Date No

Slices

A slice is convenient, flexible and powerful wrapper on top of array. Slices do not own any data on their own. They are just references to existing array.

creating a slice.

```
func main() {
    a := [5] int { 76, 77, 78, 79, 80 }
    var b [] int = a [1:4]
              // create a slice from a[1] to a[3]
    fmt.Println (b)
}  ->[77  78  79]
```

no-1

The syntax a [start : end] create a slice from array a starting from index start to index end-1

In the above program a [1:4] creates a slice representation of the array a starting from index 1 through 3.

Another way to create slice

```
func main() {
   c := []int { 3,4,5 }  // create an array and
   fmt.Print ( c)           returns a slice reference.
}
```

Modifying a Slice.

A Slice does not own any data of its own. It is just a representation of the underlying array. Any modifications done to the slice will be reflected in the underlying array.

affect

```
func main() {
  darr := [...]int {57, 89, 90, 82, 100, 78, 67, 69, 59}
  dslice := darr [2:5] (5-1)     2,3,4
  fmt.Println ("array befory" , darr)
  for i := range dslice {
       dslice[i] ++
  }
  fmt.Println("array after", darr)
                          [2]  [3]   [4]
-> array before [ 57  89  90  82  100  78  67  69  59 ]
   array after  [ 57  89  91  83  101  78  67  69  59 ]
```

When a number of slices shares the same underlying array, the changes that each one makes will be reflected in the array.

```
func main() {
  numa := [3]int {78, 79, 80}
  num1 := numa [:] // Slice contains all the elements
  num2 := numa [:]
  fmt.Println ("array before chang 1", numa)
  num1 [0] = 100
  fmt.Println ("array after modification to slice num1",
```