To discuss this course and customizations:
Call: +1 434-509-5680 or Email: sales@training4programmers.com

Zig Essentials

Duration

3 days

Description

The Zig Essentials class teaches students how to write software using the Zig programming language. Zig is a general-purpose programming language and toolchain for creating robust, optimal, and reusable software. Zig is a compiled, lower-level, hardware-friendly, system-level language. Zig improves on C through customized memory control, null reference protection, and required error handling. Unlike Rust, Zig allows low-level memory control with syntax and memory model features to help avoid memory leaks. Finally, Zig provides interoperability with existing C libraries. This class focuses on learning the data types, control flow structures, code organization, memory management, and other Zig language features.

Objectives

- Learn how to build and run Zig programs.
- Explore features unique to Zig that set it apart from other languages.
- Understand where Zig is a good choice for writing software.
- Practicing using the Zig toolchain.
- Explore how Zig can be used as a drop-in replacement for C.
- Experience how Zig enables Performance and Safety.
- Apply modern techniques of memory control, null reference handling, and error handling with a lower-level language.

Prerequisites

All students must have programming experience.

Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub in the form of documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Student will need permission to install Zig, GCC, and Visual Studio Code on their computers. Also, students will

need permission to install Visual Studio Extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

Outline

- Introduction
  - What is Zig?
  - What problems does Zig solve?
  - Zig compared to C
  - Zig compared to other Languages
  - Zig Zen
- Getting Started
  - Zig Toolchain
  - Hello, Zig!
  - Code and Debug with VSCode
  - Zig Standard Library
  - Zig Source Files
  - Cross Compilation
- Language Features in Hello Zig
  - Importing from the Standard Library
  - Constants
  - Define a Public “main” Function
  - Try Statement
  - Error Union Types
  - String Interpolation
  - Comments
- Zig Project Scaffolding
  - Create a New Executable Project
  - Create a New Library Project
  - Build and Run
  - Build and Test
- Console Apps
  - Print Output to the Terminal
  - Format Specifiers
  - Anonymous Struct Literals
  - Capture Input from the Terminal
  - String Comparison
  - While Loop

  - Error Handling
- Data Types
  - Integers
  - Floats
  - Arrays
  - Pointers
  - Slices
- Data Structures
  - Struct
  - Enum
  - Union
- Variables
  - Variable Name Rules
  - Container Level Variables
  - Compile-Time vs. Run-Time Variables
  - Local Variables
- Control Flow
  - Expressions and Operators
  - While/For Loops
  - Break/Continue Statements
  - If Statement
  - Switch Statement
  - Try/Catch Statement
  - Defer/ErrDefer Statement
- Functions
  - What is a Function?
  - Define a Function
  - Call a Function
  - Pass Parameters to a Function
  - Immutable vs Mutable Parameters
  - Importing Functions from Other Zig Code Files
- Strings
  - UTF-8 Data Type
  - Character Arrays
  - Buffers
  - Print Formatted Strings

  - Capture Strings Console Input
  - String Copy
  - String Comparison
- Memory Control
  - Memory Allocation Philosophy
  - Memory Control vs. Memory Safety
  - Choosing an Allocators
  - Heap Allocation Failure
  - Lifetime and Ownership
  - Optional and Optional Pointers
  - Null References
- Program a Zig “Object”
  - Compared to C/C++/Python/JavaScript
  - Anonymous Structs
  - Anonymous Struct Literals
  - Data Fields
  - Constant Fields
  - Error Enums
  - Function Members
  - Function Patterns
  - Dynamic Memory Allocation
- Testing
  - Zig’s built-in testing
  - What can be tested?
  - Assert Output with Expect
- Conclusion