootstrap: Prettifying our static blog using the Bootstrap CSS framework - 9.6 - Deploy: Optionally deploying our static blog to a Git repository - 9.7 - DeployTimestamp: Displaying the [.md](#) file last-modified time on each blog post

### Chapter 10: Static Build Pipelines - 10.1 - Simple: Simple linear build pipeline - 10.2 - Nonlinear: Simple non-linear build pipeline with two branches - 10.3 - Modules: Simple non-linear build pipeline, replicated in several modules - 10.4 - NestedModules: Nested modules in a Mill build pipeline - 10.5 - CrossModules: Using cross-modules to dynamically construct a build graph based on the filesystem - 10.6 - Blog: Our static blog generator, converted into an incremental Mill build pipeline - 10.7 - ExtendedBlog: Adding previews and bundled Bootstrap CSS to our static blog build pipeline - 10.8 - Push: Re-adding the ability to deploy our static blog by defining a [T.command](#) - 10.9 - PostPdf: Static blog pipeline which can generate PDFs for each blog post using Puppeteer - 10.10 - ConcatPdf: Static blog pipeline which can generate PDFs for each blog post and concatenate them using Apache PDFBox

## Part III Web Services ### Chapter 11: Scraping Websites - 11.1 - ScrapingWiki: Scraping headlines off the Wikipedia front page using Jsoup - 11.2 - ScrapingDocs: Scraping semi-structured content off of the Mozilla Development Network Web API docs - 11.3 - ApiStatus: Using Jsoup to scrape the status annotations for every Web API on MDN - 11.4 - ExternalLinks: Crawling the pages of a static website to extract every external link on the site - 11.5 - ScrapingTrees: Using Jsoup to scrape the tree-shaped comments of a threaded discussion forum

### Chapter 12: Working with HTTP APIs - 12.1 - IssueMigrator: Simple Github issue migrator - 12.2 - IssueMigratorLink: Github issue migrator that adds a link back from every new issue to the original issue - 12.3 - IssueMigratorClosed: Github issue migrator that also preserves issue open/closed status during the migration

### Chapter 13: Fork-Join Parallelism with Futures - 13.1 - Hashing: Hashing files in parallel using Futures - 13.2

---

- Crawler: Simple sequential wikipedia crawler - 13.3 - ParallelCrawler: Simple batch-by-batch parallel wikipedia crawler, using Futures and Requests-Scala - 13.4 - RecursiveCrawler: Parallel wikipedia crawler written in a recursive fashion - 13.5 - AsyncCrawler: Asynchronous parallel wikipedia crawler, using the Java AsyncHttpClient - 13.6 - ParallelScrapingDocs: Scraping MDN Web API documentation pages in parallel using Futures - 13.7 - ParallelMergeSort: Merge-sort implementation parallelized using Futures - 13.8 - AsyncCrawlerThrottled: Asynchronous wikipedia crawler which limits the number of open requests - 13.9 - AsyncThrottledScrapingDocs: Asynchronous parallel MDN scraper that limits the number of open requests

### Chapter 14: Simple Web and API Servers - 14.1 - Mock: Dummy non-interactive HTML chat website and server - 14.2 - Forms: Form-based interactive chat website - 14.3 - Ajax: Ajax-based interactive chat website - 14.4 - Websockets: Websocket-based real-time chat website - 14.5 - WebsocketsFilter: Websocket-based chat website with ability to filter chats - 14.6 - WebsocketsSynchronized: Websocket-based chat website with shared mutable state properly synchronized - 14.7 - WebsocketsJsoup: Websocket-based chat website with HTML validation tests using Jsoup - 14.8 - CrawlerWebsite: Website that performs parallel crawls of the Wikipedia article graph on behalf of the user.

### Chapter 15: Querying SQL Databases - 15.1 - Queries: Simple Quill database queries - 15.2 - Website: Database-backed websocket chat website - 15.3 - FancyQueries: Fancier Quill database queries with `groupBy` and `sortBy` - 15.4 - WebsiteTimestamps: Database-backed websocket chat website with timestamps on every chat message - 15.5 - ThreadedChat: Database-backed websocket threaded-chat website - 15.6 - ListenNotify: Database-backed websocket chat website that uses the Postgres `LISTEN/NOTIFY` feature to propagate push notifications across all connected websockets

## Part IV Program Design

### Chapter 16: Message-based Parallelism with Actors - 16.1 - Simple: Simple logging actor that asynchronously uploads log messages to `httpbin.org` servers - 16.2 - Batch: Batch logging actor that uploads batches of log messages to `httpbin.org` servers - 16.3 - StateMachine: Actor that uploads batches of logs to `httpbin.org`, with a minimum interval between each batch - 16.4 - LoggingSimple: Logging actor that asynchronously logs messages to disk - 16.5 - LoggingPipeline: Two-stage logging actor pipeline that asynchronously logs base64-encoded messages to disk - 16.6 - LoggingLongPipeline: Four-actor pipeline that logs sanitized, base64-encoded messages both to disk and to `httpbin.org` - 16.7 - LoggingRearrangedPipeline1: Three-actor logging pipeline without sanitization - 16.8 - LoggingRearrangedPipeline2: Four-actor pipeline, with only disk logs base64-encoded and only `httpbin.org` uploads sanitized - 16.9 - WebCrawler: A concurrent actor-based web crawler that avoids the limitations of earlier batch-by-batch crawlers - 16.10 - WebCrawlerPipeline: A concurrent actor-based web crawler that streams the crawled pages to disk - 16.11 - WebCrawlerThrottled: A concurrent actor-based web crawler that limits the number of open connections

### Chapter 17: Multi-Process Applications - 17.1 - Main: Simple two-process batch file synchronizer that could work over a network - 17.2 - FileSyncer: Simple two-process batch file synchronizer that could work over a network - 17.3 - Pipelined: Pipelined version of our two-process file synchronizer, minimizing the chattiness of the protocol - 17.4 - Deletes: Pipelined two-process file synchronizer that supports deletions - 17.5 - Gzip: File syncer with compressed data exchange between

---

---

the Sync and Agent processes - 17.6 - Ssh: Networked of our two-process file synchronizer, running the agent on a separate computer and interacting with it over SSH ### Chapter 18: Building a Real-time File Synchronizer - 18.1 - Simple: Simple real-time file synchronizer that uses `os.watch` to react to filesystem changes as they happen - 18.2 - Pipelined: Pipelined real-time file synchronizer, allowing RPCs and hashing to take place in parallel - 18.3 - InitialFiles: Real-time file synchronizer that supports syncing an initial set of files - 18.4 - ForkJoinHashing: Real-time file synchronizer that does hashing of files in parallel - 18.5 - Deletes: Real-time file synchronizer supporting deletion of files - 18.6 - Virtual-FileSystem: Real-time file synchronizer that keeps track of synced files, avoiding RPCs for destination file hashes ### Chapter 19: Parsing Structured Text - 19.1 - Phrases: Parser for simple “hello world”-like phrases - 19.2 - Arithmetic: Parser and evaluator for simple english-like arithmetic expressions - 19.3 - ArithmeticChained: English-like arithmetic parser, allowing more than one binary operator to be chained together - 19.4 - ArithmeticPrecedence: English-like arithmetic parser, supporting operator precedence of chained binary operators - 19.5 - ArithmeticDirect: English-like arithmetic parser that evaluates the arithmetic without constructing a syntax tree ### Chapter 20: Implementing a Programming Language - 20.1 - Parse: FastParse parser and syntax tree for Jsonnet - 20.2 - Evaluate: Evaluating Jsonnet syntax tree into Values - 20.3 - Serialize: Serializing Jsonnet Values into an output JSON string - 20.4 - Jsonnet: Interpreter for a minimal subset of the Jsonnet programming language - 20.5 - JsonnetNumbers: Jsonnet interpreter with added support for simple integer arithmetic - 20.6 - Jsonnet-PrettyPrinting: Jsonnet interpreter using the `uJson` library for pretty-printing the output JSON - 20.7 - JsonnetStackTraces: Jsonnet interpreter which provides line and column numbers when things fail