Slices

Slices

Arrays:

An array type definition specifies a length and an element type.

[4]int represents an array of four integers.

Slices

Arrays -

- An array's size is fixed.
- Length is part of its type ([4]int and [5]int are distinct, **incompatible types**)
- indexed in the usual way, s[n] accesses the nth element.
- The in-memory representation of [4]int is just four integer values laid out sequentially.

```
b := [2]string{"Penn", "Teller"}

b := [...]string{"Penn", "Teller"}
```

Slices

- build on arrays.
- type specification for a slice is []T, where T is the type of the elements of the slice.
- slice type has no specified length.

Slices - declare/create

- A slice literal is declared just like an array literal, except you leave out the element count.
- A slice can be created with the built-in function called make, which has the signature, **func make([]T, len, cap) []T**

```
var s []byte
s = make([]byte, 5, 5)
/* same as */
s := []byte{0, 0, 0, 0, 0}
```

- When the capacity argument is omitted, it defaults to the specified length.

```
s := make([]byte, 5)
```

Slices

- The length and capacity of a slice can be inspected using the built-in len and cap functions.
- The zero value of a slice is nil. The len and cap functions will both return 0 for a nil slice.

```
len(s) == 5
cap(s) == 5
```

Slices

- Slicing is done by specifying a half-open range with two indices separated by a colon.

```
b := []byte{'g', 'o', 'l', 'a', 'n', 'g'}
Following shares the same storage as b
b[1:4] == []byte{'o', 'l', 'a'}, sharing the same storage as b
b[:2]  == []byte{'g', 'o'}
b[2:]  == []byte{'l', 'a', 'n', 'g'}
b[:]   == b
```

- A slice can also be formed by "slicing" an existing slice or array.

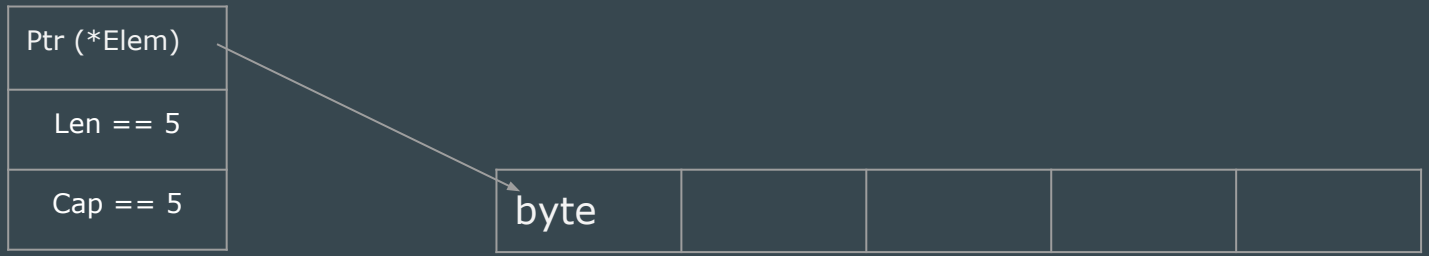
```
x := [3]string{"golang", “Scala", "java"}
s := x[:] // a slice referencing the storage of x
```

Slices - internals

A slice is a descriptor of an array segment. It consists of a pointer to the array, the length of the segment, and its capacity (the maximum length of the segment).

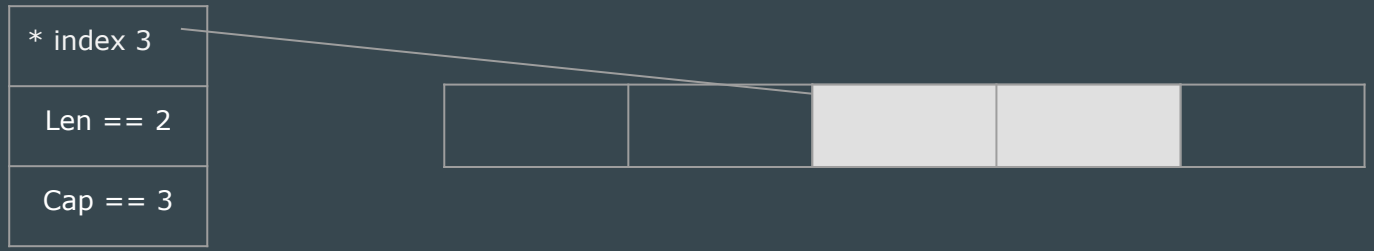
```
type sliceHeader struct {
      Length        int
      Capacity      int
      ZerothElement *byte
}
```

For `s := make([]byte, 5)`



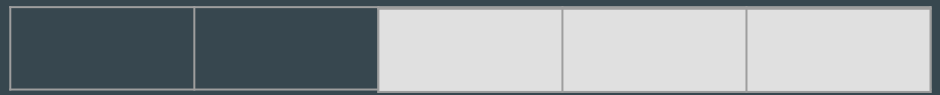
Slices - shrinking and growing

```
s = s[2:4]
```



We can grow s to its capacity by slicing it again:

```
s = s[:cap(s)]
```



Slices - growing beyond the capacity

- A slice cannot be grown beyond its capacity.
- To increase the capacity of a slice one must create a new, larger slice and copy the contents of the original slice into it.

```
t := make([]byte, len(s), (cap(s)+1)*2)
copy(t, s) /* built-in copy function, func copy(dst, src []T) int */
s = t
```

- The append function appends the elements x to the end of the slice s, and grows the slice if a greater capacity is needed.

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

Slices - Tricks

Slice Tricks
Go Slices: usage and internals