






























# Go

Ein Überblick

# Ablauf



























1. Einordnung
2. Berechtigung
3. Charakterisierung
4. Pro/Contra
5. Datentypen/Kontrollstrukturen
6. Nebenläufigkeit/Parallelität
7. Vergleich(e)

# 1. Einordnung

Language Rank	Types	Spectrum Ranking
1. C	  	100.0
2. Java	  	98.1
3. Python	 	97.9
4. C++	  	95.8
5. R		87.7
6. C#	  	86.4
7. PHP		82.4
8. JavaScript	 	81.9
9. Ruby	 	74.0
10. Go	 	71.5
11. Arduino		69.5
12. Matlab		68.7
13. Assembly		68.0
14. Swift	 	67.6

Top Programming Languages 2016 (spectrum)

# 1. Einordnung

Language Rank	Types	Spectrum Ranking
1. C	  	100.0
2. Java	  	98.1
3. Python	 	97.9
4. C++	  	95.8
5. R		87.7
6. C#	  	86.4
7. JavaScript	 	81.9
8. Ruby	 	74.0
9. Go	 	71.5
10. Arduino		69.5
11. Matlab		68.7
12. Assembly		68.0
13. Swift	 	67.6

Top Web Programming Languages (spectrum)

# 1. Einordnung

## Entwicklerpräferenzen von *stackoverflow*

Stand 18.08.2016

- 1 Rust: 79.1% (Mozilla)
- 2 Swift: 72.1% (Apple)
- 3 F#: 70.7%
- 4 Scala: 69.4%
- 5 Go: 68.7%
- 6 Clojure: 66.7%
- 7 React: 66.0%
- 8 Haskell: 64.7%
- 9 Python: 62.5%
- 10 C#: 62.0%
- 11 Node.js: 59.6%

# Ablauf

1. Einordnung
- 2. Berechtigung**
3. Charakterisierung
4. Pro/Contra
5. Datentypen/Kontrollstrukturen
6. Nebenläufigkeit/Parallelität
7. Vergleich(e)

## 2. Berechtigung

- entwickelt: Robert Griesemer, Rob Pike, Ken Thompson ab 2007 bei *google*
- public open source: Mitte November 2009
- Robert Griesemer auf der google i/o 2015:
  - „...clear goal in mind: we needed a better language ...“
  - „...wanted a clean, small, compiled language with modern features...“

# 2. Berechtigung

- Frustration mit existierenden Sprachen und Umgebungen für die Systementwicklung
- Effizientes Kompilieren - Effizientes Ausführen - Einfaches Programmieren gab es in keiner anderen der vorhanden Mainstream-Sprachen
- alle anderen wichtigen Sprachen waren zu dem Zeitpunkt älter als 10 Jahre
- andere Anforderungen: Parallelität, komplexe Frameworks, Netzwerkprogrammierung



# Ablauf

1. Einordnung
2. Berechtigung
- 3. Charakterisierung**
4. Pro/Contra
5. Datentypen/Kontrollstrukturen
6. Nebenläufigkeit/Parallelität
7. Vergleich(e)

# 3. Charakterisierung

imperativ

strukturiert

modular

objektorientiert

nebenläufig

# Ablauf

1. Einordnung
2. Berechtigung
3. Charakterisierung
- 4. Pro/Contra**
5. Datentypen/Kontrollstrukturen
6. Nebenläufigkeit/Parallelität
7. Vergleich(e)

# 4. Pros

## Minimalismus

- begrenzter Umfang an Sprachmitteln (eine Schleife, keine Klassen, keine Vererbung, keine generische Programmierung, keine Zeigerarithmetik, zwei Sichtbarkeitsebenen)
- Sprache ist einfach zu lernen
- Gleichförmigkeit des Quelltextes (erleichtert Wartung)

# 4. Pros

## Statisches Duck-Typing

- Schnittstellen werden nicht explizit implementiert
- wenn eine Komponente alle Methoden einer Schnittstelle enthält, erfüllt sie automatisch die Schnittstelle
- lose Kopplung
- führt tendenziell zu minimalen Schnittstellen

# 4. Pros

## Aufgeräumte Syntax

for.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     sum := 0
7     for i := 0; i < 60; i++ {
8         sum += i
9     }
10    fmt.Println(sum)
11 }
12
```

if.go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func sqrt(x float64) string {
9     if x < 0 {
10         return sqrt(-x) + "i"
11     }
12     return fmt.Sprintf(math.Sqrt(x))
13 }
14
15 func main() {
16     fmt.Println(sqrt(2), sqrt(-4))
17 }
18
```

# 4. Pros

## Aufgeräumte Syntax

- für lokale Variablen gibt es Typableitung
- Quelltext muss also nicht durch explizite Typangaben aufgebläht werden
- **`i := 42`**

# 4. Pros

## Schneller Compiler

- durch reduzierte Sprachmittel
- schnelle Quelltextübersetzung
- damit eignet sich Go für Einsatzbereiche, die eher Skriptsprachen vorbehalten sind



# 4. Pros

## Laufzeiteigenschaften

- kommt der Performance von z.B. Java ziemlich nahe
- deutlich geringerer Speicherbedarf
- am Ende mehr dazu

# 4. Pros

## Unit-Test-Framework

- Go bringt eigenes Framework mit
- ist schlicht, erfüllt aber seinen Zweck

**import „testing“**

# 4. Pros

## Paketmanager

- Go bringt eigenen Paketmanager mit
- URL einer externen Bibliothek
- allerdings recht primitiv, erlaubt keine Versionseinschränkungen

**package main**

# 4. Pros

## Playground

The Go Playground Run Format Imports Share

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     fmt.Println("Hello, playground")
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

Hello, playground

Program exited.

# 4. Pros

## Garbage Collection

- automatische Speicherbereinigung
- ein Programm sucht im Hintergrund nach nicht mehr referenziertem Speicher und gibt diesen automatisch wieder frei
- meist höherer Ressourcenverbrauch, im Vergleich zu anderen GCs aber besser

# 4. Contras

## **Starke Typisierung, keine generische Programmierung**

- Mehrfachverwendung von Code ist eingeschränkt
- Umgehungslösung wäre Implementierung für alle benötigten Datentypen
- ist weniger wohlstrukturiert und weniger elegant
- Go-typische Alternative wäre for-Schleife, bläht aber den Code auf

# 4. Contras

## wenig grundlegende Datenstrukturen

- Beschränkung auf Arrays, Slices, Maps
- *set* muss z.B. durch *map* improvisiert werden

# 4. Contras

## umständliches Mocking

- Go erlaubt es nicht, einen Mock zur Laufzeit aus einer Schnittstelle heraus zu erzeugen und einzelne Methoden zu implementieren
- man muss eigene Mock-Implementierung programmieren
- Unittests werden so lang und umständlich



# 4. Contras

## kleines Ökosystem

- Texteditor funktioniert zwar
- kaum gute Entwicklungsumgebungen (mittlerweile gibt es einige, z.B. Komodo IDE, Eclipse)

# 4. Contras

## **nicht wiederholbare Builds**

- es ist nicht möglich ein fremdes Paket auf eine bestimmte Version oder einen Versionsbereich festzulegen
- man kann sich nie sicher sein, welche Version welcher Bibliothek tatsächlich verwendet wird

# Ablauf

1. Einordnung
2. Berechtigung
3. Charakterisierung
4. Pro/Contra
- 5. Datentypen/Kontrollstrukturen**
6. Nebenläufigkeit/Parallelität
7. Vergleich(e)

# 5. Datentypen

## Slice

- dynamische Größe
- flexibler Blick in die Elemente eines Arrays
- es beschreibt im Prinzip ein darunter liegendes Array

```
slices.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     primes := [6]int{2, 3, 5, 7, 11, 13}
7
8     var s []int = primes[0:3]
9     fmt.Println(s)
10 }
11
```

[2 3 5]

# 5. Datentypen

## Slice

slices-pointers.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     names := [4]string{
7         "John",
8         "Paul",
9         "George",
10        "Ringo",
11    }
12    fmt.Println(names)
13
14    a := names[0:2]
15    b := names[1:3]
16    fmt.Println(a, b)
17
18    b[0] = "XXX"
19    fmt.Println(a, b)
20    fmt.Println(names)
21 }
22
```

```
[John Paul George Ringo]
[John Paul] [Paul George]
[John XXX] [XXX George]
[John XXX George Ringo]
```

slice-bounds.go

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s := []int{2, 3, 5, 7, 11, 13}
7
8     s = s[1:4]
9     fmt.Println(s)
10
11    s = s[:2]
12    fmt.Println(s)
13
14    s = s[1:]
15    fmt.Println(s)
16 }
17
```

```
[3 5 7]
[3 5]
[5]
```

# 5. Kontrollstrukturen

## Switch

```
if i == 0 {  
    fmt.Println("Zero")  
} else if i == 1 {  
    fmt.Println("One")  
} else if i == 2 {  
    fmt.Println("Two")  
} else if i == 3 {  
    fmt.Println("Three")  
} else if i == 4 {  
    fmt.Println("Four")  
} else if i == 5 {  
    fmt.Println("Five")  
}
```

```
switch i {  
case 0: fmt.Println("Zero")  
case 1: fmt.Println("One")  
case 2: fmt.Println("Two")  
case 3: fmt.Println("Three")  
case 4: fmt.Println("Four")  
case 5: fmt.Println("Five")  
default: fmt.Println("Unknown Number")  
}
```

# Ablauf

1. Einordnung
2. Berechtigung
3. Charakterisierung
4. Pro/Contra
5. Datentypen/Kontrollstrukturen
- 6. Nebenläufigkeit/Parallelität**
7. Vergleich(e)

# 6. Nebenläufigkeit/Parallelität

## Nebenläufigkeit

- Komposition eigenständig ausführbarer Vorgänge
- eigenständige Abläufe können sich überlappen
- Strukturierung einer Problemlösung durch Modularisierung



# 6. Nebenläufigkeit/Parallelität

## Parallelität

- simultane Ausführung von Vorgängen
- sehr ähnliche Abläufe und stark aneinander gekoppelt
- beschleunigte Ausführung durch Aufteilung des Rechenaufwandes

# 6. Nebenläufigkeit/Parallelität

- Shared Memory
- Problem des Data Race
- Message Passing

# 6. Nebenläufigkeit/Parallelität

## Goroutines

```
func main() {  
    printMe("Synchroner Peter", 5)  
  
    go printMe("Asynchroner Markus", 5)  
    go printMe("Asynchrone Anna", 5)  
  
    printMe("Synchoner Fabian", 5)  
  
    var input string  
    fmt.Scanln(&input)  
}  
  
func printMe(text string, times int) {  
    for i := 0; i < times; i++ {  
        time.Sleep(1000)  
        fmt.Println(text, ":", i)  
    }  
}
```

```
Synchroner Peter : 0  
Synchroner Peter : 1  
Synchroner Peter : 2  
Synchroner Peter : 3  
Synchroner Peter : 4  
Asynchroner Markus : 0  
Synchoner Fabian : 0  
Asynchrone Anna : 0  
Synchoner Fabian : 1  
Asynchroner Markus : 1  
Asynchrone Anna : 1  
Asynchroner Markus : 2  
Synchoner Fabian : 2  
Asynchrone Anna : 2  
Synchoner Fabian : 3  
Asynchroner Markus : 3  
Asynchrone Anna : 3  
Asynchroner Markus : 4  
Synchoner Fabian : 4
```

# 6. Nebenläufigkeit/Parallelität

## Channels

```
func main() {  
    fruits := make(chan string)  
  
    go func() {  
        fruits <- "Apple"  
        time.Sleep(3 * time.Second)  
        fruits <- "Banana"  
        fruits <- "Orange"  
    } ()  
  
    fruit := <-fruits  
    fmt.Println(fruit)  
  
    fruit = <-fruits  
    fmt.Println(fruit)  
  
    time.Sleep(3 * time.Second)  
  
    fruit = <-fruits  
    fmt.Println(fruit)  
}
```

**Channel erzeugen**

**Werte in den  
Channel senden**

**Werte aus dem  
Channel empfangen**

# 6. Nebenläufigkeit/Parallelität

## Select

```
func main() {  
    chanApple := make(chan string)  
    chanBanana := make(chan string)  
  
    go func() {  
        for {  
            time.Sleep(time.Second * 1)  
            chanApple <- "Apple"  
        }  
    }()  
  
    go func() {  
        for {  
            time.Sleep(time.Second * 2)  
            chanBanana <- "Banana"  
        }  
    }()  
}
```

```
    for {  
        select {  
            case apple := <-chanApple:  
                fmt.Println("Fruit: ", apple)  
            case banana := <-chanBanana:  
                fmt.Println("Fruit: ", banana)  
        }  
    }  
}
```

## 6. Nebenläufigkeit/Parallelität

# Worker Pools

```
func picker(fruit string, fruits chan<- string, timeToPick int) {  
    for {  
        fmt.Println("Pick ", fruit)  
        fruits <- fruit  
        time.Sleep(time.Duration(timeToPick) * time.Second)  
    }  
}
```

## 6. Nebenläufigkeit/Parallelität

# Worker Pools

```
func eater(i int, fruits <-chan string) {  
    for fruit := range fruits {  
        fmt.Println("Esser ", i, "Mhmmm ", fruit)  
        time.Sleep(time.Second * 8)  
    }  
}
```

## 6. Nebenläufigkeit/Parallelität

# Worker Pools

```
func main() {  
    fruits := make(chan string, 1000)  
  
    for i := 1; i <= 10; i++ {  
        time.Sleep(time.Millisecond * 600)  
        go eater(i, fruits)  
    }  
  
    go picker("Apple", fruits, 5)  
    time.Sleep(time.Millisecond * 600)  
    go picker("Orange", fruits, 3)  
    time.Sleep(time.Millisecond * 600)  
    go picker("Strawberry", fruits, 2)  
  
    time.Sleep(time.Second * 360)  
}
```



# Ablauf

1. Einordnung
2. Berechtigung
3. Charakterisierung
4. Pro/Contra
5. Datentypen/Kontrollstrukturen
6. Nebenläufigkeit/Parallelität
- 7. Vergleich(e)**

# 7. Vergleiche

(Python 3, Go 1.8.1, C 6.3, C++ 6.3, Java 1.8 / 64bit  
Quadcore - Ubuntu)

reverse-complement

**Go 0.49**

Java 1.10

Node.js 3,72

C++ 0.60

C 0.42

Python 2.82

---

ATGCGCTAGCTCATT  
TATCGGATCGAATTA

---

# 7. Vergleiche

(Python 3, Go 1.8.1, C 6.3, C++ 6.3, Java 1.8 / 64bit  
Quadcore - Ubuntu)

pidigits

**Go 2.02**

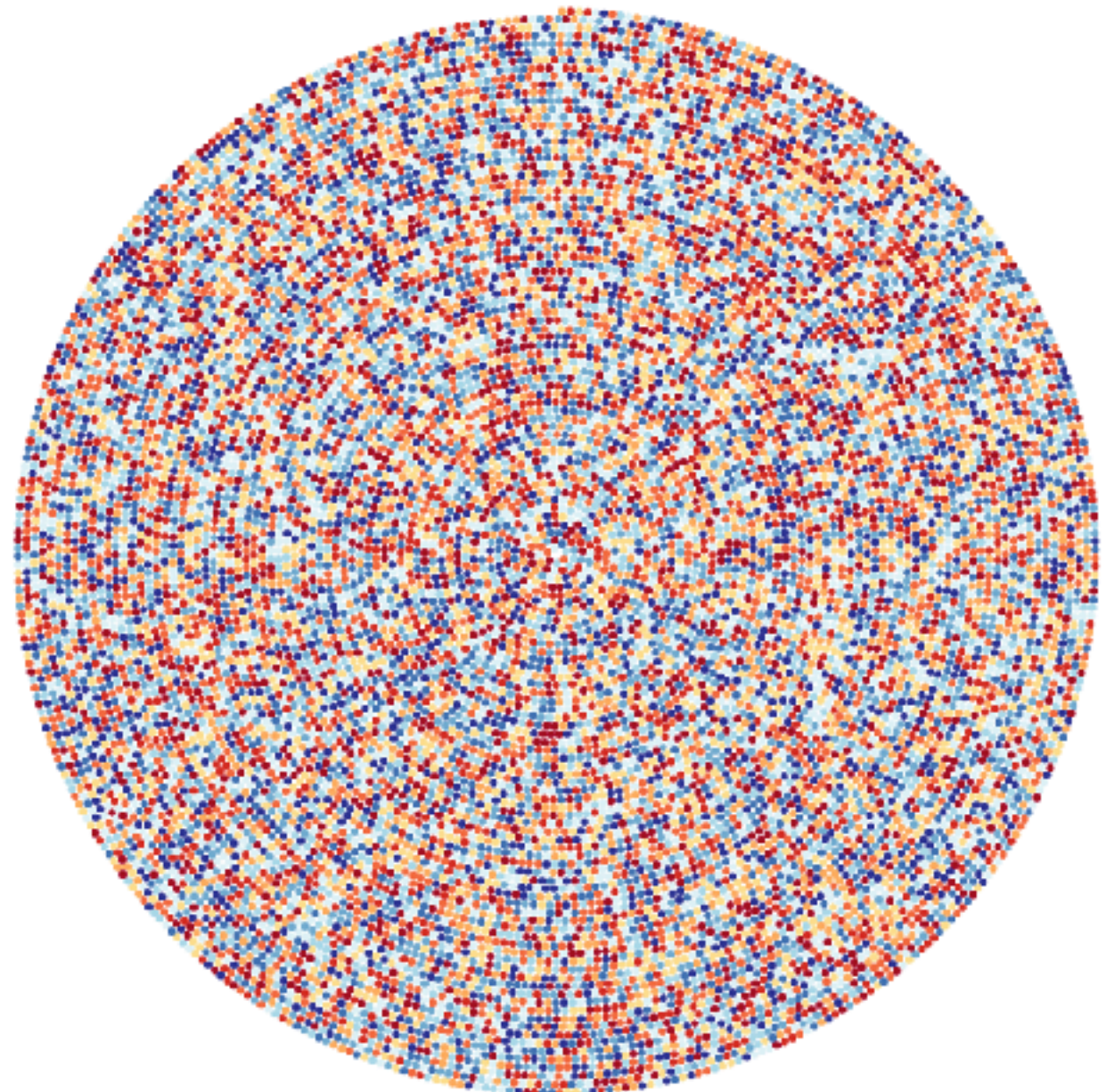
Java 3.06

Node.js Bad Output

C++ 1.89

C 1.73

Python Failed



# 7. Vergleiche

(Python 3, Go 1.8.1, C 6.3, C++ 6.3, Java 1.8 / 64bit  
Quadcore - Ubuntu)

## Mandelbrot

**Go 5.64**

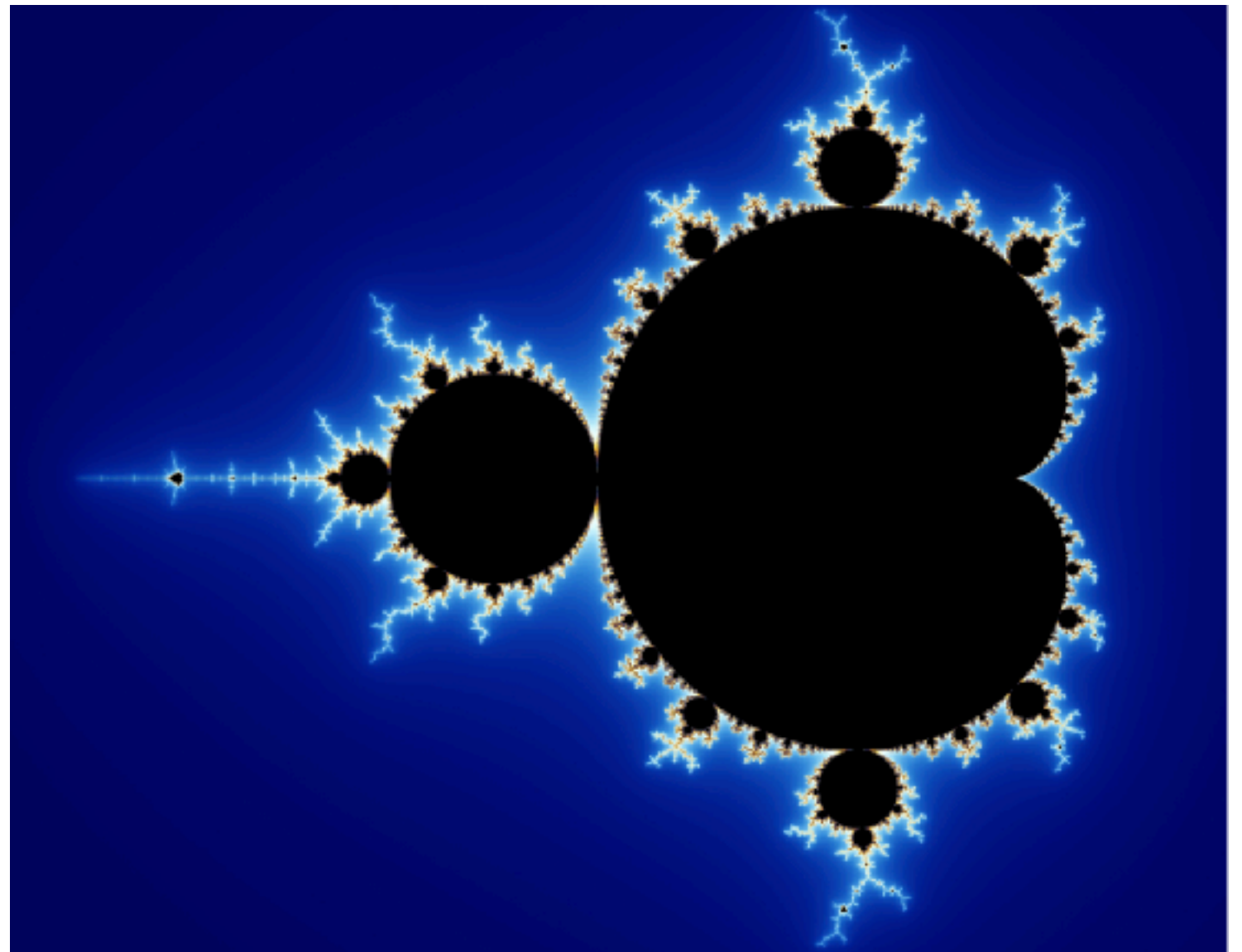
Java 7.10

Node.js 17.95

C++ 1.73

C 1.65

Python 273.43



# 7. Vergleiche

(Python 3, Go 1.8.1, C 6.3, C++ 6.3, Java 1.8 / 64bit  
Quadcore - Ubuntu)

fannkuch-redux

**Go 15.81**

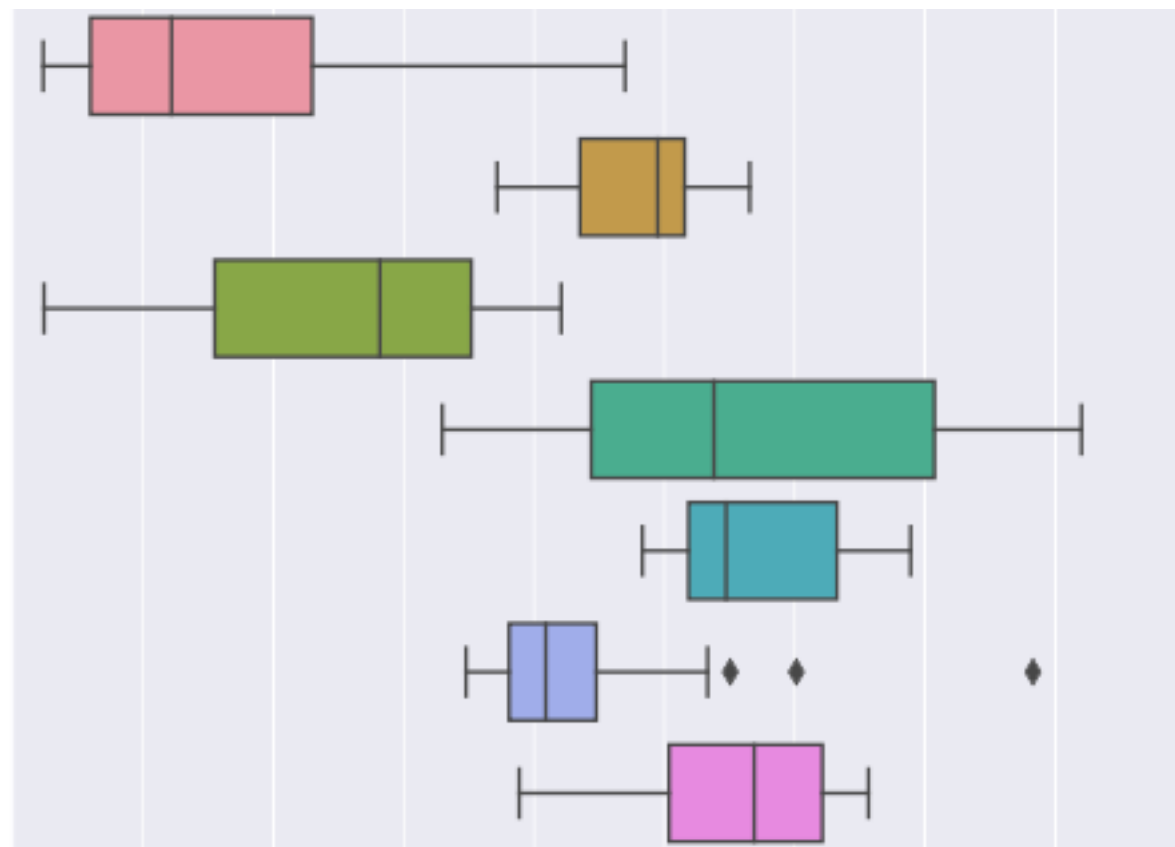
Java 13.74

Node.js 78.16

C++ 10.35

C 8.97

Python 483.79





# 7. Vergleiche

```
test.c x test.go x
1 package main
2 import "fmt"
3
4 func main() {
5     var sum int64 = 0
6
7     for e := 0; e < 200; e++ {
8         sum = 0
9
10        var x []int64
11        for i := 0; i < 1000000; i++ {
12            x = append(x, int64(i));
13        }
14
15        var y []int64
16        for i := 0; i < 1000000-1; i++ {
17            y = append(y, x[i]+x[i+1]);
18        }
19
20        for i := 0; i < 1000000; i += 100 {
21            sum += y[i];
22        }
23    }
24
25    fmt.Println(sum);
26 }
27 }
```

```
test.c x test.go
1 #include <vector>
2 #include <iostream>
3
4 int main() {
5     int64_t sum = 0;
6
7     for(int e = 0; e < 200; e++) {
8         sum = 0;
9
10        std::vector<int64_t> x;
11        for(int i = 0; i < 1000000; i++) {
12            x.push_back(i);
13        }
14
15        std::vector<int64_t> y;
16        for(int i = 0; i < 1000000-1; i++) {
17            y.push_back(x[i] + x[i+1]);
18        }
19
20        for (int i = 0; i < 1000000; i += 100) {
21            sum += y[i];
22        }
23    }
24
25    std::cout << sum << std::endl;
26 }
```

```
var sum : Int64 = 0

for e in 0..<200 {
    sum = 0

    var x : [Int64] = []
    for (var i = 0; i < 1000000; i++) {
        x.append(Int64(i));
    }

    var y: [Int64] = []
    for (var i = 0; i < 1000000-1; i++) {
        y.append(x[i] + x[i+1]);
    }

    for (var i = 0; i < 1000000; i+=100) {
        sum += y[i]
    }

    print(sum)
}
```

# 7. Vergleiche

Macbook Air (1.7 GHz Intel Core i5), Go 1.5.3, Swift 2.1.1, C++ clang-700.1.81.

```
go build -o test-go go/test.go
swiftc -O -o test-swift swift/test.swift
c++ -O3 -o test-c cplusplus/test.cc
GOMAXPROCS=2 time ./test-go
9999010000
```

4.32 real	5.99 user	0.38 sys
-----------	-----------	----------

```
GOMAXPROCS=1 time ./test-go
9999010000
```

3.93 real	3.89 user	0.05 sys
-----------	-----------	----------

```
time ./test-swift
9999010000
```

3.96 real	2.82 user	1.12 sys
-----------	-----------	----------

```
time ./test-c
9999010000
```

2.37 real	1.74 user	0.62 sys
-----------	-----------	----------

```
go build -o test-go go/test.go
swiftc -O -o test-swift swift/test.swift
c++ -O3 -o test-c cplusplus/test.cc
GOMAXPROCS=2 time ./test-go
99901000
```

4.40 real	5.65 user	1.10 sys
-----------	-----------	----------

```
GOMAXPROCS=1 time ./test-go
99901000
```

5.06 real	5.07 user	0.83 sys
-----------	-----------	----------

```
time ./test-swift
99901000
```

2.32 real	2.29 user	0.01 sys
-----------	-----------	----------

```
time ./test-c
99901000
```

1.39 real	1.37 user	0.01 sys
-----------	-----------	----------

# Danke!





# Quellen

- <https://making.pusher.com/golangs-real-time-gc-in-theory-and-practice/>
- [https://books.google.de/books?id=BZncDgAAQBAJ&pg=PA25&lpg=PA25&dq=toolchain+go&source=bl&ots=ULJx3ioZan&sig=BKbzU6DTY1TfFLaTu7P3nsKNSrw&hl=de&sa=X&ved=0ahUKEwibovfnidHUAhXHPFAKHbd\\_BmcQ6AEIZDAI#v=onepage&q=toolchain%20go&f=false](https://books.google.de/books?id=BZncDgAAQBAJ&pg=PA25&lpg=PA25&dq=toolchain+go&source=bl&ots=ULJx3ioZan&sig=BKbzU6DTY1TfFLaTu7P3nsKNSrw&hl=de&sa=X&ved=0ahUKEwibovfnidHUAhXHPFAKHbd_BmcQ6AEIZDAI#v=onepage&q=toolchain%20go&f=false)
- <http://lionet.livejournal.com/137511.html>
- <https://astaxie.gitbooks.io/build-web-application-with-golang/de/01.1.html>
- <http://www.golangbootcamp.com/book>
- <https://gowebexamples.github.io/templates/>
- [https://www.fernuni-hagen.de/imperia/md/content/ps/lehrveranstaltungen/masterarbeit\\_isensee\\_bastian.pdf](https://www.fernuni-hagen.de/imperia/md/content/ps/lehrveranstaltungen/masterarbeit_isensee_bastian.pdf)
- <http://benchmarksgame.alioth.debian.org>
- [http://www.bitloeffel.de/DOC/golang/code\\_de.html](http://www.bitloeffel.de/DOC/golang/code_de.html)
- <https://play.golang.org>
- <https://tour.golang.org/welcome/1>
- <https://alexsuter.gitbooks.io/golang-experience/content/concurrency.html>