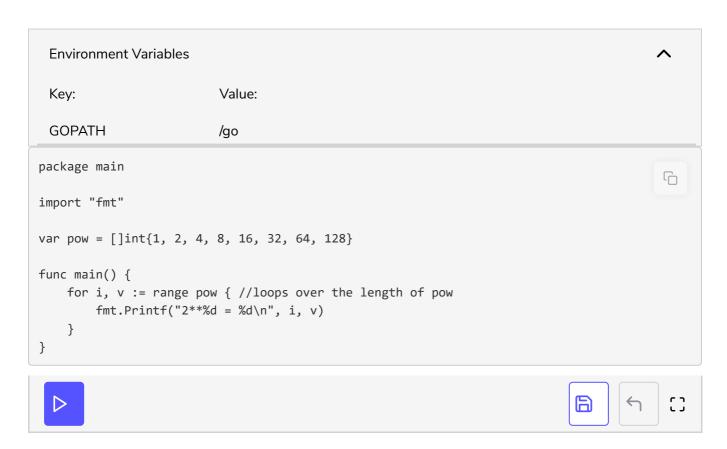
Range in for loops

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

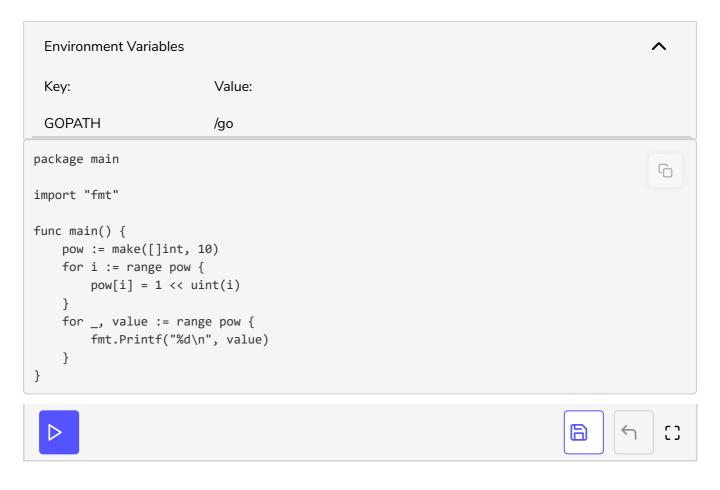
we'll cover the following ^ Range Break & continue Range and maps

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.



You can skip the index or value by assigning to ___. If you only want the index, drop the ", value" entirely.



Break & continue

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

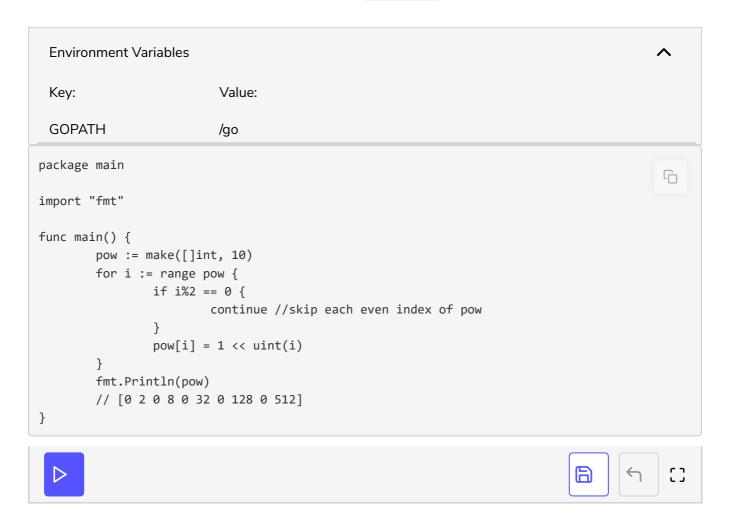
```
Environment Variables
                         Value:
 Key:
 GOPATH
                         /go
package main
                                                                                         import "fmt"
func main() {
        pow := make([]int, 10)
        for i := range pow {
                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]
}
```





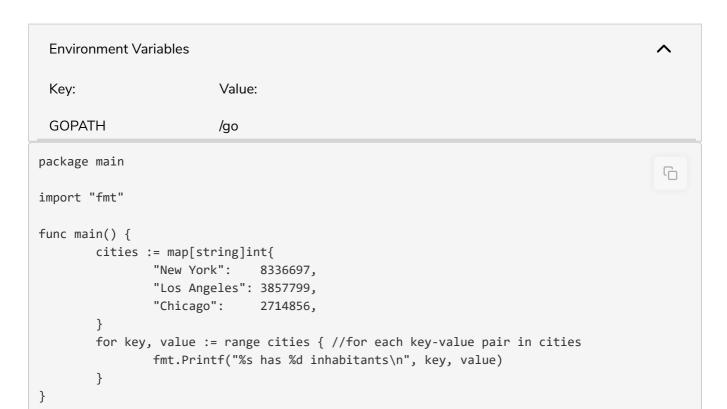


You can also skip an iteration by using continue:



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:









ני

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.