Range in for loops

This lesson discusses range form of for loops and their use to iterate slices in Go

WE'LL COVER THE FOLLOWING

Range

The `range` form of the for loop iterates over a slice or a map. Being able to iterate over all the elements of a data structure is very useful and `range` simplifies the iteration.

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
    for i, v := range pow { //loops over the length of pow
        fmt.Printf("2**%d = %d\n", i, v)
    }
}
```

You can skip the index or value by assigning to `_`. If you only want the index, drop the “, value” entirely.

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
    pow := make([]int, 10)
    for i := range pow {
        pow[i] = 1 << uint(i)
    }
    for _, value := range pow {
        fmt.Printf("%d\n", value)
    }
}
```

Break & continue

As if you were using a normal for loop, you can stop the iteration anytime by using `break`:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        pow := make([]int, 10)
        for i := range pow {
                pow[i] = 1 << uint(i)
                if pow[i] >= 16 {
                        break //stop iterating over pow when it reaches 16
                }
        }
        fmt.Println(pow)
        // [1 2 4 8 16 0 0 0 0 0]
}
```

You can also skip an iteration by using `continue`:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        pow := make([]int, 10)
        for i := range pow {
                if i%2 == 0 {
                        continue //skip each even index of pow
                }
                pow[i] = 1 << uint(i)
        }
        fmt.Println(pow)
        // [0 2 0 8 0 32 0 128 0 512]
}
```

Range and maps

Range can also be used on maps, another commonly used data structure. (discussed in the following lesson) In that case, the first parameter isn't an incremental integer but the map key:

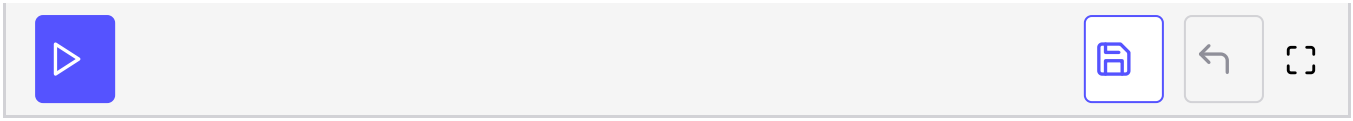
Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        cities := map[string]int{
                "New York":    8336697,
                "Los Angeles": 3857799,
                "Chicago":     2714856,
        }
        for key, value := range cities { //for each key-value pair in cities
                fmt.Printf("%s has %d inhabitants\n", key, value)
        }
}
```



In the code above `map` takes as input the `string` type, in this case, the name of cities, and maps them to integers (a concept explained in the following lesson). The loop iterates over each `key`, `value` pair in cities as `range cities` goes from **zero** to the size of **cities**.

In the next lesson, we will discuss the interesting concept of *maps* in Go.