Functions

What is a Function?

A function is a named, reusable block of code responsible for a certain task. It consists of a definition that includes the `func` keyword, name, optional parameters, and return type as well as a body that contains the code block needed to execute its task.

```
func washCar() -> Void {
  print("Soap")
  print("Scrub")
  print("Rinse")
  print("Dry")
}
```

Calling a Function

The process of executing the code inside the body of a function is known as calling it. A function is called by typing its name followed by a set of parentheses, `()`, which should hold any necessary arguments.

```
func greetLearner() {
 print("Welcome to Codecademy!")
}

// Function call:
greetLearner() // Prints: Welcome to
Codecademy!
```

Returning a Value

A value can be passed from a function with a `return` statement. If a function returns a value, the return type must be specified in the function definition.

```
let birthYear = 1994
var currentYear = 2020

func findAge() -> Int {
  return currentYear - birthYear
}

print(findAge()) // Prints: 26
```

Multiple Parameters

A function can accept multiple parameters in its definition. All parameters must be separated by commas and assigned arguments during the function call.

```
func convertFracToDec(numerator: Double,
denominator: Double) -> Double {
  return numerator / denominator
}

let decimal = convertFracToDec(numerator: 1.0,
denominator: 2.0)
print(decimal) // Prints:  0.5
```

Returning Multiple Values

A function can return multiple values in the form of a tuple. When a tuple is returned, each value within its parentheses needs to be labeled and assigned a type in the function definition.

```
func smartphoneModel() -> (name: String,
version: String, yearReleased: Int) {
  return ("iPhone", "8 Plus", 2017)
}

let phone = smartphoneModel()

print(phone.name) // Prints: iPhone
print(phone.version) // Prints: 8 Plus
print(phone.yearReleased) // Prints: 2017
```

Omitting Argument Labels

An argument label can be omitted in the function call when an `_` is prepended to a parameter in the function definition.

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

print(findDifference(6, b: 4)) // Prints: 2
```

Parameters and Arguments

Parameters are optional input values that exist between the `()` in a function definition. For each defined parameter, a real value must be passed in as an argument during the function call.

```
func findSquarePerimeter(side: Int) -> Int {
  return side * 4
}

let perimeter = findSquarePerimeter(side: 5)
print(perimeter) // Prints: 20

// Parameter: side
// Argument:  5
```

Implicit Return

Implicit returns were introduced in Swift version 5.1 to reduce the amount of code within a function body. An implicit return allows for an omitted `return` keyword from a function that returns a single value or expression.

```
func nextTotalSolarEclipse() -> String {
  "April 8th, 2024 🌎"
}

print(nextTotalSolarEclipse()) // Prints:
April 8th, 2024 🌎
```

Default Parameters

A default parameter has a real value assigned to a parameter in the function's definition. When a function with a default parameter is called, an argument for that parameter is not required. If the argument is included, that value will overwrite the default value and be used in the function body.

```
func timeToFinishBook(numWords: Double,
wordsPerMin: Double = 200) -> Double {
  let totalMinutes = numWords / wordsPerMin
  return totalMinutes / 60
}

print("\(timeToFinishBook(numWords: 93000))
hours")
// Prints: 7.75 hours
```

Variadic Parameters

A variadic parameter is a parameter that accepts zero or more values of a certain type. It is denoted by three consecutive dots, `...`, following a parameter's data type in the function definition.

```
func totalStudents(students: String...) -> Int
{
  let numStudents = students.count
  return numStudents
}

print(totalStudents(students: "Jamie",
"Michael", "Rose", "Idris")) // Prints: 4
```

In-Out Parameters

An in-out parameter allows a function to reassign the value of a variable passed in as an argument. An in-out parameter is denoted by `inout` in the function definition, and when the function is called, its variable argument must be prepended with an `&`.

```
var currentSeason = "Winter"

func determineSeason(monthNum: Int, season:
inout String) {

switch monthNum {
  case 1...2:
    season = "Winter ⛄️"
  case 3...6:
    season = "Spring 🌱"
  case 7...9:
    season = "Summer ⛱"
  case 10...11:
    season = "Autumn 🍂"
  default:
    season = "Unknown"
  }
}

determineSeason(monthNum: 4, season:
&currentSeason)

print(currentSeason) // Spring 🌱
```