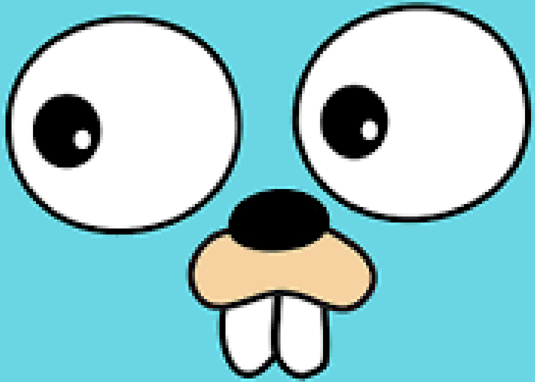
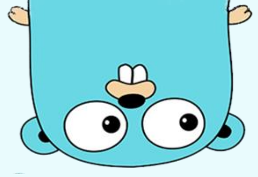


**Phase 1: Introduction to Golang (Sessions 1-8)**

**Objective:** Build a strong foundation in Golang basics.

# Session 4: Functions



## Discussion Points

- Function declarations and returns
- Multiple return values
- Recursion
- Variadic functions
- Defer, panic, and recover

Function declarations and returns

## What is Function?

We use functions to divide our code into smaller chunks to make our code looks clean and easier to understand.

Basically, a function is a block of code that performs a specific task. For example, suppose we want to write code to create a circle and rectangle and color them.

In this case, we can organize our code by creating three different functions:

- function to create a circle
- function to create a rectangle
- function to color

```
func addNumbers() {
  // code
}

addNumbers()
```

Function declarations and returns

## What is Parameters?

Parameters provided in a function call are called **arguments**

```
func addNumbers(parameters) {
   // code
}

addNumbers(parameters)
```

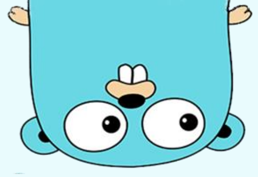
Note: Go doesn't support default parameter values.

Function declarations and returns

## What is Return value?

If a function should return some value, the data type of the value should be specified right after function parameter parentheses

```go
func addNumbers(parameters) int {
   // code
   return sum
}

result := addNumbers(parameters)
```



**Function declarations and returns**

## What is Multiple Return value?

Multiple values can be returned from a function by specifying the return types separated by commas

If we do not need some of the values after the function call, we should assign it as _ (blank identifier)

```
func addNumbers(parameters) (int, string) {
    // code
    return sum, str
}

int1, str1 := addNumbers(parameters)

int2, _ := addNumbers(parameters)
```

Function declarations and returns

## What is Named return values?

Explicitly mentioned return variables in function definition is a different way of returning values in Go

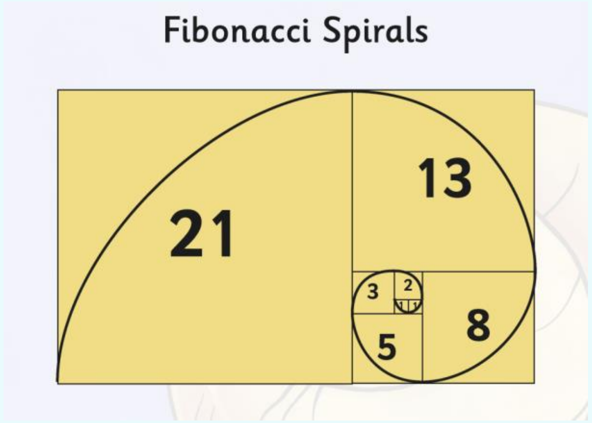
```
func addMultiply(parameters) (add int, mul int) {
   add = …
   mul = …
   return
}

add1, mul1 := addMultiply(parameters)
```

Function declarations and returns

## What is Recursion?

Sometimes, the solution for bigger instances depends on the solution of smaller instances. A function calls itself for smaller instances until the problem is solved. This approach is called recursion.

```go
func fibonacci(n int)(res int) {
   if n <= 1 {
     res = 1
   } else {
     res = fibonacci(n - 1) + fibonacci(n - 2) // recursive call
   }
   return
}
```



Function declarations and returns

## What is Function as a type?

A function in Golang is also a **type**. If two function accepts the same parameters and returns the same values, these two functions are of the same type

We can pass a function as an argument to another function or a function can return another function as a type

**Same type**

```
func add(a, b int) int {}

func subtract(a, b int) int {}
```

**Different type**

```
func add(a string) string {}

func subtract(a, b int) int {}
```

```
func calc(a, b int, fn func(int, int) int) int {
    r := fn(a, b)
    return r
}
calc(5, 10, add)
calc(10,4, subtract)
```

Function declarations and returns

## What is Anonymous function?

A function in Go can also be a value. This means you can assign a function to a variable, return an anonymous function and etc.

**Use cases:**
Short-lived functions, Passing functions as arguments, Closures with state, Deferred execution, Concurrency with Goroutines

```
var var1 = func(n1, n2 int) int {
    ...
}


var1(5, 3)
```

```
func displayNumber() func() int {
    return func() int {
        ...
    }
}
```

Function declarations and returns

## What is Closure?

Go closure is a nested function that allows us to access variables of the outer function even after the outer function is closed.

Before we learn about closure, let's first revise the following concepts:

- Nested Functions
- Returning a function

```
func greet() func() {

  return func() {
    ...
  }

}

func main() {

  g1 := greet()
  g1()
}
```

```
// outer function
func greet() func() string {

  // variable defined outside the inner function

  // return a nested anonymous function
  return func() string {
    ...
  }
}

func main() {

  // call the outer function
  message := greet()

  // call the inner function
  fmt.Println(message())

}
```

## What is Immediately-invoked function?

In Golang, we can create an anonymous function that can be defined and executed at the same time:

```
func main() {
   addRes := func(a, b int) int {
      return a + b
   }(10, 50)

   fmt.Println(“10 + 50 =", addRes)
}
```

## What is Variadic functions?

Variadic functions are functions can take an infinite or variable number of arguments

A typical syntax of a variadic function looks like this. **...** (unpack operator) called as pack operator instructs to store all arguments of type ***Type*** in ***elem*** parameter

**IMPORTANT Note:** Only the last argument of a function is allowed to be variadic

```
func print(elem ...Type)
```

```
func print(a int, arg ...int) []int {
    ...
    return arg
}
```

Defer, Panic, Recover

## What is Defer?

In Go, we use defer, panic and recover statements to handle errors.

We use **defer** to delay the execution of functions that might cause an error. The **panic** statement terminates the program immediately and **recover** is used to recover the message during panic.

```
defer fmt.Println("I'm quit")
```

```
defer func() {
	fmt.Println("I'm quit")
}()
```

## What is Recover?

The **panic** statement immediately terminates the program. However, sometimes it might be important for a program to complete its execution and get some required results.

In such cases, we use the **recover** statement to handle **panic** in Go. The **recover** statement prevents the termination of the program and recovers the program from **panic**.

```
defer func() {
    if a := recover() ; a != nil {
      fmt.Println("RECOVER", a)
    }
}()
```

## Session Quiz

## Go?

Join at

**slido.com**

**# 3213656**

## Thank you.
## Questions?

- Function declarations and returns
- Multiple return values
- Recursion
- Variadic functions
- Defer, panic, and recover

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

**- Albert Einstein**