fmt package

fmt package, variadic

fmt package

We have been using fmt.Println

firstFile.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello world!")
}
```

http://godoc.org/fmt

let’s look at this package

package fmt

```
import "fmt"
```

Package fmt implements formatted I/O with functions analogous to C's printf and scanf. The format 'verbs' are derived from C's but are simpler.

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

What do these do?

func Print

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

Print formats using the default formats for its operands and writes to standard output. Spaces are added between operands when neither is a string. It returns the number of bytes written and any write error encountered.

func Printf ¶

```
func Printf(format string, a ...interface{}) (n int, err error)
```

Printf formats according to a format specifier and writes to standard output. It returns the number of bytes written and any write error encountered.

func Println

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

Println formats using the default formats for its operands and writes to standard output. Spaces are always added between operands and a newline is appended. It returns the number of bytes written and any write error encountered.

what do these ... mean?

func Print

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

Print formats using the default formats for its operands and writes to standard output. Spaces are added between operands when neither is a string. It returns the number of bytes written and any write error encountered.

func Printf ¶

```
func Printf(format string, a ...interface{}) (n int, err error)
```

Printf formats according to a format specifier and writes to standard output. It returns the number of bytes written and any write error encountered.

func Println

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

Println formats using the default formats for its operands and writes to standard output. Spaces are always added between operands and a newline is appended. It returns the number of bytes written and any write error encountered.

variadic

...params
args...

what does “…” mean in English?
it means there’s more, or there was more, or there was more here, doesn’t it?
example: The president said, “In these hard times … we all need to help.”
example: There were many options: cheese, wine, bread ….
example: … until the end of time.

func Print

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

Print formats using the default formats for its operands and writes to standard output. Space
between operands when neither is a string. It returns the number of bytes written and a
encountered.

func Printf ¶

```
func Printf(format string, a ...interface{})
```

Printf formats according to a format specifier a
written and any write error encountered.

func Println

```
func Println(a ...i
```

Println formats using the defa
added between operands and a
encountered.

The final parameter in a function signature may have a type prefixed with `...`. A function with such a parameter is called *variadic* and may be invoked with zero or more arguments for that parameter.

```
func()
func(x int) int
func(a, _ int, z float32) bool
func(a, b int, z float32) (bool)
func(prefix string, values ...int)
func(a, b int, z float64, opt ...interface{}) (success bool)
func(int, int, float64) (float64, *[]int)
func(n int) func(p *T)
```

t. Spaces are always
ytes written and any write error

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

func Print

```
func Print(a ...interface
```

Print formats using the default f
between operands when neith
encountered.

func Printf ¶

```
func Printf(format
```

Printf formats according
written and any write er

func Println

```
func Println(a
```

Println formats us
added between operan
encountered.

https://golang.org/ref/spec#Passing_arguments_to_..._parameters

Passing arguments to ... parameters

If f is variadic with a final parameter p of type ...T, then within f the type of p is equivalent to type []T. If f is invoked with no actual arguments for p, the value passed to p is nil. Otherwise, the value passed is a new slice of type []T with a new underlying array whose successive elements are the actual arguments, which all must be assignable to T. The length and capacity of the slice is therefore the number of arguments bound to p and may differ for each call site.

Given the function and calls

```
func Greeting(prefix string, who ...string)
Greeting("nobody")
Greeting("hello:", "Joe", "Anna", "Eileen")
```

within Greeting, who will have the value nil in the first call, and []string{"Joe", "Anna", "Eileen"} in the second.

If the final argument is assignable to a slice type []T, it may be passed unchanged as the value for a ...T parameter if the argument is followed by ... . In this case no new slice is created.

Given the slice s and call

```
s := []string{"James", "Jasmine"}
Greeting("goodbye:", s...)
```

within Greeting, who will have the same value as s with the same underlying array.

tion_types

main.go

```go
package main

import "fmt"

func main() {
    Greeting("hello: ", "James", "Jasmine")
}

func Greeting(prefix string, who ...string) {
    fmt.Println(prefix)
    for _, value := range who {
        fmt.Println(value)
    }
}
```

Terminal

```
01_variadic-params $ go run main.go
hello:
James
Jasmine
01_variadic-params $
```

main.go

```go
package main

import "fmt"

func main() {
    s := []string{"James", "Jasmine"}
    Greeting("hello: ", s...)
}

func Greeting(prefix string, who ...string) {
    fmt.Println(prefix)
    for _, value := range who {
        fmt.Println(value)
    }
}
```

Terminal

```
02_variadic-args $ go run main.go
hello:
James
Jasmine
02_variadic-args $
```

main.go

```go
package main

import "fmt"

func main() {
    s := []string{"James", "Jasmine"}
    Greeting("Goodbye", s)
}

func Greeting(prefix string, who []string) {
    fmt.Println(prefix)
    for _, value := range who {
        fmt.Println(value)
    }
}
```

Terminal

```
03_slice-instead-of-variadic $ go run main.go
Goodbye
James
Jasmine
03_slice-instead-of-variadic $
```

main.go

```go
package main

import "fmt"

func main() {
    f(1, 2)
    f(1, 2, 3)
    aSlice := []int{1, 2, 3, 4}
    f(aSlice...)
    f()
}

func f(numbers ...int) {
    fmt.Println(numbers)
}
```

Terminal

```
04_params-and-args $ go run main.go
[1 2]
[1 2 3]
[1 2 3 4]
[]
04_params-and-args $
```

exercise

create a program
that uses variadic params and variadic args

exercise

create a program
that uses a slice of ints param
and a slice of ints arg

fmt package

back to the fmt package

func Print

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

Print formats using the default formats for its operands and w ces are added between operands when neither is a string. It returns the n te error encountered.

func Printf ¶

```
func Printf(format string, a ... r)
```

Printf formats according to a format put. It returns the number of bytes written and any write error enco

func Println

```
func Println(a error)
```

Println formats usi perands and writes to standard output. Spaces are always added between opera ppended. It returns the number of bytes written and any write error encountered.

saw this

func Sprint

```
func Sprint(a ...interface{}) string
```

Sprint formats using the default formats for its operands and returns the resulting string. Spaces are added between operands when neither is a string.

func Sprintf

```
func Sprintf(format string, a ...interface{}) string
```

Sprintf formats according to a format specifier and returns the resulting string.

func Sprintln

```
func Sprintln(a ...interface{}) string
```

Sprintln formats using the default formats for its operands and returns the resulting string. Spaces are always added between operands and a newline is appended.

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.

func Scanf

```
func Scanf(format string, a ...interface{}) (n int, err error)
```

Scanf scans text read from standard input, storing successive space-separated values into successive arguments as determined by the format. It returns the number of items successfully scanned.

func Scanln

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

Scanln is similar to Scan, but stops scanning at a newline and after the final item there must be a newline or EOF.

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

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

GolangTraining 〉 05_fmt-package 〉 00_verbs 〉 fmtVerbs.go

Project

- 03_variables
- 04_memory-address
  - memoryAddress.go
- 05_fmt-package
  - 00_verbs
    - fmtVerbs.go
  - 01_verbs
  - 02_verbs

fmtVerbs.go

```go
package main

import "fmt"

func main() {
    s := fmt.Sprintf("%s", "STRING PRINT, meaning, print to a string")
    fmt.Println(s)
}
```

Terminal

```
00_verbs $ go run fmtVerbs.go
STRING PRINT, meaning, print to a string
00_verbs $
```

Printing

The verbs:

General:

```
%v      the value in a default format
        when printing structs, the plus flag (%+v) adds field names
%#v     a Go-syntax representation of the value
%T      a Go-syntax representation of the type of the value
%%      a literal percent sign; consumes no value
```

Boolean:

```
%t      the word true or false
```

Integer:

```
%b      base 2
%c      the character represented by the corresponding Unicode code point
%d      base 10
%o      base 8
%q      a single-quoted character literal safely escaped with Go syntax.
%x      base 16, with lower-case letters for a-f
%X      base 16, with upper-case letters for A-F
%U      Unicode format: U+1234; same as "U+%04X"
```

Floating-point and complex constituents:

```
%b      decimalless scientific notation with exponent a power of two,
        in the manner of strconv.FormatFloat with the 'b' format,
        e.g. -123456p-78
%e      scientific notation, e.g. -1234.456e+78
%E      scientific notation, e.g. -1234.456E+78
%f      decimal point but no exponent, e.g. 123.456
%F      synonym for %f
%g      %e for large exponents, %f otherwise
%G      %E for large exponents, %F otherwise
```

String and slice of bytes:

String and slice of bytes:

```
%s      the uninterpreted bytes of the string or slice
%q      a double-quoted string safely escaped with Go syntax
%x      base 16, lower-case, two characters per byte
%X      base 16, upper-case, two characters per byte
```

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

fmtVerbs.go - GolangTraining - [~/Document

GolangTraining 05_fmt-package

1: Project

7: Structure

Project

- GolangTraining (~/Documents/go/src/github.
  - 01_helloWorld
  - 02_library
  - 03_variables
  - 04_memory-address
    - memoryAddress.go
  - 05_fmt-package
    - 01_verbs
      - fmtVerbs.go
    - 02_verbs
      - fmtVerbs.go
  - xx_typeOf
  - .gitignore
  - README.md
- External Libraries
  - Go SDK

fmtVerbs.go

```go
1  package main
2
3  import "fmt"
4
5  func main() {
6
7      a := "Hello World!"
8
9      fmt.Println(a)
10     fmt.Println(&a)
11     fmt.Printf("%v\n", a)
12     fmt.Printf("%T\n", a)
13     fmt.Printf("%t\n", a)
14     fmt.Printf("%s\n", a)
15     fmt.Printf("%q\n", a)
16 }
```

Terminal

```
01_verbs $ go run fmtVerbs.go
Hello World!
0x20818a220
Hello World!
string
%!t(string=Hello World!)
Hello World!
"Hello World!"
01_verbs $
```

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

fmtVerbs.go - GolangTraining - [~/Documents/go/src/g

GolangTraining 〉 05_fmt-package 〉 02_verbs 〉 fmtVerbs.go

Project

- GolangTraining (~/Documents/go/src/github.
  - 01_helloWorld
  - 02_library
  - 03_variables
  - 04_memory-address
    - memoryAddress.go
  - 05_fmt-package
    - 01_verbs
      - fmtVerbs.go
    - 02_verbs
      - fmtVerbs.go
  - xx_typeOf
  - .gitignore
  - README.md
- External Libraries
  - Go SDK

fmtVerbs.go

```go
package main

import "fmt"

func main() {

    a := "Hello World!"

    fmt.Println("a - ", a)
    fmt.Println("address - ", &a)
    fmt.Printf("%%v - %v\n", a)
    fmt.Printf("%%T - %T\n", a)
    fmt.Printf("%%t - %t\n", a)
    fmt.Printf("%%s - %s\n", a)
    fmt.Printf("%%q - %q\n", a)
}
```

Terminal

```
02_verbs $ go run fmtVerbs.go
a -  Hello World!
address -  0x20818a220
%v - Hello World!
%T - string
%t - %!t(string=Hello World!)
%s - Hello World!
%q - "Hello World!"
02_verbs $
```

%% just prints %

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

fmtVerbs.go - GolangTraining - [~/Documents/go/src/gith

GolangTraining 〉 05_fmt-package 〉 03_verbs 〉 fmtVerbs.go

Project

- GolangTraining (~/Documents/go/src/github.
  - 01_helloWorld
  - 02_library
  - 03_variables
  - 04_memory-address
    - memoryAddress.go
  - 05_fmt-package
    - 01_verbs
    - 02_verbs
    - 03_verbs
      - fmtVerbs.go
  - xx_typeOf
  - .gitignore
  - README.md
- External Libraries
  - Go SDK

fmtVerbs.go

```go
import "fmt"

func main() {

    a := "Hello World!"
    b := false

    fmt.Println("a - ", a)
    fmt.Println("address - ", &a)
    fmt.Printf("%%v - %v\n", a)
    fmt.Printf("%%T - %T\n", a)
    fmt.Printf("%%t - %t\n", b)
    fmt.Printf("%%s - %s\n", a)
    fmt.Printf("%%q - %q\n", a)
}
```

Terminal

```
03_verbs $ go run fmtVerbs.go
a -  Hello World!
address -  0x20818a220
%v - Hello World!
%T - string
%t - false
%s - Hello World!
%q - "Hello World!"
03_verbs $
```

%v

The default format for %v is:

```
bool:                    %t
int, int8 etc.:          %d
uint, uint8 etc.:        %d, %x if printed with %#v
float32, complex64, etc: %g
string:                  %s
chan:                    %p
pointer:                 %p
```

Width is specified by an optional decimal number immediately following the verb. If absent, the width is whatever is necessary to represent the value. Precision is specified after the (optional) width by a period followed by a decimal number. If no period is present, a default precision is used. A period with no following number specifies a precision of zero. Examples:

```
%f:    default width, default precision
%9f    width 9, default precision
%.2f   default width, precision 2
%9.2f  width 9, precision 2
%9.f   width 9, precision 0
```

For floating-point values, width sets the minimum width of the field and precision sets the number of places after the decimal, if appropriate, except that for %g/%G it sets the total number of digits. For example, given 123.45 the format %6.2f prints 123.45 while %.4g prints 123.5. The default precision for %e and %f is 6; for %g it is the smallest number of digits necessary to identify the value uniquely.

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

fmtVerbs.go - GolangTraining - [~/Documents/go/src/github.com/goe

GolangTraining 〉 05_fmt-package 〉 04_verbs-width 〉 fmtVerbs.go

Project

- GolangTraining (~/Documents/go/src/github.
  - 01_helloWorld
  - 02_library
  - 03_variables
  - 04_memory-address
    - memoryAddress.go
  - 05_fmt-package
    - 01_verbs
    - 02_verbs
    - 03_verbs
    - 04_verbs-width
      - fmtVerbs.go
  - xx_typeOf
  - .gitignore
  - README.md
- External Libraries

fmtVerbs.go

```go
package main

import "fmt"

func main() {

    a := 762324.84274232380972

    fmt.Printf("%%v    - %v\n", a)
    fmt.Printf("%%f    - %f\n", a)
    fmt.Printf("%%.2f  - %.2f\n", a)
    fmt.Printf("%%9.2f - %9.2f\n", a)
    fmt.Printf("%%4.3f - %4.3f\n", a)
    fmt.Printf("%%20.5f - %20.5f\n", a)
    fmt.Printf("%%9.f  - %9.f\n", a)
}
```

Terminal

```
04_verbs-width $ go run fmtVerbs.go
%v    - 762324.8427423238
%f    - 762324.842742
%.2f  - 762324.84
%9.2f - 762324.84
%4.3f - 762324.843
%20.5f -         762324.84274
%9.f  -    762325
04_verbs-width $
```

examples of other verbs

https://raw.githubusercontent.com/GoesToEleven/GolangTraining/master/15_fmt-package/05_verbs/goByExample.go

os.Stdin
What is standard input for your operating system?

Scanning

An analogous set of functions scans formatted text to yield values. Scan, Scanf and Scanln read from os.Stdin; Fscan, Fscanf and Fscanln read from a specified io.Reader; Sscan, Sscanf and Sscanln read from an argument string. Scanln, Fscanln and Sscanln stop scanning at a newline and require that the items be followed by one; Scanf, Fscanf and Sscanf require newlines in the input to match newlines in the format; the other routines treat newlines as spaces.

F as in File
“... read from a specified io.Reader …”
io as in input output

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

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

GolangTraining > 05_fmt-package > 06_scan > scanf.go

Project

- GolangTraining (~/Documents/go/src/github.
  - 01_helloWorld
  - 02_library
  - 03_variables
  - 04_memory-address
  - 05_fmt-package
    - 00_verbs
    - 01_verbs
    - 02_verbs
    - 03_verbs
    - 04_verbs-width
    - 05_verbs
    - 06_scan
      - scanf.go

scanf.go

```go
package main

import "fmt"

const metersToYards float64 = 1.09361

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

Terminal

```
06_scan $ go run scanf.go
Enter meters swam: 3000
3000  meters is  3280.83  yards.
06_scan $
```

exercise

create a program
that uses **fmt.ScanIn** to receive keyboard input
does some calculation using that input
and then uses **fmt.PrintIn** to display results

Review

- fmt package
  - Print
    - Print
    - Printf
    - Println
  - Scan
    - Scan
    - Scanf
    - Scanln
  - Sprint
    - Sprint
    - Sprintf
    - Sprintln
- variadic
  - func sum(numbers ...int) {

    }
  - sum(aSlice...)
- verbs
  - %v
- receiving user input

```
var meters float64
fmt.Print("Enter meters swam: ")
fmt.Scanln(&meters)
```

- http://godoc.org/reflect
  - fmt.Println("a - ", reflect.TypeOf(a), " - ", a)

Review Questions

fmt package

- Describe the difference between Print, Printf, and Println

fmt package

- What are formatting verbs and how are they used?
- What is the default formatting verb?

fmt package

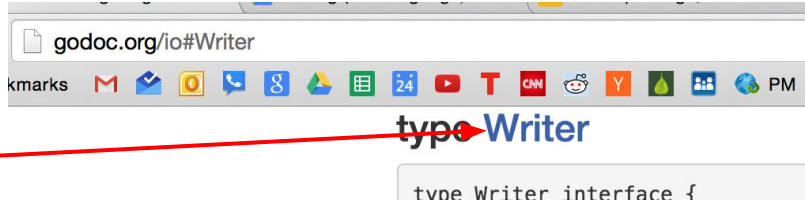
- What is the difference between **Print**, **Fprint**, & **Sprint**?
  - Read the documentation
    - describe how **Sprint** is used
      - provide a screenshot of code you wrote to illustrate **Sprint**
    - speculate as to how **Fprint** might be used
  - What is an **io.Writer**
    - describe an **io.Writer** using quotes from the documentation
      - take a screenshot of the **io.Writer** source code and include it as a curio in your description
        - (a curio is something that you don't understand yet but which is interesting)

cu·ri·o
/ˈkyo͝orēˌō/

noun

a rare, unusual, or intriguing object.
synonyms: trinket, knickknack, bibelot, ornament, bauble; More

click that to get to source code:



fmt package

- What is the difference between **Scan**, **Fscan**, & **Sscan**?
  - Read the documentation
    - describe how **Sscan** is used
      - provide a screenshot of code you wrote to illustrate **Sscan**
    - speculate as to how **Fscan** might be used
  - What is an **io.Reader**
    - describe an **io.Reader** using quotes from the documentation
      - take a screenshot of the **io.Reader** source code and include it as a curio in your description

fmt package

- Describe what Sprint does.
  - Provide a screenshot of code you wrote to illustrate your description.

variadic

- Describe the difference between variadic params & variadic args.
  - Provide a screenshot of code you wrote to illustrate your description.

...params

args...