Working with arrays

This lesson explains arrays, multidimensional arrays and how to print them in GO

WE'LL COVER THE FOLLOWING

Arrays

The type `[n]T` is an array of `n` values of type `T`.

The expression:

Environment Variables

Key:	Value:
GOPATH	/go

```
var a [10]int
```

declares a variable `a` as an array of ten integers.

An array's length is part of its type, so arrays *cannot* be resized. This seems limiting, but don't worry; Go provides a convenient way of working with arrays.

Let's take a look at an example below:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
    var a [2]string   //array of size 2
    a[0] = "Hello"    //Zero index of "a" has "Hello"
    a[1] = "World"    //1st index of "a" has "World"
    fmt.Println(a[0], a[1]) // will print Hello World
    fmt.Println(a)           // will print [Hello World]
}
```

Below is another example to further help understand the concept:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        var a [2]string
        a[0] = "Hello"
        a[1] = "World"
        fmt.Println(a[0], a[1])
        fmt.Println(a)

        primes := [6]int{2, 3, 5, 7, 11, 13}  // array of prime numbers of size 6
        fmt.Println(primes)
}
```

You can also set the array entries as you declare the array:

Environment Variables

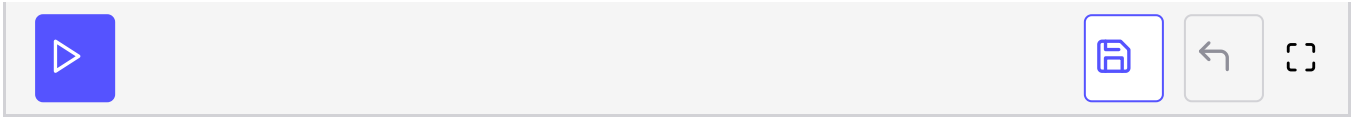
Key:	Value:
GOPATH	/go

```
package main

import "fmt"
```

```
func main() {
    a  := [2]string{"hello", "world!"}

    fmt.Printf("%q", a)
}
```



Finally, you can use an **ellipsis** to use an implicit length when you pass the values:

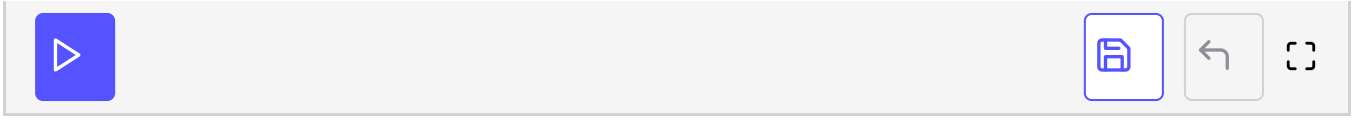
Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        a := [...]string{"hello", "world!"}
        fmt.Printf("%q", a)
}
```



Printing arrays

Note how we used the `fmt` package using `Printf` and used the `%q` "verb" to print each element quoted.

If we had used `Println` or the `%s` verb, we would have had a different result:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        a := [2]string{"hello", "world!"}
        fmt.Println(a)
        // [hello world!]
```

```
        fmt.Printf("%s\n", a)
        // [hello world!]
        fmt.Printf("%q\n", a)
        // ["hello" "world!"]
}
```

Multi-dimensional arrays

You can also create multi-dimensional arrays:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        var a [2][3]string
        for i := 0; i < 2; i++ {
                for j := 0; j < 3; j++ {
                        a[i][j] = fmt.Sprintf("row %d - column %d", i+1, j+1)
                }
        }
        fmt.Printf("%q", a)
        // [["row 1 - column 1" "row 1 - column 2" "row 1 - column 3"]
        //  ["row 2 - column 1" "row 2 - column 2" "row 2 - column 3"]]
}
```

Trying to access or set a value at an index that doesn't exist will prevent your program from compiling, for instance, try to compile the following code:

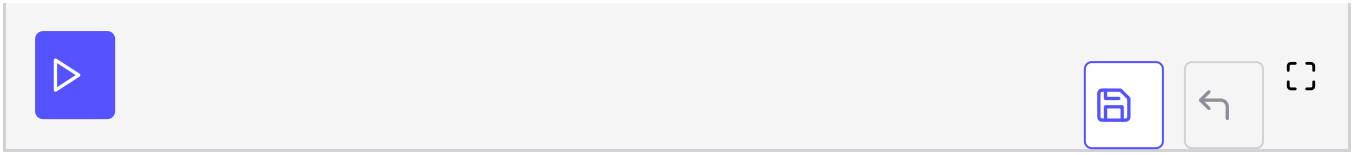
Environment Variables

Key:	Value:
GOPATH	/go

```
package main

func main() {
    var a [2]string
    a[3] = "Hello"
```

```
}
```



Compiler will generate Error

That's because our array is of length **2** meaning that the only 2 available indexes are **0** and **1**. Trying to access index **3** results in an error that tells us that we are trying to access an index that is *out of bounds* since our array only contains 2 elements and we are trying to access the 4th element of the array.

Slices, the type that we are going to see in the next lesson is more often used, due to the fact that we don't always know in advance the length of the array we need.