Functions

Created	@Jan 28, 2021 9:00 PM
Tags	Lesson

Using Functions in Go

- The `func` keyword is used again here, like in `func main()`

```
func summonNicole() {
  fmt.Println("Hey Nicole, get over here!")
}

func main() {
  // We call our function for the first time
  summonNicole()

  // We call our function for the second time
  summonNicole()
}
```

Scope

- Scope is a concept that refers to where the values and functions are defined and where they can be accessed.
    - For instance, when a variable is defined within a function, that variable is only accessible within that function. When we try to access that same variable from a different function, we get an error because we can't do it. Each function has its own specific scope

Returning

- `return` keyword
- Return type is declared **AFTER** function name and parameters

```
func getLengthOfCentralPark() int32 {
  var lengthInBlocks int32
  lengthInBlocks = 51
```

```
  return lengthInBlocks
}
```

Parameters

- Parameters are declared between the `()` parentheses of a function header
- Type is declared **AFTER** parameter name

```
func multiplier(x int32, y int32) int32 {
  return x * y
}
```

Reusing Code with Functions

```
func main() {
  var a, b, c, d float64
  a = .0214
  b = 1.02
  c = 0.312
  d = 4.001

  a = math.Log2(math.Sqrt(math.T
anh(a)))
  b = math.Log(math.Sqrt(math.Ta
nh(b)))
  c = math.Log(math.Sqrt( math.T
an(c) ))
  d = math.Log2(math.Sqrt(math.T
anh(d)))

  fmt.Println(a, b, c, d)
}
```

```
func specialComputation(x float64) float64
{
  return math.Log2(math.Sqrt(math.Tanh(x)))
}

func main() {
  var a, b, c, d float64
  a = .0214
  b = 1.02
  c = 0.312
  d = 4.001

  a = specialComputation(a)
  b = specialComputation(b)
  c = specialComputation(c)
  d = specialComputation(d)

  fmt.Println(a, b, c, d)
}
```

- If we needed to change `tanh` to `tan` for instance, we would only have to change it in 1 location

Multiple Return Values

- Go can return multiple values in a single function!

- Return types are declared in `()` parentheses in the function header separated with a `,` comma
- The `return` statement separates values (in ORDER) with a `,` comma

```
func GPA(midtermGrade float32, finalGrade float32) (string, float32) {
  averageGrade := (midtermGrade + finalGrade) / 2
  var gradeLetter string
  if averageGrade > 90 {
    gradeLetter = "A"
  } else if averageGrade > 80 {
    gradeLetter = "B"
  } else if averageGrade > 70 {
    gradeLetter = "C"
  } else if averageGrade > 60 {
    gradeLetter = "D"
  } else {
    gradeLetter = "F"
  }

  return gradeLetter, averageGrade
}
```

- The above values could be accessed with `myGrade, myAverage = GPA(myMidterm, myFinal)` in the `main()` function

Deferring Resolution

- We can delay a function call to the end of the current scope by using the `defer` keyword

```
func calculateTaxes(revenue, deductions, credits float64) float64 {
  defer fmt.Println("Taxes Calculated!")
  taxRate := .06143
  fmt.Println("Calculating Taxes")

  if deductions == 0 || credits == 0 {
    return revenue * taxRate
  }


  taxValue := (revenue - (deductions * credits)) * taxRate
  if taxValue >= 0 {
    return taxValue
  } else {
    return 0
```

```
  }
}
```

- The above function will not print `Taxes Calculated!` until the very end