Getting Started with Google GoLang

...

Week 1

Why Learn GO ?

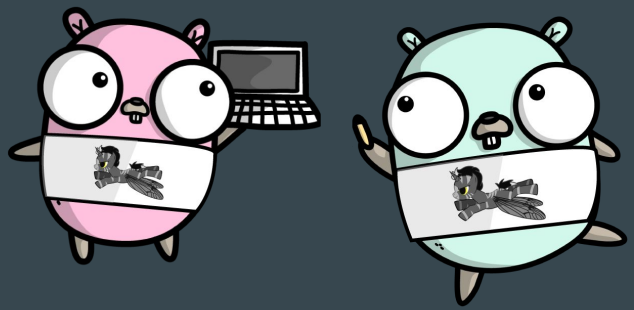
- Insanely fast
- Super Simple
- Extremely Powerful

Installing Go

Official Website: **golang.org**

Supports Linux, Windows and Mac.

Install guide: **golang.org/doc/install**



Code Editor Setup

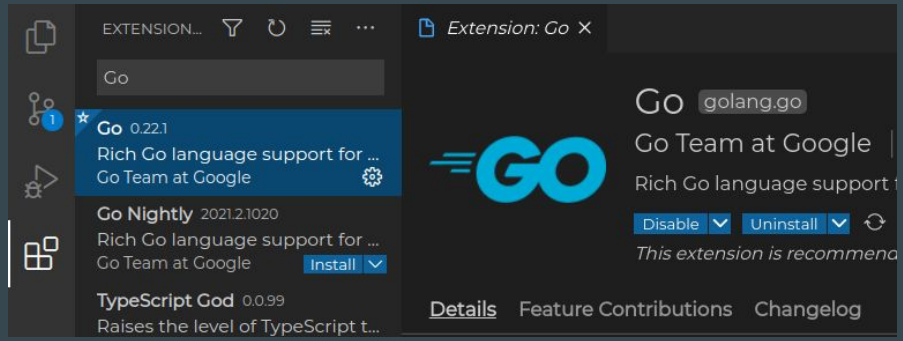
Atom (**atom.io**)

VS Code (**code.visualstudio.com**)

Vim (**vim.org**)

Go Plugins for VS Code

Simply search for Go in the extensions tab!



Hello World!

hello.go
1 package main
2
3 import "fmt"
4
5 func main() {
6 fmt.Println("Hello World!")
7 }
8

Execution and Output

```
 Week-1 ❯ ls
hello.go
 Week-1 ❯ go build hello.go
 Week-1 ❯ ls
hello  hello.go
 Week-1 ❯ ./hello
Hello World!
 Week-1 ❯ go run hello.go
Hello World!
 Week-1 ❯
```

Data types in Go

Go supports data types like int, float, strings, etc.

For more info about Go types, take a look at the following link:

geeksforgeeks.org/data-types-in-go/

Variables and Data types

data-types.go

```go
package main

import "fmt"

func main() {
  // Decleration followed by intitialisation
  var x int
  x = 0

  // Decleration and intitialisation
  var y float64 = 1.2

  // Type inferring
  z := "Hi!"

  // Unused variable, not allowed
  a := 123

  fmt.Println("Int:", x)
  fmt.Println("Float:", y)
  fmt.Println("String:", z)
}
```

Execution and Output

```
Week-1 ❯ go run data-types.go
Int: 0
Float: 1.2
String: Hi!
Week-1 ❯
```

Composite types in Go

Go supports composite types like arrays, slices, maps, structs, etc.

For more info about composite types in Go, take a look at the following link:
tutorialedge.net/golang/go-complex-types-tutorial

Composite types - Arrays

arrays.go

```go
package main

import "fmt"

func main() {
  // Declaring an empty array
  // var variableName [size]type
  var arr [2]int

  // Assigning values to array elements
  arr[0] = 1
  arr[1] = 2

  fmt.Println("Array:", arr, "Length:", len(arr))

  // Declaring and intitialising an array
  arr2 := [...]string{"Hello", "There"}

  // In the above declartion ... is used to
  // infer size, you can explicitly specify
  // a number too. Eg: a := [3]int{1, 2, 3}

  fmt.Println("Array 2:", arr2, "Length:", len(arr2))
}
```

Composite types - Slices

slices.go

```go
package main

import "fmt"

func main() {
  // Declaring an empty slice
  // var variableName []type
  var slice []int

  fmt.Println("Slice before appends")
  fmt.Println("Slice:", slice, "Length:", len(slice))

  // Adding values to slices
  // slice = append(slice, elements)
  slice = append(slice, 1)
  slice = append(slice, 2)

  fmt.Println("Slice after appends")
  fmt.Println("Slice:", slice, "Length:", len(slice))

  // Declaring and intitialising a slice
  slice2 := []string{"Hello", "There", "General", "kenobi"}

  // Deriving slice from other slice
  // You can also derive slices from arrays
  slice3 := slice2[1:3]

  fmt.Println("Slice 2", slice2)
  fmt.Println("Slice 3", slice3)
}
```

Other Composite types

More about Arrays and Slices:

- technobeans.com/2019/01/27/golang-composite-data-types-arrays-and-slices/

Structs and Maps:

- tutorialedge.net/golang/go-complex-types-tutorial
- geeksforgeeks.org/structures-in-golang/
- geeksforgeeks.org/golang-maps/

Conditionals: if-else

if.go

```go
package main

import "fmt"

func main() {
  x := 10
  y := false
  z := "hi!"

  if x > 5 {
    fmt.Println("X is greater than 5")
  }

  if x == 20 {
    fmt.Println("20!")
  } else if x < 20 && !y {
    fmt.Println("< 20")
  } else {
    fmt.Println(z)
  }
}
```

Loops

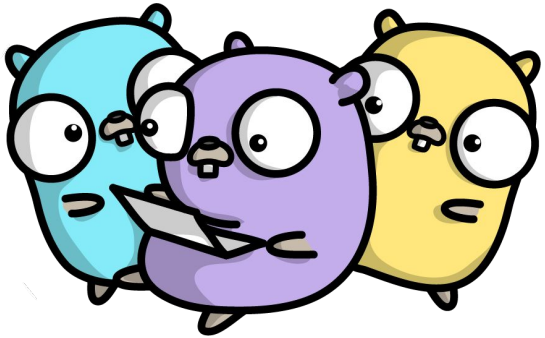
loops.go

```go
package main

import "fmt"

func main() {
  arr := []int{}

  // 3 condition for loop
  // for <initialisation>; <condition>; <update>
  for i := 0; i < 10; i++ {
    arr = append(arr, i*i)
  }

  y := len(arr) - 1

  // While loop
  // for <condition>
  for y >= 0 {
    arr[y]++
    y--
  }

  // Infinite loop
  for {
    fmt.Println("Infinite!")
  }
}

```

Iterating over arrays

arr-loop.go

```go
package main

import "fmt"

func main() {
  arr := []int{1, 2, 3, 4, 5, 6}

  // Iterating over a slice
  for index, value := range arr {
    fmt.Println(index, value)
  }

  sum := 0
  // Use _ when we don't require a variable
  for _, value := range arr {
    sum += value
  }

  fmt.Println(sum)
}
```



Thank You!

Source Code and Slides available at:
github.com/Gituser143/PESU-IO-Go