Pointers in Go

& => ampersand => address operator

- stores memory address

```
answer := 42
fmt.Println(&answer) //0x1040c108
```

You can use 'answer' to obtain the value

* => asterisk => dereference operator
- provides the value that a memory address refers to

```
answer := 42
fmt.Println(&answer) -> 0x1040c108

address := &answer
fmt.Println(*address) -> 42
```

Type aliases

byte → uint8

rune → int32

Functions

- Functions, variables, & other identifiers that begin with an uppercase letter are exported & become available to other packages

Parameter vs argument

func Multiply (number1 int64 , number2 int64) int64

parameters

Multiply(7,3)

arguments

Variadic function → ellipsis (...) → variable # of args

ex. func Println (a ...interface{}) (n int, err error)

Declaring new types

```
type celsius float64
var temperature celsius = 20
fmt.Println(temperature)
```

Celsius is a unique type, not a type alias.
Trying to use a celsius w/ a float64 will result in a mismatched types error

Methods

- Used to operate on types
- Have something like a parameter called a receiver

Functions in Go are first class

Function types

```
type sensor func() kelvin
func measureTemperature(samples int, s func() kelvin)
func measureTemperature(samples int, s sensor)
```

Anonymous fn aka fn literal is a *closure*
- keeps references to variables in the surrounding scope

Go arrays

Composite literal syntax

dwarfs := [5] string {"Ceres", "Pluto", "Haumea", "Makemake", "Eris"}

Arrays are values (and are passed by value)

Slices

- planets[0:4] → First 4 planets in array
- Half-open range - starts at x and continues up to, but does not include y → [x:y]
- You can index into slices
- Modifying an element in a slice alters the underlying array
- Omitting 1st number defaults to beginning of array
- Omitting last number defaults to array length

Composite literal syntax

- dwarfs := [] string {"One", "Two", "Three"}

- Arrays are hardly used directly; use slices instead
- 'sort' package → StringSlice type w/ methods

- String append function is variadic

Three-index slicing

- limits capacity of resulting slice

```
terrestial := planets[0:4:4]
```

- Use 'make' to pre-allocate slices

```
dwarfs := make([]string, 0, 10)
```

Declaring variadic functions

```
func terraform(prefix string, worlds ... string)[]string{
    ⋮
}
```

To pass slice instead of arguments, expand with ...

Maps

```
map[string]int
```

key ↑ value ↑

For key that doesn't exist in map, zero value of value is returned

Maps

ok syntax accounts for not finding value

```
if moon, ok := temperature["Moon"]; ok {
        // do moon stuff
} else {
    //no moon
}
```

- Maps are not copied
- Pre-allocate maps with make

```
temperature := make(map[float]int, 8)
```

Use a map of map[T] bool to make it a set