Multidimensionality

This lesson discusses how to handle multidimensional sequences of data in Go.

WE'LL COVER THE FOLLOWING

Multidimensional arrays

Arrays are always *1-dimensional*, but they may be composed to form multidimensional arrays, like:

```
[3][5]int
[2][2][2]float64
```

The inner arrays always have the same length. Go's multidimensional arrays are rectangular. Here is a code snippet which uses such an array:

```
package main

const (
    WIDTH = 1920    // columns of 2D array
    HEIGHT = 1080   // rows of 2D array
)

type pixel int  // aliasing int as pixel
var screen [WIDTH][HEIGHT]pixel     // global 2D array

func main() {
    for y := 0; y < HEIGHT; y++ {
        for x := 0; x < WIDTH; x++ {
            screen[x][y] = 0    // initializing value to 2D array
        }
    }
}
```

In the code above, we set two *constants* `WIDTH` and `HEIGHT` at **line 4** and **line 5**, which determine **columns** and **rows** of a 2D array in our program. Then, at **line 8**, we alias the type `int` as `pixel`, which means every integer in our program is actually a pixel. In the next line, we make a global 2D array `screen` of type `pixel`. In the `main` function, we have *two* nested for loops (because `screen` is 2-D array). The outer loop at **line 12** controls the rows and the inner loop at **line 13** controls the columns. The pixel of each column for every row is assigned **0** value at **line 14**.

That's how we can work with a multidimensional array. For an n-D array, n for loops will be used.

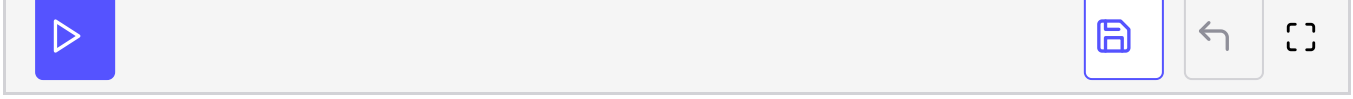
Multidimensional slices

Like arrays, slices are always 1-dimensional but may be composed to construct higher-dimensional objects. With slices of slices (or arrays of slices), the lengths may vary dynamically, so Go's multidimensional slices can be jagged. Moreover, the inner slices must be allocated individually (with make). Here is a simple example of where the inner slices are created literally:

```
package main
import "fmt"

func main() {
  values := [][]int{} // multidimensional slice
  // These are the first two rows.
  row1 := []int{1, 2, 3}
  row2 := []int{4, 5, 6}

  // Append each row to the two-dimensional slice.
  values = append(values, row1)
  values = append(values, row2)
  // Display first row.
  fmt.Println("Row 1")
  fmt.Println(values[0])
  // Display second row.
  fmt.Println("Row 2")
  fmt.Println(values[1])
  // Access an element.
  fmt.Println("First element")
  fmt.Println(values[0][0])
  // Display entire slice.
  fmt.Println("Values")
  fmt.Println(values)
}
```



Multidimensional Slices

In the program above, in `main` at **line 5**, we made a multidimensional slice called `values`. In the next two lines, we made two more slices called `row1` and `row2`, which are rows of the `values`. We append these two rows in `values` at **line 11** and **line 12**. We print the first row: `values[0]` at **line 15**, and the second row: `values[1]` at **line 18**. What if we want to access the first element of the first row? Look at **line 21**. The line `values[0][0]` is used to access the first element present in the first row, which is **1**. Whole 2D slice can also be printed by just typing **values**, as we did in **line 24**.

Now that you know about multidimensional arrays and slices, let's study the concepts of bytes and slices.