

## SWIFT TRAINING - WEEK 3-4

## Arrays

# Chapter 5: Arrays

## Introduction

Often when you're dealing with data you don't just have a fixed amount of elements. Take for example a program where you compute the average of multiple grades in a class:

```
var grade1 = 4
var grade2 = 3
```

```
var average = Double(grade1 + grade2) / 2.0
print("Average grade: \(average)")
```

What if we wanted the program to also work when we have 3 grades? We'd have to change our program to work with 3 grades.

```
var grade1 = 4
var grade2 = 3
var grade3 = 5
```

```
var average = Double(grade1 + grade2 + grade3) / 3.0
print("Average grade: \(average)")
```

After doing this it will no longer work with 2 grades. What if we wanted our program to work with any number of grades between 1 grade and 10 grades.

It's not practical to write a separate program for each case. We would like to have something like a list of grades. This list would contain any number of grades. This is where arrays come in.

## What is an array?

An array is an ordered collection that stores multiple values of the same type. That means that an array of `Int` can only store `Int` values. And you can only insert `Int` values in it.

## Declaring Arrays

To declare an array you can use the square brackets syntax([Type]).

```
var arrayOfInts: [Int]
```

You can initialize an array with an array literal. An array literal is a list of values, separated by commas, surrounded by a pair of square brackets:

[value, value, ...]

```
var arrayOfInts: [Int] = [1, 2, 3]
```

```
var arrayOfStrings: [String] = ["We", "♥", "Swift"]
```

Keep in mind that you can create empty arrays if you don't write any values.

```
var emptyArray: [Int] = []
```

## Getting values

To get all the values from an array you can use the for-in syntax. This is called iterating through an array.

```
var listOfNumbers = [1, 2, 3, 10, 100] // an array of numbers
```

```
var listOfNames = ["Andrei", "Silviu", "Claudiu"] // an array of strings
```

```
for number in listOfNumbers {  
    print(number)
```

```
}
```

```
// 1
```

```
// 2
```

```
// 3
```

```
// 10
```

```
// 100
```

```
for name in listOfNames {  
    print("Hello " + name + "!")
```

```
}
```

```
// Hello Andrei!
```

```
// Hello Silviu!
```

```
// Hello Claudiu!
```

To get the number of elements in an array you can use the count property.

```
var listOfNumbers = [1, 2, 3, 10, 100] // an array of numbers
```

```
print(listOfNumbers.count) // 5
```

You can access specific elements from an array using the subscript syntax. To do this pass the index of the value you want to retrieve within square brackets immediately after the name of the array. Also you can get a subsequence from the array if you pass a range instead of an index. Element in an array are indexed from 0 to the number of elements minus one. So an array with 3 elements will have elements at index 0, 1 and 2.

```
var listOfNumbers = [1, 2, 3, 10, 100] // an array of numbers
```

```
listOfNumbers[0] // 1
```

```
listOfNumbers[1] // 2
```

```
listOfNumbers[2] // 3
```

```
listOfNumbers[3] // 10
```

```
listOfNumbers[4] // 100
```

```
//listOfNumbers[5]// this gives an error uncomment this line to see it
```

```
listOfNumbers[1...2] // [2, 3] this is a subsequence of the original array
```

## Adding values

You can add elements to the end of an array using the `append` method.

```
// create a empty array of integers
```

```
var numbers: [Int] = []
```

```
for i in 1...5 {  
    numbers.append(i)  
    print(numbers)  
    // [1]  
    // [1, 2]  
    // [1, 2, 3]  
    // [1, 2, 3, 4]  
    // [1, 2, 3, 4, 5]  
}
```

```
print(numbers)
```

```
// [1, 2, 3, 4, 5]
```

To insert an item into the array at a specified index, call the array's `insert(at:)` method.

```
var numbers: [Int] = [1, 2, 3]
```

```
numbers.insert(0, at: 0) // numbers will be [0, 1, 2, 3]
numbers.insert(9, at: 1) // numbers will be [0, 9, 1, 2, 3]
```

You can also append another array using the `+=` operator.

```
var numbers: [Int] = [1, 2, 3]
```

```
numbers += [4, 5, 6] // numbers will be [1, 2, 3, 4, 5, 6]
```

```
// or just one value
```

```
numbers += [7] // numbers will be [1, 2, 3, 4, 5, 6, 7]
```

## Removing Values

To remove an item from a specific index call the `remove(at:)` method.

```
var numbers: [Int] = [1, 2, 3]
```

```
numbers.remove(at: 0) // numbers will be [2, 3]
```

## Changing values

To change a value use the assignment operator (`=`) after the subscript syntax.

```
var numbers: [Int] = [1, 2, 3]
```

```
numbers[0] = 7 // numbers will be [7, 2, 3]
```

```
numbers[1] = 5 // numbers will be [7, 5, 3]
```

```
numbers[2] = 4 // numbers will be [7, 5, 4]
```

Or you could replace a subsequence of values using range subscripting.

```
var numbers: [Int] = [1, 2, 3, 4, 5, 6]
```

```
numbers[2...4] = [0, 0] // numbers will now be [1, 2, 0, 0, 6].
```

Keep in mind that you don't need to replace a sequence with another sequence with the same number of elements. In the example above numbers had 6 elements and after the replacement of the subsequence `2...4` (`[3, 4, 5]`) it had 5.

## Type Inference

Thanks to Swift's type inference, you don't have to declare the type of an array if you initialize it with something other than an empty array literal(`[]`).

```
// arrayOfNumbers will be of type [Int]
var arrayOfNumbers = [1, 2, 3]

// arrayOfStrings will be of type [String]
var arrayOfStrings = ["We", "♥", "Swift"]

// arrayOfBools will be of type [Bool]
var arrayOfBools = [true, false, true, true, false]

// this is the proper way of declaring a empty array of Int - [Int]
var emptyArrayOfInts: [Int] = []

// this will infer into [Int] because the right hand side of the
// assignment has a known type [Int]
var anotherEmptyArray = emptyArrayOfInts
```

## Copy Behavior

Swift's Array types are implemented as structures. This means that arrays are copied when they are assigned to a new constant or variable, or when they are passed to a function or method.

```
var numbers = [1, 2, 3]
var otherNumbers = numbers // this will create a copy of numbers

// this will append 4 to otherNumbers but not to numbers
otherNumbers.append(4)

// numbers = [1, 2, 3]
// otherNumbers = [1, 2, 3, 4]
```

## Mutability

If you create an array and assign it to a variable, the collection that is created will be mutable. This means that you can change (or mutate) the collection after it is created. Changes can be done by adding, removing, or changing items in the collection. Conversely, if you assign an array to a constant, that array is immutable, and its size and contents cannot be changed. In other words if you want to be able to change an array declare it using the `var` keyword, and if you don't want to be able to change it use the `let` keyword.

```
var numbers = [1, 2, 3, 4, 5, 6]
```

```
numbers.append(7) // [1, 2, 3, 4, 5, 6, 7]  
numbers.remove(at: 0) // [2, 3, 4, 5, 6, 7]
```

```
let strings = ["We", "♥", "Swift"]
```

```
// the next lines will not compile!  
strings.append("!") // this will give an error because strings is  
immutable  
strings.remove(at: 0) // this will give a similar error
```

### 5.1 Max

Print the maximum value from listOfNumbers.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
// your code here
```

### 5.2 Odd

Print all the odd numbers in listOfNumbers.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
// your code here
```

### 5.3 Sum

Print the sum of all the numbers in listOfNumbers.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
// your code here
```

### 5.4 Odd Index

Print all the numbers in listOfNumbers that are located at odd indexes.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
// your code here
```

### 5.5 Going backwards

Print listOfNumbers in reverse order.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
// your code here
```

## 5.6 Reverse

Reverse the order of the elements in listOfNumbers without creating a new array. Use the algorithm from the Swap problem in Chapter 1.

[1, 2, 3] -> [3, 2, 1]

[3, 4, 5, 6] -> [6, 5, 4, 3]

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

**Solution Below - Insert Comment lines explaining how it works.**

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
var firstIndex = 0
```

```
var lastIndex = listOfNumbers.count - 1
```

```
while firstIndex < lastIndex {
```

```
    // swap
```

```
    var tmp = listOfNumbers[firstIndex]
```

```
    listOfNumbers[firstIndex] = listOfNumbers[lastIndex]
```

```
    listOfNumbers[lastIndex] = tmp
```

```
    // go to next pair
```

```
    ++firstIndex
```

```
    ++lastIndex
```

```
}
```

## 5.7 Sorting

Sort the values in `listOfNumbers` in descending order.

**HINT:**

```
array.sort(<) // will sort the array in ascending order  
array.sort(>) // will sort the array in descending order
```

**5.8 Search**

Find out if `x` appears in `listOfNumbers`. Print yes if true and no otherwise.

```
var listOfNumbers = [1, 2, 3, 10, 100]
```

```
var x = 10
```

```
var xAppears = false
```

```
// your code here
```

**5.9 Divisors**

Given a number find and store all its divisors in an array called `divisors`, then print the divisors in order on separate lines.

```
var number = 60
```

```
var divisors: [Int] = []
```

```
// your code here
```