Chapter 9

Functions

- all parameters are represented as **constants (creates a shallow copy if modified)**

```
func foo(arg_label para_name: Int) {
    print("yes")
}
_ = foo(arg_label: 10) // result returned by the function is not used

func foo(para_name: Int) { // parameter name is used as the argument label
    print("yes")
}
_ = foo(para_name: 10)
```

```
// return statement is omitted if function contains a only single expression
// argument labels make the code more readable within the function calls
// parameter names make the code more readable within the function bodies
// "_" overwrites the default parameter name (grammatrical phrases)

var myValue = 10

func doubleValue (_ value: inout Int) -> Int {
    value += value
    return value
}

print("doubleValue call returned \(doubleValue(&myValue))")
```

```
func foo(_ name: String = "Customer") -> String { // default parameter value
    "\(name)"
}
print(foo())
```

```
func foo(_ name: String...) -> [String] { // parameters are stored as an array of strings: [String]
    return name
}
print(foo("a", "b", "c"))

// multiple return types
func sizeConverter(_ length: Float) -> (yards: Float, centimeters: Float, meters: Float) {
    let yards = length * 0.0277778
    let centimeters = length * 2.54
    let meters = length * 0.0254
    return (yards, centimeters, meters)
}
```

```
let foo = {(a: String) -> String in "hello"}
let bar = {(a: String) -> String in "bye"}

// returing a function
func outputfunc(_ cond: (Bool)) -> (String) -> String {
    if cond {
        return foo
    } else {
        return bar
    }
}

print(outputfunc(false)("sasdfdfg"))
```

Closures

- encloses its scope from its function call
- **does not use argument labels in function calls**

```
let sayHello = {
    print("Hello")
}
sayHello()

var greet: (String) -> Void = {
    print("Welcome \($0)")
}

let multiply = {(_ val1: Int, _ val2: Int) -> Int in
    return val1 * val2
}
print(multiply(10, 20))

// compiler can infer the data types and return types of the closure expression
let multiply = {(val1, val2) -> Double in return val1 * val2}
```

```
print(multiply(10, 20))

func foo() -> () -> Int { // returns a closure expression
    var counter = 10
    return {() -> Int in counter + 10}
}
print(foo()())
```