Cheatsheets / **Learn Go: Loops and Arrays**

Learn Go: Arrays

Go Arrays

In Go, an array is a fixed size ordered list of elements with the same data type. Arrays are useful for collecting and accessing multiple related values. Example use cases include:

- `Storing sensor values`
- `Computing averages`
- `Holding lists of information`

Empty Arrays

In Go, an empty array can be created by specifying its number of intended elements and its type.

```
var my_array [3]int
```

Array Creation

In Go, an array can be created with elements in two ways:

1. By specifying the number of elements, the type, and a list of values:

```
animals := [4]string{"Dog", "Hipp
```

1. Or by using the `...` syntax to automatically determine the number of elements, along with the type and a list of values:

```
animals := [...]string{"Dog", "Hi
```

Access Array Values

In Go, values within an array can be accessed using a square bracket `[]` syntax.

```
value := array[index]
```

Modify Array Values

In Go, values can be modified using square brackets `[]`, an index, and the assignment operator `=`.

```
array[index] = value
```

Array Length

In Go, the length of an array or slice can be accessed using the `len` function.

```
length := len(arrayOrSlice)
```

Arrays and Slices

In Go, both arrays and slices are collections of multiple elements of the same data type. However, a slice can be resized to hold additional elements, whereas an array cannot.

Array and Slice Capacity

In Go, an array's capacity is its length, and this cannot change. A slice has both a length and a capacity, where:

- The slice's length is the current number of elements it holds
- The slice's capacity is the number of elements it can hold before needing to resize itself.

Slice Creation

In Go, a slice can be created with or without elements. Without elements, an empty set of square brackets `[]` and the data type is provided:

```
var numberSlice []int
```

With elements, a list of items enclosed in curly braces `{}` is also provided:

```
names := []string{""Kathryn"", ""
```

Array and Slice Length

In Go, the length of an array or slice can be accessed using the `len` function.

```
length := len(sliceOrArray)
```

The capacity of a slice can be accessed using the `cap` function.

```
capacity := cap(slice)
```

Slice Append

In Go, an element can be added to the end of a slice using the `append` function.

```
slice := append(slice, newElement)
```

Read Arrays

In Go, arrays or slices can be passed into functions as parameters. To pass an array parameter into a function, provide a local name, square brackets, and the data type. The difference between slice and array parameters is whether the number of elements is stated.

```
func printFirstLastArray(array [4]int) {
    fmt.Println("""First""", array[0])
    fmt.Println("""Last""", array[3])
}

func printFirstLastSlice(slice []int) {
    length := len(slice)
    if (length > 0) {
        fmt.Println("""First""",
slice[0])
        fmt.Println("""Last""",
slice[length-1])
    }
}
```

Write to Arrays

In Go, arrays are passed into functions by value, which means that changes remain local to the function. Slices, which are pointers, are passed by reference so changes affect the original variable. To modify an array within a function, it must be converted into a slice.

```
func changeFirst(slice []int, value int)
{
    if (len(slice) > 0) {
        slice[0] = value
    }
}
```

Print Share