



Level 3-2

Following the Trail

Arrays & Slices

ON TRACK
with
GOLANG

Declaring Arrays

When creating arrays via manual type declaration, we must set the **max number of elements**.

src/hello/main.go

```
package main
```

```
import "fmt"
```

```
func main() {
```

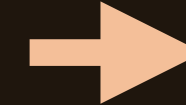
```
    var langs [3]string
```

```
    fmt.Println(langs)
```

```
}
```

\$

go run main.go



[]

← Holds no more than 3 values of type string

Writing to Arrays

We can add elements to arrays by assigning to each specific index.

src/hello/main.go

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var langs [3]string
```

```
    langs[0] = "Go"  
    langs[1] = "Ruby"  
    langs[2] = "JavaScript"
```

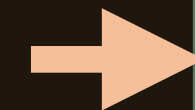
```
    fmt.Println(langs)
```

```
}
```

Index count starts at 0.

\$

go run main.go



[Go Ruby JavaScript]

Arrays Are Not Dynamic

Adding more elements to an array than what was initially expected **raises an out-of-bounds error**.

src/hello/main.go



\$

go run main.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var langs [3]string
7
8     langs[0] = "Go"
9     langs[1] = "Ruby"
10    langs[2] = "JavaScript"
11    langs[3] = "LOLcode"
12    fmt.Println(langs)
13 }
```

→ ./main.go:11: invalid array index 3 (out of bounds for 3-element array)

← Adding to nonexistent space

Slices Are Like Arrays

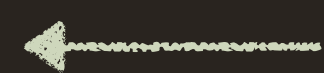
The **slice** type is built on top of arrays to provide more power and convenience. Most array programming in Go is done with **slices** rather than simple arrays.

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var langs []string
```



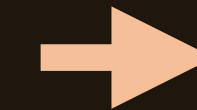
Leaving out max elements creates a slice with a zero value of nil.

```
    fmt.Println(langs)
```

```
}
```

\$

go run main.go



[]

Slices Are Dynamic

A **nil** slice in Go behaves **the same as an empty slice**. It can be appended to using the built-in function `append()`, which takes two arguments: a slice and a variable number of elements.

```
package main
```

```
import "fmt"
```

Returns a new slice that contains the new element

```
func main() {  
    var langs []string
```

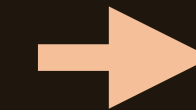
```
    langs = append(langs, "Go")  
    langs = append(langs, "Ruby")  
    langs = append(langs, "JavaScript")  
    langs = append(langs, "LOLcode")  
    fmt.Println(langs)
```

```
}
```



\$

```
go run main.go
```



```
[Go Ruby JavaScript LOLcode]
```

If capacity is not sufficient, a new underlying array will be allocated.