Funktionen (in Go)

Modul 346, BBZW

Patrick Bucher

30.11.2023

Just a function...

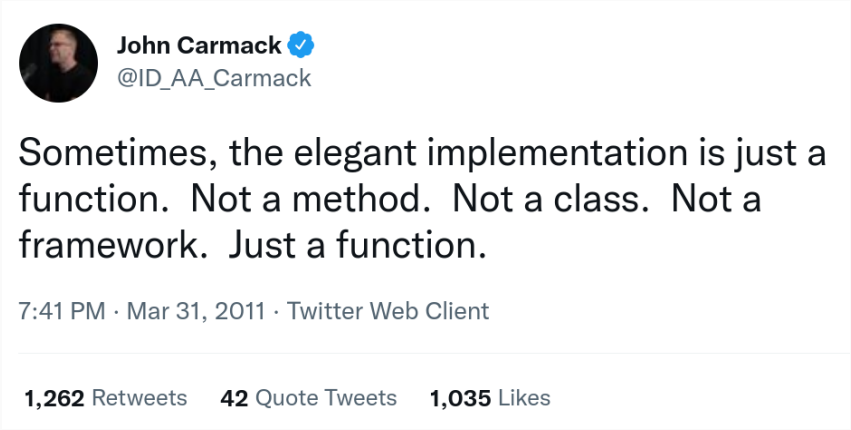
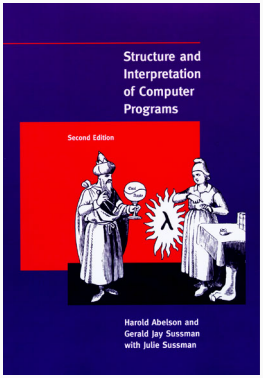


Abbildung 1: John Carmack über Funktionen

“Just” a function…?

Wenn Sie es genau wissen wollen:



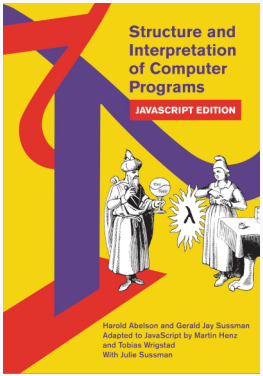


Abbildung 2: *Structure and Interpretation of Computer Programs* (links: Scheme-Ausgabe von 1984/1996, 688 Seiten; rechts: JavaScript-Ausgabe von 2022, 640 Seiten)

Ein verwandtes Konzept: Cloud Functions

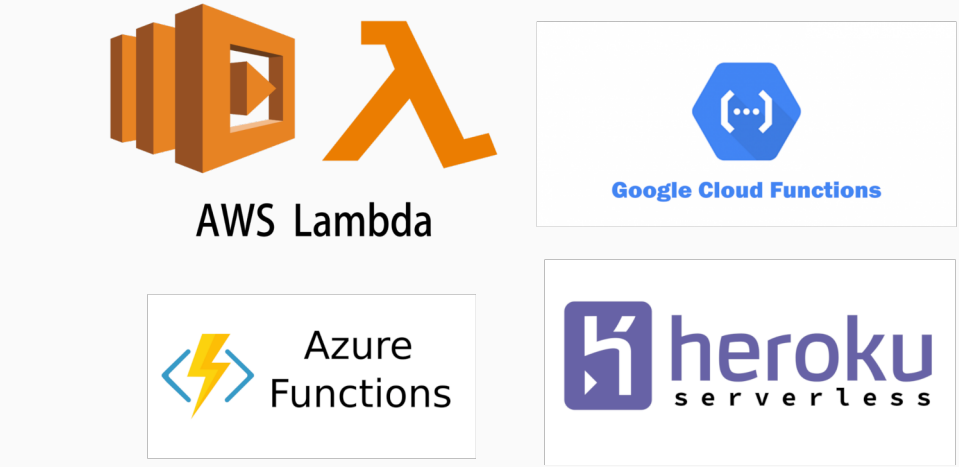


Abbildung 3: “serverless” Computing in der Cloud (im Uhrzeigersinn): AWS Lambda, Google Cloud Function, heroku serverless, Azure Functions

Funktionen in der Mathematik (I)

Definition:

$$y = f(x) = 2x$$

Wertetabelle:

x	y
-2	-4
-1	-2
0	0
1	2
2	4

Funktionen in der Mathematik (II)

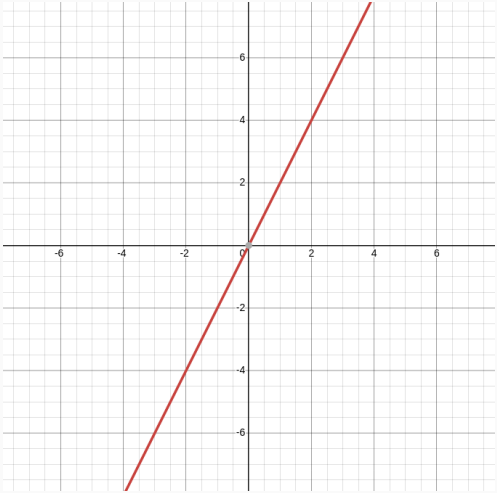


Abbildung 4: Die Funktion $y = f(x) = 2x$ als Plot

Funktionen in Go (I): Definition

Die Funktion $y = f(x) = 2x$ in Go:

```
func f(x int) int {
    y := 2 * x
    return y
}
f(2) // 4
```

Oder:

```
func f(x int) int {
    return 2 * x
}
f(-2) // -4
```

Funktionen in Go (II): Aufruf

Erweiterter Aufruf (zur Erzeugung der Wertetabelle):

```
xs := []int{-2, -1, 0, 1, 2}
for _, x := range xs {
    y := f(x)
    fmt.Printf("y=f(%d)=%d\n", x, y)
}
```

Ausgabe:

```
y=f(-2)=-4
y=f(-1)=-2
y=f(0)=0
y=f(1)=2
y=f(2)=4
```

Funktionen in Go (III): Ohne Parameter, ohne Rückgabewert

```
func sayHello() {
    fmt.Println("Hello")
}
sayHello()
```

Ausgabe:

```
Hello
```

Funktionen in Go (IV): Mit Parameter, ohne Rückgabewert

```
func sayHelloTo(whom string) {
    fmt.Println("Hello,", whom)
}
sayHelloTo("Alice")
```

Ausgabe:

```
Hello, Alice
```

Funktionen in Go (V): Mehrere Parameter, ohne Rückgabewert

```
func outputCurrency(amount float32, currency rune) {
    fmt.Printf("%8.2f %c\n", amount, currency)
}
outputCurrency(2.5, '$')
outputCurrency(10.0/3.0, '€')
outputCurrency(1234.567, '¥')
```

Ausgabe:

```
    2.50 $
    3.33 €
 1234.57 ¥
```

Funktionen in Go (VI): Mehrere Parameter, mit Rückgabewert

```
func formatCurrency(amount float32, currency rune) string {
    return fmt.Sprintf("%8.2f %c", amount, currency)
}
dollars := formatCurrency(2.5, '$')
euros := formatCurrency(10.0/3.0, '€')
fmt.Println(dollars)
fmt.Println(euros)
```

Ausgabe:

```
    2.50 $
    3.33 €
```

Funktionen in Go (VII): Keine Parameter, mit Rückgabewert

```
func rollDice() int {
    return rand.Intn(6) + 1
}
rand.Seed(time.Now().Unix)
fmt.Println(rollDice())
fmt.Println(rollDice())
fmt.Println(rollDice())
```

Ausgabe (nicht deterministisch):

```
1
2
5
```

Funktionen in Go (VIII): Zwei Parameter, mehrere Rückgabewerte

```
func divide(dividend, divisor float32) (float32, error) {
    if divisor == 0.0 {
        return 0.0, errors.New("divide by 0")
    }
    return dividend / divisor, nil
}
fmt.Println(divide(10.0, 3.0))
fmt.Println(divide(10.0, 0.0))
```

Ausgabe:

```
3.3333333 <nil>
0 divide by 0
```

Funktionen in Go (IX): Fehlerbehandlung I

Diese Funktion gibt möglicherweise einen Fehler zurück…:

```
func computeAverage(values []float32) (float32, error) {
    if len(values) == 0 {
        return 0.0, fmt.Errorf("cannot compute average of %v", values)
    }
    var sum float32
    for _, value := range values {
        sum += float32(value)
    }
    return sum / float32(len(values)), nil
}
```

Funktionen in Go (X): Fehlerbehandlung II

...worauf der Aufrufer reagieren muss:

```
grades := makeRandomGrades() // returns 0..2 grades
average, err := computeAverage(grades)
if err != nil {
    fmt.Fprintf(os.Stderr, "compute average of %v: %v\n", grades, err)
} else {
    fmt.Printf("the average of %v is %.2f\n", grades, average)
}
```

Ausgabe:

```
compute average of []: cannot compute average of []
the average of [2.41 5.07] is 3.74
```

Methoden (I): Ist Go objektorientiert?

> *Question:* ***Is Go an object-oriented language?***
> *Answer: Yes and no. Although Go has types and methods and allows an object-oriented style of programming, there is no type hierarchy. […]*

Volle Antwort: Go FAQ

Go hat keine Klassen und keine Vererbung, Funktionen können aber als *Methoden* implementiert werden.

Methoden (II): Neuer Typ mit Funktion

Funktionen können Parameter beliebiger Typen erwarten:

```
type Celsius float32

func outputCelsius(c Celsius) {
    fmt.Printf("%.2f°C\n", c)
}

var coldest Celsius = -273.15
var warm Celsius = 32.5

outputCelsius(coldest) // -273.15°C
outputCelsius(warm) // 32.50°C
```

Methoden (III): Neuer Typ mit Methode

Funktionen können auch an genau einen Typen “angehängt” werden:

```
type Celsius float32

func (c Celsius) Output() {
    fmt.Printf("%.2f°C\n", c)
}

var coldest Celsius = -273.15
var warm Celsius = 32.5

coldest.Output() // -273.15°C
warm.Output() // 32.50°C
```