Published in Dev Genius

Randil Tennakoon
Dec 19, 2021 · 3 min read · Listen

The Difference between Arrays and Slices in Golang

This post will discuss the main differences that you could find between Arrays and Slices in Go

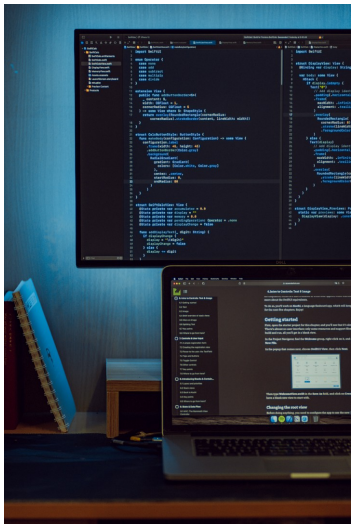

Photo by Safar Safarov on Unsplash

In **Golang**, defining Arrays and Slices might look similar but it has a certain amount of differences that you need to know. Also, you should know when to use Arrays and when to use Slices in your Go code. This post's objective is to give you a basic understanding of the differences between Arrays and Slices in the Go programming language.

Also, before you run any code which is written in Go, you need to download & install Go to your local machine *(you can download Go to your local machine via the link below)*

Download and install
Download and install Go quickly with the steps described here. For other content on installing, you might be interested...
golang.org

Arrays in Go

Arrays can be defined in Golang as below.

```
arr1 := [3]int {7,9,4}

// array size is 3
```

This is the easiest way to declare an Array and assign its values in Golang. But this method can only be used when you initialize a variable inside a function. Other than this, you can follow two other ways to declare an Array in your Go code.

```
ex 1:
var arr1 [3]int = [3]int {2,3}

ex 2:
var arr1 = [3]int {1,7,9}
```

When you declare an Array, you need to clearly define its size and when you initialize the Array, you can assign values up to that size.
Once you define and set up Array values, you can easily print them using the **"fmt"**.

```
ex:
fmt.Println(arr1)
```

When you need to change Array values, you can use the relevant index and change its value accordingly.

```
ex:
arr1[2] = 6
fmt.Println(arr1)
```

Slices in Go

You can use the same method which we used to declare an Array to declare a Slice in Golang. But the difference is, in slices you don't have to mention the **size** for that particular slice. But in Arrays, we had to define its size when declaring.

```
cities := []string {"London", "NYC", "Colombo", "Tokyo"}

OR

var cities = []string {"London", "NYC", "Colombo", "Tokyo"}

OR

var cities []string = []string {"London", "NYC", "Colombo", "Tokyo"}
```

You can use any method from the above list when you declare a Slice in your Go code. Also, you can easily print a slice using **"fmt"** as below.

Randil Tennakoon
128 Followers
Technical Writer | DevOps | Python | Software Engineering
Follow

More from Medium
Alexander Nguyen in Level Up Coding
$150,000 Amazon Engineer vs. $300,000 Google Engineer
Edmond fagbuyi
Part 1: SOLID Design Principles In Golang
Kevin Takano
How to DESTROY your career as a JR software engineer in 15 steps (with memes) :D
Oleksii Trekhleb in ITNEXT
SOLID Principles Sketches

```
fmt.Println(cities)
```

Since a slice doesn't have a specific size, we can append values to a slice in addition to changing its existing values. But in Arrays, we couldn't append values and all we could do is change its existing values.
You can **append values** to a slice using two different methods. If you want to append values to the original slice, you can do it as below.

```
cities = append(cities, "LA")
fmt.Println(cities)
```

This will add **LA** to the end of the original slice and update it. Then, when you print the original slice, it will be outputted LA along with other city names.
But in slices, you can use the built-in **append()** function to add values and it will be creating another slice and append values without overwriting the original slice.

```
ex:

cities := []string {"London", "NYC", "Colombo", "Tokyo"}
fmt.Println(cities)

addCity := append(cities, "Auckland")

fmt.Println(addCity)
fmt.Println(cities)
```

In the above example, I created a separate variable called **addCity** and used the append function to add values. When you print the **addCity** variable, it will output cities along with **Auckland** but it will not overwrite the original slice. You can verify it by printing the original slice once again.

```
// output
[London NYC Colombo Tokyo]
[London NYC Colombo Tokyo Auckland]
[London NYC Colombo Tokyo]
```

Also, you can use this append function directly with the **fmt.Println()** and it will also do the same as above.

```
fmt.Println(append(cities, "Auckland"))
```

Thank you for reading!