Swift - Arrays

Swift 4 arrays are used to store ordered lists of values of the same type. Swift 4 puts strict checking which does not allow you to enter a wrong type in an array, even by mistake.

If you assign a created array to a variable, then it is always mutable, which means you can change it by adding, removing, or changing its items; but if you assign an array to a constant, then that array is immutable, and its size and contents cannot be changed.

Creating Arrays

You can create an empty array of a certain type using the following initializer syntax −

```
var someArray = [SomeType]()
```

Here is the syntax to create an array of a given size a* and initialize it with a value −

```
var someArray = [SomeType](count: NumbeOfElements, repeatedValue: InitialVa
```

You can use the following statement to create an empty array of **Int** type having 3 elements and the initial value as zero −

```
var someInts = [Int](count: 3, repeatedValue: 0)
```

Following is one more example to create an array of three elements and assign three values to that array −

```
var someInts:[Int] = [10, 20, 30]
```

Accessing Arrays

You can retrieve a value from an array by using **subscript** syntax, passing the index of the value you want to retrieve within square brackets immediately after the name of the array as follows −

```
var someVar = someArray[index]
```

Here, the **index** starts from 0 which means the first element can be accessed using the index as 0, the second element can be accessed using the index as 1 and so on. The following example shows how to create, initialize, and access arrays −

```
var someInts = [Int](count: 3, repeatedValue: 10)

var someVar = someInts[0]
```

```
print( "Value of first element is \(someVar)" )
print( "Value of second element is \(someInts[1])" )
print( "Value of third element is \(someInts[2])" )
```

When the above code is compiled and executed, it produces the following result −

```
Value of first element is 10
Value of second element is 10
Value of third element is 10
```

Modifying Arrays

You can use **append()** method or addition assignment operator (+=) to add a new item at the end of an array. Take a look at the following example. Here, initially, we create an empty array and then add new elements into the same array −

Live Demo

```
var someInts = [Int]()

someInts.append(20)
someInts.append(30)
someInts += [40]

var someVar = someInts[0]

print( "Value of first element is \(someVar)" )
print( "Value of second element is \(someInts[1])" )
print( "Value of third element is \(someInts[2])" )
```

When the above code is compiled and executed, it produces the following result −

```
Value of first element is 20
Value of second element is 30
Value of third element is 40
```

You can modify an existing element of an Array by assigning a new value at a given index as shown in the following example −

Live Demo

```
var someInts = [Int]()

someInts.append(20)
someInts.append(30)
someInts += [40]

// Modify last element
someInts[2] = 50
```

```
var someVar = someInts[0]

print( "Value of first element is \(someVar)" )
print( "Value of second element is \(someInts[1])" )
print( "Value of third element is \(someInts[2])" )
```

When the above code is compiled and executed, it produces the following result −

```
Value of first element is 20
Value of second element is 30
Value of third element is 50
```

Iterating Over an Array

You can use **for-in** loop to iterate over the entire set of values in an array as shown in the following example −

Live Demo

```
var someStrs = [String]()

someStrs.append("Apple")
someStrs.append("Amazon")
someStrs += ["Google"]
for item in someStrs {
   print(item)
}
```

When the above code is compiled and executed, it produces the following result −

```
Apple
Amazon
Google
```

You can use **enumerate()** function which returns the index of an item along with its value as shown below in the following example −

Live Demo

```
var someStrs = [String]()

someStrs.append("Apple")
someStrs.append("Amazon")
someStrs += ["Google"]

for (index, item) in someStrs.enumerated() {
   print("Value at index = \(index) is \(item)")
}
```

When the above code is compiled and executed, it produces the following result −

```
Value at index = 0 is Apple
Value at index = 1 is Amazon
Value at index = 2 is Google
```

Adding Two Arrays

You can use the addition operator (+) to add two arrays of the same type which will yield a new array with a combination of values from the two arrays as follows −

```
var intsA = [Int](count:2, repeatedValue: 2)
var intsB = [Int](count:3, repeatedValue: 1)

var intsC = intsA + intsB
for item in intsC {
   print(item)
}
```

When the above code is compiled and executed, it produces the following result −

```
2
2
1
1
1
```

The count Property

You can use the read-only **count** property of an array to find out the number of items in an array shown below −

```
var intsA = [Int](count:2, repeatedValue: 2)
var intsB = [Int](count:3, repeatedValue: 1)

var intsC = intsA + intsB

print("Total items in intsA = \(intsA.count)")
print("Total items in intsB = \(intsB.count)")
print("Total items in intsC = \(intsC.count)")
```

When the above code is compiled and executed, it produces the following result −

```
Total items in intsA = 2
Total items in intsB = 3
Total items in intsC = 5
```

The empty Property

You can use the read-only **empty** property of an array to find out whether an array is empty or not as shown below −

```
var intsA = [Int](count:2, repeatedValue: 2)
var intsB = [Int](count:3, repeatedValue: 1)
var intsC = [Int]()

print("intsA.isEmpty = \(intsA.isEmpty)")
print("intsB.isEmpty = \(intsB.isEmpty)")
print("intsC.isEmpty = \(intsC.isEmpty)")
```

When the above code is compiled and executed, it produces the following result −

```
intsA.isEmpty = false
intsB.isEmpty = false
intsC.isEmpty = true
```