MODULE PRACTICE

Arrays of Structs

```
points := []Point{{1, 1}, {7, 27}, {12, 7}, {9, 25}}
```

In Go, arrays can be used to store many of the same struct's instances.

Struct Instances

```
p1 := Point{x: 10, y: 12}
```

In Go, an instance of a defined struct can be created by providing its name followed by a set of curly braces with optional values.

Nested Structs

```
type Name struct{
        firstName string
        lastName string
}

type Employee struct{
        name Name
        age int
        title string
}
```

In Go, a struct can contain fields that are themselves other structs.

Access Struct Fields

```
p1 := Point{x:10, y:12}
fmt.Println(p1.x)
```

In Go, fields within a struct can be accessed or modified using the `.` operator.

Access Pointer Struct Fields

```
steve := Employee{“Steve”, “Stevens”, 34, “Junior Manager”}
pointerToSteve := &steve
fmt.Println(pointerToSteve.firstName)
```

In Go, accessing the fields of a pointer to a struct does not require dereferencing. The fields of the struct pointer can be accessed using the normal `.` syntax.

Struct Definition

```
type Point struct{
        x int
        y int
}
```

In Go, a struct must be defined before it can be used in a program. The definition of a struct includes its name and its fields.

Structs and Fields

In Go, a group of related variables can be defined as a struct. Each variable within a struct is known as a field.

Passing Structs as Pointers

```
func (rectangle *Rectangle) modify(newLength float32){
        rectangle.length = newLength
}
```

In Go, the values of a struct can only be modified in a function if the struct is passed as a pointer.