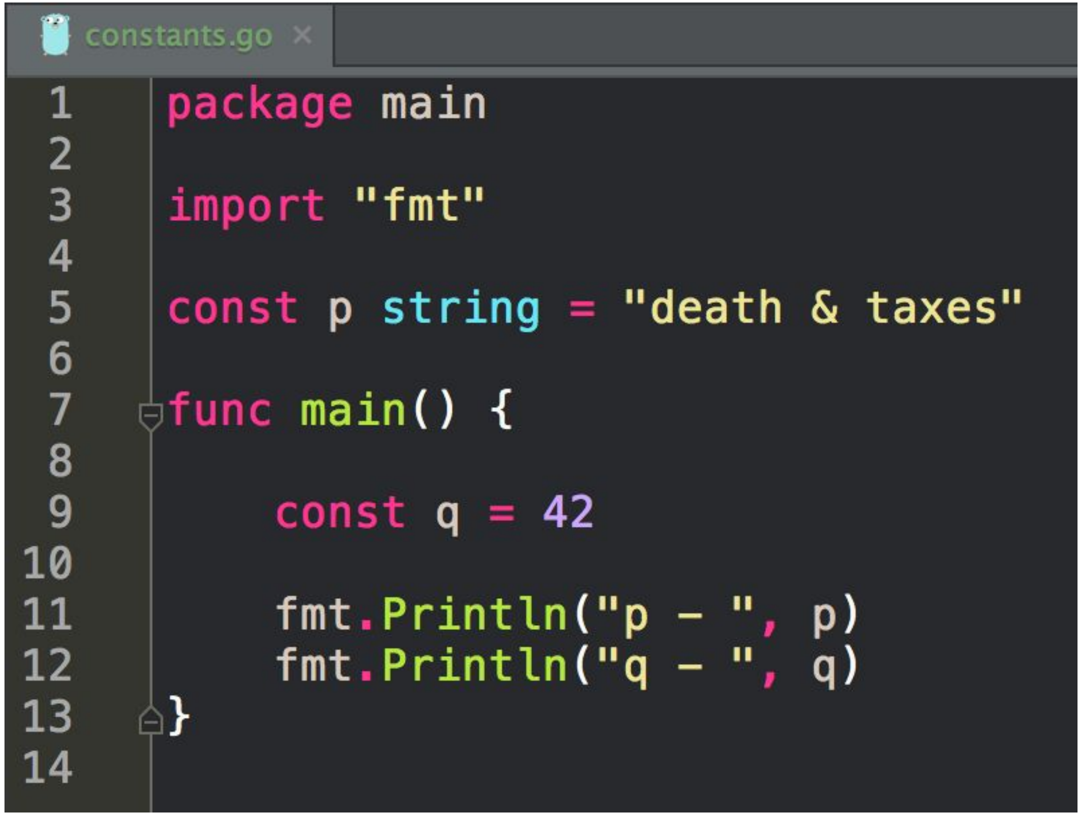
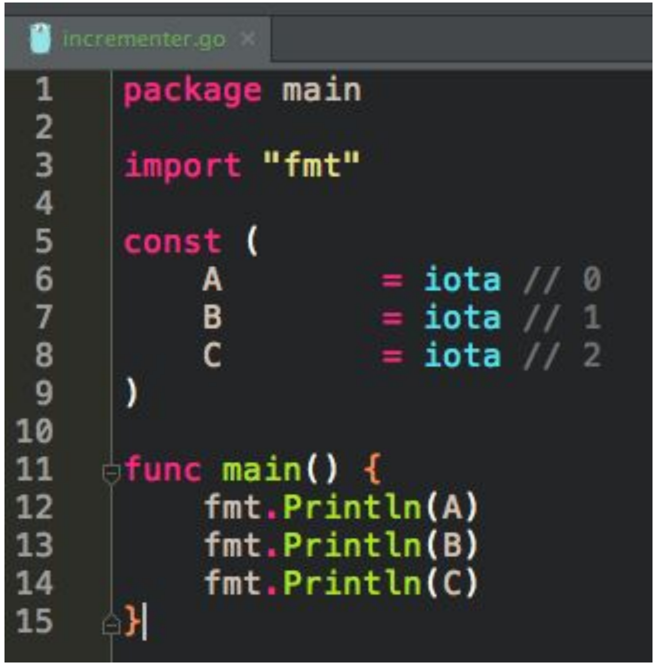
Constants & Pointers

constants, iota, memory addresses, pointers

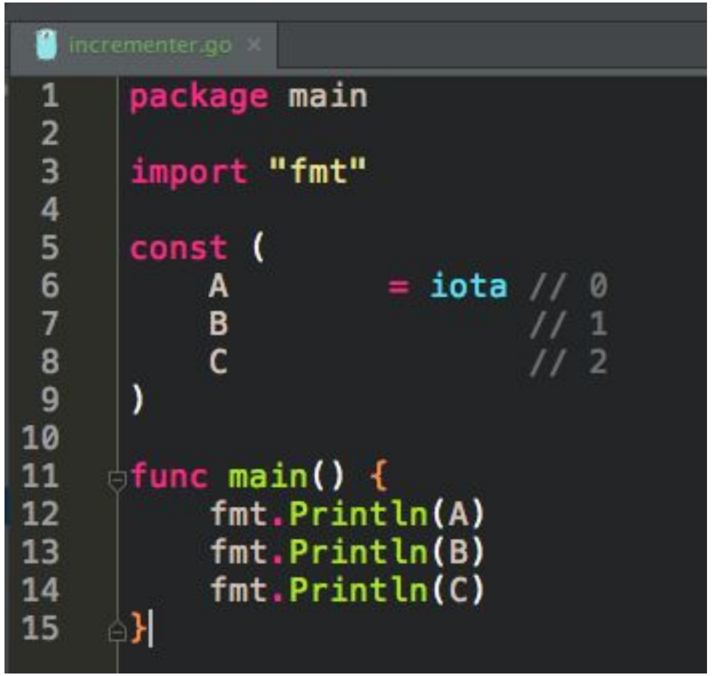
constants



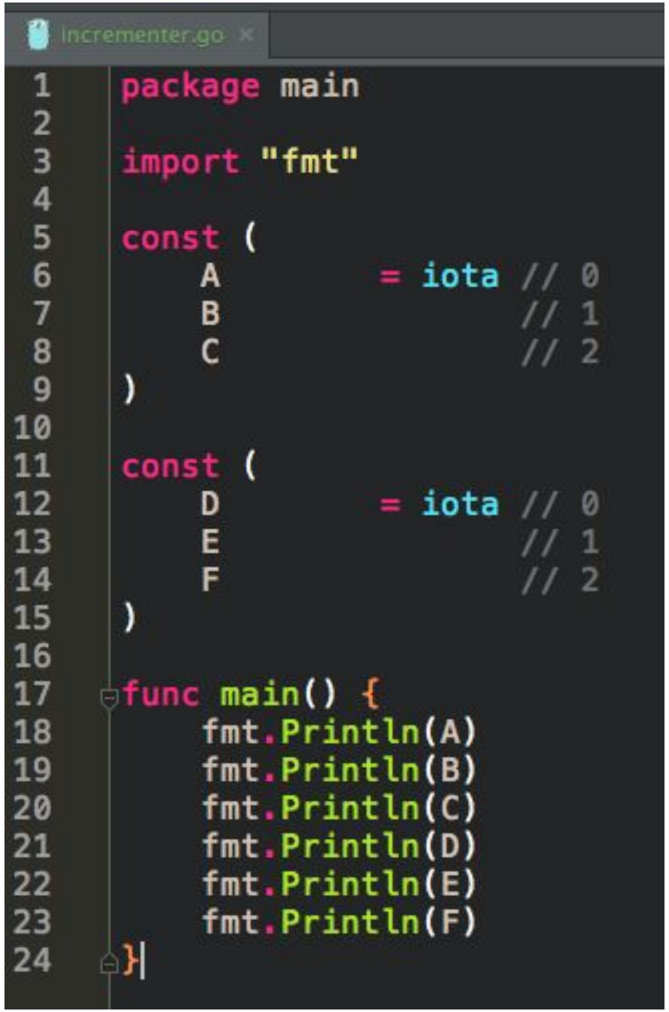
```go
package main

import "fmt"

const p string = "death & taxes"

func main() {

	const q = 42

	fmt.Println("p - ", p)
	fmt.Println("q - ", q)
}

```



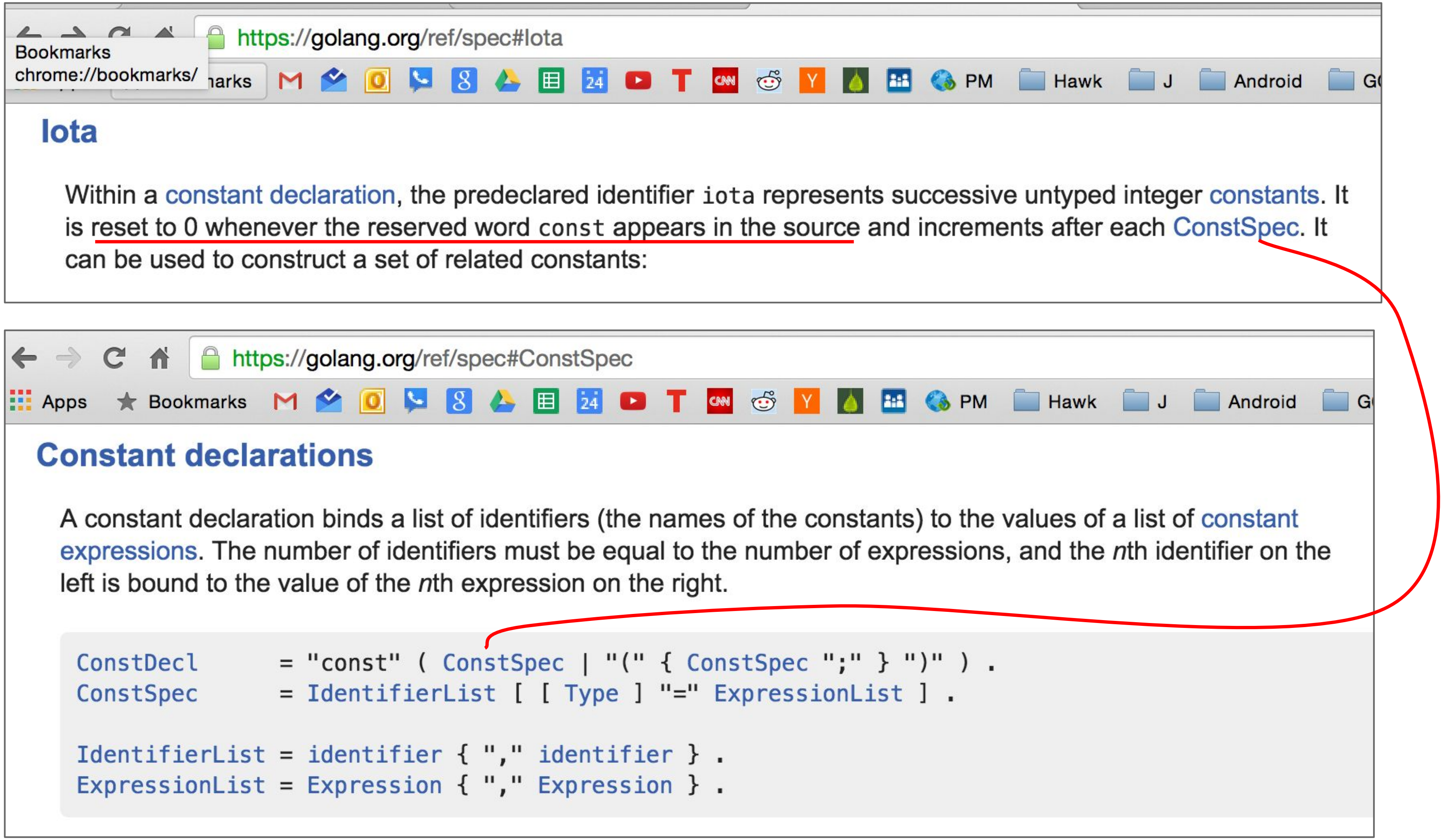
```go
package main

import "fmt"

const (
	A        = iota // 0
	B        = iota // 1
	C        = iota // 2
)

func main() {
	fmt.Println(A)
	fmt.Println(B)
	fmt.Println(C)
}
```



```go
package main

import "fmt"

const (
	A        = iota // 0
	B               // 1
	C               // 2
)

func main() {
	fmt.Println(A)
	fmt.Println(B)
	fmt.Println(C)
}
```



```go
package main

import "fmt"

const (
	A        = iota // 0
	B               // 1
	C               // 2
)

const (
	D        = iota // 0
	E               // 1
	F               // 2
)

func main() {
	fmt.Println(A)
	fmt.Println(B)
	fmt.Println(C)
	fmt.Println(D)
	fmt.Println(E)
	fmt.Println(F)
}
```



https://golang.org/ref/spec#Iota

Iota

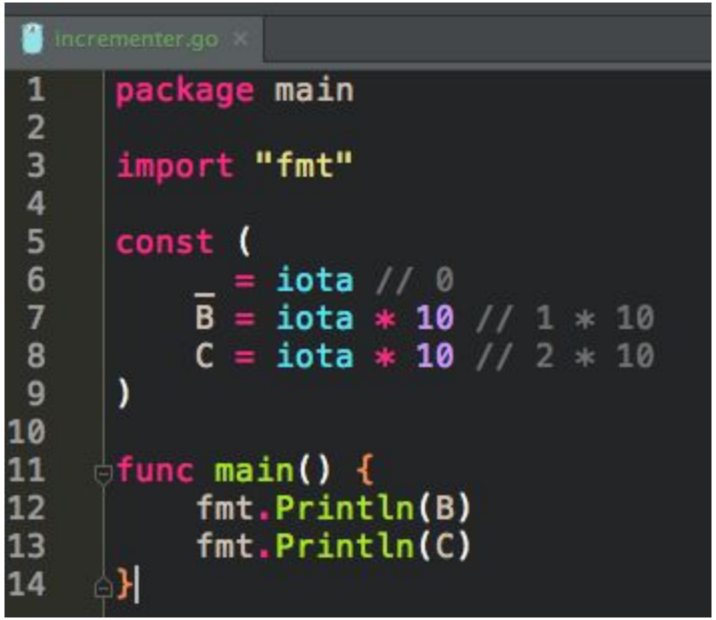
Within a constant declaration, the predeclared identifier `iota` represents successive untyped integer constants. It is reset to 0 whenever the reserved word `const` appears in the source and increments after each ConstSpec. It can be used to construct a set of related constants:

https://golang.org/ref/spec#ConstSpec

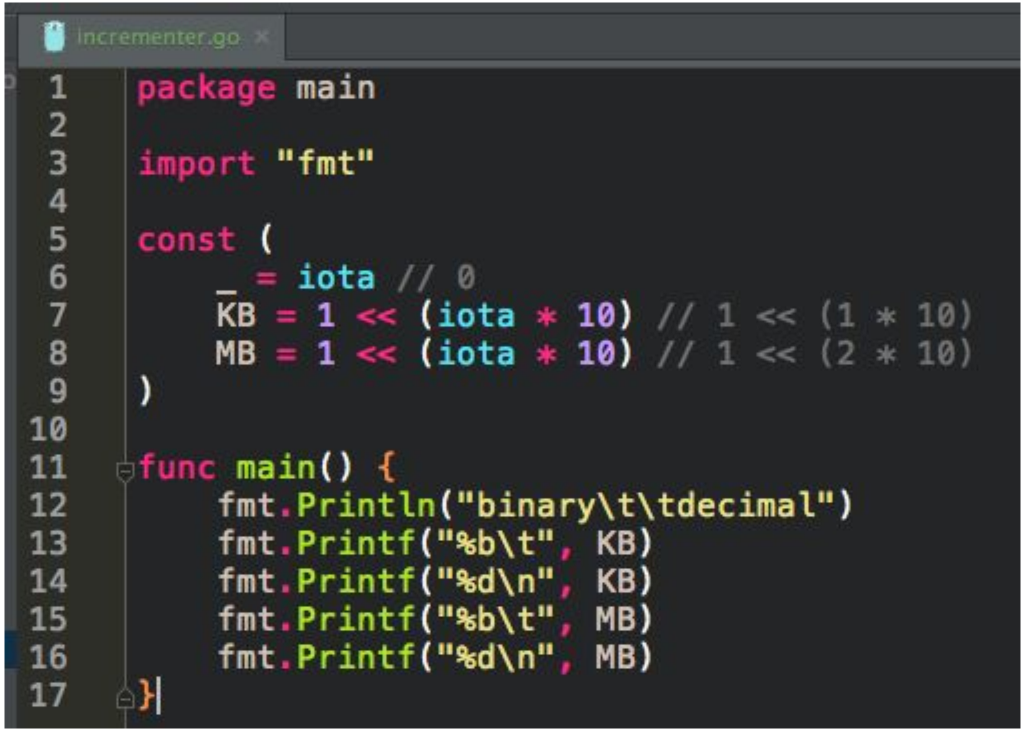
Constant declarations

A constant declaration binds a list of identifiers (the names of the constants) to the values of a list of constant expressions. The number of identifiers must be equal to the number of expressions, and the *n*th identifier on the left is bound to the value of the *n*th expression on the right.

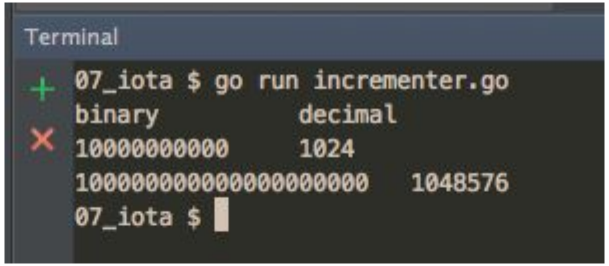
```
ConstDecl      = "const" ( ConstSpec | "(" { ConstSpec ";" } ")" ) .
ConstSpec      = IdentifierList [ [ Type ] "=" ExpressionList ] .

IdentifierList = identifier { "," identifier } .
ExpressionList = Expression { "," Expression } .
```


incrementer.go

```go
package main

import "fmt"

const (
	_ = iota // 0
	B = iota * 10 // 1 * 10
	C = iota * 10 // 2 * 10
)

func main() {
	fmt.Println(B)
	fmt.Println(C)
}
```



```go
package main

import "fmt"

const (
	_ = iota // 0
	KB = 1 << (iota * 10) // 1 << (1 * 10)
	MB = 1 << (iota * 10) // 1 << (2 * 10)
)

func main() {
	fmt.Println("binary\t\tdecimal")
	fmt.Printf("%b\t", KB)
	fmt.Printf("%d\n", KB)
	fmt.Printf("%b\t", MB)
	fmt.Printf("%d\n", MB)
}
```



```
07_iota $ go run incrementer.go
binary          decimal
10000000000     1024
100000000000000000000   1048576
07_iota $
```

bitwise operations

learn more if you want

exercise

declare a constant of type int
assign it the value of your age
use the constant in a statement

exercise

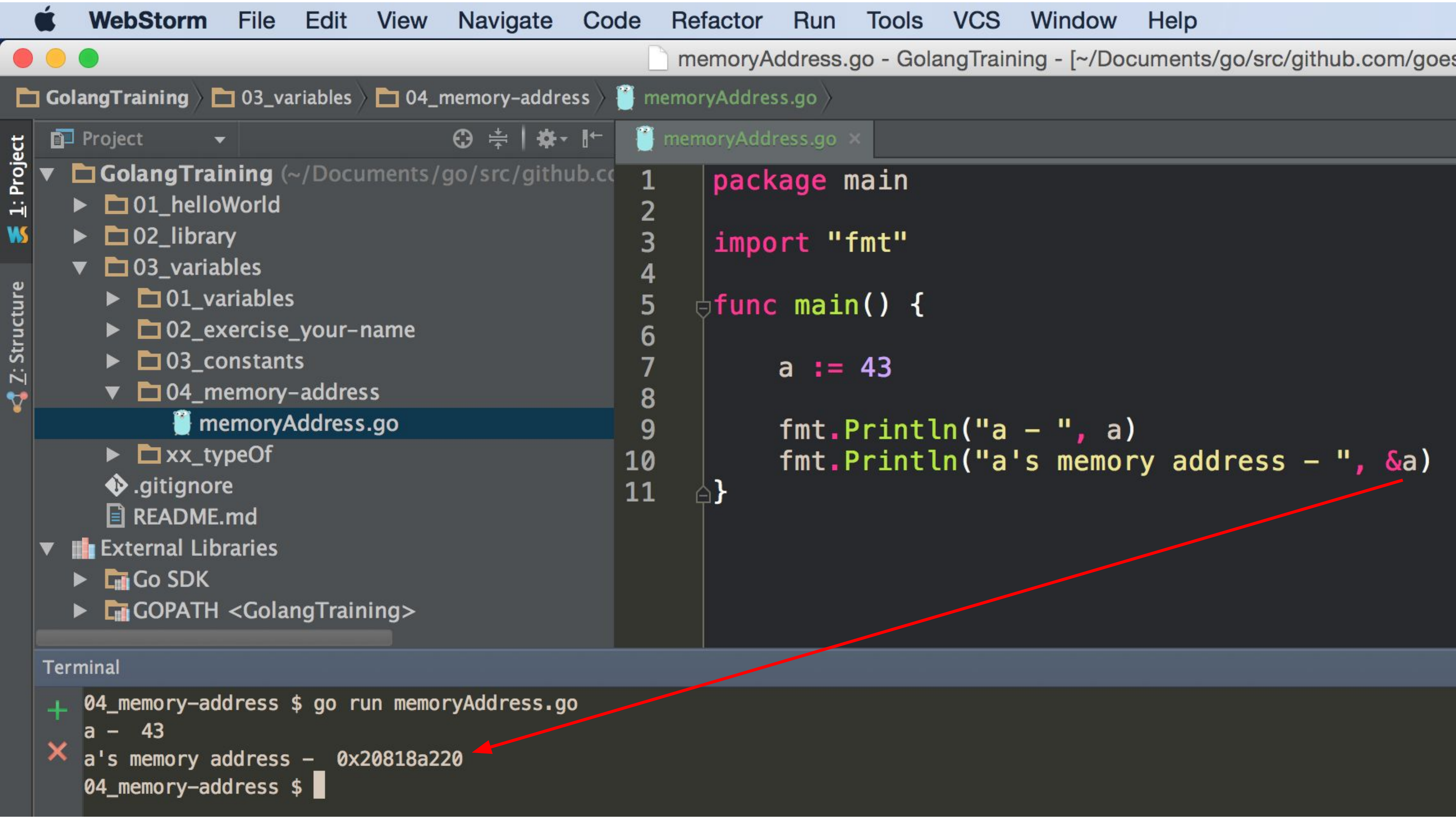
write some code using iota

memory address

And where do you live?

&

& where do you live?


WebStorm File Edit View Navigate Code Refactor Run Tools VCS Window Help
memoryAddress.go - GolangTraining - [~/Documents/go/src/github.com/goes
GolangTraining 03_variables 04_memory-address memoryAddress.go
Project
1: Project
Z: Structure
GolangTraining (~/Documents/go/src/github.c
01_helloWorld
02_library
03_variables
01_variables
02_exercise_your-name
03_constants
04_memory-address
memoryAddress.go
xx_typeOf
.gitignore
README.md
External Libraries
Go SDK
GOPATH <GolangTraining>
Terminal

```go
package main

import "fmt"

func main() {

	a := 43

	fmt.Println("a - ", a)
	fmt.Println("a's memory address - ", &a)
}
```

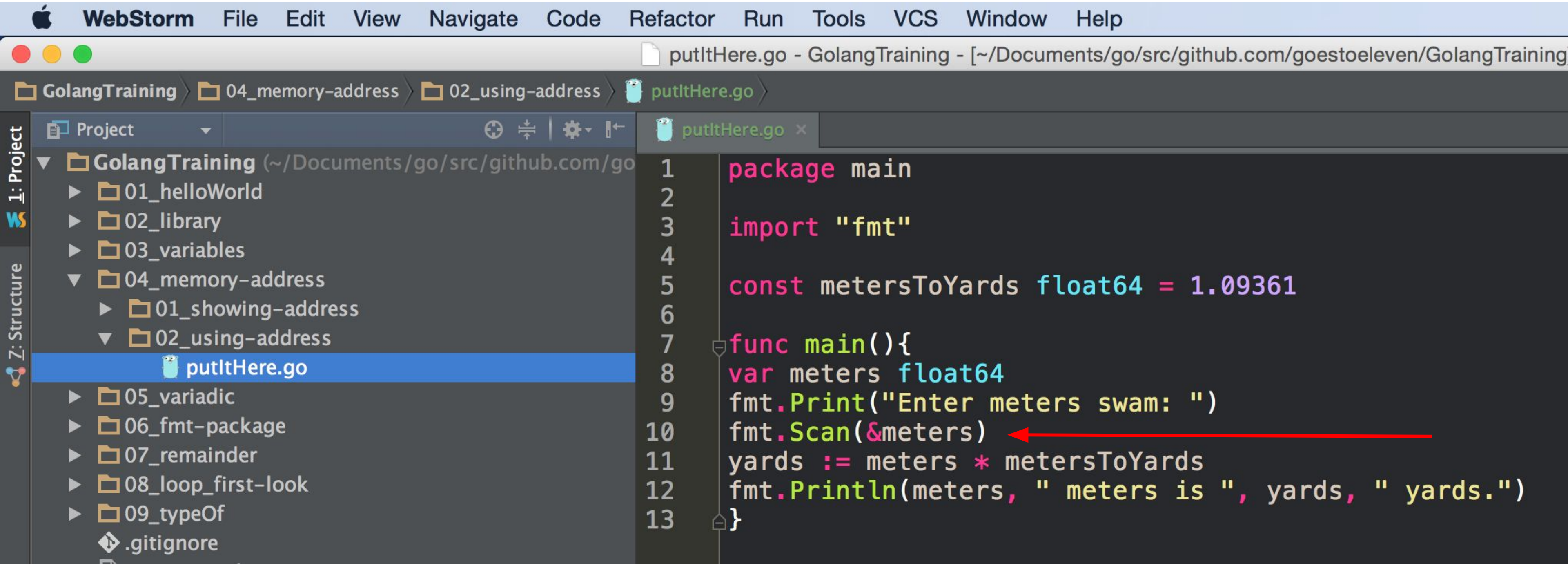
```
04_memory-address $ go run memoryAddress.go
a -  43
a's memory address -  0x20818a220
04_memory-address $
```

exercise

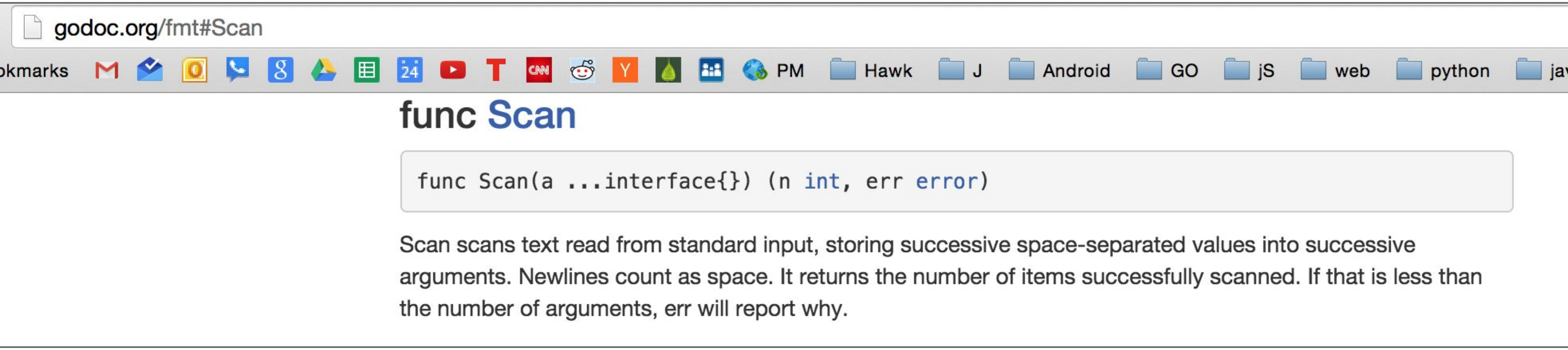
declare a variable
print the variable’s memory address

memory address

using the memory address



```go
package main

import "fmt"

const metersToYards float64 = 1.09361

func main(){
var meters float64
fmt.Print("Enter meters swam: ")
fmt.Scan(&meters)
yards := meters * metersToYards
fmt.Println(meters, " meters is ", yards, " yards.")
}
```

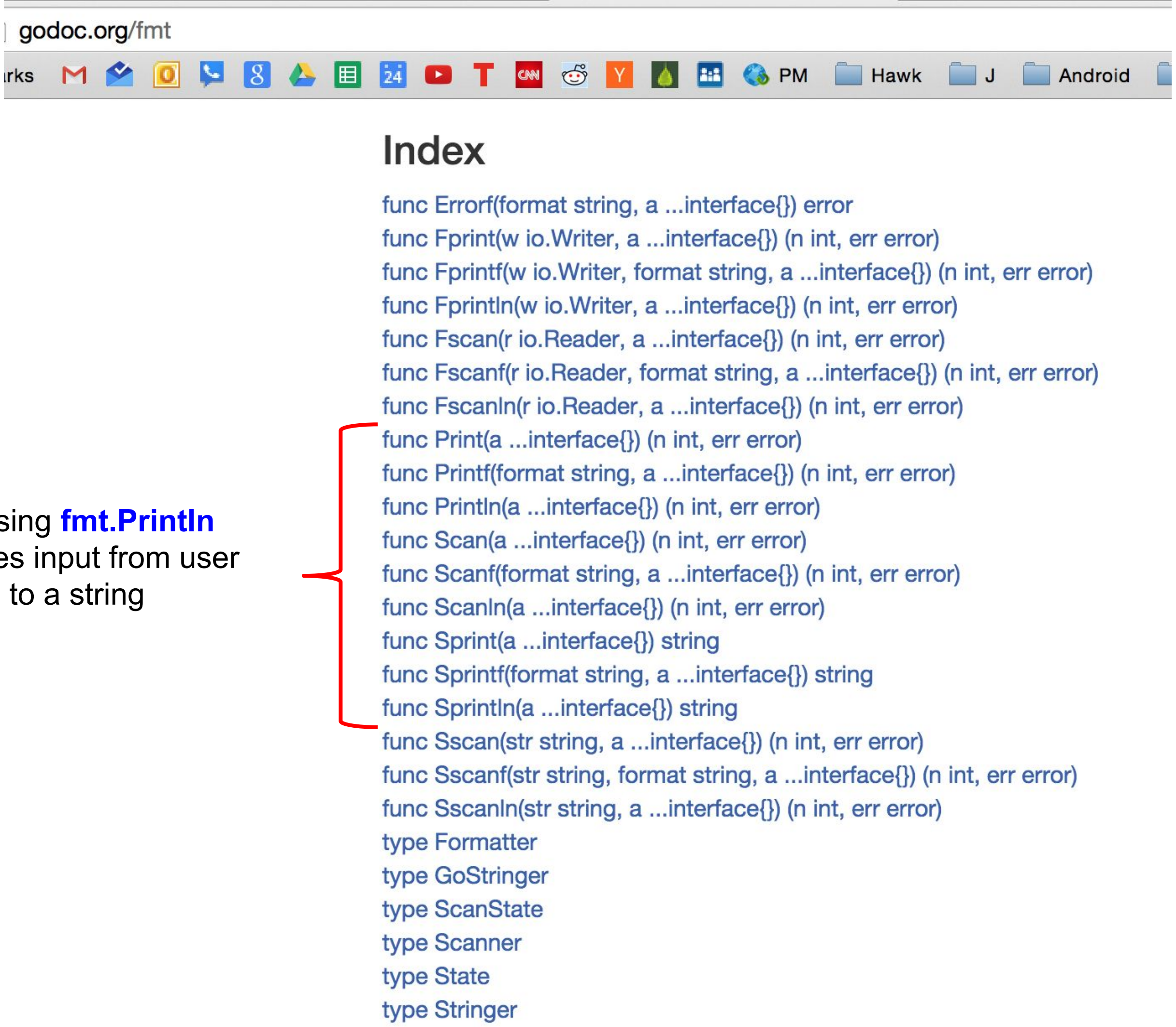


func Scan

```
func Scan(a ...interface{}) (n int, err error)
```

Scan scans text read from standard input, storing successive space-separated values into successive arguments. Newlines count as space. It returns the number of items successfully scanned. If that is less than the number of arguments, err will report why.

- We have been using **fmt.Println**
- **fmt.Scan** receives input from user
- **fmt.Sprint** prints to a string



Index

func Errorf(format string, a ...interface{}) error
func Fprint(w io.Writer, a ...interface{}) (n int, err error)
func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
func Fprintln(w io.Writer, a ...interface{}) (n int, err error)
func Fscan(r io.Reader, a ...interface{}) (n int, err error)
func Fscanf(r io.Reader, format string, a ...interface{}) (n int, err error)
func Fscanln(r io.Reader, a ...interface{}) (n int, err error)
func Print(a ...interface{}) (n int, err error)
func Printf(format string, a ...interface{}) (n int, err error)
func Println(a ...interface{}) (n int, err error)
func Scan(a ...interface{}) (n int, err error)
func Scanf(format string, a ...interface{}) (n int, err error)
func Scanln(a ...interface{}) (n int, err error)
func Sprint(a ...interface{}) string
func Sprintf(format string, a ...interface{}) string
func Sprintln(a ...interface{}) string
func Sscan(str string, a ...interface{}) (n int, err error)
func Sscanf(str string, format string, a ...interface{}) (n int, err error)
func Sscanln(str string, a ...interface{}) (n int, err error)
type Formatter
type GoStringer
type ScanState
type Scanner
type State
type Stringer

exercise

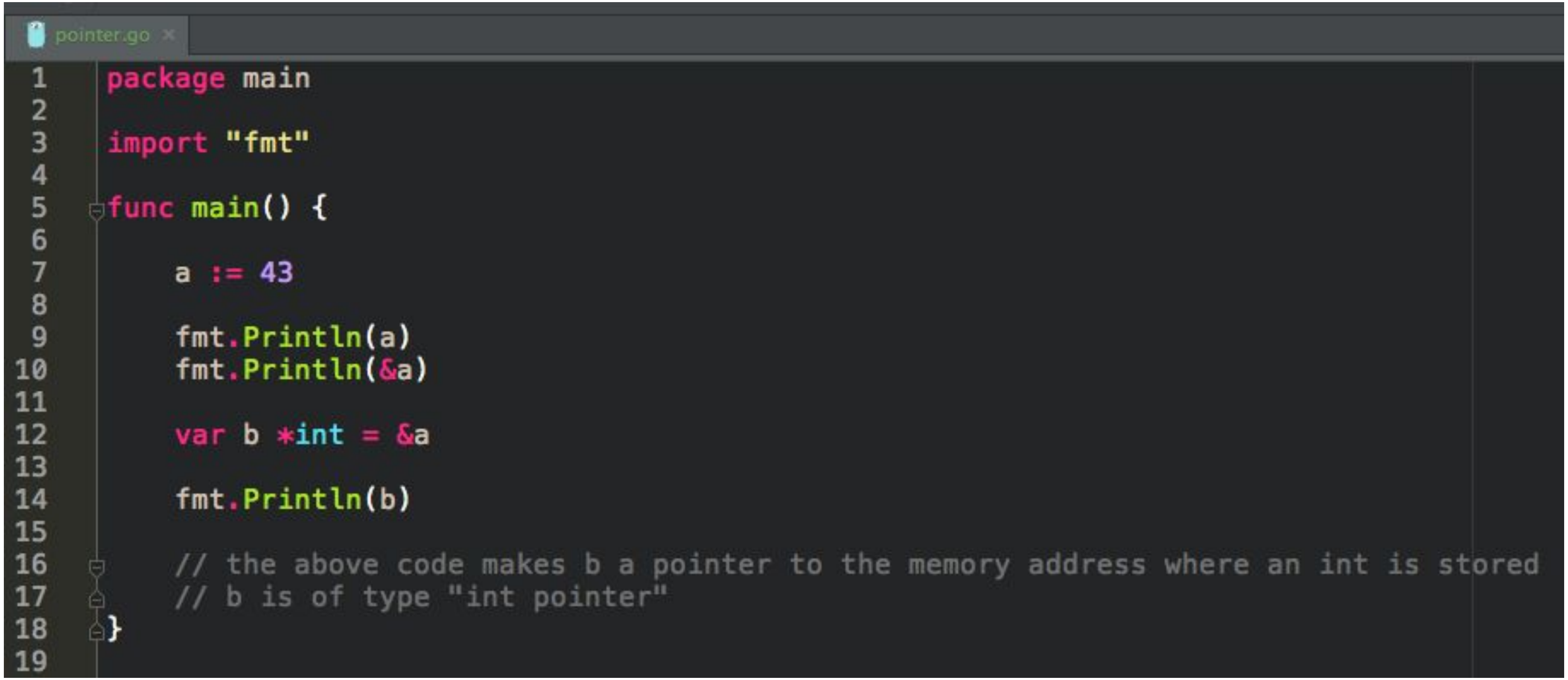
write a program
that receives input from the user
then does something with the input

pointers

pointing to memory addresses

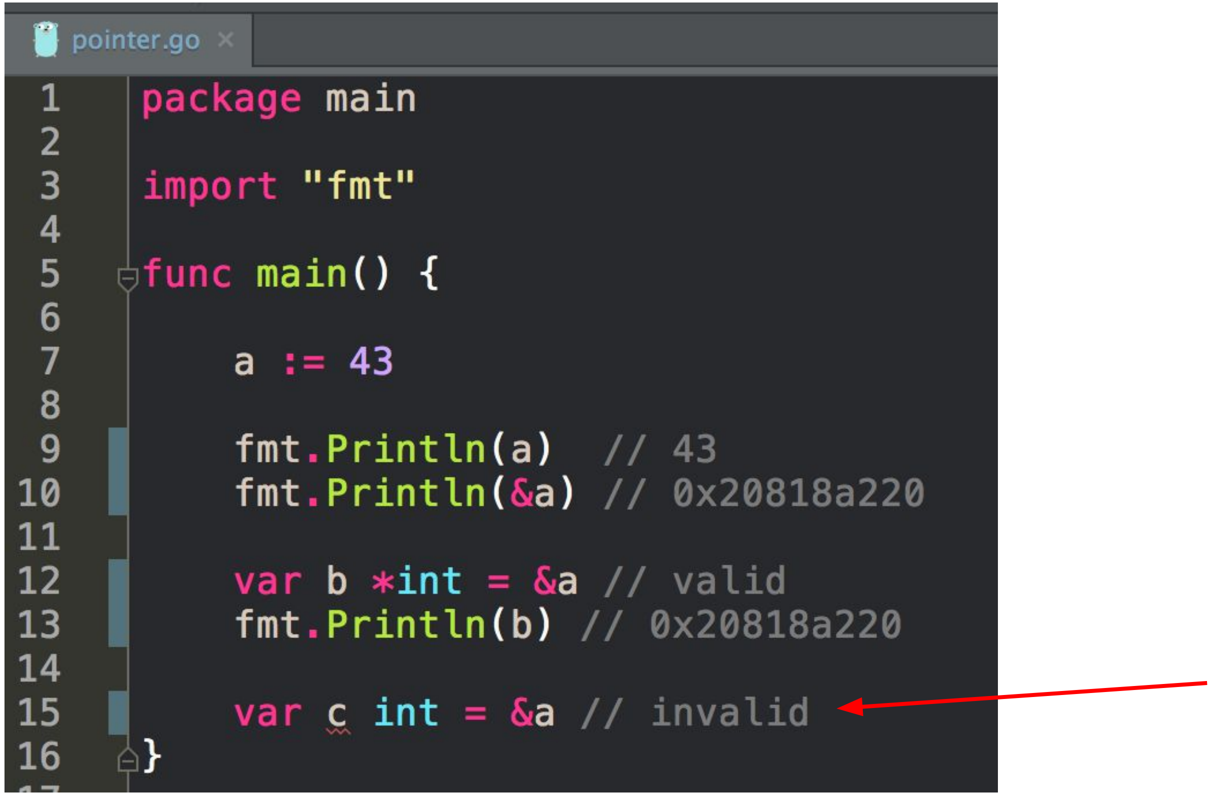
*

*The asterisk symbol is used in books to reference something more, is it not?



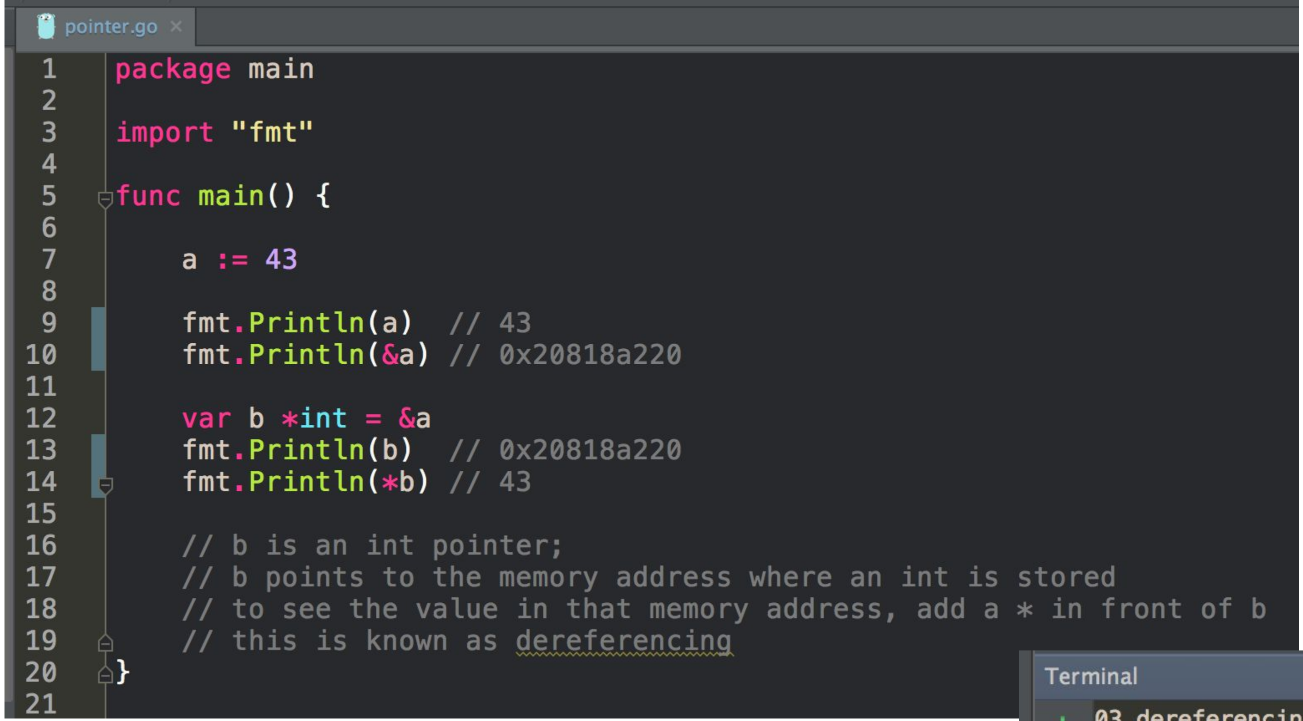
```go
package main

import "fmt"

func main() {

    a := 43

    fmt.Println(a)
    fmt.Println(&a)

    var b *int = &a

    fmt.Println(b)

    // the above code makes b a pointer to the memory address where an int is stored
    // b is of type "int pointer"
}

```

this is how we store, or **reference**, a memory address
the variable b is storing a memory address, and in that memory address, an int is stored
b is of type “int pointer”



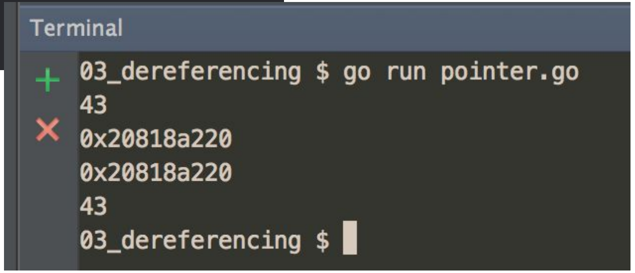
```
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      a := 43
8
9      fmt.Println(a)  // 43
10     fmt.Println(&a) // 0x20818a220
11
12     var b *int = &a // valid
13     fmt.Println(b) // 0x20818a220
14
15     var c int = &a // invalid
16 }
```

invalid code:
we’re saying on line 15 that c is of type int but then we’re assigning it a memory address, not an int



pointer.go

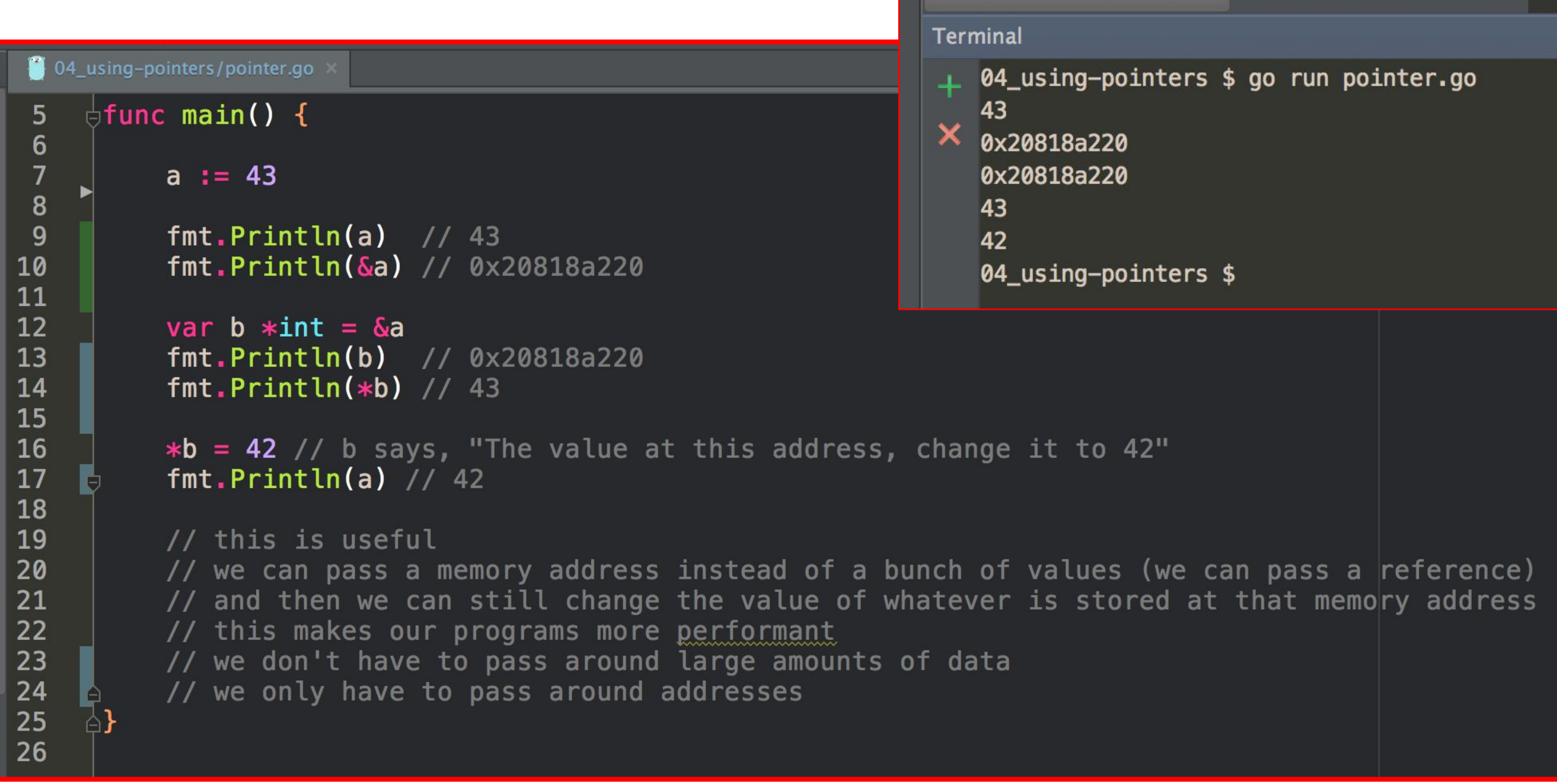
```go
package main

import "fmt"

func main() {

	a := 43

	fmt.Println(a)  // 43
	fmt.Println(&a) // 0x20818a220

	var b *int = &a
	fmt.Println(b)  // 0x20818a220
	fmt.Println(*b) // 43

	// b is an int pointer;
	// b points to the memory address where an int is stored
	// to see the value in that memory address, add a * in front of b
	// this is known as dereferencing
}
```



Terminal

```
03_dereferencing $ go run pointer.go
43
0x20818a220
0x20818a220
43
03_dereferencing $
```

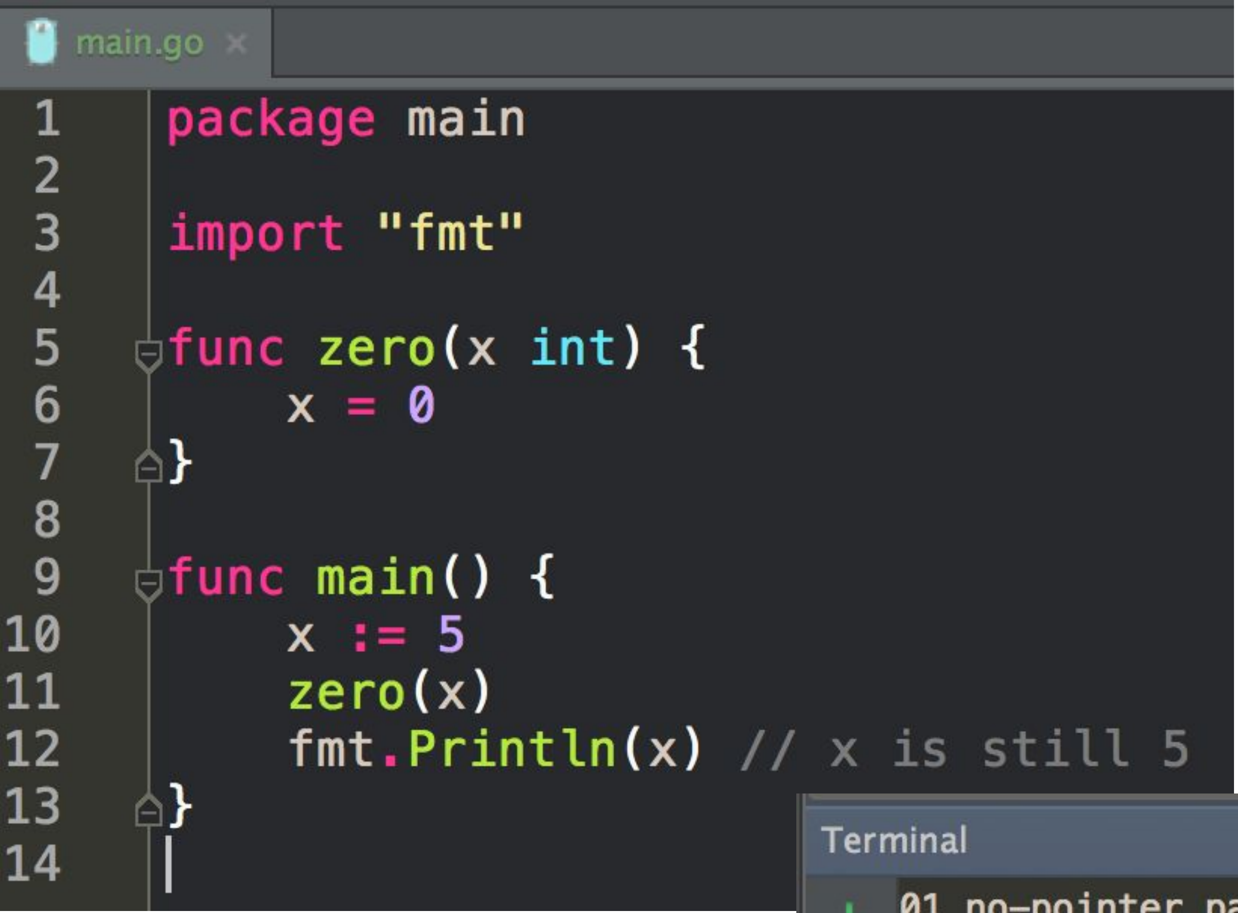
this is how we **dereference**
the variable b is storing a memory address
adding an asterisk in front of b says, “Show me the value in the memory address you’re storing.”


04_using-pointers/pointer.go

```go
func main() {

	a := 43

	fmt.Println(a)  // 43
	fmt.Println(&a) // 0x20818a220

	var b *int = &a
	fmt.Println(b)  // 0x20818a220
	fmt.Println(*b) // 43

	*b = 42 // b says, "The value at this address, change it to 42"
	fmt.Println(a) // 42

	// this is useful
	// we can pass a memory address instead of a bunch of values (we can pass a reference)
	// and then we can still change the value of whatever is stored at that memory address
	// this makes our programs more performant
	// we don't have to pass around large amounts of data
	// we only have to pass around addresses
}
```

Terminal

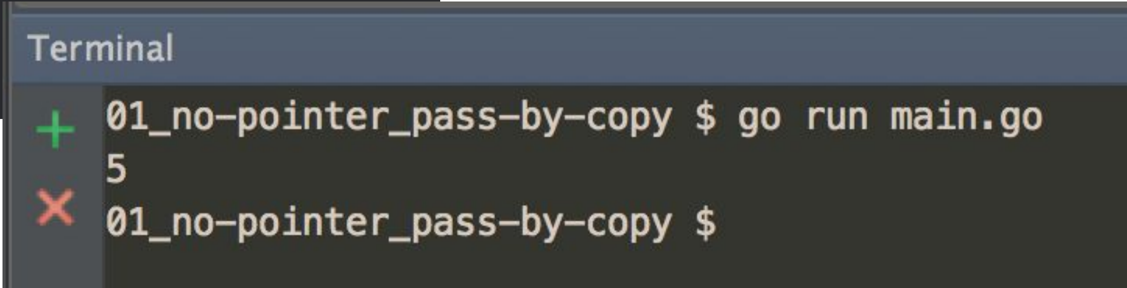
```
04_using-pointers $ go run pointer.go
43
0x20818a220
0x20818a220
43
42
04_using-pointers $
```


main.go

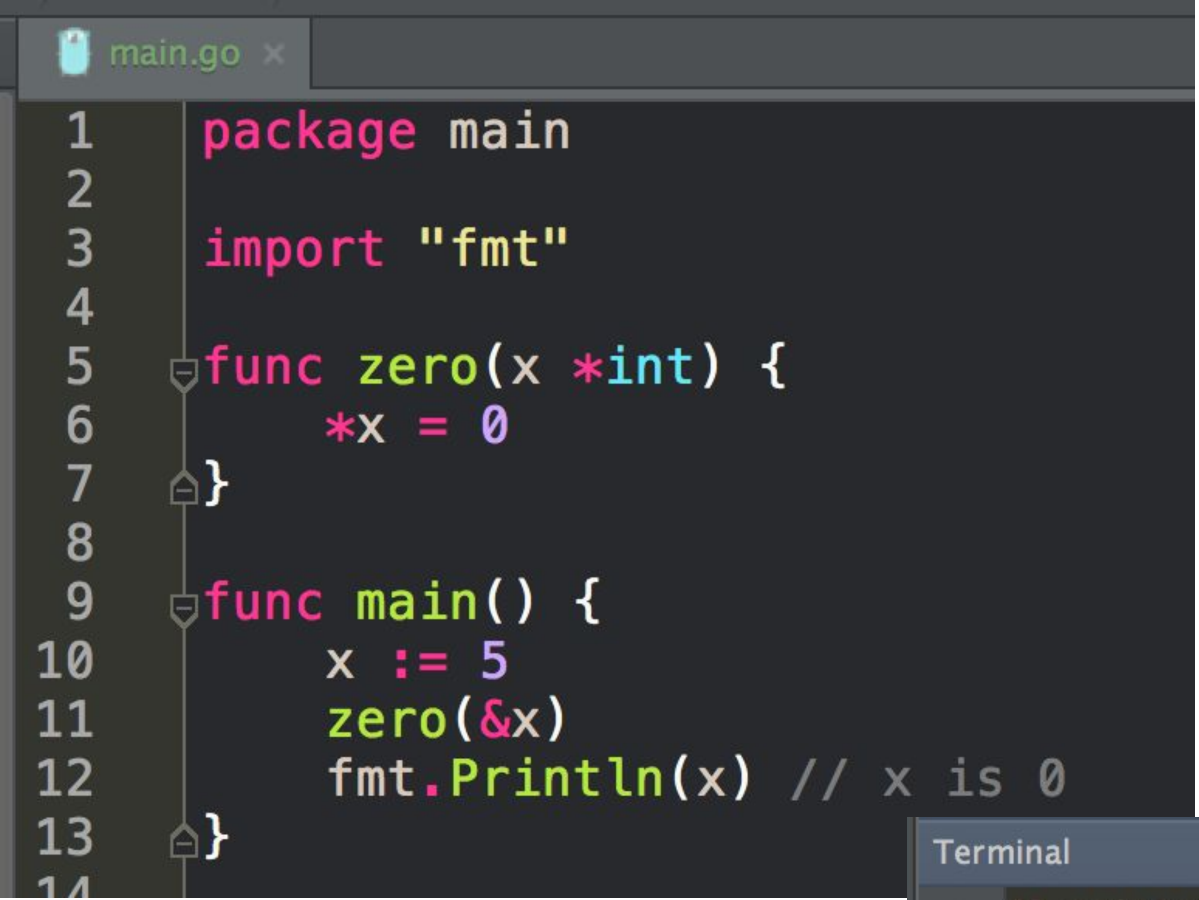
```go
package main

import "fmt"

func zero(x int) {
	x = 0
}

func main() {
	x := 5
	zero(x)
	fmt.Println(x) // x is still 5
}

```

Pass By Copy

A **copy** of the variable is passed into the function


Terminal

```
01_no-pointer_pass-by-copy $ go run main.go
5
01_no-pointer_pass-by-copy $
```

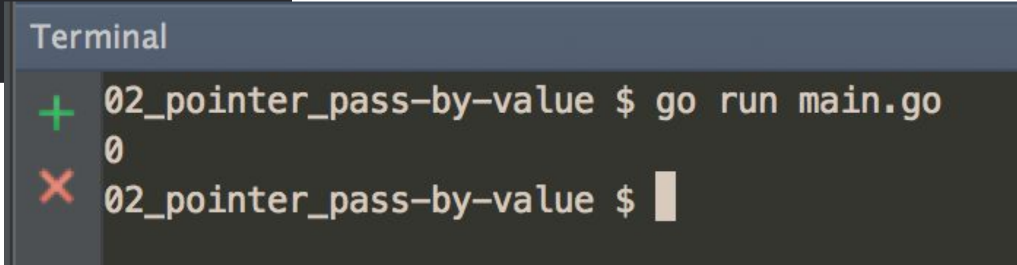


```go
package main

import "fmt"

func zero(x *int) {
	*x = 0
}

func main() {
	x := 5
	zero(&x)
	fmt.Println(x) // x is 0
}
```

Pass By Value

The memory address of the variable is passed into the function

the **value** of the variable can now be changed



```
02_pointer_pass-by-value $ go run main.go
0
02_pointer_pass-by-value $
```

exercise

write a program
that uses memory addresses and pointers

Review

- constants
- iota
    - bitwise operations
- memory address
    - **&**
        - *And where do you live?*
- pointers
    - *
        - referencing / dereferencing

Review Questions

bitwise

- Research and then explain how bitwise operations work.
    - Write a program that uses bitwise operations and use screenshots of your code and output in your explanation.

Memory Addresses & Pointers

- Describe the role that memory addresses and pointers play in go programming.
    - How might you use a memory address?
    - How might you use a pointer?
        - Why is it useful to use a pointer?
- What is the relationship between a memory address and a pointer?