FUNCTIONS

Monday, July 24, 2023   10:31 PM

INTRO:

- Functions in GO may take ZERO or MORE Arguments
- A Function in general is a small instruction set for a purpose
- Larger code can be split into functions to improve the readability and so will the management.
  Better maintainability
- Easy to understand the goal of the use-case
- Keyword used for syntax is **func** in GO
- **Example:**

```
func sub(x int,y int) int {
    returnx-y
}
```

- The above function takes two input parameters of type integer, and returns the value of type integer.

-
```
func sub(x int,y int) int
```
- The above snippet is referred as **Function Signature**
- **Function Signature:** Tells about the goal of the function, information about input parameters and the return type.

MULTIPLE PARAMETERS:

- If the parameters are of same time , then the type of the input param can be defined after the last parameter
- **Example:**
  -
```
func add(a int, b int, c string) {
    // code
}
```
  - The above function can be same as the below one.
  -
```
func add (a, b int , c string) {
    // code
}
```

PASS BY VALUE:

- Copy of the variable will be passed given as an input to the function
- Original data which is in the caller function will not be mutated / changed
- **Example:**

```
func main(){
    x := 5
    increment(x)

    fmt.Println(x)
   }

func increment(x int){
    x++
}
```

This particular snippet:
- still prints 5,
- because the increment function received a copy of x

Fix for above snippet:

```
func main(){
    x := 5
    x = increment(x)
    fmt.Println(x)
}

func increment(x int) int {
    x++
     return x
}
```

- Added return type of int and returning - the incremented value to fix the issue SO now the output would be 6

IGNORING RETURN VALUES:

- For example, if a function is returning multiple values , and you would only be needing just one of it (the very required one) , in such cases return value van be ignored by that function
- _ (**underscore**) is used for ignoring the returned value from the invoked function
- **Example Code Snippet:**

```
package main
import "fmt"

func main() {
     // ignore y value
     x, _ := getPoint()
     fmt.Println(x);
     // ignore x value
     _, y := getPoint()
     fmt.Println(y);
}
func getPoint() (x, y int) {
  return 3, 4
}
```

```
The output here would be:
3
4
```

NAMED RETURN VALUES:

- Acts as new variables if return values are named
- A return statement without arguments returns the **named return values**.
- Great for function documentation because the information is self-explanatory from the function signature
- Majorly useful in longer functions with more number of return values
- Blank / implicit return should be used only in **short** functions.

- **Example:**

```
func getCoords() (x, y int){

  return
}
```

- In the above function x and y are initialized with zero values
  And x, y will be returned automatically
- The above example snippet is same as below

```
func getCoords() (int, int){
  var x int
  var y int
  return x, y
}
```

EXPLICIT RETURNS

Examples:

- **Explicit Return:**

```
func getValues() (x, y int) {
     return x, y
}
```

- **Overriding Return Values:**

```
func getValues() (x, y int) {
     return 5, 7
}
```

- **Blank Return - Returns Implicity:**

```
Func getValues()(x, y int) {
     return
}
```

EARLY RETURNS:

- **Guard Clauses**: Provides the ability to return early from a function (or continue through a loop) to make nested conditionals one-dimensional
- GO supports **early return from a function** which is a powerful feature that can clean up code, especially when used as **guard clauses.**
- Return early from the function at the end of each conditional block.
- **Example:**

```
func divide(dividend, divisor int) (int, error) {
     if divisor == 0 {
          return 0, error.New("Cannot divide by zero") // early return
     }
     return dividend/divisor, nil
}
```

  - Note: Error handling in GO will be covered in upcoming chapters