Hello World

There is no better way to learn a programming language than getting our hands dirty with code. Let's go ahead and write our first Go program.

Setting up the Development Environment

Let's create a directory where we want to write our hello world program. Open the terminal and run the following command.

```
mkdir ~/Documents/learngo/
```

The above command will create a directory named `learngo` inside the current user's `Documents` directory. Feel free to create the directory wherever you want the code to reside.

Creating a Go Module

Go modules are used to track our application's dependencies and their versions. Next step is to create a go module named `learngo` in the `~/Documents/learngo/` folder.

Run `go mod init learngo` inside the `~/Documents/learngo/` directory. This will create a file named `go.mod`. The contents of the file are provided below.

```
module learngo

go 1.18
```

The first line `module learngo` specifies the module name. The next line `go 1.18` indicates that the files in this module use go version 1.18.

Hello World

Create a file named `main.go` in the `learngo` directory using your favorite text editor with the following contents.

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

It's a convention in Go to name the file that contains the `main` function as `main.go`, but other names work as well.

Running a Go Program

There are a couple of different ways to run a Go program. Let's look at them one by one.

go install

The first method to run a Go program is using the `go install` command. Let's `cd` into the `learngo` directory we just created.

```
cd ~/Documents/learngo/
```

Run the following command next.

```
go install
```

The above command will compile the program and install (copy) the binary to location `~/go/bin`. The name of the binary will be the folder name containing `main.go`. In our case, it will be named `learngo`.

You might encounter the following error when you try to install the program.

```
go install: no install location for directory /home/kpteja/Documents/learngo outside GOPATH
For more details see: 'go help gopath'
```

What the above error actually means is, `go install` is unable to find a location to install the compiled binary. So let's go ahead and give it a location. This location is governed by the `GOBIN` environment variable.

```
export GOBIN=~/go/bin/
```

The above command specifies that `go install` should copy the compiled binary to the path `~/go/bin/`. This is the conventional location for a Go binary but feel free to change it to any location you want. Now try running `go install` again and the program should compile and run without any problems.

You can type `ls -al ~/go/bin/learngo` in the terminal and you can find that in fact `go install` has placed the binary in the path `~/go/bin`.

Now let's run the compiled binary.

```
~/go/bin/learngo
```

The above command will run the `learngo` binary and print the following output.

```
Hello World
```

Congrats! You have successfully run your first Go Program.

If you want to avoid typing the entire path `~/go/bin/learngo` each time you run the program, you can add `~/go/bin/` to your PATH.

```
export PATH=$PATH:~/go/bin
```

Now you can just type `learngo` in the terminal to run the program.

You might be wondering what will happen when the `learngo` directory contains muliple go files instead of just `main.go`. How will `go install` work in this case? Please hold on, we will discuss these when we learn about packages and go modules.

go build

The second option to run the program is using `go build`. `go build` is much similar to `go install` except that it doesn't install (copy) the compiled binary to the path `~/go/bin/`, rather it creates the binary inside the location from which `go build` was installed.

Type the following command in the terminal

```
cd ~/Documents/learngo/
```

to change the current directory to `learngo`.

After that, enter the following command.

```
go build
```

The above command will create a binary named `learngo` in the current directory. `ls -al` will reveal that a file named `learngo` is created.

Type `./learngo` to run the program. This will also print

```
Hello World
```

We have successfully run our first Go program using go build too :)

go run

The third way to run the program is using `go run` command.

Type the command `cd ~/Documents/learngo/` in the terminal to change the current directory to `learngo`.

After that, enter the following command.

```
go run main.go
```

After the above command is entered, we can see the output

```
Hello World
```

One subtle difference between the `go run` and `go build/go install` commands is, `go run` requires the name of the `.go` file as an argument.

Under the hood, `go run` works much similar to `go build`. Instead of compiling and installing the program to the current directory, it compiles the file to a temporary location and runs the file from that location. If you are interested to know the location where go run compiles the file to, please run `go run` with the `--work` argument.

```
go run --work main.go
```

Running the above command in my case outputs

```
WORK=/var/folders/23/vdjz4kt972g5nzr86wzrj9740000gq/T/go-build698353814
Hello World
```

The `WORK` key's value specifies the temporary location to which the program will be compiled.

In my case, the program has been compiled to the location `/var/folders/23/vdjz4kt972g5nzr86wzrj9740000gq/T/go-build69`
This might vary in your case :)

Go Playground

The final way of running the program is using the go playground. Although this has restrictions, this method comes in handy when we want to run simple programs since it uses the browser and doesn't need Go installed in your local :). I have created a playground for the hello world program. Click here to run the program online.

You can also use the go playground to share your source code with others.

Now that we know 4 different ways to run a program, you might be in a confusion to decide which method to use. The answer is, it depends. I generally use the playground when I want to do a quick check of logic or find out how a standard library function works. In most other cases, I prefer `go install` since it gives me an option to run the program from any directory in the terminal as it compiles all programs to the standard `~/go/bin/` path.

A short explanation of the hello world program

Here is the hello world program we just wrote

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

We will discuss in brief what each line of the program does. We will dwell deep into each section of the program in the upcoming tutorials.

`package main` - **Every go file must start with the package name statement**. Packages are used to provide code compartmentalization and reusability. The package name main is used here. The `main` function should always reside in the `main` package.

`import "fmt"` - The import statement is used to import other packages. In our case, `fmt` package is imported and it will be used inside the main function to print text to the standard output.

`func main()` - The `func` keyword marks the beginning of a function. The `main` is a special function. The program execution starts from the `main` function. The `{` and `}` braces indicate the start and end of the `main` function.

`fmt.Println("Hello World")` - The `Println` function of the `fmt` package is used to write text to the standard output. `package.Function()` is the syntax to call a function in a package.