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 as result.

Let's take a simple example:

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

isOdd(number: 1) // true
isOdd(number: 2) // false
isOdd(number: 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 sayHello(to name: String) {
    print("Hello \(name)!")
}

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

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 square(number: Int) -> Int {
    return number * number
}

square(number: 1) // 1
square(number: 2) // 4
square(number: 3) // 9
```

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(from: 5, to: 10)
// 5
// 6
// 7
// 8
// 9
// 10
```

Notice that all parameters have the name in the function call. That is called the external parameter name. 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 this time neither parameter appeared in the function call. This is because of the `_` character in front of `a` and `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

Sometimes it's useful to name your parameters differently when you call a function.

For example:

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

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

In this case it would have more sense name the parameter `to` because then the function call would read `sayHello(to: "Batman")`. But then the code in the function would make less sense.

To do this you must define *external parameter names* for them. You can write external parameter names before the local name. All parameters have the external parameter name set to the local one by default. You can change it by writing a different name before it.

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

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

You can make a function ignore the external parameter name by writing `_` in front of the parameter name:

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

External parameter names make your code clear. Don't remove them unless you have to.

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 i in (to...from).reversed() {
        print(i)
    }
}

countdown(from: 3)
// 3
// 2
// 1

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

In-Out 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.

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 double(number: inout Int) {
    number = number * 2
}

var n = 10
```

```
double(number: &n)

print(n) // 20
```

Quick tip

If you want to change the value of a parameter and those changes don't need to be reflected outside the function then you can just declare a variable with the same name:

```
func printNumber(after number: Int) {
    var number = number
    number += 1
    print(number)
}

printNumber(after: 2) // 3
```