Theorie-Programmieraufgaben

22. November 2023

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo1() {
    x := 8
    y := 7
    k := x + 2*y - 2

    fmt.Println(x, y, k)
    fmt.Printf("%d-%d-%d\n", x, y, k)
    fmt.Println(k + 2 - x)
    fmt.Println(float64(k) / float64(x))
}
```

Lösung

```
    // Output:
    // 8 7 20
    // 8-7-20
    // 14
    // 2.5
```

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo2(n int) {
    s := "*"
    for i := 0; i < n; i++ {
        fmt.Println(s)
        s += s
    }
}
```

Lösung

```
    // Output:
    // *
    // **
    // ****
    //
    // *
    // **
    // ****
    // ********
    //
    // *
    // **
    // ****
    // ********
    // ****************
```

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo3(n int) {
    if n == 0 {
        return
    }
    Foo3(n - 1)
    fmt.Println(strings.Repeat("*", n))
}
```

Lösung

```
    // Output:
    // *
    // **
    // ***
    //
    // *
    // **
    // ***
    // ****
    //
    // *
    // **
    // ***
    // ****
    // *****
```

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo4(n int) {
    fmt.Print(n)
    n = 42
    fmt.Println(n)
}
```

Lösung

```
    // Output:
    // 342
    // 442
    // 542
```

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo5() {
    n := 23
    Foo4(n)
    fmt.Println(n)
}
```

Lösung

```
    // Output:
    // 2342
    // 23
```

Aufgabe: Ausgabe von Programmen

Was gibt diese Funktion auf die Konsole aus?

```
func Foo6() {
    x := 1
    {
        x := 2
        {
            x := 3
            fmt.Println(x)
        }
        fmt.Println(x)
    }
    fmt.Println(x)
}
```

Lösung

```
    // Output:
    // 3
    // 2
    // 1
```

Aufgabe: Ausgabe von Programmen

Was gibt dieses Programm auf die Konsole aus?

```
func Foo() {
    x := 3
    y := 4
    fmt.Println(x, y)
    x = Bar(x, y)
    fmt.Println(x, y)
    Baz(x, y)
    fmt.Println(x, y)
}

// Bar ist eine Hilfsfunktion für Foo.
func Bar(x, y int) int {
    x = y - x
    return x
}

// Baz ist eine Hilfsfunktion für Foo.
func Baz(x, y int) {
    x, y = y, x
    fmt.Println(x, y)
}
```

Lösung

```
    // Output:
    // 3 4
    // 1 4
    // 4 1
    // 1 4
```

Aufgabe: Syntaxfehler

Welche Typen haben die Variablen `x`, `y`, `z` und `w`?

```
x := 42
y := x - 2
z := 3.14
w := "Hello"
```

Lösung

```
// Output:
// int int float64 string
```

Aufgabe: Syntaxfehler

Welche Typen haben die Variablen `x`, `y` und `z`?

```
x := Foo1()
y := Foo2()
z := Foo4(x)
z = z / 2.3
if y {
    x += Foo3(z)
}
x += 3
```

Lösung

```
// Output:
// int bool float64
```

Aufgabe: Fehlersuche - Inhaltliche Fehler

Diese Funktion für einen Primzahltest ist fehlerhaft. Was ist falsch?

```
1  func IsPrimeBuggy(n int) bool {
2      for i := 2; i < n-1; i++ {
3          if n%i == 0 {
4              return false
5          } else {
6              return true
7          }
8      }
9      return true
10 }
```

Lösung

```
1  func IsPrimeCorrect(n int) bool {
2      //* Fehler 1: Spezialfall hat gefehlt, 0 und 1
           sind keine Primzahlen
3      if n <= 1 {
4          return false
5      }
6      // Im
7      for i := 2; i < n-1; i++ {
8          if n%i == 0 {
9              return false
10         }
11         //* Fehler 2: Der else-Zweig war falsch: Wenn
               n%i != 0,
12         //* dann ist n nicht automatisch prim, denn
               es kann immer
13         //*  noch ein anderer Teiler gefunden werden.
14     }
15     return true
16 }
```

Aufgabe: Syntaxfehler

Finden Sie alle Fehler in den folgenden Variablendefinitionen.

```
1     x := 42
2     var int y 55
3     int z = 42
4     s := string([]byte{'a', 'b', 'c'})
5     b := []byte{'a', 'b', 'c'}
6     var l1 []int := make([]int, 0)
7     string := hallo
```

Lösung

```
1     x := 42
2     var y int = 55 //* var int y 55
3     var z int = 77 //* int z = 42
4     s := string([]byte{'a', 'b', 'c'})
5     b := []byte{'a', 'b', 'c'}
6     l := make([]int, 0) //* var l1 []int :=
          make([]int, 0)
7     string := "hallo"   //* string := hallo
```

Aufgabe: Syntaxfehler

Finden Sie alle Fehler in diesem Programm.

```
0  package foo
1
2  import "fmt"
3
4  func PrintSomething(what string) string {
5      fmt.Print(what)
6      fmt.Print("\n")
7  }
8
9  func ComputeProduct(numbers int) int {
10     result := 1
11     for _, num := range numbers {
12         result *= num
13     }
14     return result
15 }
16
17 func main() {
18     p = ComputeProduct(1, 3, 5, 2, 0, 2)
19     PrintSomething(string fmt.Sprint(p))
20 }
```

Lösung

```
0  // Package muss main sein, weil es eine main-Funktion
       gibt.
1  package main
2
3  import "fmt"
4
5  // Funktion hat Return-Typ, aber kein return.
6  func PrintSomething(what string) {
7      fmt.Print(what)
```

```
8       fmt.Print("\n")
9   }
10
11  // numbers ist int, unten wird aber iteriert.
12  func ComputeProduct(numbers ...int) int {
13      result := 1
14      // (s.o.) Schleife mit int geht nicht.
15      for _, num := range numbers {
16          result *= num
17      }
18      return result
19  }
20
21  func main() {
22      // Neue Variable nicht mit = definieren.
23      p := ComputeProduct(1, 3, 5, 2, 0, 2)
24      // Typ nicht beim Aufruf mit angeben.
25      PrintSomething(fmt.Sprint(p))
26  }
```

Aufgabe: Signaturen

Welche Signaturen haben die Funktionen `Foo1`, `Foo2` etc.?

```
x1 := Foo1("Hallo", 15)
x2 := Foo2(x1)
x3 := Foo3(127, x2)
if x2 {
    x3 = append(x3, Foo1("Welt", x1))
}
x1 += Foo4(x2, true)
Foo5(x2 && x1 != Foo4(x2, x2))
return x2 && !(x1 > x3[0])
```

Hinweis: Die Signatur einer Funktion gibt an, welche Parameter sie erwartet und welchen Typ der Rückgabewert hat.

Lösung

```
// Output:
// Foo1: func(string, int) int
// Foo2: func(int) bool
// Foo3: func(int, bool) []int
// Foo4: func(bool, bool) int
// Foo5: func(bool)
```

Aufgabe: Signaturen

Welche Signaturen haben die Funktionen `Foo1`, `Foo2` etc.?

```
x1 := Foo1(42)
Foo2(x1)
s1 := []string{"Hallo"}
s2 := []string{"Welt"}
if Foo3(s1) {
    x1 = Foo4(Foo3(s2), 15)
}
e1, e2 := Foo5(s1[0])
fmt.Printf("%d\n", x1+e1+e2)
```

Hinweis: Die Signatur einer Funktion gibt an, welche Parameter sie erwartet und welchen Typ der Rückgabewert hat.

Lösung

```
// Output:
// Foo1: func(int) int
// Foo2: func(int)
// Foo3: func([]string) bool
// Foo4: func(bool, int) int
// Foo5: func(string) (int, int)
```