# For-in vs While Loop

The 'for-in' loop in Swift is optimized for performance, particularly when working with a large array, as 'while' loops involve the use of a variable for incrementation. The variable increment operation in 'while' loops can become resource-intensive, especially with very large arrays. It is advisable to use 'for' loops when dealing with a finite range and 'while' loops when your logic requires dynamic range adjustments.

```swift
import Dispatch

func measureTime(block: () -> Void) -> Double {
    let startTime = DispatchTime.now()                  // DispatchTime
    block()
    let endTime = DispatchTime.now()                    // DispatchTime
    let nanoseconds = endTime.uptimeNanoseconds -       // 37,972,891,583
        startTime.uptimeNanoseconds
    let timeInterval = Double(nanoseconds) /            // 37.972891583
        1_000_000_000.0
    return timeInterval                                 // 37.972891583
}

func forLoop(array: [Int]) {
    var k = 0                                           // 0
    for i in 0..<array.count {
        k += 1                                          // 900,000
    }
}

func whileLoop(array: [Int]) {
    var i = 0                                           // 0
    var k = 0                                           // 0
    while  i < array.count {
        k += 1                                          // 900,000
        i += 1                                          // 900,000
    }
}

var largeArray = Array(repeating: 1, count: 900000)     // Array of 900,000 Int elements

let elapsedTimeForLoop = measureTime {                  // 18.4537685
    forLoop(array: largeArray)                          // ()
}

let elapsedTimeWhileLoop = measureTime {                // 37.972891583
    whileLoop(array: largeArray)                        // ()
}
```

The 'for-in' loop maintains constant starting and ending points. In the following example, I added elements to the array in the middle of the loop. Despite the dynamic change in the array's count, the loop still executed only four times.

```
2
3
4   var array = [1,2,3,4]                          Array of 4 Int elements
5   var numberOfTimesLoopRuns = 0                  0
6   for i in 0..<array.count {
7       if i < 3 {
8           if let value = array.last {
9               array.append(value + 1)            Array of 7 Int elements
10          }
11      }
12      numberOfTimesLoopRuns += 1                 4
13  }
14  print(numberOfTimesLoopRuns)                   "4\n"
15  print(array)                                   "[1, 2, 3, 4, 5, 6, 7]\n"
16
```

In this second example, I incremented the 'i' variable within the loop. However, it is evident that the loop ignores the updated 'i' value and continues with the originally incremented value. Consequently, the loop still executed only four times.

```
2
3
4   var array = [1,2,3,4]                          Array of 4 Int elements
5   var numberOfTimesLoopRuns = 0                  0
6   for var i in 0..<array.count {
7       print("before increment value of i = \(i)")   "before increment value of i = 3\n"
8       i += 2                                     5
9       print("after  increment value of i = \(i)")   "after  increment value of i = 5\n"
10      numberOfTimesLoopRuns += 1                 4
11  }
12  print(numberOfTimesLoopRuns)                   "4\n"
13  print(array)                                   "[1, 2, 3, 4]\n"
14
15
```

```
before increment value of i = 0
after  increment value of i = 2
before increment value of i = 1
after  increment value of i = 3
before increment value of i = 2
after  increment value of i = 4
before increment value of i = 3
after  increment value of i = 5
4
[1, 2, 3, 4]
```

The while loop, on the other hand, adjusts its starting and ending points based on changes within the loop. In the following example, I added elements to the array in the middle of the loop. Due to the dynamic change in the array's count, the loop executed seven times, matching the new array count.

```
1
2   var array = [1,2,3,4]                          Array of 4 Int elements
3   var numberOfTimesLoopRuns = 0                  0
4   var j = 0                                      0
5
6   while j < array.count {
7       if j < 3 {
8           if let value = array.last {
9               array.append(value + 1)            Array of 7 Int elements
10          }
11      }
12      numberOfTimesLoopRuns += 1                 7
13      j += 1                                     7
14  }
15
16  print(numberOfTimesLoopRuns)                   "7\n"
17  print(array)                                   "[1, 2, 3, 4, 5, 6, 7]\n"
18
```

```
7
[1, 2, 3, 4, 5, 6, 7]
```

In the second example using a while loop, I incremented the 'j' variable within the loop. It is evident that the loop considers the updated 'j' value and continues with the newly incremented value. Consequently, the loop executed only two times less than the original four times.

```
1
2   var array = [1,2,3,4]                          Array of 4 Int elements
3   var numberOfTimesLoopRuns = 0                  0
4   var j = 0                                      0
5
6   while j < array.count {
7
8       j += 2                                     4
9       numberOfTimesLoopRuns += 1                 2
10  }
11
12  print(numberOfTimesLoopRuns)                   "2\n"
13  print(array)                                   "[1, 2, 3, 4]\n"
14
```

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