Functions

-Devendranath

Functions are self-contained chunks of code that perform a specific task. You give a function a name that identifies what it does, and this name is used to "call" the function to perform its task when needed.

```
let c = a + b
```

Syntax:

```
func  functionName(arg1, arg2 …) —> ReturnType
{
    Statement 1
    ……………..
    ……………..
    Statement N
    return  ReturnTypeValue;
}
// Calling a method
 var aVar: RT =  functionName(arg1, arg2…);
```

NOTE: Returntype is optional if it is Void.

C-Lang:

```
ReturnType aFunctionName(DataType arg1, DataType arg2)
{
  // Method Body goes here
}
```

Swift:

```
func  aFunctionName(arg1:  DataType,  arg2:  DataType)  ->
ReturnType
{
    // Method Body goes here
}
```

Examples:

```
void add(int a, int b)
func  add(a: Int, b: Int) -> Void
```

Functions with descriptive input arguments

```
    func add(aVar: Int, bVar: Int)
    {
        print("Addition of two numbers \(aVar
+ bVar)")
    }
    add(aVar: 10, bVar: 20);

    func add(aValue aVar: Int, with bVar: Int)
    {
        print("Additio of two numbers: \(aVar
+ bVar)")
    }

    // Method calling
    add(aValue: 10, with: 20)
```

```swift
func playWithFunctions()
{
    sayHello()
    sayHello(to person: "Obama");
    addition(aValue:10, bValue: 20);
    subtract(aValue:20, with: 10, and: 10.50);
}
// Function with no input args and no output
func sayHello() -> Void
{
    print("Hello");
}
// Function with one input args and no output
func sayHello(to person: String) -> Void
{
    print("Hello \(person)");
}

// Function with two input args and Int reuturn value
func addition(aValue: Int, bValue: Int) -> Int
{
    return aValue + bValue;
}

// Function with Three input args and Int reuturn value
func subtract(aValue: Int, with bValue:Int, and cValue:Float) -> Int
{
    return aValue - bValue - Int(cValue);
}
```

Functions with multiple return values

```
func returnsTuple() -> (firstValue: Int, secondValue: Int)
{
    return (20, 100);
}

   let aTuple = returnsTuple();
   print(aTuple.0); // 20
   print(aTuple.1); // 100
   print(aTuple.firstValue); // 20
   print(aTuple.secondValue);// 100
```

Functions with optional return types

```
func optionalReturnMethod(input: Int)-> Int?
{
    if input <= 1
    {
        return 0;
    }
    return nil;
}

let result: Int? = optionalReturnMethod(10);
```

inout parameter

Function parameters are lets (constants). by default. Trying to change the value of a function parameter from within the body of that function results in a compile-time error. Use inout parameter to reference input arguments.

Example:

```
func swapTwoInts(a: Int, b: Int) {
    let temp = a
    a = b
    b = temp
}
swapTwoInts(a: a, b: b)

// ERROR: a and b are let. Can’t be modified

func swapTwoInts(a: inout Int, b: inout Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}
swapTwoInts(a: &a, b: &b)
```

Functions with variadic number of parameters

```
func methodWithVariadicNumberOfParams(a: Int ...) -> Int
{
    var sum: Int = 0

    for i in a
    {
        sum += i
    }
    return sum;
}


var sum = methodWithVariadicNumberOfParams(a: 1)   // 1
sum = methodWithVariadicNumberOfParams(a: 1,2)     // 3
sum = methodWithVariadicNumberOfParams(a: 1,2,3)   // 6
sum = methodWithVariadicNumberOfParams(a: 1,2,3,4) // 10
```

Nested Functions

When we define a function **inside a global function**, we refer to that function as a **nested function**. A nested function has **access to the values defined in its enclosing function**.

```swift
func printMessage(_ message: String) {
    let a = "hello world"
    func printHelloWorld() {
        print(a)
    }
}
```

`a is accessible in printMessage() and printHelloWorld() methods, printHelloWorld() function is not accessible outside of printMessage(:) method.`

Thank You