Copying and Appending Slices

This lesson describes the method of copying a slice or appending a slice to provide flexibility.

WE'LL COVER THE FOLLOWING

- Modifying slices
  - Use of the `copy` function
  - Use of the `append` function

Modifying slices

To *increase* the capacity of a slice one must create a *new* and *larger* slice and *copy* the contents of the original slice into it.

Use of the `copy` function

Its syntax is as:

```
func copy(dst, src []T) int
```

The function `copy` copies slice elements of type **T** from a source `src` to a destination `dst`, overwriting the corresponding elements in `dst`, and it returns the number of elements copied. Source and destination may overlap. The number of arguments copied is the minimum of `len(src)` and `len(dst)`. When `src` is a *string*, the element type is *byte*.

Use of the `append` function

Its syntax is:

```
func append(s[]T, x ...T) []T
```

The function `append` appends zero or more values to a slice `s` and returns the resulting slice with the same type as `s`. The values, of course, have to be of the

same type as the element-type **T** of `s`. If the capacity of `s` is not large enough to fit the additional values, `append` allocates a new, sufficiently large slice that fits both the existing slice elements and the additional values. Thus, the returned slice may refer to a different underlying array. The `append` always succeeds unless the computer runs out of memory.

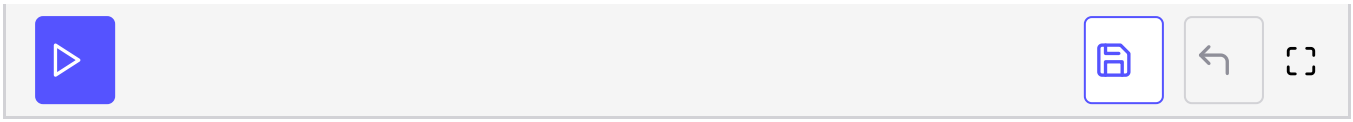
If you want to append a slice `y` to a slice `x`, use the following form to expand the second argument to a list of arguments:

```
x = append(x, y...)
```

The following code illustrates the functions `copy` for copying slices, and `append` for appending new values to a slice.

```
package main
import "fmt"

func main() {
  sl_from := []int{1,2,3}
  sl_to := make([]int,10)
  n := copy(sl_to, sl_from)
  fmt.Println(sl_to) // output: [1 2 3 0 0 0 0 0 0 0]
  fmt.Printf("Copied %d elements\n", n) // n == 3
  sl3 := []int{1,2,3}
  sl3 = append(sl3, 4, 5, 6)
  fmt.Println(sl3) // output: [1 2 3 4 5 6]
}
```



Copy and Appending

In the code above, we make two slices: `sl_from` at **line 5** and `sl_to` at **line 6**. The length of `sl_from` is **3** and the length of `sl_to` is **10** (all zeros). At **line 7**, we copy the contents of `sl_from` to `sl_to` and store total number of elements copied in `n`. Printing `sl_to` at **line 8** gives **[1 2 3 0 0 0 0 0 0 0]**. The variable `n` here is **3** because only three elements (1,2,3) were copied from `sl_from` to `sl_to`. At **line 10**, we make another slice called `sl3` and initialize it to {1,2,3}. **Line 11** shows how to use the `append` function. The elements **4**, **5** and **6** are further appended at the end of slice `sl3`. Now, the modified value of `sl3` is **{1,2,3,4,5,6}**.

The flexibility of slices and especially the append method makes slices

comparable to the ArrayList from Java/C#.

The `copy` and `append` functions make it easy for programmers to copy and append slices when needed without having to write loops. In the next lesson, you have to write a function to solve a problem.