# Getting Started with Google GoLang

...

Week 2

# What are functions and why use them?

Function types:

- Functions (Call By Value and Reference)
- Variadic and Deferred Functions
- Receiver Functions
- Anonymous functions

# Declaring Functions

functions.go

```go
import "fmt"

// Simple function with no
// parameters and no return type
func hello() {
  fmt.Println("Hello!")
}

// Function taking 2 integer
// parameters and returning 1 integer
func sum(a, b int) int {
  return a + b
}

// Function taking and returning
// 2 integer parameters
func multiplyAndSum(a, b int) (int, int) {
  return a * b, a + b
}

func main() {
  hello()

  x, y := 5, 10

  sum := sum(x, y)
  prod, _ := multiplyAndSum(x, y)

  fmt.Println(sum, prod)
}
```

# Call by value vs Reference

call-by-value-ref.go

```go
package main

import "fmt"

func byValue(a int) {
  a += 5
}

func byReference(a *int) {
  *a += 5
}

func main() {
  x := 10
  byValue(x)
  fmt.Println(x)

  byReference(&x)
  fmt.Println(x)
}
```

# Points on Pointers

Use functions with call by reference only when you have to modify data passed to the function.

Use ‘&’ to send address in functions.

Use ‘*’ to dereference pointer to get value.

Resources:

- **digitalocean.com/community/conceptual_articles/understanding-pointers-in-go**
- **geeksforgeeks.org/pointers-in-golang/**
- **gobyexample.com/pointers**

# Variadic functions

variadic.go

```go
package main

import "fmt"

// Variadic function, takes one
// or more values of same type
func vSum(a ...int) int {
  sum := 0
  for _, value := range a {
    sum += value
  }
  return sum
}

func main() {
  // Calling vSum with aribitrary
  // number of arguments
  s1 := vSum(1, 2, 15, 7)

  arr := []int{1, 5, 6, 3, 14}

  // Providing a slice as argument
  s2 := vSum(arr...)

  fmt.Println("Sum S1:", s1)
  fmt.Println("Sum S2:", s2)
}
```

# The “defer” keyword

Used to defer function calls to the end.

More details about defer:
**geeksforgeeks.org/defer-keyword-in-golang**

# Custom types

custom-types.go

```go
package main

import "fmt"

type side float64

type rectangle struct {
  length side
  width  side
}

func main() {
  r1 := rectangle{length: 4, width: 5}
  fmt.Println(r1)
}
```

# Receivers

receivers.go

```go
package main

import "fmt"

// Creating a new type side
type side float64

// using side in a struct type
type rectangle struct {
  length side
  width  side
}

// receiver for type rectangle
func (r rectangle) getArea() float64 {
  return float64(r.length * r.width)
}

// receiver for type side
func (s side) getSquare() side {
  return s * s
}
```

```go
func main() {

  var l, w side = 4, 5
  r1 := rectangle{length: l, width: w}

  area := r1.getArea()
  lSquare := l.getSquare()

  fmt.Println("Area:", area)
  fmt.Println("Square:", lSquare)
}
```

# Receivers: value/reference

receivers-value-ref.go

```go
package main

import "fmt"

type list []int

// Receiver using call by value
func (l list) append(num int) {
  l = append(l, num)
}

// Receiver using call by reference
func (l *list) appendRef(num int) {
  *l = append(*l, num)
}

func main() {
  l1 := list{1, 2, 3}

  l1.append(4)
  fmt.Println(l1)

  l1.appendRef(4)
  fmt.Println(l1)
}
```

# Interfaces

```go
// custom type rect
type rect struct {
  width, height float64
  name          string
}

// Receivers for rect
func (r rect) area() float64 {
  return r.width * r.height
}

func (r rect) perim() float64 {
  return 2*r.width + 2*r.height
}

func (r rect) getName() string {
  return r.name
}
```

```go
// custom type circle
type circle struct {
  radius float64
  name   string
}

// Receivers for circle
func (c circle) area() float64 {
  return math.Pi * c.radius * c.radius
}

func (c circle) perim() float64 {
  return 2 * math.Pi * c.radius
}

func (c circle) getName() string {
  return c.name
}
```

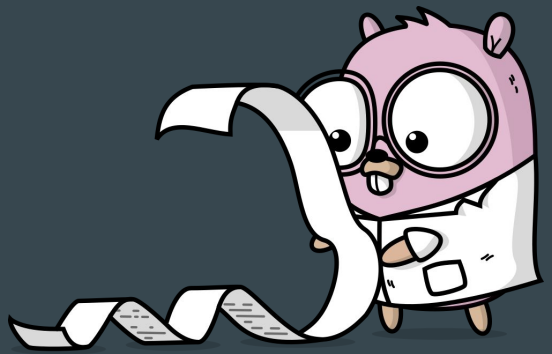
# Interfaces: Continued

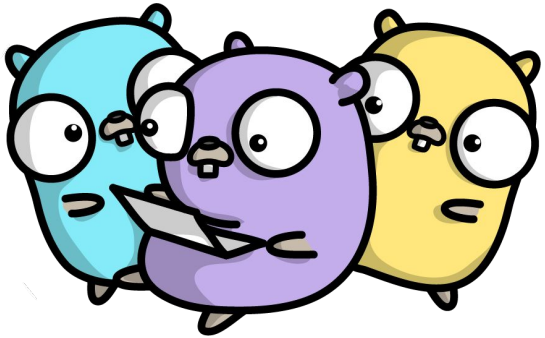
```go
func main() {
  r := rect{width: 3, height: 4, name: "Rectangle"}
  c := circle{radius: 5, name: "Circle"}

  measure(r)
  measure(c)
}
```

```go
// Interface decleration
type geometry interface {
  getName() string
  area() float64
  perim() float64
}

// function to work on interface
func measure(g geometry) {
  fmt.Println("Shape:", g.getName())
  fmt.Println("Area:", g.area())
  fmt.Println("Perim:", g.perim())
}
```

# Interfaces resources

- geeksforgeeks.org/interfaces-in-golang/
- digitalocean.com/community/tutorials/how-to-use-interfaces-in-go
- golangbot.com/interfaces-part-1/





# Thank You!

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