An introduction into go

Hauke Stieler

January 16, 2018

Universität Hamburg, Department of Informatics

Agenda

Thank to Fred, on whose slides I was able to create these :)

Agenda

- some history
- basic features
- cool web stuff
- concurrency
- interfaces

Why go?

In 2007, three guys at Google were frustrated with the existing languages for writing server software:

- Compiling C++ was too slow
- Writing Java felt too verbose
- Aversion against inheritance and design patterns
- Getting concurrency right was hard

C++

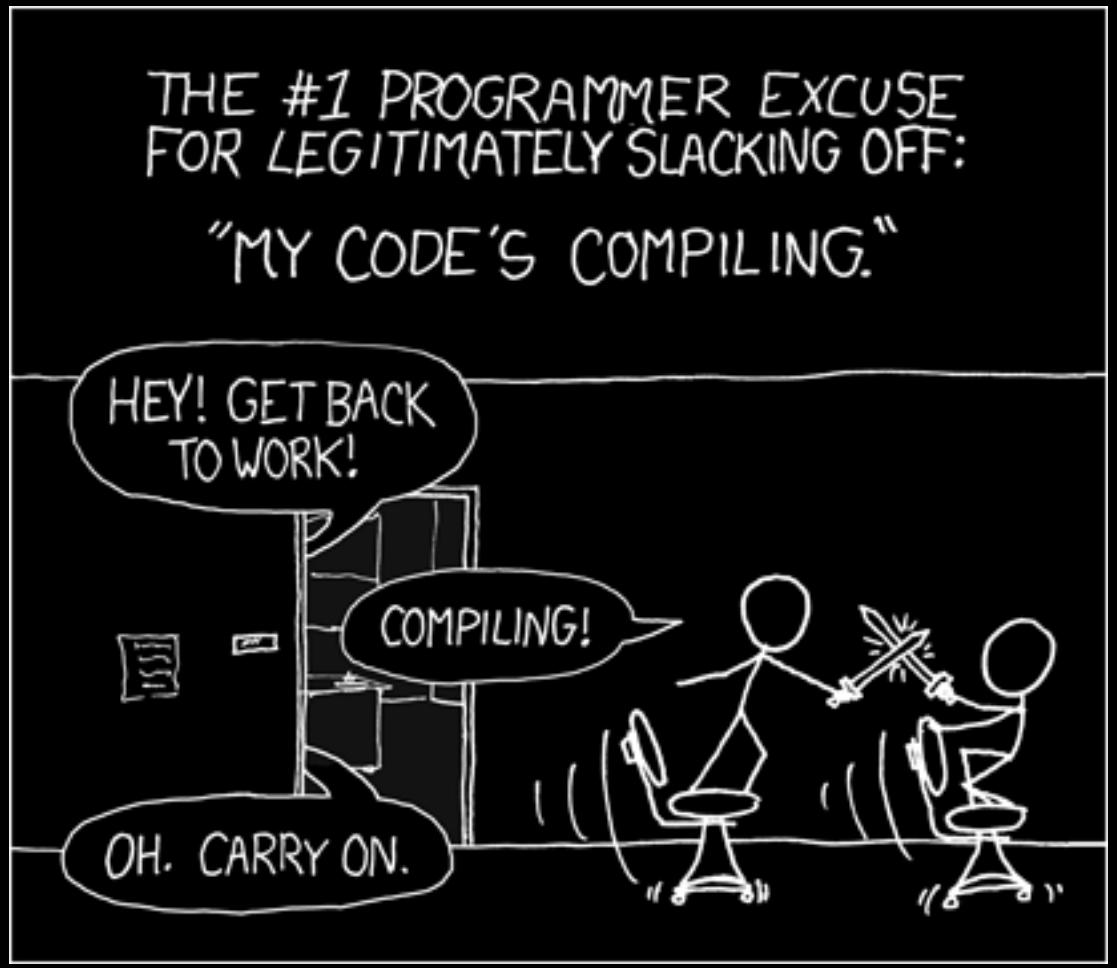
```
1  // Within large projects, popular header files
2  // get included thousands of times and hence
3  // have to be recompiled over and over again
4  #include <iostream>
5  #include <string>
6  #include <vector>
```

C++

```
1  // Within large projects, popular header files
2  // get included thousands of times and hence
3  // have to be recompiled over and over again
4  #include <iostream>
5  #include <string>
6  #include <vector>
```

gcc copies specified file by `#include` recursively into source file. The *same* header file gets recompiled over and over again.

→ Rob Pike: Public Static Void at OSCON 2010


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Java

Let's do some Java.

Write a `public class Person` that does the following:

- store a string `name`
- store an int age

Simple, right?

Java

Let's do some Java.
Write a `public class Person` that does the following:

- store a string `name`
- store an int `age`

Simple, right?

NO :(

Java I

```java
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

```

Java II

```
18      public int getAge() {
19          return age;
20      }
21
22      public void setAge(int age) {
23          this.age = age;
24      }
25
26      @Override
27      public String toString() {
28          return "Person [" + "name=" + name + ", " +    "age=" +
             ↪  age + "]";
29      }
30
31      @Override
32      public int hashCode() {
33          final int prime = 31;
```

Java III

```
34          int result = 1;
35          result = prime * result + age;
36          result = prime * result + ((name == null) ? 0 :
             ↪  name.hashCode());
37          return result;
38      }
39
40      @Override
41      public boolean equals(Object obj) {
42          if (this == obj)
43              return true;
44          if (obj == null)
45              return false;
46          if (getClass() != obj.getClass())
47              return false;
48          Person other = (Person) obj;
49          if (age != other.age)
```

Java IV

```
50              return false;
51          if (name == null) {
52              if (other.name != null)
53                  return false;
54          } else if (!name.equals(other.name))
55              return false;
56          return true;
57      }
58  }
```

Design

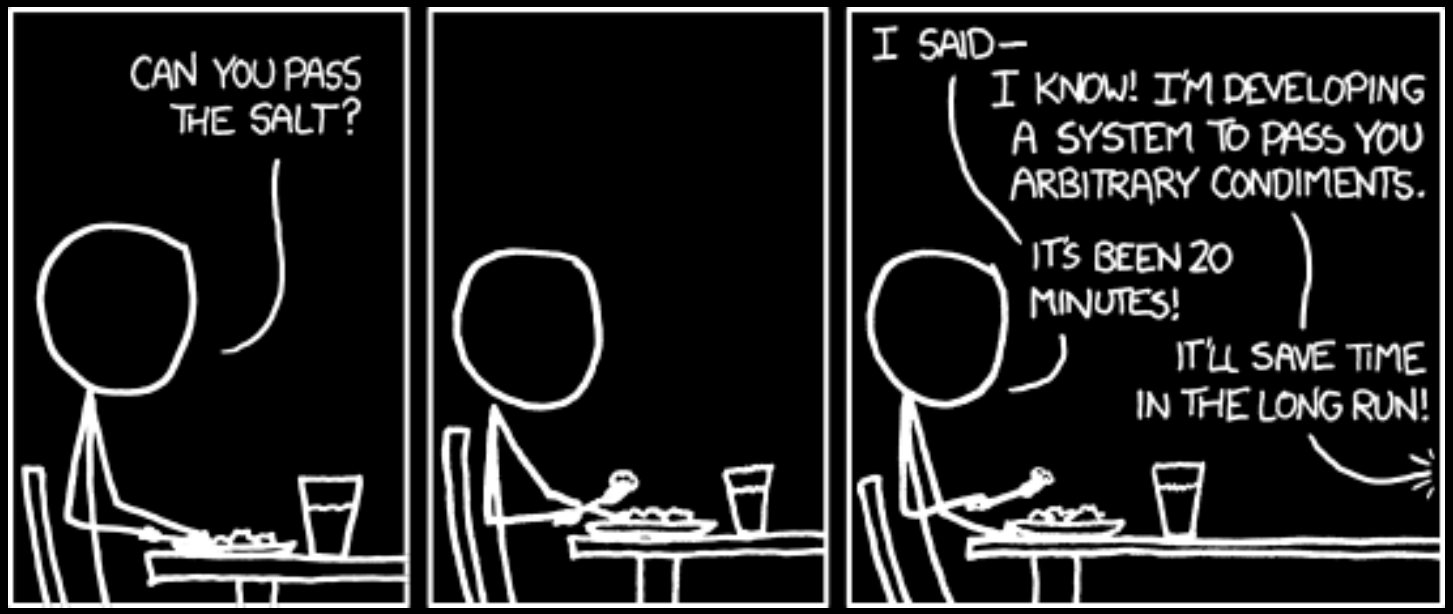
Initial design by 3 people with different backgrounds:

- Rob Pike (Concurrency)
- Robert Griesemer (Modules)
- Ken Thompson (Operating Systems)

All design decisions had to be agreed upon unanimously. Design team later joined by more people at Google.

Design

- **simplicity**
- **simplicity**
- **simplicity**
- clean package model for fast compilation
- built-in concurrency based on CSP
- interfaces instead of inheritance
- no radical changes after Go 1.0


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

Hello world!

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("hello world")
7  }
```

- The import declaration imports entire packages
- All imported names must be qualified
- Uppercase names are visible to other packages
- Unused imports are compile-time errors!

Hello world!

Get the go compiler:

```
$ sudo apt-get install golang-go
$ sudo pacman -S go

...or download from https://golang.org/dl
```

Run the code[1]:

```
$ go run hello.go
```

[1] The `go run` command works for single files, not always for projects

Basics

Keywords:

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

Constants:

```
true         false        nil          iota
```

Basics

Functions:

```
new         len         complex     panic
make        cap         real        recover
close       append      imag
            copy
            delete
```

Basics

Basic types:

```
int     int8     int16     int32     int64
uint    uint8    uint16    uint32    uint64    uintptr

float32      float64
complex64    complex128

bool    byte     rune      string    error
```

- `int` and `uint` are platform-dependent
- `byte` is the same as `uint8`
- `rune` is the same as `uint32`
- `uintptr` is large enough to hold pointers
- `error` is a special type for error handling

Basics

Operators:

```
*    /    %    &    &^   <<   >>
+    -    ^    |
==   !=   <    <=   >    >=
&&
||
```

- only 5 precedence levels!
- `^` is both bitwise-xor (infix) and bitwise-not (prefix)
- `&^` is bitwise-andn

Declarations

```
1  // three semantically identical alternatives
2  var x int = 0
3  var x int
4  var x = 0
5
6  // fourth alternative for local variables only
7  x := 0
```

Strings

```go
package main

import "fmt"

func main() {
    var s = "Käsebrötchen"
    fmt.Println(s)
    fmt.Println(len(s)) // 14

    // loop over the bytes (UTF-8 code units)
    for i := 0; i < len(s); i++ {
        fmt.Printf("%02d: %c\n", i, s[i])
    }

    // loop over the runes (Unicode code points)
    for i, r := range s {
        fmt.Printf("%02d: %c\n", i, r)
    }
```

Arrays

```
1  func initSha1(a [20]byte) {
2      // a is an array of 20 bytes
3      a[0] = 0x01
4      a[1] = 0x23
5      a[2] = 0x45
6      a[3] = 0x67
7      // ...
8      fmt.Println(a)
9  }
10
11 func main() {
12     var x [20]byte // arrays have a fixed size
13     initSha1(x)    // arrays are passed by value!
14     fmt.Println(x) // x remains unchanged
15 }
```

Pointers

```
1  func initSha1(a *[20]byte) {
2      a[0] = 0x01
3      a[1] = 0x23
4      a[2] = 0x45
5      // ...
6      fmt.Println(*a)
7  }
8
9  func main() {
10     var x [20]byte
11     initSha1(&x) // &x is a pointer to x
12     fmt.Println(x)
13 }
```

Slices

```
1  func initSha1(a []byte) {
2      a[0] = 0x01
3      a[1] = 0x23
4      a[2] = 0x45
5      // ...
6      fmt.Println(a)
7  }
8
9  func main() {
10     var x [20]byte
11     initSha1(x[:]) // x[:] is a slice (view) into x
12     fmt.Println(x) // x[:] is short-hand for x[0:20]
13 }
```

Slices can grow

```
1  func main() {
2      var compound [100]bool
3      for i := 2; i*i < len(compound); i++ {
4          if !compound[i] {
5              for j := i * i; j < len(compound); j += i {
6                  compound[j] = true
7              }
8          }
9      }
10     var primes []int
11     for i := 2; i < len(compound); i++ {
12         if !compound[i] {
13             primes = append(primes, i)
14         }
15     }
16     fmt.Println(primes)
17 }
```

Exercise

I. Extract two functions from the last main function:
   - `markCompounds`
   - `gatherPrimes`

II. Determine the growth strategy of append by printing after each call:
   - either a pointer to the first element

     or
   - the result of calling the special cap function

Maps

```go
func main() {
    birth := map[string]int{
        "C":    1972,
        "Java": 1994,
    }
    birth["Go"] = 2007
    examine(birth, "Go")
    delete(birth, "Go")
    examine(birth, "Go")
}

func examine(birth map[string]int, language string) {
    if year, present := birth[language]; present {
        fmt.Printf("%s was released in %d\n", language, year)
    } else {
        fmt.Printf("Never heard of %s...\n", language)
    }
}
```

Maps

```
1  func main() {
2      birth := map[string]int{
3          "C":    1972,
4          "Java": 1994,
5          "go":  2007,
6      }
7      for language, year := range birth {
8          fmt.Printf("%s was released in %d\n", language, year)
9      }
10 }
```

Exercise

Write a function that counts the occurrences of all characters in a given string.

Which value is returned by map lookup for missing keys?

Structs

```go
type Person struct {
    Name string
    Age  int
}

func main() {
    alice := Person{"Alice", 21}

    bob := alice
    bob.Name = "Bob"
    bob.Age++

    fmt.Printf("%v\n", alice) // {Alice 21}
    fmt.Printf("%#v\n", bob)  // main.Person{Name:"Bob",
    ↪  Age:22}
}
```

Structs

```
1  type Person struct {
2      Name string
3      Age  int
4  }
5
6  func main() {
7      myBoss := &Person{"Guido", 60}
8
9      yourBoss := myBoss
10     yourBoss.Age++
11
12     fmt.Println(myBoss, yourBoss) // &{Guido 61} &{Guido 61}
13 }
```

Web client I

```go
type Xkcd struct { // full spec at https://xkcd.com/json.html
    Title string
    Hover string `json:"alt"`
}

func FetchCurrentXkcdComic() (*Xkcd, error) {
    response, err := http.Get("https://xkcd.com/info.0.json")
    if err != nil {
        return nil, err
    }
    defer response.Body.Close()
    if response.StatusCode != http.StatusOK {
        return nil, errors.New(response.Status)
    }
    var xkcd Xkcd
    decoder := json.NewDecoder(response.Body)
    if err := decoder.Decode(&xkcd); err != nil {
        return nil, err
    }
    return &xkcd, nil
}
```

Web client II

```
1  func main() {
2      xkcd, err := FetchCurrentXkcdComic()
3      if err != nil {
4          fmt.Println(err)
5      } else {
6          fmt.Printf("%s\n\n%s\n", xkcd.Title, xkcd.Hover)
7      }
8  }
```

Exercise

Fetch the 3 most recent XKCD comics and print additional information of your choice.

Web server I

```
1  func main() {
2      http.HandleFunc("/", root)
3      fmt.Println("waiting for requests...")
4      http.ListenAndServe(":8080", nil)
5  }
6  var counter = 0
7  func root(writer http.ResponseWriter, request *http.Request) {
8      fmt.Printf("%s\n\n", request)
9      counter++
10     fmt.Fprintf(writer, "<html><body><pre>")
11     fmt.Fprintln(writer, request.URL.Path)
12     fmt.Fprintf(writer, "%d visits\n", counter)
13     fmt.Fprintf(writer, "</pre></body></html>")
14 }
```

Web server I

```
1  func main() {
2      http.HandleFunc("/", root)
3      fmt.Println("waiting for requests...")
4      http.ListenAndServe(":8080", nil)
5  }
6  var counter = 0
7  func root(writer http.ResponseWriter, request *http.Request) {
8      fmt.Printf("%s\n\n", request)
9      counter++
10     fmt.Fprintf(writer, "<html><body><pre>")
11     fmt.Fprintln(writer, request.URL.Path)
12     fmt.Fprintf(writer, "%d visits\n", counter)
13     fmt.Fprintf(writer, "</pre></body></html>")
14 }
```

Compiles and runs. Where's the bug?

Web server II

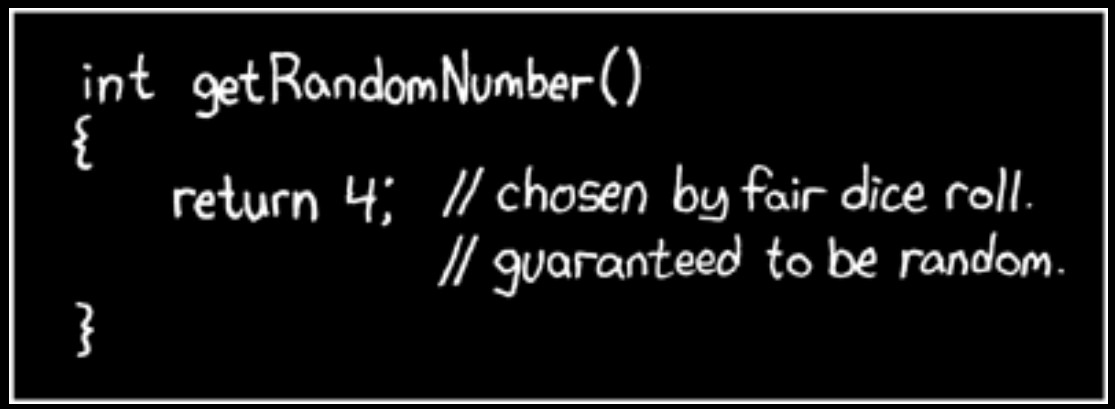
```
1  var counter = 0
2  var mutex sync.Mutex
3
4  func root(writer http.ResponseWriter, request *http.Request) {
5      fmt.Printf("%s\n\n", request)
6
7      mutex.Lock()
8      counter++
9      count := counter
10     mutex.Unlock()
11
12     fmt.Fprintf(writer, "<html><body><pre>")
13     fmt.Fprintln(writer, request.URL.Path)
14     fmt.Fprintf(writer, "%d visits\n", count)
15     fmt.Fprintf(writer, "</pre></body></html>")
16 }
```

Web server III

```
1  func main() {
2      http.HandleFunc("/", root)
3      fmt.Println("waiting for requests...")
4      http.ListenAndServe(":8080", nil)
5  }
6
7  var counter int32
8
9  func root(writer http.ResponseWriter, request *http.Request) {
10     fmt.Printf("%s\n\n", request)
11
12     count := atomic.AddInt32(&counter, 1)
13
14     fmt.Fprintf(writer, "<html><body><pre>")
15     fmt.Fprintln(writer, request.URL.Path)
16     fmt.Fprintf(writer, "%d visits\n", count)
17     fmt.Fprintf(writer, "</pre></body></html>")
18 }
```

Exercise

Write a web server that generates a web page with 3 random XKCD comics.



Channels and goroutines I

```
1  // full spec at https://api.stackexchange.com/docs/info
2  type Item struct {
3      Users     int `json:"total_users"`
4      Questions int `json:"total_questions"`
5      Answers   int `json:"total_answers"`
6      Comments  int `json:"total_comments"`
7  }
8
9  type Info struct {
10     Site  string
11     Items []Item
12 }
```

Channels and goroutines II

```
13  func FetchInfo(site string, infos chan *Info, errs chan error) {
14      url := "https://api.stackexchange.com/2.2/info?site=" + site
15      response, err := http.Get(url)
16      if err != nil {
17          errs <- err
18          return
19      }
20      defer response.Body.Close()
21      if response.StatusCode != http.StatusOK {
22          errs <- errors.New(response.Status)
23          return
24      }
25      var info Info
26      decoder := json.NewDecoder(response.Body)
27      if err := decoder.Decode(&info); err != nil {
28          errs <- err
29          return
30      }
31      info.Site = site
32      infos <- &info
33  }
```

Channels and goroutines III

```
34  func main() {
35      sites := [...]string{
36          "stackoverflow",
37          "serverfault",
38          "superuser"}
39
40      infos := make(chan *Info)
41      errs := make(chan error)
42
43      for _, site := range sites {
44          go FetchInfo(site, infos, errs)
45      }
46
47      for range sites {
48          select {
49          case info := <-infos:
50              fmt.Printf("%#v\n\n", info)
51          case err := <-errs:
52              fmt.Printf("%s\n\n", err)
53          }
54      }
```

Exercise

Have your web server fetch the random XKCD comics concurrently.

Circles and rectangles

```go
type Circle struct {
    Radius float64
}

type Rectangle struct {
    Width  float64
    Height float64
}

func areaCircle(circ *Circle) float64 {
    return math.Pi * circ.Radius * circ.Radius
}

// You need distinct function names for areas
// because Go does not support overloading
func areaRectangle(rect *Rectangle) float64 {
    return rect.Width * rect.Height
}

func main() {
    c := Circle{Radius: 2}
    r := Rectangle{Width: 16, Height: 9}
    fmt.Printf("%#v -> %f\n", c, areaCircle(&c))
    fmt.Printf("%#v -> %f\n", r, areaRectangle(&r))
}
```

Methods

```go
type Circle struct {
    Radius float64
}

type Rectangle struct {
    Width  float64
    Height float64
}

func (circ *Circle) Area() float64 {
    return math.Pi * circ.Radius * circ.Radius
}

// Methods have an additional receiver argument
// and can be overloaded by their receiver
func (rect *Rectangle) Area() float64 {
    return rect.Width * rect.Height
}

func main() {
    c := Circle{Radius: 2}
    r := Rectangle{Width: 16, Height: 9}
    fmt.Printf("%#v -> %f\n", c, c.Area())
    fmt.Printf("%#v -> %f\n", r, r.Area())
}
```

Interfaces I

```
1  type Shape interface {
2      Area() float64
3  }
4
5  type Circle struct {
6      Radius float64
7  }
8
9  type Rectangle struct {
10     Width  float64
11     Height float64
12 }
13
14 func (circ *Circle) Area() float64 {
15     return math.Pi * circ.Radius * circ.Radius
16 }
17
18 func (rect *Rectangle) Area() float64 {
19     return rect.Width * rect.Height
20 }
```

Interfaces II

```
21  // *Circle and *Rectangle implicitly implement Shape
22  // because they provide an Area() float64 method
23  func main() {
24      shapes := [...]Shape{
25          &Circle{Radius: 2},
26          &Rectangle{Width: 16, Height: 9},
27      }
28
29      for _, shape := range shapes {
30          fmt.Printf("%#v -> %f\n", shape, shape.Area())
31      }
32  }
```

Exercise

Implement 2 additional shapes: `Square` and `Triangle`.

Where to go

https://golang.org/doc/

https://gobyexample.com/

https://blog.golang.org/

https://forum.golangbridge.org/

https://groups.google.com/forum/#!forum/golang-nuts

https://stackoverflow.com/questions/tagged/go

https://www.reddit.com/r/golang/

http://www.gopl.io/

```
fmt.Println("Thanks for not falling asleep :)")
```