Functions - Part 2

by Ash Dreyer & Donald Pinckney

Table of Contents

Parameters

Review

```
func print_something(input_int: Int) {
    print(“You gave me the number \(input_int)!”)
}
```

- Whatever is in the function’s parentheses is the function’s parameter(s)
  - Parameters are values taken in / used by the function
- `input_int` is the parameter name and `Int` is its type
- When calling the function, put parameters in parentheses
- Don’t have to have parameters

Multiple Parameters

- A lot of real-world problems involve multiple inputs
- Functions can take in multiple inputs (parameters) just separated by commas

```
func say_hello(name: String, times: Int) {
   for num in 0..<times {
        print(“Hello, \(name)!”)
    }
}
```

Same Name!

- You can have separate functions with the same name as long as they take in different types of inputs

```
func greet(person: String) -> String {
    return “Hello, \(person)!”
}
func greet(person: String, alreadyGreeted: Bool) -> String {
    if alreadyGreeted {
        return “Hello again, \(person)!”
    } else {
        return “Hello, \(person)!”
    }
}
```

Parameter Names & Argument Labels

- Each function parameter has both an argument label and a parameter name
- When calling a function, you use the argument label before the value you are giving the function (a.k.a. the argument)
- The parameter name is used inside of the function
- Default: parameters' parameter names are their argument labels

Parameter Names & Argument Labels

```
func someFunction(firstArgumentLabel firstParameterName: Int,
                  secondArgumentLabel secondParameterName: Int){
    let sum = firstParameterName + secondParameterName
    print(sum)
}
someFunction(firstArgumentLabel: 1, secondArgumentLabel: 2)
```

Default Parameter Values

- Default value for a parameter: If no value is given for that parameter, it becomes the default value
- If a default value is defined, you can omit that parameter when calling the function
- Write parameters without default values first

Default Parameter Values

```
func someFunction(parameterWithoutDefault: Int,
                  parameterWithDefault: Int = 12) {
    print("Parameter without default value:
          \(parameterWithoutDefault)")
    print("Parameter with default value:
          \(parameterWithDefault)")
}

someFunction(parameterWithoutDefault: 3,
    parameterWithDefault: 6)
print()
someFunction(parameterWithoutDefault: 4)
```

What will this output?

Returning Values

Review

```
func does_something(num: Int) -> String {
    var something = “You gave me the number
        \(num)!”
    return something
}
let fifty_five = 55
print(does_something(num: fifty_five))
```

Output:
```
You gave me
the number
55!
```

- Function returns constant of type String
- -> denotes that the function will actually be returning something
- return ends the function & gives the call the value of something
- Not all functions return a value

Multiple Return Values

- Why? Useful for tasks that need to produce two values or time/resource-consuming tasks
  - Finding coordinates (x, y)
  - Finding min and max numbers of an array

- How? Tuples!
  - Tuples group multiple values into a single compound value
  - Values in a tuple can be any type and do not have to be the same type

Tuples Explained

```
let http404Error = (code: 404, description: “Not found”)
```

- `http404Error` is a tuple of type `(Int, String)`, and equals `(404, “Not found”)`
- Tuples are denoted by parentheses ( )
- Can contain as many values as you’d like
- Does not need labels (“code:” and “description:”), however, labels make it clearer what you are accessing in the tuple
  - Ex. `http404Error.code` is the value 404
  - To access 404 without a label, you would write `http404Error.0`

Tuples Returned From Functions

```
func min_max(array: [Int]) -> (min: Int, max: Int) {
  var current_min = array[0]
  var current_max = array[0]
  for value in array[1..<array.count] {
    if value < current_min {
      current_min = value
    } else if value > current_max {
      current_max = value
    }
  }
  return (current_min, current_max)
}
```

Tuples Returned From Functions

```
func min_max(array: [Int]) -> (min: Int, max: Int) {
  var current_min = array[0]
  var current_max = array[0]
  for value in array[1..<array.count] {
    if value < current_min {
      current_min = value
    } else if value > current_max {
      current_max = value
    }
  }
  return (current_min, current_max)
}
```

- min and max are the labels of the returned tuple and CAN be used outside of the function
- current_min and current_max are the values in the tuple but their *names* CANNOT be used outside of the function

Accessing Returned Tuples

```
let data = [8, -6, 2, 109, 3, 71]
let bounds = min_max(array: data)
// bounds is a tuple of type (Int, Int) because the
// value being returned from min_max(array:) is a
// tuple of type (Int, Int)
print(“min is \(bounds.min) and max is
  \(bounds.max)”)
```

Output:

```
min is -6 and max is 109
```

Miscellaneous

- You don’t have to use a function’s return value or store it in a constant when the function ends
  - Simply call the function on one line
  - However, if a function says it will return a value, it must do so and the type must be correct