Overview of Functions

This lesson explains how to write a simple function in Go.

WE'LL COVER THE FOLLOWING

Functions

The simplest function declaration has the format:

```
func functionName()
```

Between the *mandatory* parentheses **( )** *no*, *one*, or *more parameters* (separated by ,) can be given as input to the function. After the name of each parameter variable must come its type.

The main function as a starting point is required (usually the first function), otherwise the build-error: `undefined: main.main` occurs. The `main` function has *no* parameters and *no* return type (in contrary to the C-family) otherwise, you get the build-error: `func main must have no arguments and no return values`. When the program executes, after initializations the first function called (the entry-point of the application) will be the `main.main()` (like in C). The program exits immediately and successfully when `main.main` returns.

The code or body in functions is enclosed between braces **{ }**. The first { *must* be on the same line as the declaration otherwise you get the error: `syntax error: unexpected semicolon or newline before { )`. The last **}** is positioned after the function-code in the column beneath the function. The syntax is as follows:

follows:

```
func func_Name(param1 type1, param2 type2, ...){
  ...
}
```

If the function is returning an object of type `type1`, we follow the syntax as:

```
func func_Name(param1 type1, param2 type2, ...) type1 {
  ...
}
```

or:

```
func func_Name(param1 type1, param2 type2, ...) ret1 type1 {
  ...
}
```

where `ret1` is a variable of type `type1` to be returned. So a general function returning multiple variables looks like:

```
func func_Name(param1 type1, param2 type2, ...) (ret1 type1, ret2 ret2,
...) {
...
}
```

Smaller functions can be written on one line like:

```
func Sum(a, b int) int { return a + b }
```

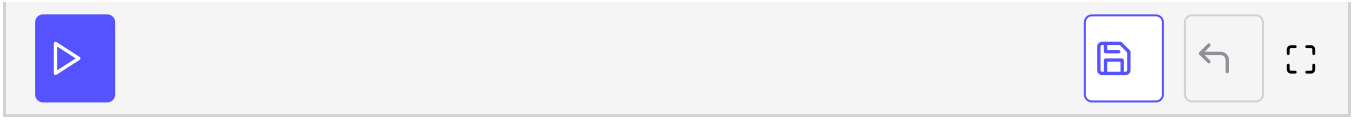
Let's create the main function now as an entry point.

```
package main
import "fmt"

func main(){
}
```

The main structure of a Go program is ready. How about making it capable of doing something?

Hello World 🌍

```
package main // making package for standalone executable
import "fmt" // importing a package

func main() { // making an entry point
    // printing using fmt functionality
    fmt.Println("Hello World")
} // exiting the program
```



Hello World

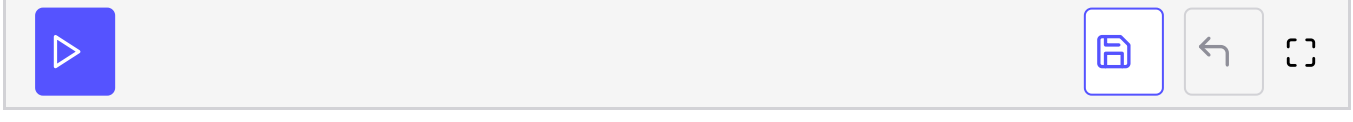
BINGO! We just made our first Go program using the components above. At **line 1**, we make a package *main* and then import package `fmt` to get the functionality of formatted IO at **line 2**. At **line 4**, we make a *main* function to add an entry point, and at **line 6**, we print **Hello World** with `fmt.Println("Hello World")`. This function `Println` from the `fmt` package takes a parameter and prints it on the screen.

Comments

Explanation of source code added to a program as a text note is called a **comment**. Comments are un-compilable. They are just for the understanding of the user. In Go, a one-line comment starts with `//`. A multi-line or block-comment starts with `/*` and ends with `*/`, where nesting is not allowed. You can see we had added comments (the sentences in green) to the above program.

Run the following program and see the magic.

```
package main
import "fmt" // Package implementing formatted I/O.
func main() {
    fmt.Printf("Καλημέρα κόσμε; or こんにちは 世界\n")
}
```



Go an International Language

This illustrates the international character of Go by printing **Καλημέρα**

κόσμε; or こんにちは 世界.

Naming things in Go

Clean, readable code and simplicity are major goals of Go development. Therefore, the names of things in Go should be short, concise, and evocative. Long names with mixed caps and underscores which are often seen e.g., in Java or Python code, sometimes hinder readability. Names should not contain an indication of the package. A method or function which returns an object is named as a noun, no *Get*... is needed. To change an object, use *SetName*. If necessary, Go uses *MixedCaps* or *mixedCaps* rather than underscores to write multiword names.

You have finally run your first Go program. Now let's get an overview of elementary types.