

# Higher Order Functions in Swift (Sorted, Map, Filter, Reduce)



Doug Galante [Follow](#)

May 18, 2017 · 5 min read

Higher order functions are simply functions that operate on other functions by either taking a function as an argument, or returning a function. Swift's Array type has a few methods that are higher order functions: sorted, map, filter, and reduce. These methods use closures to allow us to pass in functionality that can then determine how we want the method to sort, map, filter, or reduce an array of objects.

In this post I'll give an example of each of these methods and talk a little bit about how the syntax of the closure can change based on what swift can infer.

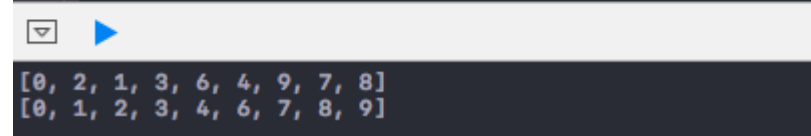
If you are unfamiliar with closures I recommend reading up on them a bit first.

Wouldn't hurt to be familiar with generics as well.

## Sorted

Lets start with the sorted function. If we call sorted on an array it will return a new array that sorted in ascending order. In order for this method to work the elements in the array need to conform to the Comparable protocol.

```
4 let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]
5
6 let ascendingNumbers = numbers.sorted()
7
8 print(numbers)
9 print(ascendingNumbers)
10
```

The image shows a Swift Playground interface. The code is entered in a dark-themed editor. Below the code, there is a play button (a blue triangle) and a dropdown menu. The output of the code is displayed in a light-colored area at the bottom, showing two arrays: the original array [0, 2, 1, 3, 6, 4, 9, 7, 8] and the sorted array [0, 1, 2, 3, 4, 6, 7, 8, 9].

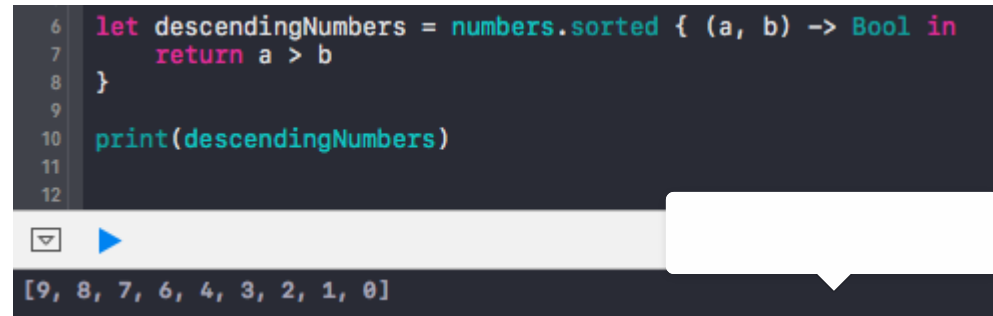
```
[0, 2, 1, 3, 6, 4, 9, 7, 8]
[0, 1, 2, 3, 4, 6, 7, 8, 9]
```

If we want to specify exactly how we want to sort the array — we will need to use the higher order function *sorted(by:)*

```
numbers.sorted(by: (Int, Int) -> Bool)
```

Notice that the argument is a closure. Within this closure we can define exactly how we want to sort our array of numbers. ***note:** Swift is able to infer that numbers is an array of Ints and can show us that the function we need to pass as an argument should be of type  $(Int, Int) \rightarrow Bool$ . If we were sorting an array of strings the argument would be of type  $(String, String) \rightarrow Bool$ .*

For the first example lets keep things simple and sort the array in descending order.



```
6 let descendingNumbers = numbers.sorted { (a, b) -> Bool in
7     return a > b
8 }
9
10 print(descendingNumbers)
11
12
```

The screenshot shows a Swift code editor with a dark background. The code defines a variable `descendingNumbers` by calling `numbers.sorted` with a closure that returns `a > b`. A `print` statement is used to output the sorted array. Below the code, a console window shows the output: `[9, 8, 7, 6, 4, 3, 2, 1, 0]`. A white speech bubble is positioned over the console output.

Using trailing closure syntax we can specify that within our closure we want to return the bool `a > b`. If this is your first time seeing the `sorted` method you might be wondering what `a` and `b` represent. Lets add a print statement to get a better look.

```
4 let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]
5
6 let descendingNumbers = numbers.sorted { (a, b) -> Bool in
7     print("a = \(a)")
8     print("b = \(b)")
9     print(a > b)
10    return a > b
11 }
12
13 print(descendingNumbers)
14
```

▼ ▶


```
a = 2
b = 0
true
a = 1
b = 0
true
a = 1
b = 2
false
a = 3
b = 0
true
a = 3
b = 1
true
a = 3
b = 2
true
```

If we look at our original array *numbers*, we can see that the sorted method is iterating through the array and comparing various numbers within it. In the first iteration **a = numbers[1]**, and **b = numbers[0]**. If the boolean **a > b** returns **true** the sorted method performs some background logic to put them in the correct order. The next iteration then steps to the next numbers to compare and repeats this process until the array is sorted.

It isn't super important to understand how the array is being sorted, but it is important to understand what **a** and **b** represent in this context. when we return **a > b** we are saying we want larger numbers in the array to come before smaller numbers.

Lets try sorting the array by separating even and odd numbers.

```
3  
4 let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]  
5  
6 let evenNumbersFirst = numbers.sorted { (a, b) -> Bool in  
7     return a % 2 == 0  
8 }  
9  
10 print(evenNumbersFirst)  
11  
12
```

A Swift Playground interface showing the execution of the code. Below the code editor, there is a blue play button icon. Below that, a dark bar displays the output of the print statement: [8, 4, 6, 2, 0, 1, 3, 9, 7].

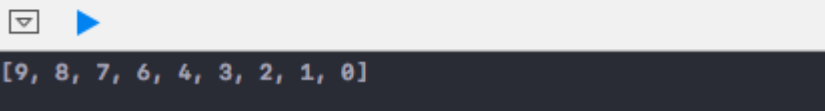
[8, 4, 6, 2, 0, 1, 3, 9, 7]

In this case all of the even numbers come before the odd numbers in the array after it has been sorted.

## Closure Syntax

With the *sorted(by:)* function we can reduce the syntax quite a bit based on what swift can infer.

```
3  
4 let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]  
5  
6 let descendingNumbers = numbers.sorted(by: >)  
7  
8 print(descendingNumbers)  
9  
10
```



```
[9, 8, 7, 6, 4, 3, 2, 1, 0]
```

So instead of writing out a closure we can just pass “>”. Seems ridiculous right? Well if we command click on “>” we can see that it is actually a function of the type (Int, Int) -> Bool which is the same type as the *by:* parameter.



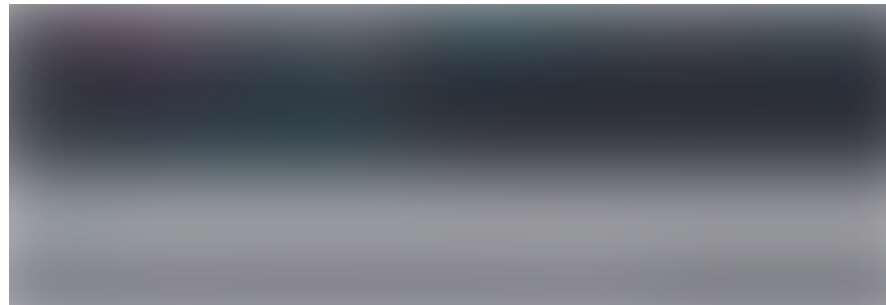
For a complete breakdown of how the syntax is being broken down check out the apple documentation on [Inferring Type From Context](#).

## Map

Mapping is similar to sort in that it iterates through the array that is calling it, but instead of sorting it changes each element of the array based on the closure passed to the method.



Notice that the return of the closure is a **T**. Since an array is a generic type and we are returning a new array, we can return an array with a type different than the starting array (if we so choose). Lets convert our numbers into strings.



If you read up on how to write inline syntax with closures you may write the same function this way.



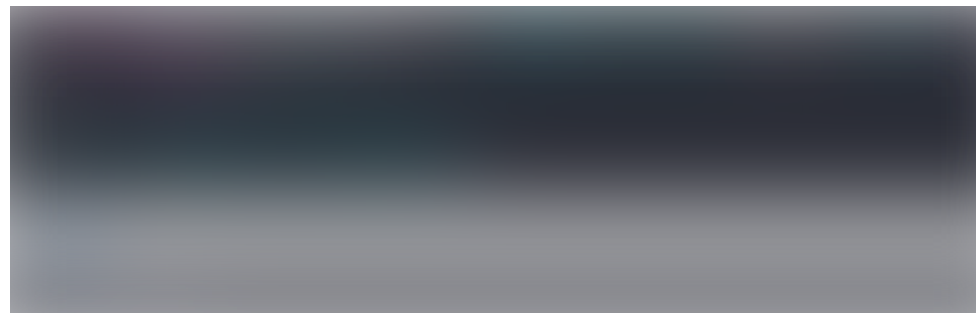


## Filter

the filter method will return an array that has only elements that pass your filter specified in your closure.



Lets get all the numbers less than 5:





Inline syntax:



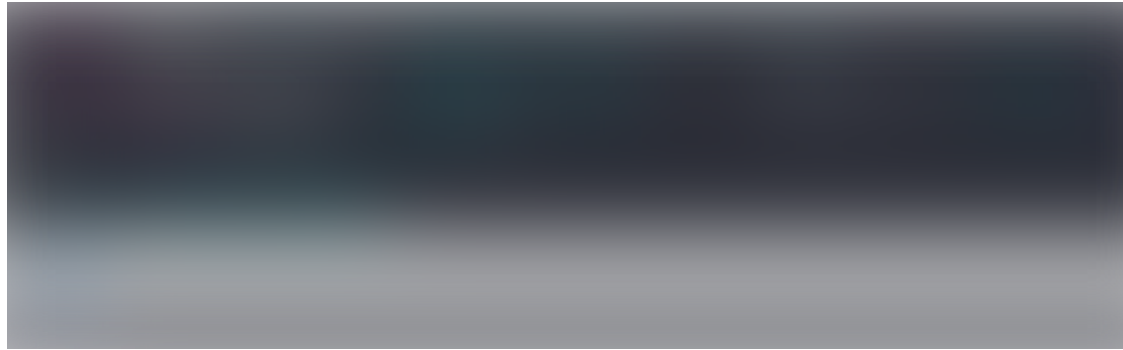
## Reduce

The reduce function allows you to combine all the elements in an array and return an object of any type (generics!!!!)

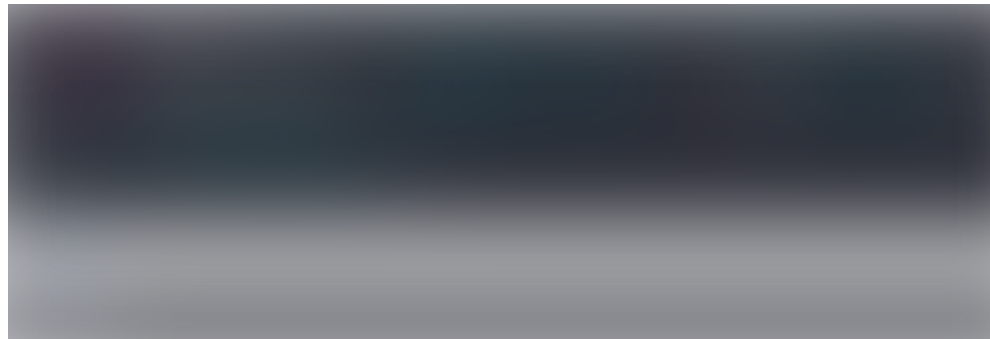


Notice that reduce has two parameters — `initialResult` and `nextPartialResult`. We need the initial result to tell us where to start, and the method then operates on that result based on the logic in the closure.

lets try turning all the numbers into one long string.



Inline syntax:



Using higher order functions may feel a bit different than other OOP concepts you have come across. Understanding them relies a lot on understanding other concepts like closures, recursion, and functional programming. Also having a grasp on generics can really ramp up the power versatility of your closures— luckily apple is doing a lot of that stuff

behind the scenes for us. Keep digging into some of these topics if you found them interesting — I hope at the very least this post was helpful :)

[Swift](#) [iOS](#) [Higher Order Function](#)

### Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

### Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

### Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade

[About](#) [Help](#) [Legal](#)