

Variables & Scope

variables, zero value, scope, capitalization

**“There are only two hard things in Computer Science:
cache invalidation and naming things.”**

~ Phil Karlton

variables

```
package main
```

```
import "fmt"
```

```
func main() {  
    var message string  
    message = "Hello World."  
    fmt.Print(message)  
}
```

```
package main

import "fmt"

func main() {
    var message string
    message = "Hello World."
    fmt.Print(message)
    fmt.Println(message)
    fmt.Println(message)
}
```

```
package main

import "fmt"

func main() {
    var message string
    var a, b, c int
    a = 1

    message = "Hello World!"

    fmt.Println(message, a, b, c)
}
```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var message string = "Hello World!"
```

```
    var a, b, c int = 1, 2, 3
```

```
    fmt.Println(message, a, b, c)
```

```
}
```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var message = "Hello World!"
```

```
    var a, b, c = 1, 2, 3
```

```
    fmt.Println(message, a, b, c)
```

```
}
```



```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    var message = "Hello World!"
```

```
    var a, b, c = 1, false, 3
```

```
    fmt.Println(message, a, b, c)
```

```
}
```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    // you can only do this inside a func
```

```
    message := "Hello World!"
```

```
    a, b, c := 1, false, 3
```

```
    fmt.Println(message, a, b, c)
```

```
}
```

GolangTraining > 03_variables > 01_variables > variables.go

Project

- ▼ GolangTraining (~/Documents/go/src/github.com/goestoeleven/GolangTraining)
 - ▶ 01_helloWorld
 - ▶ 02_library
 - ▼ 03_variables
 - ▼ 01_variables
 - variables.go**
 - ▶ 02_typeOf
 - ▶ 03_constants
 - ▶ 04_priv_pub
 - ▶ .gitignore
 - ▶ README.md
 - ▼ External Libraries
 - ▶ Go SDK
 - ▶ GOPATH <GolangTraining>

```
variables.go x
1 package main
2
3 import "fmt"
4
5 var a string = "this is stored in the variable a" // package scope
6 var b, c string = "stored in b", "stored in c" // package scope
7 var d string // package scope
8
9 func main() {
10
11     d = "stored in d" // declaration above; assignment here; package scope
12     var e int = 42 // function scope - subsequent variables have same package scope:
13     f := 43
14     g := "stored in g"
15     h, i := "stored in h", "stored in i"
16     j, k, l, m := 44.7, true, false, 'm' // single quotes
17     n := "n" // double quotes
18     o := `o` // back ticks
19
20     fmt.Println("a - ", a)
21     fmt.Println("b - ", b)
22     fmt.Println("c - ", c)
23     fmt.Println("d - ", d)
24     fmt.Println("e - ", e)
25     fmt.Println("f - ", f)
26     fmt.Println("g - ", g)
27     fmt.Println("h - ", h)
28     fmt.Println("i - ", i)
29     fmt.Println("j - ", j)
30     fmt.Println("k - ", k)
31     fmt.Println("l - ", l)
32     fmt.Println("m - ", m)
33     fmt.Println("n - ", n)
34     fmt.Println("o - ", o)
35 }
```

variables.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

```
package main

import "fmt"

var a string = "this is stored in the variable a" // package scope
var b, c string = "stored in b", "stored in c" // package scope
var d string // package scope

func main() {

    d = "stored in d" // declaration above; assignment here; package scope
    var e int = 42 // function scope - subsequent variables have function scope
    f := 43
    g := "stored in g"
    h, i := "stored in h", "stored in i"
    j, k, l, m := 44.7, true, false, 'm' // single quotes
    n := "n" // double quotes
    o := `o` // back ticks

    fmt.Println("a - ", a)
    fmt.Println("b - ", b)
    fmt.Println("c - ", c)
```

Terminal

```
+ 03_variables $ cd 01_variables/
01_variables $ go run variables.go
X a - this is stored in the variable a
  b - stored in b
  c - stored in c
  d - stored in d
  e - 42
  f - 43
  g - stored in g
  h - stored in h
  i - stored in i
  j - 44.7
  k - true
  l - false
  m - 109
  n - n
  o - o
01_variables $
```

lexical

lex·i·cal

/ˈleksək(ə)l/

adjective

of or relating to the words or vocabulary of a language.

"lexical analysis"

- relating to or of the nature of a lexicon or dictionary.

"a lexical entry"

declare, assign, initialize

declare + assign = initialize

exercise

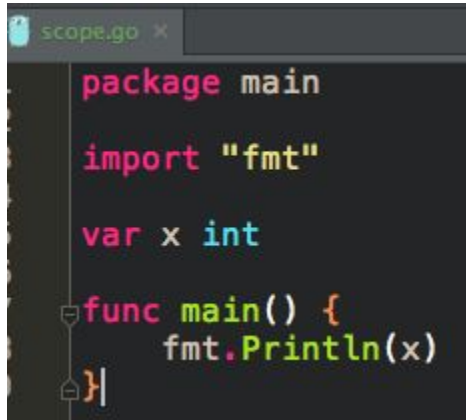
declare a variable of type string
assign it the value of your name
use the variable in an statement

zero value

false for booleans, 0 for integers, 0.0 for floats, "" for strings
nil for pointers, functions, interfaces, slices, channels, and maps

scope

universe → package → file → function → curly braces

A screenshot of a code editor window titled 'scope.go'. The code is written in Go and demonstrates package-level scope. It includes a package declaration, an import statement, a variable declaration, and a function definition. The variable 'x' is declared at the package level and is used within the 'main' function. The code is as follows:

```
package main

import "fmt"

var x int

func main() {
    fmt.Println(x)
}
```

package level scope

WebStorm File Edit View Navigate Code Refactor Run Tools VCS Window Help

main.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

GolangTraining > 05_scope > 02_package-scope > dukes > main.go

Project

- GolangTraining (~/.Documents/go/src/github.com/goestoeleven/GolangTraining)
 - 01_helloWorld
 - 02_library
 - 03_variadic
 - 04_variables
 - 05_scope
 - 01_package-scope
 - 02_package-scope
 - characters
 - dukes
 - 06_constants
 - 07_memory-address
 - a06_remainder
 - a07_for-loop
 - a08_switch-statements
 - a09_conditionals
 - a10_exercises
 - xx06_fmt-package
 - xx09_typeof
 - .gitignore
 - README.md
 - External Libraries

model.go

```
1 package characters
2
3 var sherrif string = "Boss Hogg"
4
```

controller.go

```
1 package characters
2
3 func Sherrif() string {
4     return sherrif
5 }
6
```

main.go

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/02_package-scope/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10 }
11
```

Terminal

```
+ dukes $ go run main.go
Boss Hogg
X dukes $
```

package level scope

```
model.go x
1 package characters
2
3 var sherrif string = "Boss Hogg"
4

controller.go x
1 package characters
2
3 func Sherrif() string {
4     return sherrif
5 }

main.go x
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/02_package-scope/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10    fmt.Print(characters.sherrif) // not valid - can't access the variable
11 }
```

invalid
variables declared at the top level (outside a function) have a package scope

main.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

Project

- GolangTraining (~/Documents/go/src/github.com/goestoeleven/GolangTraining)
 - 01_helloWorld
 - 02_library
 - 03_variadic
 - 04_variables
 - 05_scope
 - 01_package-scope
 - 02_package-scope
 - characters
 - controller.go
 - model.go
 - dukes
 - main.go
 - 03_package-scope
 - 06_constants
 - 07_memory-address
 - a06_remainder
 - a07_for-loop
 - a08_switch-statements
 - a09_conditionals
 - a10_exercises
 - xx06_fmt-package
 - xx09_typeOf
 - .gitignore
 - README.md

main.go

```
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4
```

controller.go

```
1 package characters
2
3 func Sherrif() string {
4     return SherrifName
5 }
```

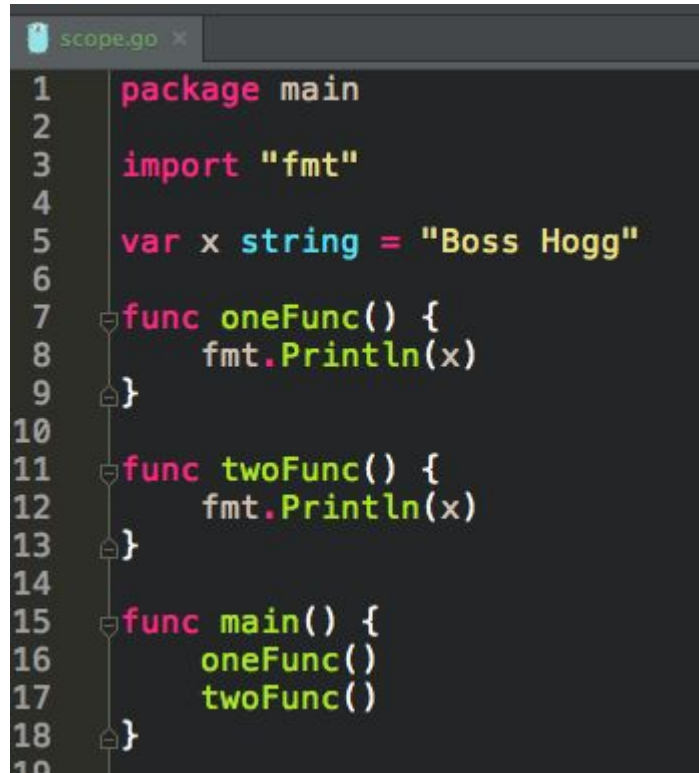
main.go

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/02_package-scope/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10    fmt.Println(characters.SherrifName) // valid - can access the variable
11 }
```

Terminal

```
+ Boss Hoggdukes $ go run main.go
Boss Hogg
Boss Hogg
dukes $
```

Capitalized variables and functions are accessible outside the package; like "public" in some languages




The image shows a code editor window with a tab labeled 'scope.go'. The code is written in Go and illustrates package-level scope. It includes a package declaration, an import, a global variable, and three functions. The variable 'x' is defined at the package level and is accessible within all functions. The functions 'oneFunc', 'twoFunc', and 'main' are defined at the package level and can access the global variable 'x'.

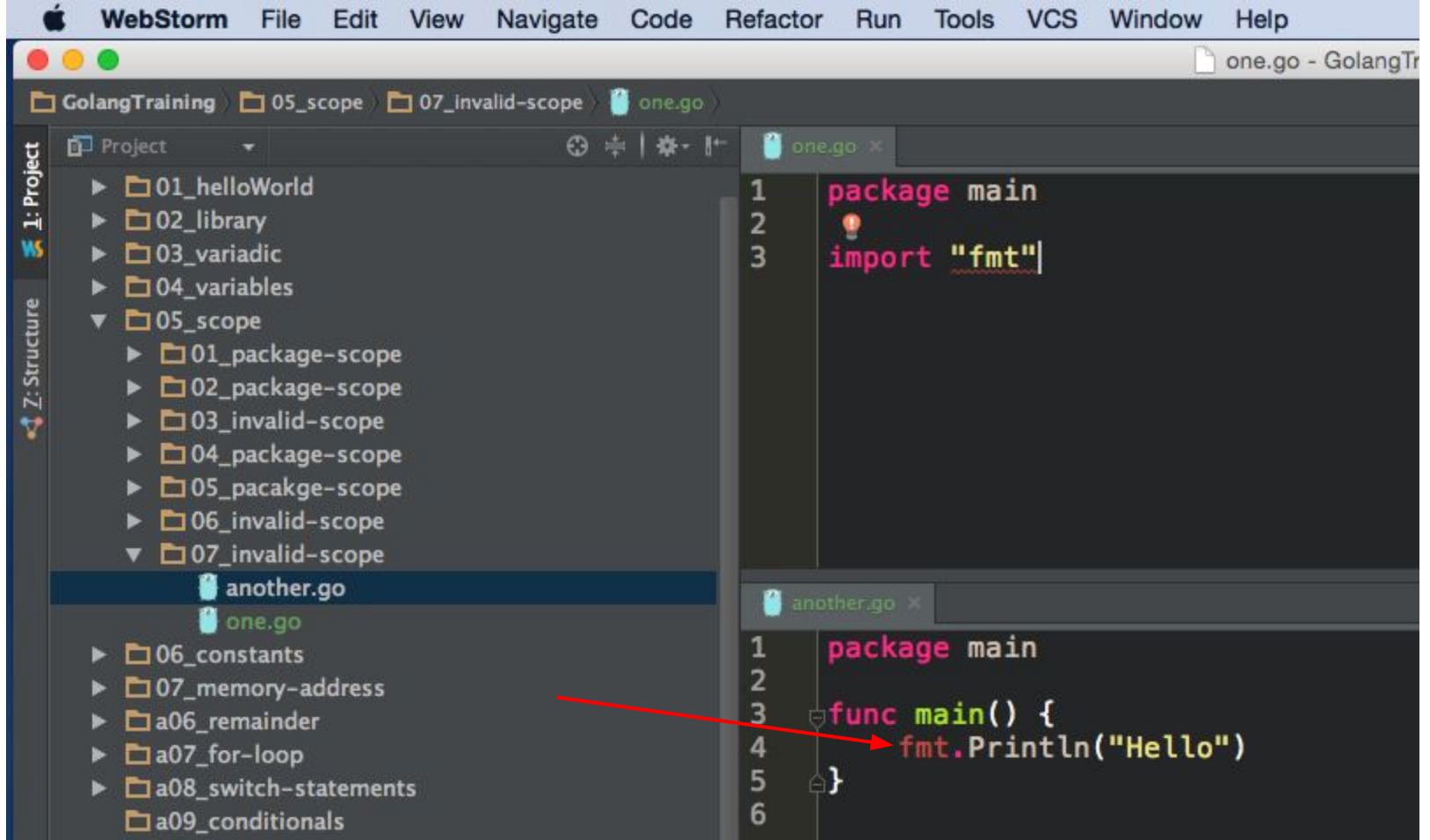
```
1 package main
2
3 import "fmt"
4
5 var x string = "Boss Hogg"
6
7 func oneFunc() {
8     fmt.Println(x)
9 }
10
11 func twoFunc() {
12     fmt.Println(x)
13 }
14
15 func main() {
16     oneFunc()
17     twoFunc()
18 }
```

package level scope

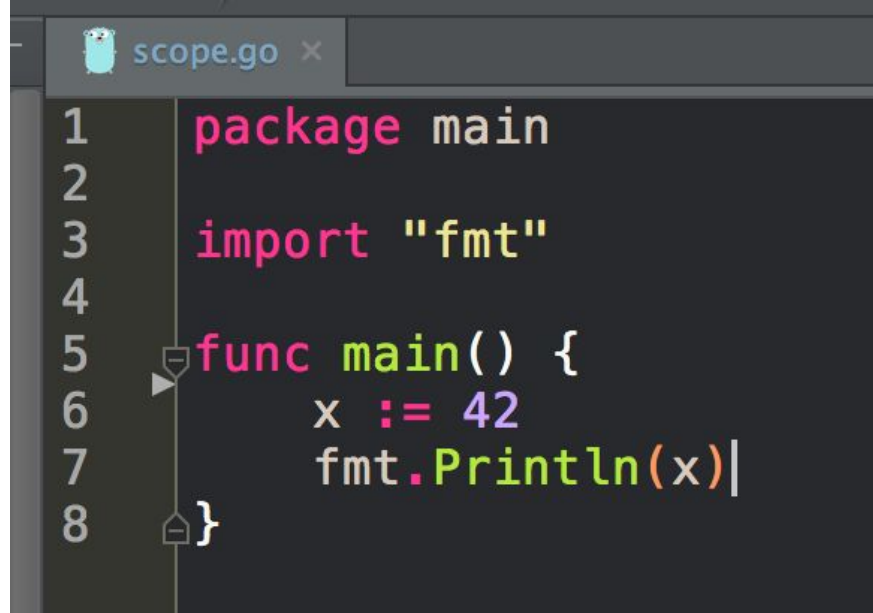
```
scope.go x
1  package main
2
3  import "fmt"
4
5  func oneFunc() {
6      var x string = "Boss Hogg"
7      fmt.Println(x)
8  }
9
10 func twoFunc() {
11     fmt.Println(x)
12 }
13
14 func main() {
15     oneFunc()
16     twoFunc()
17 }
18
```



invalid
scope of variable x is within the function in which it is declared



invalid
imports have a file scope




```
1 package main
2
3 import "fmt"
4
5 func main() {
6     x := 42
7     fmt.Println(x)
8 }
```

func level scope

```
scope.go x
package main

import "fmt"

func main() {
    fmt.Println(x)
    x := 42
}
```

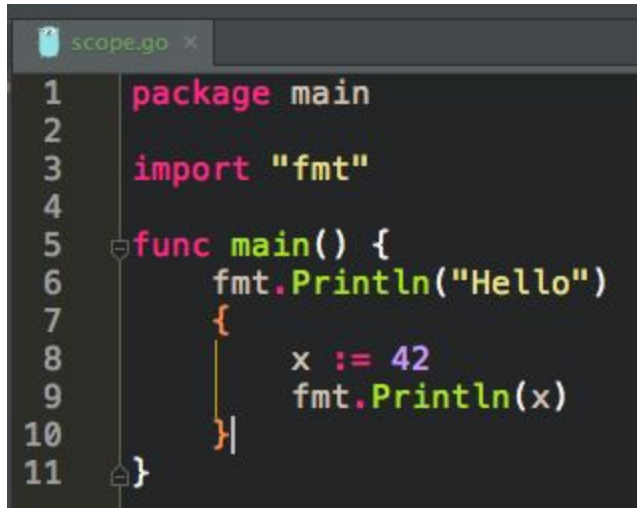


invalid

you can't use a variable before it is initialized; there is no hoisting like in javascript

```
scope.go x
1 package main
2
3 import "fmt"
4
5 func main() {
6     x := 42
7     another()
8 }
9
10 func another() {
11     fmt.Println(x)
12 }
```

invalid
scope of variable x is within the function in which it is declared



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello")
7     {
8         x := 42
9         fmt.Println(x)
10    }
11 }
```

The image shows a code editor window titled 'scope.go'. The code is written in Go and illustrates variable scope using curly braces. A function 'main' is defined, and within it, a block of code is enclosed in curly braces. A variable 'x' is declared and assigned the value 42 within this block, and its value is printed. The code is line-numbered from 1 to 11.

curly-braces level scope

```
scope.go x
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello")
7     {
8         x := 42
9         fmt.Println(x)
10    }
11    fmt.Println(x)
12 }
```

invalid
scope of variable x is within the curly-braces in which it is declared

resource

[learn more about scope](#)

capitalization

Public, private

The screenshot shows an IDE with the following structure:

- Project Explorer:** Shows a directory tree for `GolangTraining`. The `dukes` directory is selected, showing `main`.
- main.go (top):**

```
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4
```

Annotation: Capitalized variables and functions are accessible outside the package; like "public" in some languages
- controller.go:**

```
1 package characters
2
3 func Sherrif() string {
4     return SherrifName
5 }
```
- main.go (bottom):**

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/02_package-scope/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10    fmt.Println(characters.SherrifName) // valid - can access the variable
11 }
```

Annotation: // valid - can access the variable
- Terminal:**

```
+ Boss Hoggdukes $ go run main.go
Boss Hogg
Boss Hogg
dukes $
```

Callout: We already saw this

Capitalized top-level variables, and functions, are accessible outside the package.


```
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

invalid
scope on SideKick is func scope

Capitalized top-level variables, and functions, are accessible outside the package.

WebStorm File Edit View Navigate Code Refactor Run Tools VCS Window Help

controller.go - GolangTraining - [~/Documents/go/src/github.com/goestoeleven/GolangTraining]

GolangTraining 05_scope 15_capitalization characters controller.go

Project

- GolangTraining (~/Documents/go/src/github.com/goestoeleven/GolangTraining)
 - 01_helloWorld
 - 02_library
 - 03_variadic
 - 04_variables
 - 05_scope
 - 01_package-scope
 - 02_package-scope
 - 03_invalid-scope
 - 04_package-scope
 - 05_package-scope
 - 06_invalid-scope
 - 07_invalid-scope
 - 08_func-scope
 - 09_invalid-scope
 - 10_invalid-scope
 - 11_braces-scope
 - 12_invalid-scope
 - 13_capitalization
 - 14_capitalization
 - 15_capitalization
 - characters
 - controller.go
 - model.go
 - dukes
 - main.go
 - 06_constants
 - 07_memory-address
 - a06_remainder
 - a07_for-loop
 - a08_switch-statements
 - a09_conditionals

Z-Structure

model.go

```
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4 var sideKick = "Roscoe"
```

controller.go

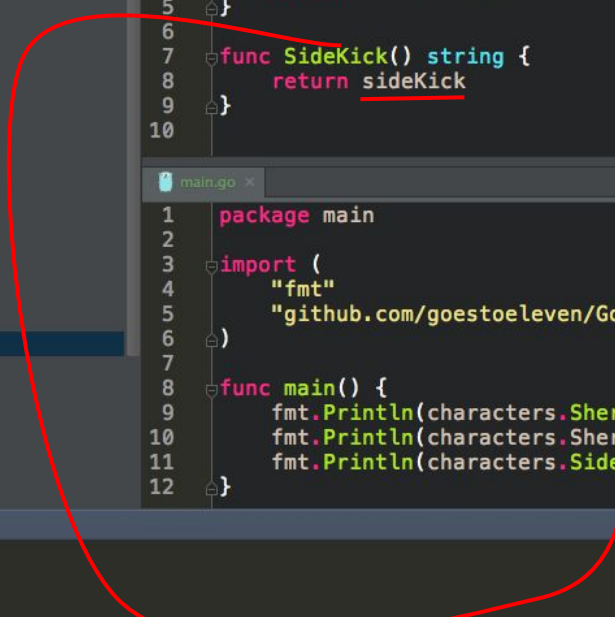
```
1 package characters
2
3 func Sherrif() string {
4     return SherrifName
5 }
6
7 func SideKick() string {
8     return sideKick
9 }
10
```

main.go

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/15_capitalization/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10    fmt.Println(characters.SherrifName)
11    fmt.Println(characters.SideKick())
12 }
```

Terminal

```
+ dukes $ go run main.go
Boss Hogg
Boss Hogg
Roscoe
dukes $
```



```
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4 var sideKick = "Roscoe"
```

```
1 package characters
2
3 func Sherrif() string {
4     return SherrifName
5 }
6
7 func sideK() string {
8     return sideKick
9 }
10
```

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/goestoeleven/GolangTraining/05_scope/16_invalid-capitalization/characters"
6 )
7
8 func main() {
9     fmt.Println(characters.Sherrif())
10    fmt.Println(characters.SherrifName)
11    fmt.Println(characters.sideK())
12 }
```

invalid
scope on **sideK()** is package scope

exercise

- create a package main
- create a package names
 - create the variable **MyName** and assign it a name
 - use MyName in package main
 - take a screenshot
 - create the variable **yourName** and assign it a name
 - what happens when you use yourName in package main?
 - take a screenshot

_blank identifier

for a variable you don't want to use

```
variables.go x
1 package main
2
3 import "fmt"
4
5 var a string = "this is stored in the variable a"|
6 var b, c string = "stored in b", "stored in c"
7
8 func main() {
9
10     fmt.Println("a - ", a)
11     fmt.Println("b - ", b)
12     fmt.Println("c is not being used - ") // invalid code
13 }
```

variables.go x

```
1 package main
2
3 import "fmt"
4
5 var a string = "this is stored in the variable a"
6 var b, _ string = "stored in b", "stored then thrown away"
7
8 func main() {
9
10     fmt.Println("a - ", a)
11     fmt.Println("b - ", b)
12     fmt.Println("c is not being used - and this is no problem")
13 }
```

```
for key, value := range oldMap {  
    newMap[key] = value  
}
```

```
sum := 0  
for _, value := range array {  
    sum += value  
}
```


Review

- can occur anywhere in your code
 - `var a string = "this is stored in the variable a"`
 - `var b, c string = "stored in b", "stored in c"`
 - `var d string`
 - `const p string = "death & taxes"`
 - `const q = 42`
- this lexical element `:=` can only occur inside a function
 - `f := 43`
 - `g := "stored in g"`
 - `h, i := "stored in h", "stored in i"`
 - `j, k, l, m := 44.7, true, false, 'm' // single quotes`
 - `n := "n"`
 - `o := `o` // backticks`
- `fmt.Println("a - ", a)`
- declare + assign = initialize
- capitalization
 - `Public`
 - `private`
- blank identifier

Review Questions

lexical elements

What are lexical elements? Give a few examples of lexical elements in golang.

golang spec

- Open the **golang spec**
- Under lexical elements, find the golang keywords
 - **take a screenshot of this**

golang spec

- Open the **golang spec**
- What are the two types of comments which may be used in golang code?

golang spec

- Open the **golang spec**
- What is an identifier?

golang spec

- Open the [golang spec](#)
- According to the golang spec, the first character in an identifier must be what?

golang spec

- Open the **golang spec**
- Find the operators which you can use in golang.
 - **take a screenshot of this**

zero value

- Describe zero value and how it occurs

zero value

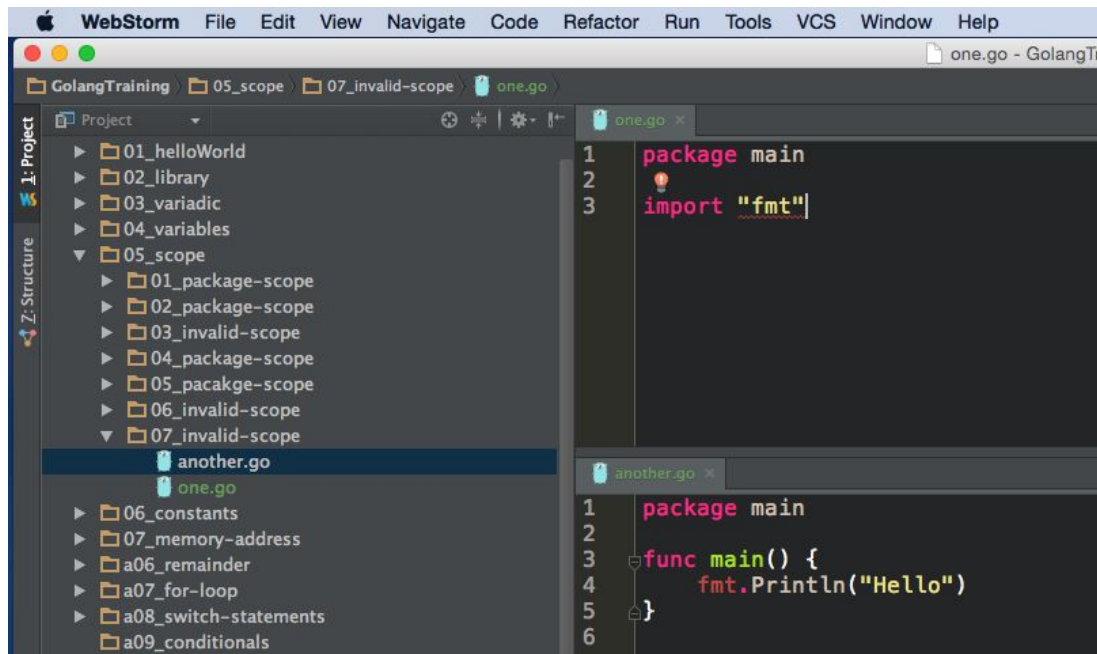
- What are the zero values for each of the following:
 - int
 - float
 - string
 - pointer
 - boolean

scope

- What are the levels of scope within golang?

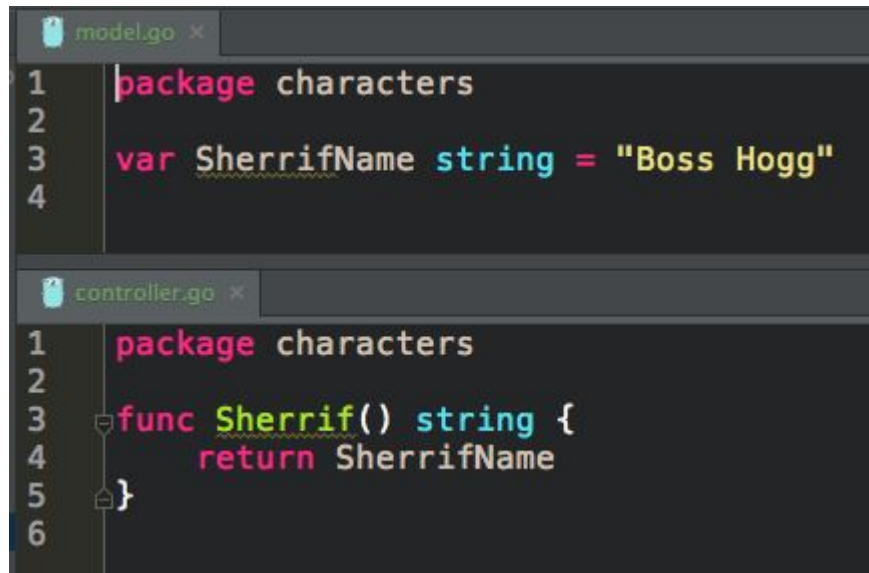
scope

- When you import a package in a file, what is that package's scope? For example, is "fmt" available in file f1.go?



scope

- When you declare a variable at the top-level (not in a function), and when you make that declaration in a file that is in a package, what is that variable's scope? For example, is the code below valid?



```
model.go x
1 package characters
2
3 var SherrifName string = "Boss Hogg"
4

controller.go x
1 package characters
2
3 func Sherrif() string {
4     return SherrifName
5 }
6
```

scope

- Is this code valid? Why or why not?

```
func oneFunc() {  
    var x string = "Boss Hogg"  
    fmt.Println(x)  
}  
  
func twoFunc() {  
    fmt.Println(x)  
}
```

scope

- Is this code valid? Why or why not?



```
one.go x
1 package main
2
3 import fmt

another.go x
1 package main
2
3 func main() {
4     fmt.Println("Hello")
5 }
6
```

scope

- Is this code valid? Why or why not?

```
func main() {  
    fmt.Println(x)  
    x := 42  
}
```


scope

- Is this code valid? Why or why not?

```
func main() {  
    fmt.Println("Hello")  
    {  
        x := 42  
        fmt.Println(x)  
    }  
    fmt.Println(x)  
}
```

scope

- How is the scope of a function changed when you **capitalize** the first letter of a function?

scope

- How is the scope of a variable changed when you **capitalize** the first letter of a variable?

scope

- Can you **capitalize** func main? Would you want to?