7. Functions

A function is a chunk of code that performs a specific task. Functions have a name that describes their purpose, that name is used to call the function to perform the task when needed. You can provide data to a function by sending parameters to it, and the function can give data back using a result value. For example here is a function that takes an `Int` and returns a `Bool`:

```
func isOdd(number: Int) -> Bool {
    if number % 2 == 1 {
        return true
    } else {
        return false
    }
}

print(isOdd(1)) // true
print(isOdd(2)) // false
print(isOdd(3)) // true
```

In the above example `isOdd` is the name of the function, `number` is the parameter and `Bool` is the return type.

Defining a function

When you define a function, you can optionally define one or more named, typed values that the function takes as input (known as parameters), and/or a type of value that the function will pass back as output when it is done (known as its return type).

The general syntax for a function is:

func `name` (`list of parameters`) -> `return type` {
`statements`
}

Some functions don't return any values. In that case the syntax doesn't have the arrow(->) and the return type.

func `name` (`list of parameters`) {
`statements`
}

Functions with no parameters with no return value

```
func sayHello() {
```

```
    print("Hello!")
}

sayHello() // Hello!
```

Functions with one parameter with no return value

Parameters are followed by their type.

```
func sayHelloTo(name: String) {
    print("Hello " + name + "!")
}

sayHelloTo("Swift") // Hello Swift!

func printNumber(number: Int) {
    print(number)
}

printNumber(12345) // 12345
```

Functions with one parameter and return value

To add a return value to a function write `->` after the list of parameter followed by the type of the result. Functions that return a value must do so using the `return` keyword. When calling return inside a function the code execution will stop at that line - similar to the `break` statement inside a loop.

```
func double(number: Int) -> Int {
    return number * 2
}

print(double(1)) // 2
print(double(2)) // 4
print(double(3)) // 6
```

Functions with multiples parameters with no return value

To declare multiple parameters use commas to separate them.

```
func count(from: Int, to: Int) {
    for i in from...to {
        print(i)
    }
}

count(5, to: 10)
// 5
// 6
// 7
// 8
```

```
// 9
// 10
```

Notice that the second parameter has the name visible from outside. Starting with the second parameter all parameters have an implicit external parameter name. The same as the local parameter name.

Functions with multiples parameters and return value

```
func sum(a: Int, _ b: Int) -> Int {
    return a + b
}

print(sum(1, 2)) // 3
```

Notice that the second parameter doesn't have an external name in this example. This is because of the `_` character in front of `b`. It this case `_` means don't give this parameter an external name. Remember this because you are going to use it in the exercises.

External parameter names

Parameter names are only used inside the functions body. They are know as *local parameter names*.

Sometimes it's useful to name your parameters when you call a function. To do this you must define *external parameter names* for them. You can write external parameter names before the local name. Starting with the second parameter all parameters have the external parameter name set to the local one. You can change it by writing a different name before it.

```
func sayHello(to name:String) {
    print("Hello " + name + "!")
}

sayHello(to: "Batman")
// Hello Batman!
```

External parameter names should be used when the purpose of a function's arguments is unclear to someone using your code for the first time.

I think you agree that this code is clear:

```
sum(1, 2) // adds 1 and 2
```

While this one isn't that clear:

```
printNumbers(1, 10)
```

Does it print `1` and `10` or the numbers from `1` to `10`?

By adding an external parameter name we can make the purpose of the function clear.

```
printNumbers(1, to: 10)
```

Default Parameter Values

You can define a default value for any parameter in a function definition. You do this by following the parameter definition with a `=` sign and the value for that parameter. If a parameter has a default value set you can omit that parameter when calling the function. To keep things clean it's recommended that you write all the parameters with default value at the end of the parameter list.

```
func countdown(from: Int, to: Int = 1) {
    for var i = from; i >= to; --i {
        print(i)
    }
}

countdown(3)
// 3
// 2
// 1

countdown(5, to: 3)
// 5
// 4
// 3
```

Note that Swift will automatically assign an external parameter name to any parameter with a default value so that it's purpose is clear. To stop this from happening write an underline (`_`) before the parameter with default value.

```
func countdown(from: Int, _ to: Int = 1) {
    for var i = from; i >= to; --i {
        print(i)
    }
}

countdown(3)
// 3
// 2
// 1

countdown(5, 3)
```

```
// 5
// 4
// 3
```

Constant and Variable Parameters

Function parameters are constant by default, that means that you cannot change the value of a parameter inside a function. Trying to change the value of a parameter will result in a compile error.
In some case you will want to change the value of a parameter. To do that declare the parameter with the `var` keyword. The changes made to the parameter will only be present inside the function call.

```
func digits(var number: Int) -> [Int] {
    var result: [Int] = []
    repeat {
        let digit = number % 10
        result.insert(digit, atIndex: 0)
        number = number / 10
    } while number > 0
    return result
}

var number = 12345

print(digits(number)) // [1, 2, 3, 4, 5]
print(number) // 12345
```

In-Out Parameters

Variable parameters can only be changed inside a function. If you want the function to change the value of a parameter and you want those changes to persist after the function call, define the parameter as an `inout` parameter.

Keep in mind that you can only pass variables as in-out parameters. You cannot pass a constant or a literal value, because they cannot be changed. You have to write an ampersand (`&`) in front of the variable name when calling the function. That will indicate that the variable can be modified by the function.

```
func doubleANumber(inout number: Int) {
    number = number * 2
}

var n = 10

doubleANumber(&n)

print(n) // 20
```