The for-range Construct

This lesson helps you learn how to run a loop on a slice using the for-range construct.

WE'LL COVER THE FOLLOWING

Introduction

This construct can be applied to arrays and slices:

```
for ix, value := range slice1 {
...
}
```

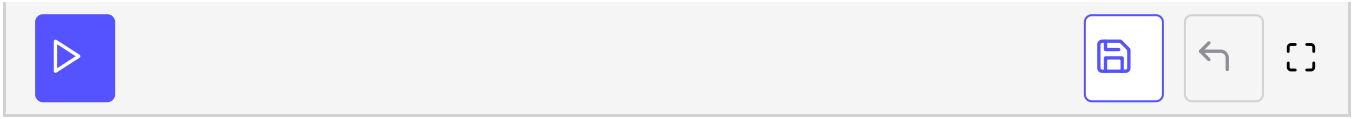
The first value `ix` is the index in the array or slice, and the second is the `value` at that index. They are local variables only known in the body of the for-statement, so `value` is a copy of the slice item at index `ix` and cannot be used to modify the slice!

Implementation of `range` on slices

The following is a simple program that iterates over a slice using the `range` construct.

```
package main
import "fmt"

func main() {
  slice1 := make([]int, 4)
  slice1[0] = 1
  slice1[1] = 2
  slice1[2] = 3
  slice1[3] = 4
```

```
  for ix, value := range slice1 {
    fmt.Printf("Slice at %d is: %d\n", ix, value)


  }
}
```



For Range on Slice

In this program, we make a slice, `slice1`, of length **4** at **line 5** using the `make` function. Then, in the next four lines, from **line 6** to **line 9**, we initialize the elements with some values. At **line 10**, we have a for loop managed by the `range` function, that will iterate `len(slice1)` times. In each iteration, it prints the values present in `slice1` at each index.

The following snippet presents an example with *strings* here:

```
seasons := []string{"Spring","Summer","Autumn","Winter"}
for ix, season := range seasons {
  fmt.Printf("Season %d is: %s\n", ix, season)
}
var season string
for _, season = range seasons {
  fmt.Printf("%s\n", season)
}
```

The above for loop iterates until `len(seasons)` and prints every season from `seasons` as:

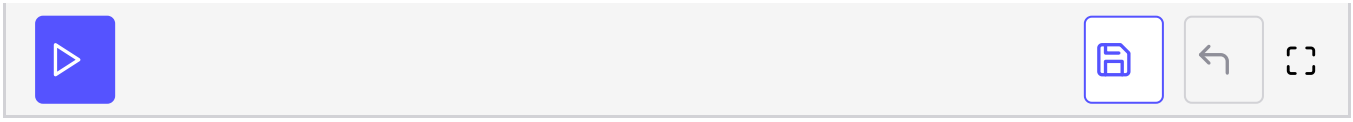
```
Season 0 is: Spring
Season 1 is: Summer
Season 2 is: Autumn
Season 3 is: Winter
```

And the second for loop prints every season from `seasons` by using `_` to discard the index:

```
Spring
Summer
Autumn
Winter
```

Use this version if you want to modify seasons, as shown in the following example:

```
package main
import (
"fmt"
"strings"
)

func main() {
  seasons := []string{"Spring","Summer","Autumn","Winter"}
  for ix, season := range seasons { // printing seasons
    fmt.Printf("Season %d is: %s \n", ix, season)
  }
  for ix := range seasons {
    seasons[ix] = strings.ToUpper(seasons[ix])  // modifying the seasons
  }
  for ix, season := range seasons {
    fmt.Printf("Season %d is: %s \n", ix, season) // printing modified seasons
  }
}
```



Modifying Slices of Strings Using range

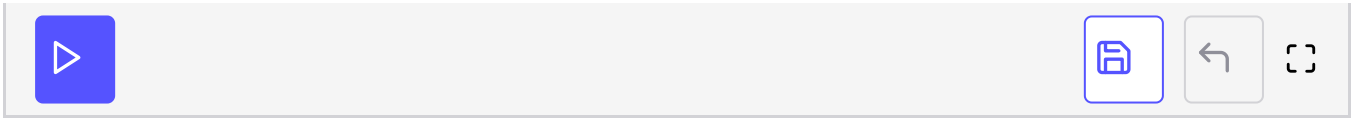
In the code above, we make a slice `seasons` at **line 8** in `main` function as: `seasons := []string{"Spring","Summer","Autumn","Winter"}`. Then, we have a *for* loop at **line 9**, which prints every season from `seasons` along with their index. At **line 12**, we have a *for* loop managed with `range` that will iterate `len(seasons)` time and change the seasons by capitalizing them as: `seasons[ix] = strings.ToUpper(seasons[ix])` at **line 13**. Then, we have a *for* loop at **line 15**, which prints every modified season from `seasons` along with their index.

For `range` with multidimensional slices

It can be convenient to write the nested for-loops as ranges over the rows and row values (columns) of a matrix, like:

```
package main
import "fmt"

func main(){
  value := 0  // used to set values to 2D array
  screen :=  [2][2]int{}  // will contain 4 values

  // asssigning values in 2D-array
  for row := range screen {
```

```
    for column := range screen[row] {
      screen[row][column] = value
      value = value+1

    }
  }

  // printing 2D-array
  for row := range screen {
    for column := range screen[0] {
      fmt.Print(screen[row][column], " ") // separates columns
    }
    fmt.Print("\n") //separates rows
  }
}
```



Range on a 2D array

In the code above, at **line 6**, we make a *2D array* of integers called `screen` with 2 *rows* and 2 *columns*. Right now, the `screen` is empty. To assign values, we need nested-loops. One loop will control the rows and the other one will control the columns. See **line 9**, where we make an outer loop controlling rows of the `screen`. In the next line, there is an inner loop that is controlling the columns. Here, the `for` is ranging over `screen[row]`, which means that this loop will cover all the columns for a `row` (the specific row) of the `screen` in each iteration. At **line 11**, the `value` is assigned to `screen[row][column]` (which is a cell of the specific row and column). Before moving to the next iteration, the `value` is incremented by **1**. Let's break down the loops:

- In the first iteration of the outer loop, the first row will be considered.
    - Inner loop will begin its first iteration and will select the first column of the first row. Value **0** will be placed in the cell.
    - Inner loop will begin its second iteration and will select the second column of the first row. Value **1** will be placed in the cell.
- As all the columns of the first row are filled, the outer loop will begin its second iteration.
    - Inner loop will begin its first iteration and will select the first column of the second row. Value **2** will be placed in the cell.
    - Inner loop will begin its second iteration and will select the second column of the second row. Value **3** will be placed in the cell.

- As all the columns of the second row are filled, the outer loop will get the control. However, all the rows are traversed now, so the control will exit from the outer loop.

Now, you know how to iterate through a slice. In the next lesson, you'll learn how to reslice a slice.