# Structs, Interfaces and Maps

This lesson is a flashback to the standard operations and their syntaxes defined on structs, interfaces, and maps.

WE'LL COVER THE FOLLOWING



## 📝 Useful code snippets for structs

### Creation

```
type struct1 struct {
  field1 type1
  field2 type2
  ...
}

ms := new(struct1)
```

### Initialization

```
ms := &struct1{10, 15.5, "Chris"}
```

```
ms := &struct1{10, 15.5, "Chris"}
```

Capitalize the first letter of the struct name to make it visible outside its package. Often, it is better to define a factory function for the struct and force using that.

```
ms := Newstruct1{10, 15.5, "Chris"}

func Newstruct1(n int, f float32, name string) *struct1 {
  return &struct1{n, f, name}
}
```

# 📝 Useful code snippets for interfaces

## Testing if a value implements an interface

```
if v, ok := v.(Stringer); ok { // test if v implements Stringer
  fmt.Printf("implements String(): %s\n", v.String());
}
```

## A type classifier

```
func classifier(items ...interface{}) {
  for i, x := range items {
    switch x.(type) {
    case bool: fmt.Printf("param #%d is a bool\n", i)
    case float64: fmt.Printf("param #%d is a float64\n", i)
    case int, int64: fmt.Printf("param #%d is an int\n", i)
    case nil: fmt.Printf("param #%d is nil\n", i)
    case string: fmt.Printf("param #%d is a string\n", i)

    default: fmt.Printf("param #%d's type is unknown\n", i)
    }
  }
}
```

# 📝 Useful code snippets for maps

## Creation

```
map1 := make(map[keytype]valuetype)
```

## Initialization

## Initialization

```
map1 := map[string]int{"one": 1, "two": 2}
```

## Looping over a map with `for` or `for-range`

```
for key, value := range map1 {
  ...
}
```

## Testing if a key value exists in a map

```
val1, isPresent = map1[key1]
```

This gives a value or zero-value for `val1`, *true* or *false* for `isPresent`.

## Deleting a key in a map

```
delete(map1, key1)
```

---

This pretty much summarizes structs, interfaces, and maps. The next lesson deals with functions and files.