List

the list maybe not be by page number. it's maybe be random.

free e-book.

this e-book for learning XpLevel in examples and info. the book think you new in XpLevel.

its not important you new at developing or not.

better run examples to you know better whats happend.

Introduction to XpLevel in Blue Sky

Welcome to **XpLevel in Blue Sky**, a journey into the world of programming with a modern and versatile language. Designed to be powerful, efficient, and user-friendly, XpLevel is the result of blending the best features from multiple programming languages such as **Assembly**, **C**, **D**, **Odin**, **Rust**, **Python**, and **Zig**. It represents the perfect harmony between low-level efficiency and high-level simplicity.

This book aims to guide both beginners and experienced developers through the unique capabilities of XpLevel. Whether you're looking to harness precise control over hardware or build accessible, modern applications, this language has something for everyone. Each chapter is designed to provide a clear, hands-on approach to mastering XpLevel's syntax, tools, and features.

By the end of this book, you will have the skills to write clean, efficient, and scalable code using XpLevel, exploring the possibilities of its endless potential under the blue sky of creativity and innovation.

Written by: Pouya Mohamadi (GitHub: [IDGAF - Pouyamohamadi-cpu])

Introduction to XpLevel

XpLevel is a modern and powerful programming language designed to combine the best features from several renowned programming languages. This language offers developers a unique and efficient experience by incorporating principles and ideas from the following languages:

- **Assembly:** For precise control over hardware and optimized low-level performance.
- **C:** For high performance, simplicity, and powerful memory handling capabilities.
- **D:** For a modern approach to combining low-level power with high-level features.
- **Odin:** For readable syntax and well-structured designs for complex projects.
- **Rust:** For memory safety and avoidance of common programming errors.
- **Python:** For simplicity, code readability, and rapid development capabilities.
- **Zig:** For reliable performance and enhanced resource management.

By blending power, simplicity, and modern capabilities, **XpLevel** empowers developers to create safer, faster, and more readable code. Its goal is to provide flexibility and high performance, carving out a unique space among programming languages.

Complete Syntax Guide for XpLevel

This document provides a comprehensive guide to the syntax of **XpLevel**, showcasing its rules and practical usage. XpLevel combines the best features from languages like Assembly, C, Python, Odin, Rust, Zig, and D to provide a powerful programming experience.

1. Basic Structure of an XpLevel Program

Every XpLevel program starts with a `ma_in` function as its entry point:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

- **ma_in:** The main entry point function.
- **storio:** Used to define functions and variables.
- **std.onVoidMain:** Initializes the runtime environment.

2. Variables

Variables are defined using the `storio` keyword:

```
storio {Instr} x = 10
storio {Instr} text = b"Hello XpLevel!"
```

- **{Instr}:** Specifies the data type of the variable.
- **b:** Indicates binary strings.

3. Functions

Functions are defined using `createFn`:

```
createFn printMessage !void {} !end void {
    InDisplay.show !void {"XpLevel is amazing!"} !end
!xple.fi
```

```
}
printMessage !void {} !end
```

- **createFn:** Used to define a function.
- **InDisplay.show:** Outputs text or results.
- **!xple.fi:** Marks the end of a function block.

4. Conditional Statements

Conditionals in XpLevel use `ifit`:

```
storio {Instr} x = 5
ifit !void {x > 0} !end void {
    InDisplay.show !void {"Positive number"} !end
}
```

- **ifit:** Executes a block of code based on conditions.

5. Loops

Loops are defined using `initLoopFor`:

```
initLoopFor i inside [0, 1, 2, 3, 4] void {
    InDisplay.show !void {f"Number: {i}"} !end
!xple.fi
}
```

- **initLoopFor:** Iterates over collections or ranges.
- **f"":** Used for formatted output.

6. Object-Oriented Programming

Classes are defined using `initCls`, and methods are created inside:

```
initCls math void {
    createFn add !void {a, b} !end void {
        return a + b
    !xple.fi
```

```
    }
}
math.add !void {5, 7} !end
```

- **initCls:** Defines a new class.
- **createFn:** Creates methods inside the class.

7. Error Handling

Error handling uses predefined functions for safety:

```
tryto !void {
    storio {Instr} result = divide !void {10, 0} !end
!xple.fi catchIn !void {
    InDisplay.show !void {"Division by zero error!"} !end
!xple.fi
```

- **try:** Marks the block of code to test.
- **catch:** Handles any error during execution.

8. Importing Libraries

Libraries are imported using `XZero`:

```
XZero inst mathlib
XZero instb /storage/mathlib.zip
```

- **inst:** Installs a library from the internet.
- **instb:** Installs a library from a local zip file.

9. Built-in Functions

XpLevel offers a variety of built-in functions:

```
storio {Instr} abs_result = xabs.absvalue !void {-15} !end
storio {Instr} word_count = xstring.countwords !void {b"Hello World"} !end
```

```
storio {Instr} matrix = convertToBinaryMatrix !void {"/path/image.pgm", 128} !end
```

10. Displaying Results

Results are displayed using `InDisplay.show`:

```
InDisplay.show !void {"Final Output"} !end
InDisplay.show !void {f"Calculated Value: {value}"} !end
```

Conclusion

XpLevel is a versatile language that combines simplicity and power. Its rich syntax supports procedural and object-oriented programming, making it suitable for a wide range of applications.

Introduction to the XpLevel Language

XpLevel is a powerful and unique programming language. It is designed with a distinctive structure, suitable for writing modern and readable code. Here, we explain the basic concepts of this language through a simple example.

Basic Structure of XpLevel Code

An XpLevel file typically starts by defining the `ma_in` function, which serves as the main entry point for the program. Example:

```
# "a example of use xple"
ma_in !void {
    storio !void {ma_in} {create} : !main
    std.onVoidMain()
    !end
} !end
```

Printing to Output

To display text in the output, use the `InDisplay.show` command:

```
InDisplay.show !void {"hello world"} !end
```

Conditional Statements

To execute code based on conditions, use the `ifit` keyword:

```
storio {Instr} hi = "hi"
ifit !void {hi == "hi"} !end void {
    InDisplay.show !void {"yes"} !end
jump void {
    InDisplay.show !void {"No"} !end
!xple.fi
```

Final Notes

Keywords like `!void` and `storio` are specifically used for defining data types and functions in this language.

To practice further, you can experiment with functions and conditional structures in the language to write simple programs.

Understanding the XpLevel Example

This example highlights a basic structure in **XpLevel** programming. Below, we'll break down each part of the code and explain its functionality.

1. Entry Point (`ma_in` function)

The `ma_in` function acts as the main entry point for the program, similar to the `main` function in many other languages. It initializes the program:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

- **storio:** Appears to register or store the `ma_in` function.
- **std.onVoidMain():** Likely a standard function call related to the main setup.
- **!void:** Indicates the function does not return any value.

2. Custom Function Creation

The `createFn` keyword defines a custom function named `hi`. Within this function, a message is printed to the output:

```
createFn hi !void {} !end void {
    InDisplay.show !void {"hello world"} !end
}
```

- **createFn:** Used to define new functions.
- **InDisplay.show:** Prints output to the console or display.
- **!void:** Indicates the function does not return anything.

3. Function Invocation

Once the `hi` function is defined, it is called to execute its logic:

```
hi !void {} !end
```

4. Execution and Output

To execute the program, the following command is used:

```
xple --c /storage/emulated/0/untitled.xple
```

Output:

```
hello world
```

Key Observations

- The structure is modular, with clear separation of function definition and invocation.
- Syntax emphasizes readability with distinct keywords like `!void` and `createFn`.
- The language appears to include both high-level constructs (e.g., printing output) and low-level control (e.g., function storage).

XpLevel Tutorial: Functions with Arguments

In this tutorial, we will explore how to define and use functions with arguments in **XpLevel**. This allows for greater flexibility and adaptability in your programs.

1. Entry Point: The `ma_in` Function

The `ma_in` function serves as the main entry point of the program, where the execution begins. Below is the code structure:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

- **storio:** Registers the main function with the program runtime.
- **std.onVoidMain():** Initializes and prepares the environment for the program.

2. Function Definition with Arguments

The function `createFn hi` is defined to accept an argument (`hm`). The argument is used to dynamically construct the output text:

```
createFn hi !void {hm} !end void {
    InDisplay.show !void {"hello world" + " " + hm} !end
}
```

- **createFn:** Used to define custom functions.
- **Arguments:** The function accepts `hm` as input, which is concatenated to form a full message.

- **InDisplay.show:** Displays the final message on the console or screen.

3. Function Invocation

Once the function is defined, it can be called with specific arguments. In this example, the `hi` function is called with the argument `"again"`:

```
hi !void {"again"} !end
```

4. Execution

The program can be executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

5. Output

The output of this program will be:

```
hello world again
```

Key Observations

- **Modularity:** Functions with arguments allow you to reuse code and handle dynamic input.
- **String Manipulation:** The use of the `+` operator facilitates basic string concatenation.
- **Readable Syntax:** The syntax for function definition and invocation is concise and easy to understand.

XpLevel Tutorial: Class Initialization and Methods

In this tutorial, we explore how to define classes, create methods inside them, and invoke these methods in **XpLevel**.

1. Entry Point: The `ma_in` Function

The program starts with the `ma_in` function, which serves as the main entry point:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

- **storio:** Registers the main function with the runtime system.
- **std.onVoidMain:** Initializes the program environment.

2. Class Initialization

The `initCls` keyword is used to define a new class named `hello`. Inside this class, we define a method:

```
initCls hello void {
    createFn say !void {} !end void {
        InDisplay.show !void {"Hello"} !end
    }
!xple.fi
}
```

- **initCls:** Initializes a new class.
- **createFn:** Defines a method called `say` inside the `hello` class.
- **InDisplay.show:** Outputs the string `"Hello"` to the console.

3. Method Invocation

The method `say` is invoked using dot notation, which associates the method with its class:

```
hello.say !void {} !end
```

4. Execution

The program is executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

5. Output

The output of this program will be:

```
Hello
```

Key Observations

- **Object-Oriented Features:** The use of `initCls` shows that **XpLevel** supports object-oriented design principles.
- **Method Encapsulation:** Methods like `say` are neatly encapsulated within the class.
- **Readability:** The syntax is intuitive and supports modular, reusable code structures.

XpLevel Tutorial: Version Handling with Classes

In this tutorial, we demonstrate how to use classes and methods in **XpLevel** to retrieve and display system-level information, such as the language's version.

1. System-Level Initialization

The `std` namespace is used to initialize specific system components. In this case, we initialize `infosys` using `std *void {onMain}` :

```
std *void {onMain} !void {infosys} !end
```

Note: Only one item can be initialized using this method.

2. Entry Point

The program starts with the `ma_in` function, registering the main function and setting up the runtime:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

3. Class Definition

The `initCls` keyword is used to define a class named `hello`. Within the class, a method `sayversionofxple` is defined to retrieve and store the current version:

```
initCls hello void {
    createFn sayversionofxple !void {} !end void {
        storio {Instr} h = getVersion !void {} !end
        storio {recall} h
    !xple.fi
    }
}
```

4. Display Output

The method is invoked using `hello.sayversionofxple`, and the result is displayed with `InDisplay.show`:

```
InDisplay.show !void {hello.sayversionofxple !void {} !end} !end
```

5. Execution

The program is executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

6. Output

The output of this program will be:

```
2.3
```

Key Observations

- **System Namespace:** `std` is used for initializing system components, like `infosys`.
- **Class Method:** The method retrieves the version using the predefined `getVersion` function.
- **Compiler Behavior:** If the compiler doesn't import components initially, recompiling resolves the issue.

XpLevel Tutorial: Importing External Files and Methods

In this tutorial, we will cover how to import external files, define and call methods, and manage compiler behavior in **XpLevel**.

1. System Initialization

We start by initializing the `peep` component for managing file imports:

```
std *void {onMain} !void {peep} !end
```

2. Importing Files

To include external files, use `preep.importAsPeepvoid`. In this example, two files are imported:

```
preep.importAsPeepvoid !void {"lib-main/infosys.xple"} !end
preep.importAsPeepvoid !void {"/storage/emulated/0/hi.xple"} !end
```

Note: The compiler may only import one file per compilation. Recompiling ensures all files are properly included.

3. Main Function

The `ma_in` function initializes the runtime environment:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

4. Class Definition and Method

A class `hello` is defined, containing a method `sayversionofxple` to retrieve the language version:

```
initCls hello void {
    createFn sayversionofxple !void {} !end void {
        storio {Instr} h = getVersion !void {} !end
        storio {recall} h
    !xple.fi
    }
}
```

5. Method Invocation

The method is invoked and its result is displayed using:

```
InDisplay.show !void {hello.sayversionofxple !void {} !end} !end
```

6. External Method Invocation

A method from the external file `hi.xple` is invoked:

```
hi.sayhifromhixple !void {} !end
```

7. Output

The program produces the following output:

```
2.3
Hello from Hi.xple
```

Key Observations

- **File Imports:** Use `preep.importAsPeepvoid` to include external scripts and libraries.
- **System Version:** Retrieve the language version with `getVersion`.
- **Compiler Note:** Recompilation ensures all dependencies are loaded.

XpLevel Tutorial: Word Count with Strings

In this tutorial, we demonstrate how to use **XpLevel** for string manipulation and word counting. This example utilizes built-in functions to process text and display results dynamically.

1. Entry Point

The program begins with the `ma_in` function, which initializes the runtime environment:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

- **storio:** Registers the entry point with the runtime.
- **std.onVoidMain:** Prepares the runtime environment for execution.

2. Declaring a String

We declare a variable `text` containing the string `"Hello world from XpLevel!"`:

```
storio {Instr} text = b"Hello world from XpLevel!"
```

- **storio:** Used for variable declaration.
- **b:** Indicates the string is binary (useful for specific processing or encoding).

3. Word Count

The built-in function `xstring.countwords` is used to count the number of words in the text:

```
storio {Instr} word_count = xstring.countwords !void {text} !end
```

- **xstring.countwords:** Counts words in the provided string.
- **word_count:** Stores the result of the function (number of words).

4. Displaying the Result

The word count is displayed dynamically using `InDisplay.show` and formatted strings:

```
InDisplay.show !void {f"Word Count: {word_count}"} !end
```

- **f"":** Formats the string by embedding variables directly into it.
- **InDisplay.show:** Displays the message in the console or output environment.

5. Execution Command

The program is executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

6. Output

The output of this program is:

```
Word Count: 4
```

Key Observations

- **Built-in Functionality:** The `xstring.countwords` function simplifies word counting and string manipulation.
- **Formatted Strings:** Using `f""` for dynamic output increases readability.
- **Modularity:** The use of variables (e.g., `word_count`) ensures clarity and modular design.

XpLevel Tutorial: String Length Calculation

In this tutorial, we demonstrate how to calculate the length of a string in **XpLevel** using the `xstring.stringlength` function.

1. Entry Point

The program begins with the `ma_in` function, which initializes the runtime environment:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

2. Declaring a String

A variable `text` is declared and assigned the string `"Hello from XpLevel!"`:

```
storio {Instr} text = b"Hello from XpLevel!"
```

- **b:** The `b` prefix indicates a binary string, potentially used for specific processing needs.
- **storio:** Declares and initializes the variable `text`.

3. Calculating String Length

The built-in function `xstring.stringlength` calculates the number of characters in `text`:

```
storio {Instr} length = xstring.stringlength !void {text} !end
```

- **length:** Stores the result of the string length calculation.
- **xstring.stringlength:** Computes the length of the provided string.

4. Displaying the Length

The length of the string is displayed using `InDisplay.show` and formatted strings:

```
InDisplay.show !void {f"String Length: {length}"} !end
```

- **f"":** Dynamically embeds the value of `length` into the string.
- **InDisplay.show:** Outputs the formatted message to the console or display.

5. Execution

The program is executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

6. Output

The output of the program is:

```
String Length: 19
```

Key Observations

- **String Processing:** The `xstring.stringlength` function makes it easy to compute the length of a string.
- **Formatted Output:** Using `f""` for dynamic strings ensures clarity and simplicity.
- **Modular Design:** Variables like `text` and `length` improve readability and reusability.

XpLevel Tutorial: Converting PGM to Binary Matrix

In this tutorial, we'll learn how to convert a PGM (Portable Graymap) image file to a binary matrix in **XpLevel**. We'll also visualize the binary matrix and a BMP file using XpLevel tools.

1. Entry Point

The program begins with the `ma_in` function, which sets up the runtime and registers the entry point:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

2. Converting PGM to Binary Matrix

A PGM file is specified, and a threshold value is used for binarizing pixel values. The `convertToBinaryMatrix` function generates a binary matrix:

```
storio {Instr} filename = "/storage/emulated/0/Testscript/test.pgm"
storio {Instr} threshold = 128
storio {Instr} binary_matrix = convertToBinaryMatrix !void {filename, threshold} !end
initLoopFor row inside binary_matrix void {
    InDisplay.show !void {row} !end
!xple.fi
}
```

- **Filename:** Specifies the path to the PGM file.
- **Threshold:** Binarizes pixel values above or below 128.
- **convertToBinaryMatrix:** Generates a binary matrix based on the file and threshold.
- **initLoopFor:** Iterates through each row of the matrix to display it.

Sample output:

```
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 1, 1, 0]
[0, 0, 0, 1]
```

3. Visualizing BMP Files

A BMP file is specified and visualized using the `xbmp.displayBMP` function:

```
storio {Instr} filename = b"/storage/emulated/0/Testscript/simple.bmp"
xbmp.displayBMP !void {filename} !end
```

- **BMP Filename:** Specifies the path to the BMP file as a binary string.
- **xbmp.displayBMP:** Displays the BMP file visually, possibly in a GUI or terminal.

4. Execution Command

The program can be executed using:

```
xple --c /storage/emulated/0/untitled.xple
```

5. Output

The output of the program includes:

```
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 1, 1, 0]
[0, 0, 0, 1]
```

Additionally, the BMP file is displayed.

6. Bash Script for Simulating the Binary Matrix

You can simulate the binary matrix using a Bash script. This script uses colored spaces to represent pixels:

```
#!/bin/bash

# Simulated binary matrix
binary_matrix=(
    "0 1 1 0"
    "1 0 0 1"
    "1 1 1 0"
    "0 0 0 1"
)

# Define pixel representation
pixel_on="█" # White pixel
pixel_off=" " # Black pixel

# Loop through each row of the binary matrix
for row in "${binary_matrix[@]}"; do
    for pixel in $row; do
        if [ "$pixel" -eq 1 ]; then
            echo -n "$pixel_on"
        else
            echo -n "$pixel_off"
        fi
    done
    echo # New line for next row
done
```

7. Simulated Bash Output

When you run the Bash script, it will simulate the binary matrix as:

Key Observations

- **PGM Conversion:** `convertToBinaryMatrix` simplifies converting images to binary matrices.
- **BMP Visualization:** Use `xbmp.displayBMP` for easy BMP file rendering.
- **Threshold Logic:** Pixels are binarized using a specified threshold (128).
- **Bash Simulation:** Lightweight, terminal-based simulation using colored spaces.

XpLevel Tutorial: Installing Libraries with XZero

Libraries in **XpLevel** are managed through **XZero**, a library installer specifically designed for the language. This tutorial explains both online and offline installation methods.

1. Installing Libraries Online

To install libraries online, make sure you have an active internet connection and use the following command:

```
XZero inst hi
```

- **hi:** The name of the library to be downloaded and installed.
- **XZero inst:** This command fetches the library from the repository and installs it.

2. Installing Libraries Offline

If you do not have an internet connection but have the library's zip file, you can install it using the following command:

```
XZero instb /storage/hi
```

- **/storage/hi:** The path to the folder containing the library's zip file.
- **XZero instb:** Installs the library from a local zip file.

3. Notes

- Ensure you use the correct path when installing libraries offline.

- Libraries are downloaded and installed in the relevant directory automatically when using the `XZero inst` command.

4. Example

Here's a practical example:

Online Installation:

```
XZero inst hi
```

Offline Installation:

```
XZero instb /storage/hi
```

5. Summary

The `XZero` installer is a versatile tool for managing libraries in **XpLevel**, offering both online and offline options. This ensures flexibility regardless of your internet connectivity.

XpLevel Tutorial: Calculating Absolute Value

In this tutorial, we'll explain how to compute the absolute value of a number in **XpLevel**, with dynamic output formatting.

1. Entry Point

The program begins with the `ma_in` function, which sets up the runtime:

```
ma_in !void {
    storio !void {ma_in} {create}: !main
    std.onVoidMain()
    !end
} !end
```

2. Defining a Negative Value

A variable `value` is defined and assigned the value `-15`:

```
storio {Instr} value = -15
```

3. Calculating Absolute Value

The `xabs.absvalue` function computes the absolute value of the variable:

```
storio {Instr} abs_result = xabs.absvalue !void {value} !end
```

- **xabs.absvalue:** A built-in function that computes the absolute value of the input.
- **abs_result:** Stores the computed absolute value.

4. Displaying the Result

The result is displayed using `InDisplay.show` and formatted strings:

```
InDisplay.show !void {f"Absolute Value: {abs_result}"} !end
```

- **f"":** Dynamically embeds variables into the string.
- **InDisplay.show:** Outputs the formatted message to the console or screen.

5. Execution

The program is executed using the following command:

```
xple --c /storage/emulated/0/untitled.xple
```

6. Output

The program produces the following output:

```
Absolute Value: 15
```

Key Observations

- **Mathematical Operations:** The `xabs.absvalue` function simplifies absolute value calculations.
- **Formatted Strings:** Using `f""` makes the output dynamic and readable.
- **Clean Syntax:** The program structure is modular and easy to understand.

XpLevel - full information about math on xple.

1. Mather is do anything its a class name in examples you learn.

2. Mather.dobule --> do *2 with input number. better know yournumber x 2 = Mather.dobule

3. Mather.radical --> get squt of ypur number better know √yournumber = Mather.radical

4. Mather.plus --> plus two number.

5. Mather.mine --> number - number2

6. Mather.equaler --> number ÷ number2

7. Mather.square --> number ** number

8. Mather.siner --> get sin of number.

```
ma_in !void {
      storio !void {ma_in} {create}: !main
      std.onVoidMain()
      !end
} !end
InDisplay.show !void {Mather.dobule !void {5} !end} !end
InDisplay.show !void {Mather.plus !void {4, 3} !end} !end
InDisplay.show !void {Mather.mine !void {5, 2} !end} !end
InDisplay.show !void {Mather.equaler !void {4, 2} !end} !end
InDisplay.show !void {Mather.square !void {3, 2} !end} !end
InDisplay.show !void {Mather.siner !void {30} !end} !end
```

output:

```
~ $ xple --c /storage/emulated/0/untitled2.xple

10
7
3
2.0
9
-0.9880316240928618

~ $
```

XpLevel - Files

Options

- FileUI.readFileWithPath - read file from path
- FileUI.readFileFromHome - read from home
- FileUI.writeFile - write intro a file
- FileUI.deleteFile - delete a file

```
ma_in !void {
      storio !void {ma_in} {create}: !main
      std.onVoidMain()
      !end
} !end
FileUI.readFileWithPath !void {"hi.txt"} !end
FileUI.readFileFromHome !void {"sh.txt"} !end
FileUI.writeFile !void {"hi.txt", "hello"} !end
FileUI.deleteFile !void {"hi.txt"} !end
```

XpLevel - Create A Lib

create a zip file with lib name

main info in --> lib_name/info_package/main.xple

main code in lib_name/main_package/lib_name.xple

XpLevel - compiler full info

```
Usage: xple [command] [options]
Commands:

--c             Build Your XpLevel Program to binary and run it.
--v             Show version of xplevel compiler installed in system.
--h             Show help guilde
--u             Update compiler to new version by github server.
```