Go Training

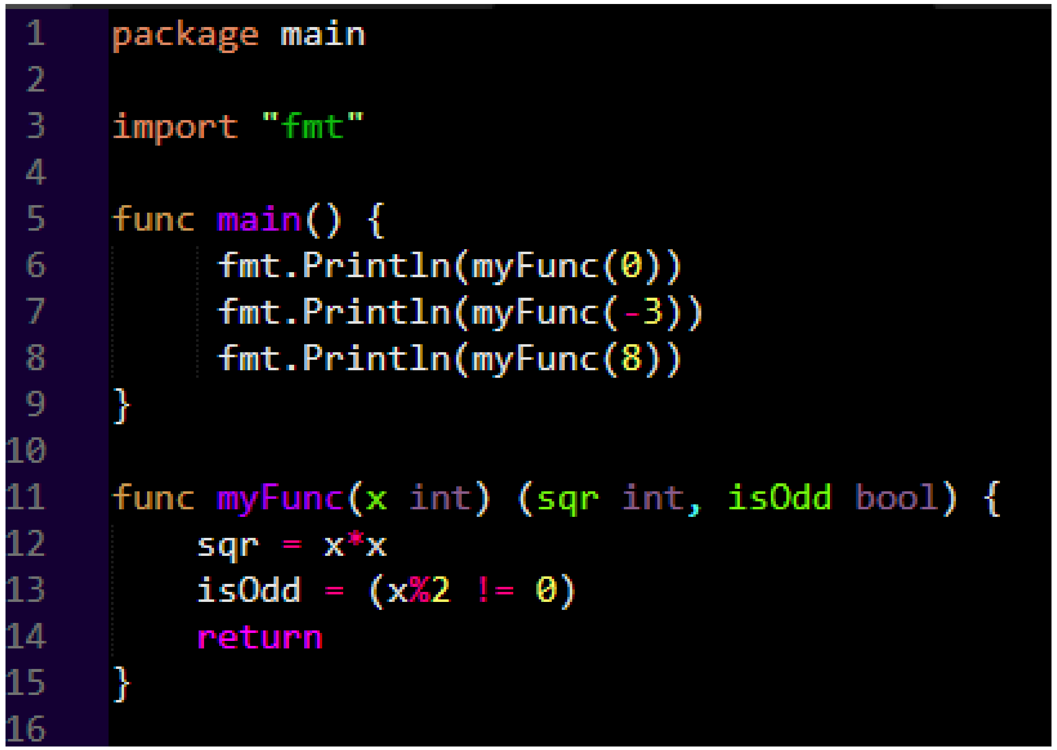
Session 3

Assignment Solution

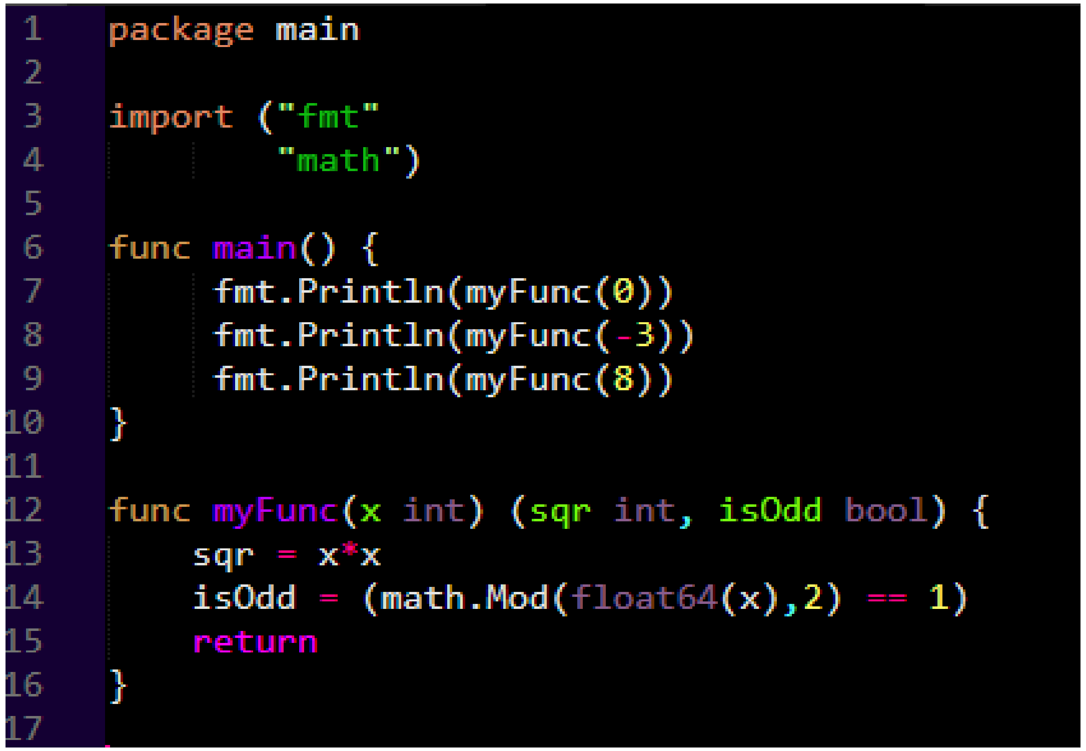
Write a go program which has the function definition ‘myFunc’.

myFunc will take an integer as an argument and will return its square value and a Boolean ‘yes’ if the given integer is odd else a Boolean ‘no’ .

Call this myFunc three times in your program with arguments 0, -3 and 8 at a time.



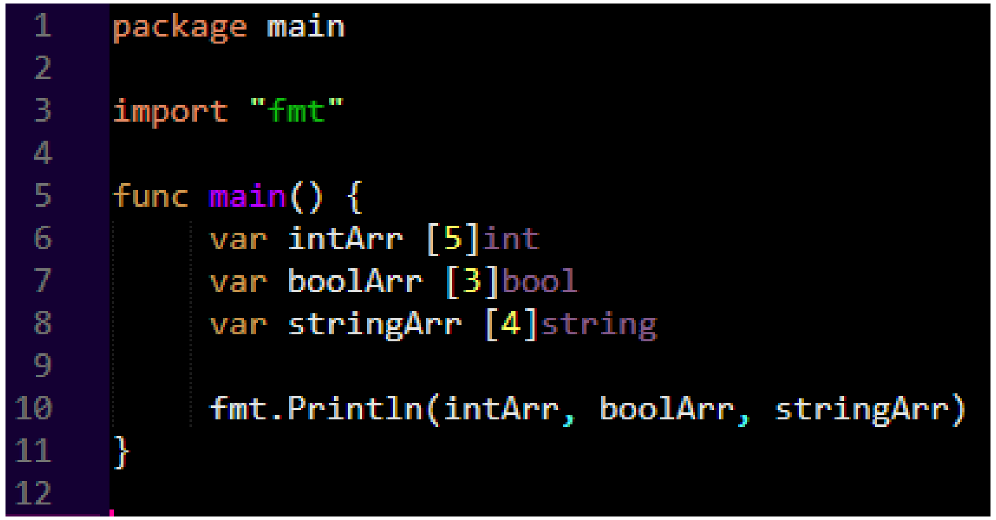
```go
package main

import "fmt"

func main() {
	fmt.Println(myFunc(0))
	fmt.Println(myFunc(-3))
	fmt.Println(myFunc(8))
}

func myFunc(x int) (sqr int, isOdd bool) {
    sqr = x*x
    isOdd = (x%2 != 0)
    return
}
```



```go
package main

import ("fmt"
		"math")

func main() {
	fmt.Println(myFunc(0))
	fmt.Println(myFunc(-3))
	fmt.Println(myFunc(8))
}

func myFunc(x int) (sqr int, isOdd bool) {
    sqr = x*x
    isOdd = (math.Mod(float64(x),2) == 1)
    return
}
```

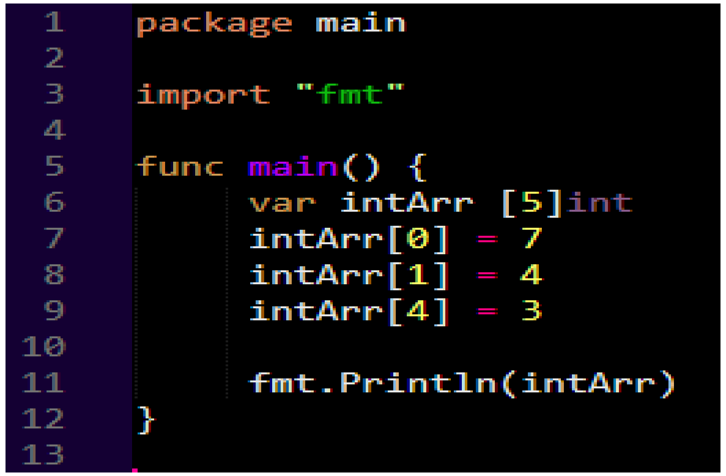
Array

- A container which holds the values of the same type
- Once defined with a size, the size of an array cannot be increased or decreased
- How to declare an array?



```go
package main

import "fmt"

func main() {
	var intArr [5]int
	var boolArr [3]bool
	var stringArr [4]string

	fmt.Println(intArr, boolArr, stringArr)
}

```

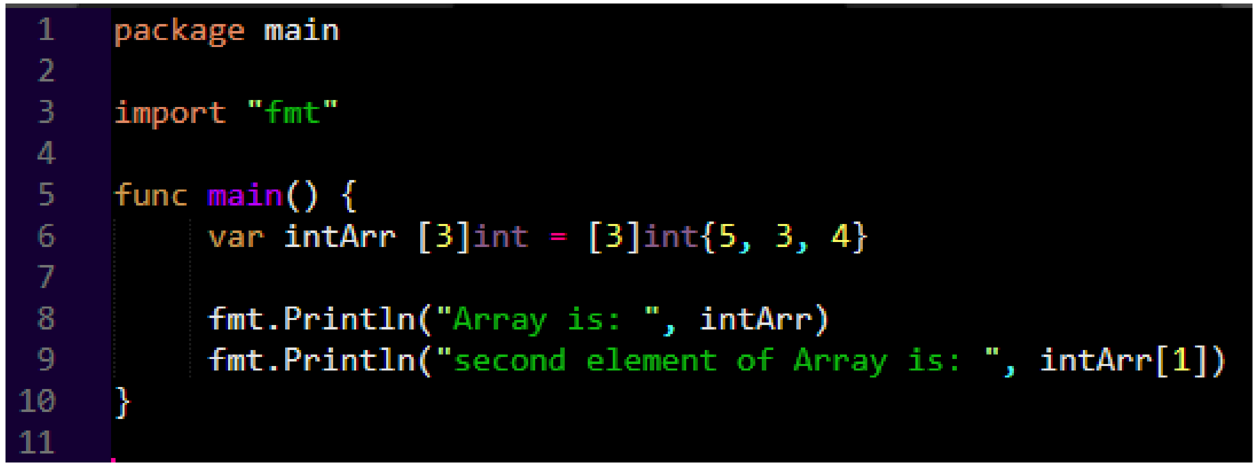
- Assigning values to an array



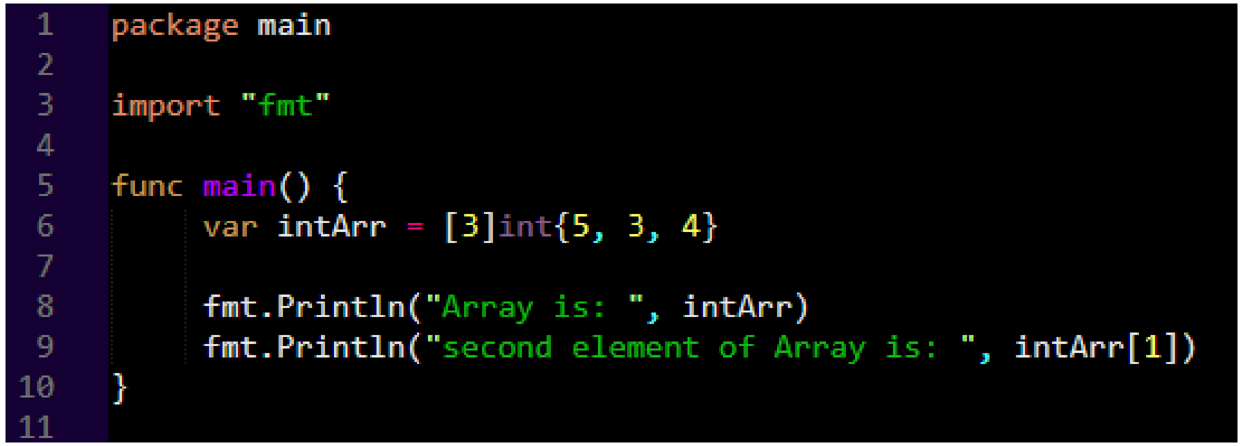
```go
package main

import "fmt"

func main() {
	var intArr [5]int
	intArr[0] = 7
	intArr[1] = 4
	intArr[4] = 3

	fmt.Println(intArr)
}

```

Array

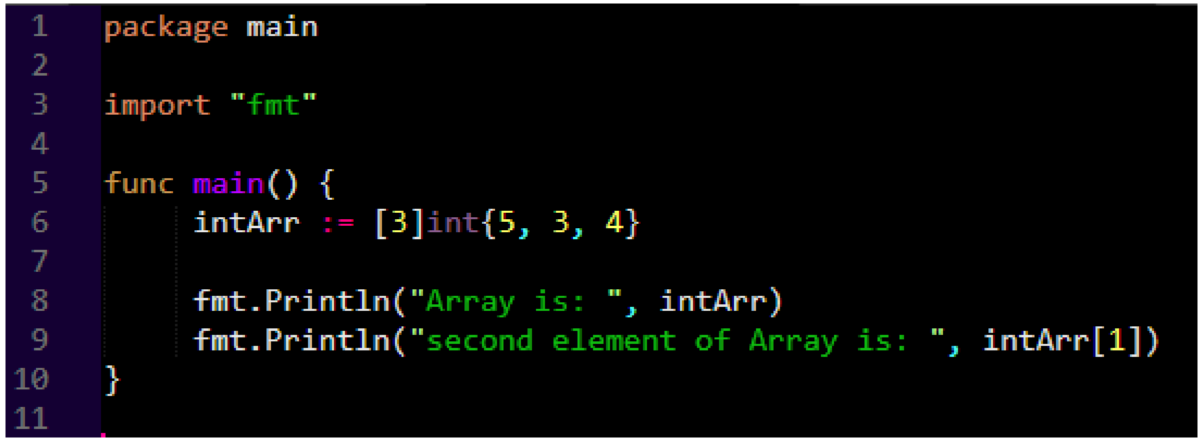
- Declaration and assignment



```go
package main

import "fmt"

func main() {
	var intArr [3]int = [3]int{5, 3, 4}

	fmt.Println("Array is: ", intArr)
	fmt.Println("second element of Array is: ", intArr[1])
}

```



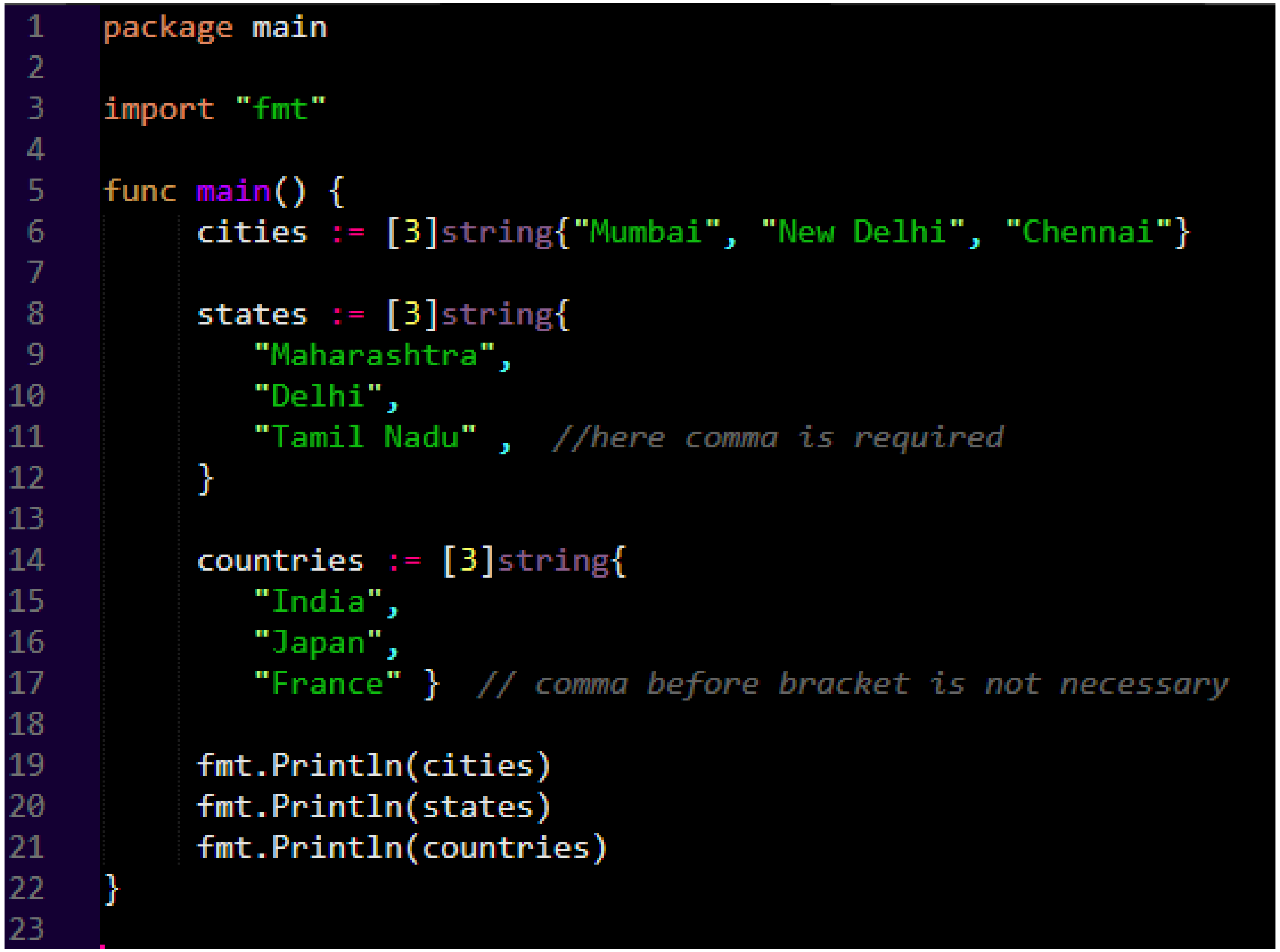
```go
package main

import "fmt"

func main() {
	var intArr = [3]int{5, 3, 4}

	fmt.Println("Array is: ", intArr)
	fmt.Println("second element of Array is: ", intArr[1])
}

```



```go
package main

import "fmt"

func main() {
	intArr := [3]int{5, 3, 4}

	fmt.Println("Array is: ", intArr)
	fmt.Println("second element of Array is: ", intArr[1])
}

```

Array

- Multiline initialization

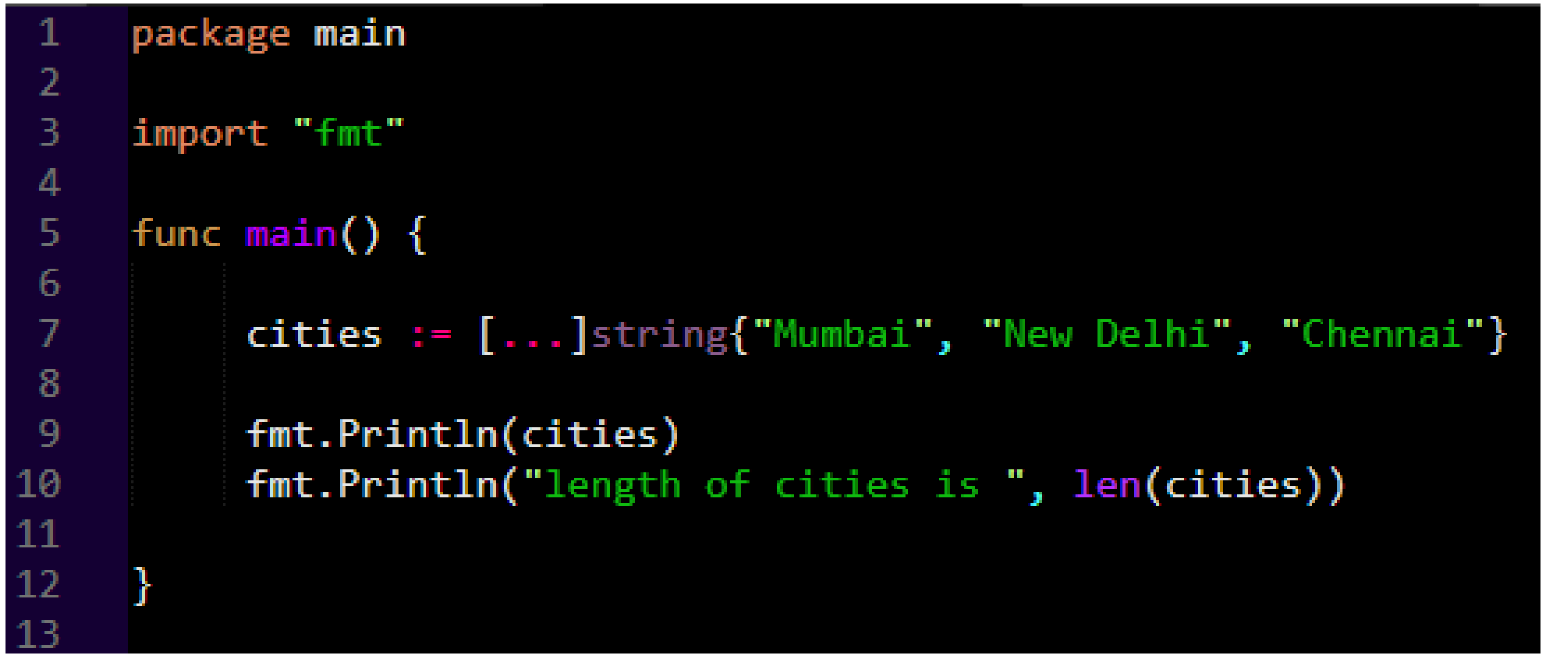


```go
package main

import "fmt"

func main() {
	cities := [3]string{"Mumbai", "New Delhi", "Chennai"}

	states := [3]string{
	   "Maharashtra",
	   "Delhi",
	   "Tamil Nadu" ,  //here comma is required
	}

	countries := [3]string{
	   "India",
	   "Japan",
	   "France" }  // comma before bracket is not necessary

	fmt.Println(cities)
	fmt.Println(states)
	fmt.Println(countries)
}

```

Array

- Not sure about length?

- Finding length of an array

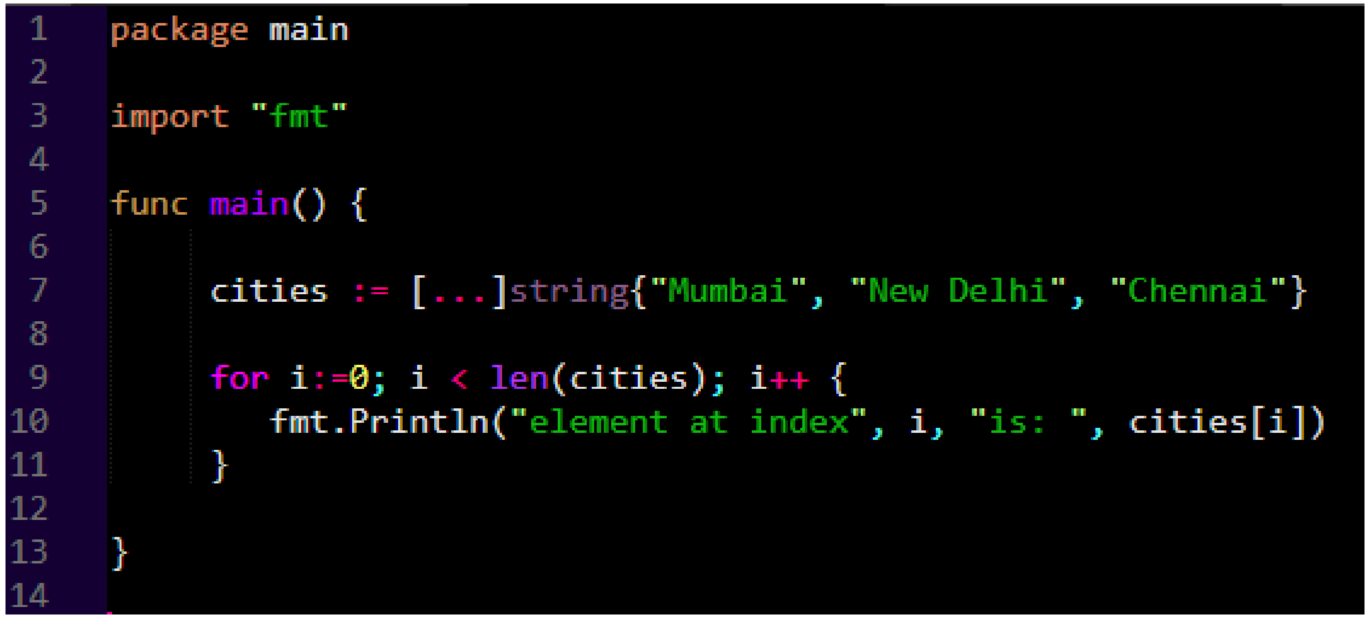


```go
package main

import "fmt"

func main() {

	cities := [...]string{"Mumbai", "New Delhi", "Chennai"}

	fmt.Println(cities)
	fmt.Println("length of cities is ", len(cities))

}

```

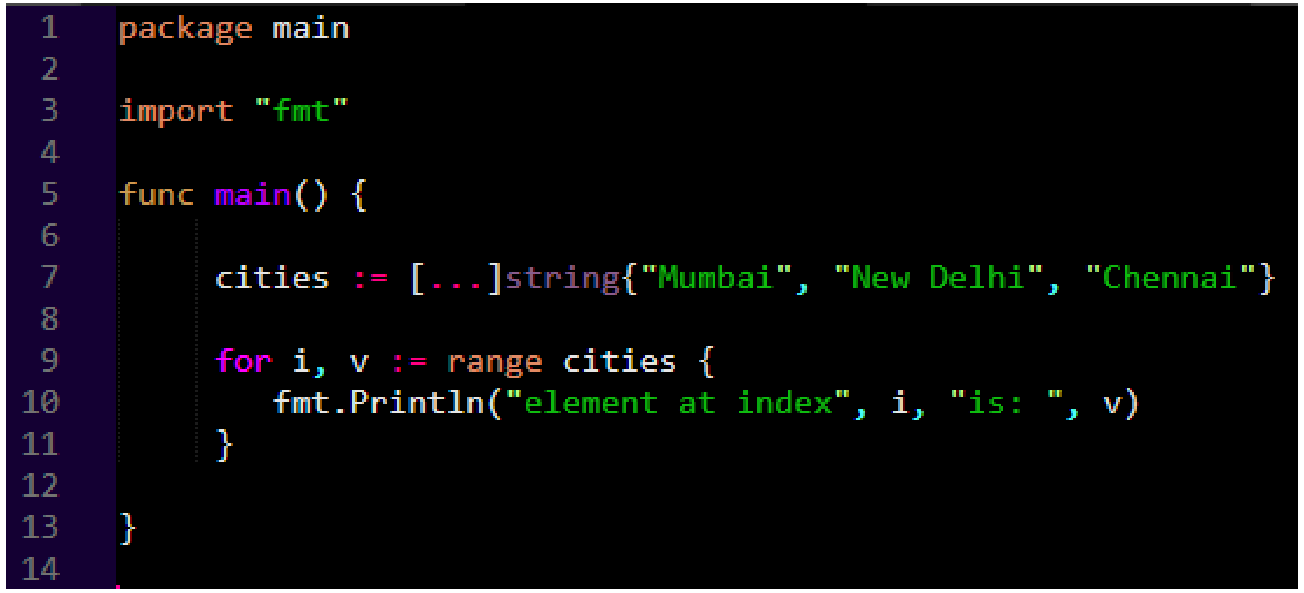
- Array A == Array B
  - len(A) == len(B)
  - Type of A == Type of B
  - Elements in A same as those in B
  - Same order of elements in A and B

Array

- Iteration over array elements



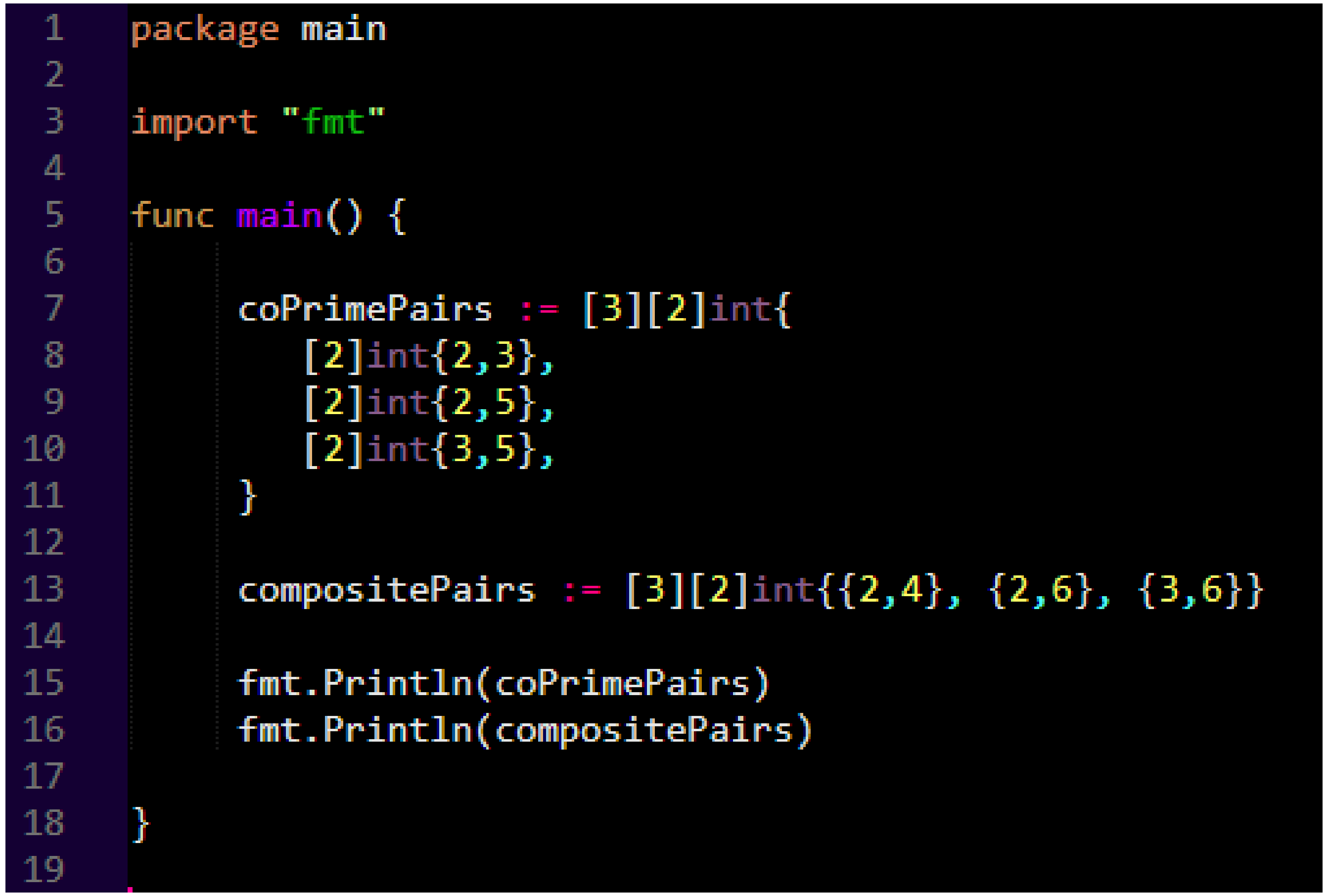
```go
package main

import "fmt"

func main() {

	cities := [...]string{"Mumbai", "New Delhi", "Chennai"}

	for i:=0; i < len(cities); i++ {
	   fmt.Println("element at index", i, "is: ", cities[i])
	}

}
```



```go
package main

import "fmt"

func main() {

	cities := [...]string{"Mumbai", "New Delhi", "Chennai"}

	for i, v := range cities {
	   fmt.Println("element at index", i, "is: ", v)
	}

}
```

Array

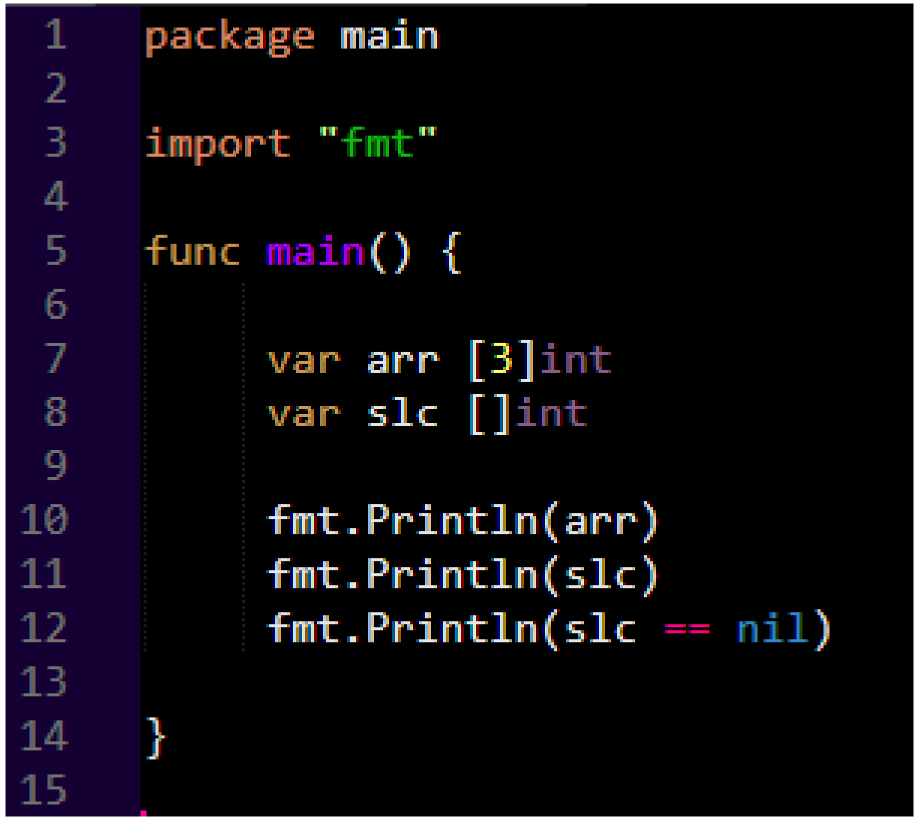
- Multi-dimensional array



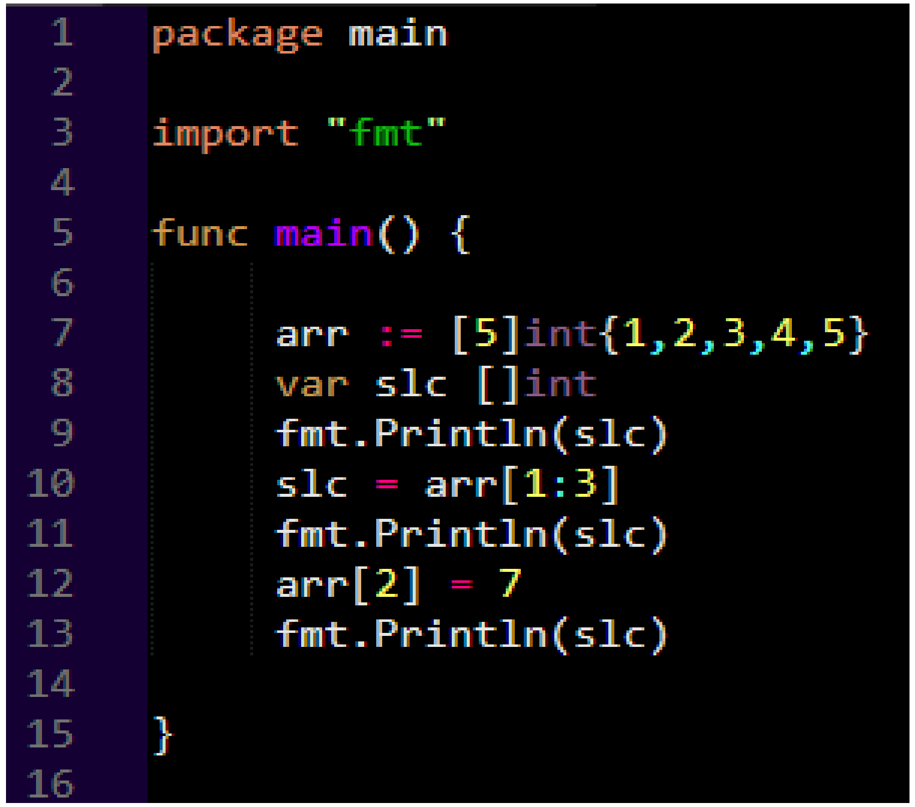
```go
package main

import "fmt"

func main() {

	coPrimePairs := [3][2]int{
		[2]int{2,3},
		[2]int{2,5},
		[2]int{3,5},
	}

	compositePairs := [3][2]int{{2,4}, {2,6}, {3,6}}

	fmt.Println(coPrimePairs)
	fmt.Println(compositePairs)

}
```

Slices

- A container to hold elements of same data type
- Size of a given slice can vary
- A slice is just a reference to an array



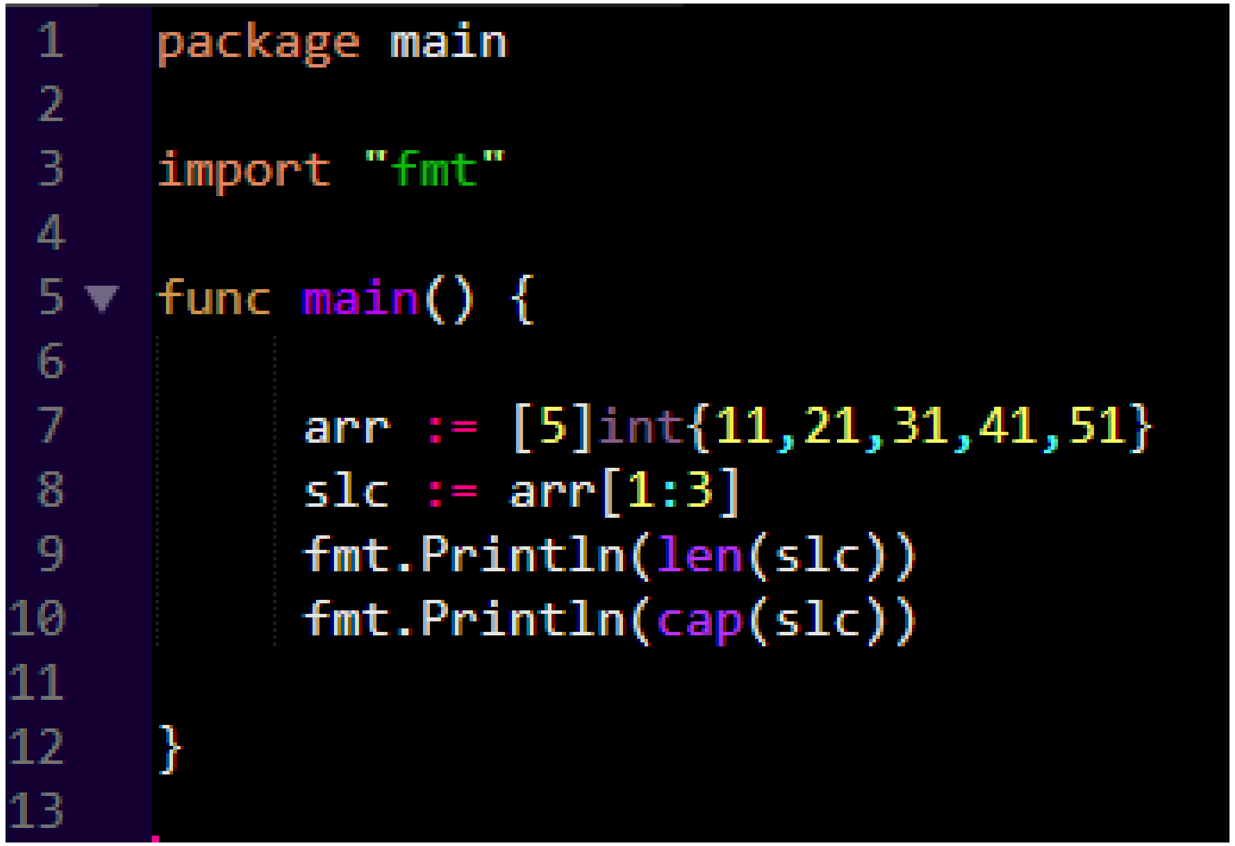
```go
package main

import "fmt"

func main() {

	var arr [3]int
	var slc []int

	fmt.Println(arr)
	fmt.Println(slc)
	fmt.Println(slc == nil)

}

```



```go
package main

import "fmt"

func main() {

	arr := [5]int{1,2,3,4,5}
	var slc []int
	fmt.Println(slc)
	slc = arr[1:3]
	fmt.Println(slc)
	arr[2] = 7
	fmt.Println(slc)

}

```

Slices

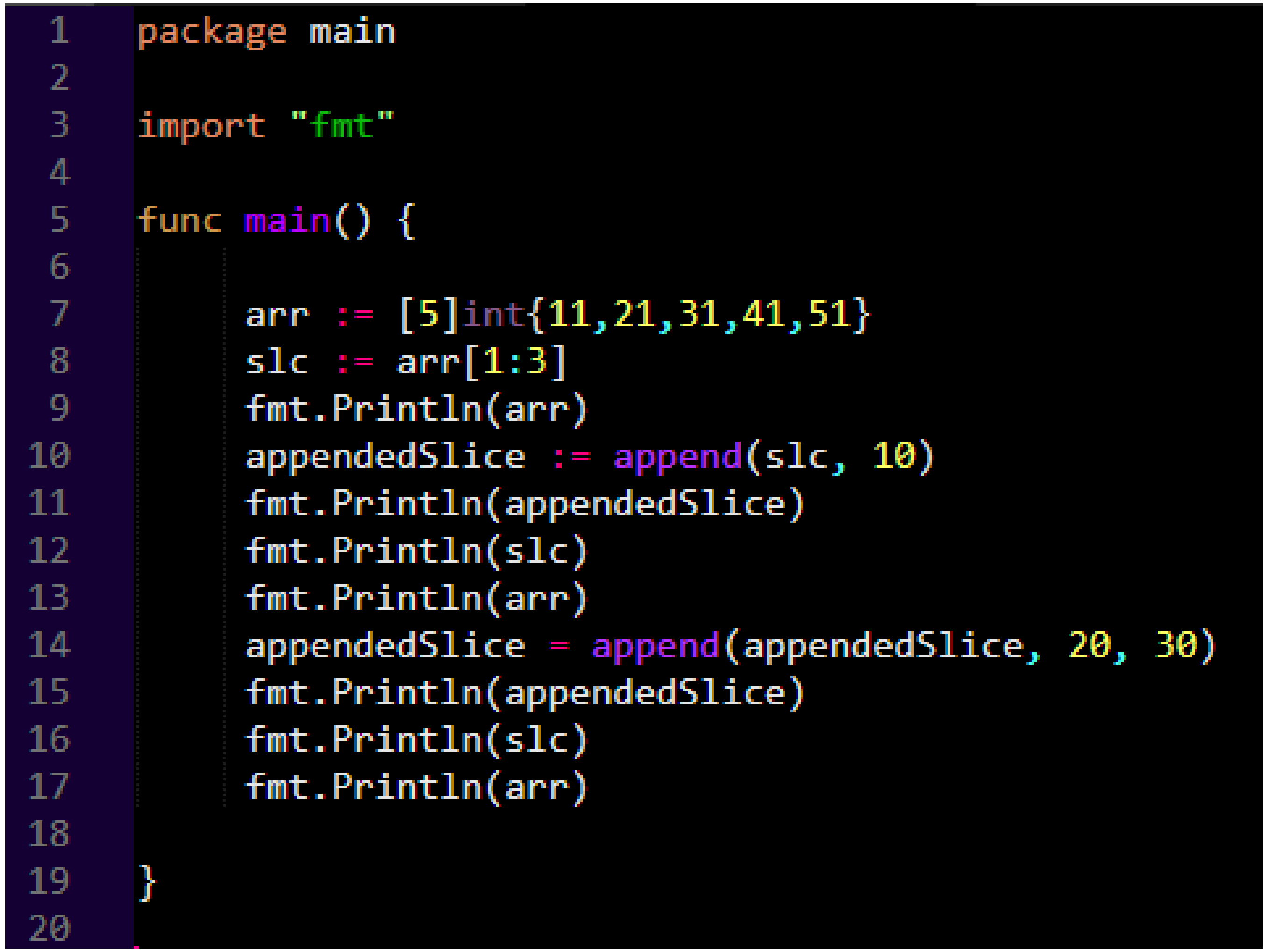
- Length and capacity



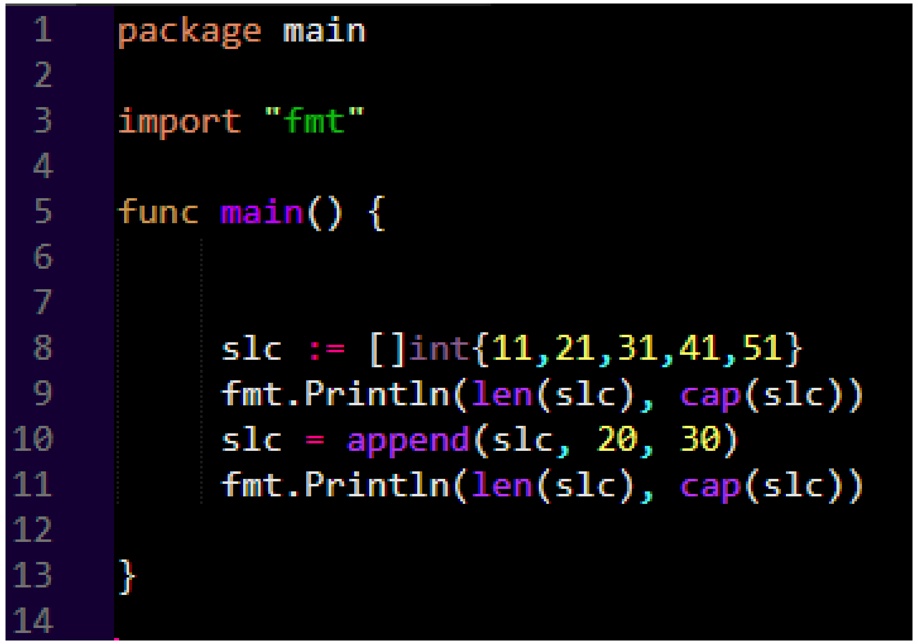
```go
package main

import "fmt"

func main() {

	arr := [5]int{11,21,31,41,51}
	slc := arr[1:3]
	fmt.Println(len(slc))
	fmt.Println(cap(slc))

}
```

Slices

- Append
  - Takes a slice as the first argument and one or more elements to append as further arguments



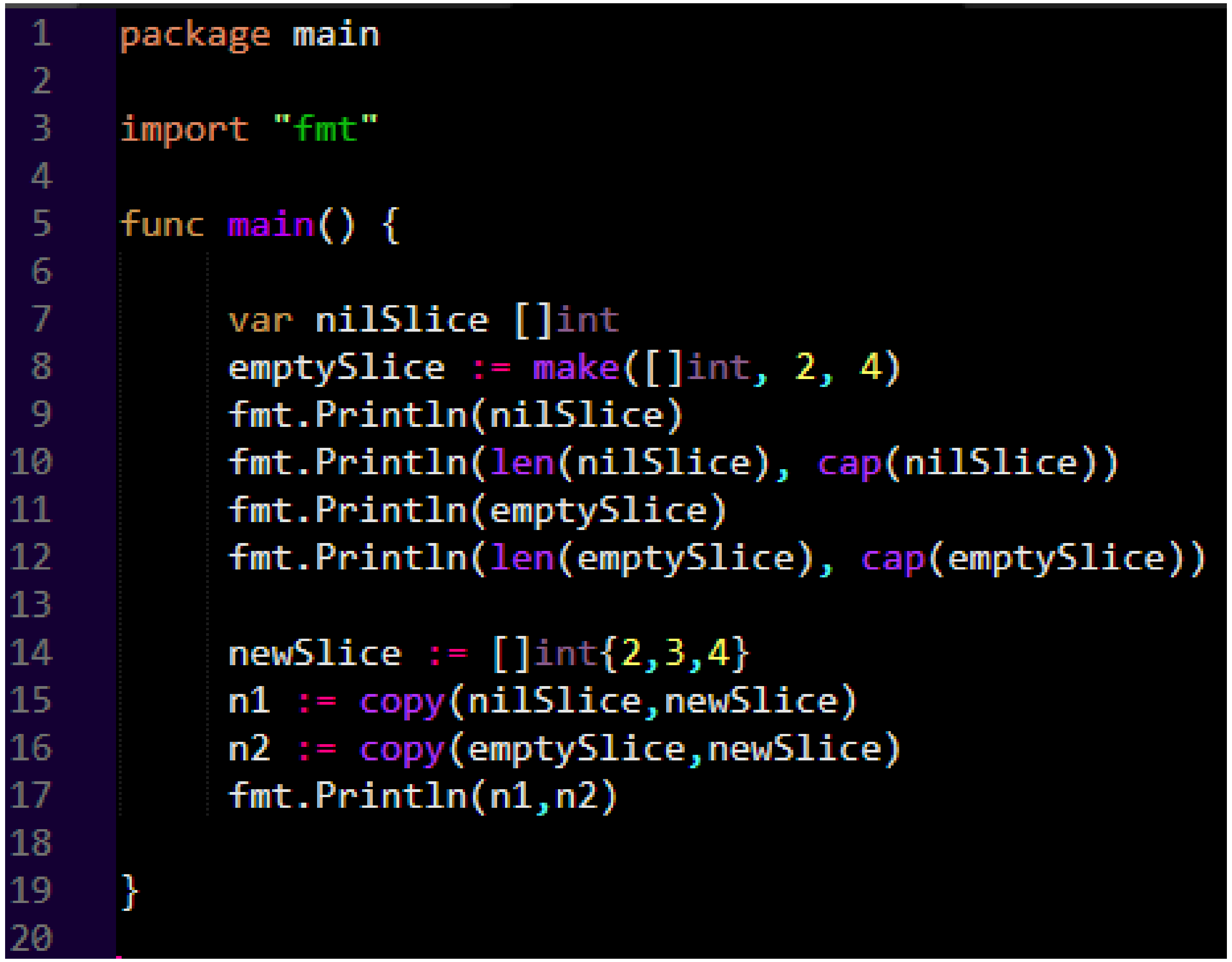
```go
package main

import "fmt"

func main() {

	arr := [5]int{11,21,31,41,51}
	slc := arr[1:3]
	fmt.Println(arr)
	appendedSlice := append(slc, 10)
	fmt.Println(appendedSlice)
	fmt.Println(slc)
	fmt.Println(arr)
	appendedSlice = append(appendedSlice, 20, 30)
	fmt.Println(appendedSlice)
	fmt.Println(slc)
	fmt.Println(arr)

}
```



```go
package main

import "fmt"

func main() {


	slc := []int{11,21,31,41,51}
	fmt.Println(len(slc), cap(slc))
	slc = append(slc, 20, 30)
	fmt.Println(len(slc), cap(slc))

}
```

Slices

- Make function
- ‘Nil slice’ and ‘Empty Slice’
- copy



```go
package main

import "fmt"

func main() {

	var nilSlice []int
	emptySlice := make([]int, 2, 4)
	fmt.Println(nilSlice)
	fmt.Println(len(nilSlice), cap(nilSlice))
	fmt.Println(emptySlice)
	fmt.Println(len(emptySlice), cap(emptySlice))

	newSlice := []int{2,3,4}
	n1 := copy(nilSlice,newSlice)
	n2 := copy(emptySlice,newSlice)
	fmt.Println(n1,n2)

}
```

Slices

- Spread operator
- Extract operator
- Iteration
- Deleting an element of a slice
- Pass by value or pass by reference?

Thank You