Aufgabe 1 (Grundlagen) [10 Punkte]

Bemerkung: Alle Aufgaben in diesem Abschnitt sind Multiple-Choice-Aufgaben, die sich auf die Grundlagen der Programmiersprache Go beziehen. Korrekt angekreuzte Zeilen in den Antworttabellen geben einen Punkt, für falsch angekreuzte Zeilen wird ein Punkt abgezogen.

a) Welche der folgenden Behauptungen sind wahr, welche falsch?

Behauptung	wahr	falsch
`int` ist ein Datentyp in Go.		
`float` ist ein Datentyp in Go.		
Jede Go-Quelldatei muss eine `main`-Funktion enthalten.		
Bei Variablen muss der Datentyp immer explizit angegeben werden.		

b) Welche der folgenden Variablendeklarationen sind korrekt, welche nicht?

Deklaration	korrekt	falsch
`var x1 int`		
`var x2 = 42`		
`var x3 string = "42"`		
`int x4`		
`x5 int := 42`		

Aufgabe 2 (Code-Verständnis) **[15 Punkte]**

Betrachten Sie den folgenden Code:

```
1  func Foo() {
2      x := 5
3      y := 38
4      y = Bar(x) + y
5      fmt.Println(x)
6      fmt.Println(y)
7  }
8
9  func Bar(x int) int {
10     fmt.Println(x)
11     for i := 0; i < x; i++ {
12         fmt.Printf("%vX", i)
13     }
14     x += 42
15     return x
16 }
```

Was gibt ein Aufruf von `Foo()` auf der Konsole aus?

Aufgabe 3 (Syntax von Programmen) **[15 Punkte]**

Betrachten Sie das folgende Programm:

```
1  package foo
2
3  import "fmt"
4
5  func PrintSomething(what string) string {
6      fmt.Print(what)
7      fmt.Print("\n")
8  }
9
10 func ComputeProduct(numbers int) int {
11     result := 1
12     for _, num := range numbers {
13         result *= num
14     }
15     return result
16 }
17
18 func main() {
19     p = ComputeProduct(1, 3, 5, 2, 0, 2)
20     PrintSomething(string fmt.Sprint(p))
21 }
```

Dieses Progamm enthält eine Reihe an Syntaxfehlern, durch die es nicht compiliert. Markieren Sie alle Zeilen, die einen Fehler enthalten und erläutern Sie kurz, was jeweils falsch ist.

Hinweis: Es geht hier nicht um inhaltliche Fehler, nur um Syntaxfehler.

Bemerkung: Für jede falsch markierte Zeile gibt es Punktabzug!

Aufgabe 4 (Datentypen) **[15 Punkte]**

Betachten Sie den folgenden Code:

```
1    x := Foo1()
2    k := Foo2([]int{x})
3    m := Foo5(42.0)
4    Foo3(m)
5    if k || m != fmt.Sprintf("%v", k) {
6        x += Foo4(!(k && x > 42))
7    }
```

Bestimmen Sie die Signaturen der Funktionen `Foo1` bis `Foo5`.

Funktion	**Signatur**
`Foo1`	
`Foo2`	
`Foo3`	
`Foo4`	
`Foo5`	

Aufgabe 5 (Debugging) **[15 Punkte]**

Die folgende Funktion ist zwar syntaktisch korrekt, sie erfüllt aber nicht ihre Aufgabe. Erläutern Sie den/die Fehler und machen Sie einen Vorschlag zur Korrektur.

```
1  package bug
2
3  // Contains liefert true, falls die Liste die Zahl x genau n
       mal enthaelt.
4  func Contains(list []int, x, n int) bool {
5      counter := 0
6      for el := range list {
7          if el == x {
8              el = counter + 1
9              if counter == n {
10                 return true
11             }
12         }
13     }
14     return false
15 }
```

Bemerkung: Ihr Korrekturvorschlag muss kein syntaktisch korrekter Code sein. Eine Erklärung in Worten genügt.

Aufgabe 6 (Rekursion) **[20 Punkte]**

Betrachten Sie die folgende Funktion:

```
1 func Foo(n float64, k int) float64 {
2     if k == 0 {
3         return n
4     }
5     x := Foo(n, k-1)
6     return (x + n/x) / 2
7 }
```

a) Berechnen Sie beispielhaft die Werte der Funktion mit $n = 25$ für $k = 0, \ldots, 5$.
Tragen Sie dazu in die Tabelle für jeden Wert von `k` den entsprechenden Wert von `x`, die konkreten Werte im **`return`**-Statement und das Ergebnis der Funktion ein.

Hinweis: Es genügt, die Werte auf zwei Nachkommastellen zu runden.

`k`	`x`	**`return`-Statement**	**Ergebnis**
0			
1			
2			
3			
4			
5			

b) Stellen Sie eine Rekursionsgleichung für die Funktion auf.

Hinweis: Die Rekursionsgleichung kann direkt abgelesen werden.

c) Beschreiben Sie in Worten, was die Funktion berechnet.

Aufgabe 7 (Structs) **[15 Punkte]**

Betrachten Sie den folgenden Code:

```
type Res struct {
    Start string
    End   string
}

func (r Res) String() string {
    return fmt.Sprintf("%s - %s", r.Start, r.End)
}

func (r Res) ChangeStart(newstart string) {
    r.Start = newstart
}

type ResData struct {
    Data []Res
}

func (rd *ResData) AddRes(start, end string) {
    rd.Data = append(rd.Data, Res{start, end})
}

func (rd ResData) String() string {
    results := []string{}
    for _, res := range rd.Data {
        results = append(results, res.String())
    }
    return strings.Join(results, "\n")
}

func (rd ResData) ChangeStart(i int, newstart string) {
    rd.Data[i].ChangeStart(newstart)
}
```

a) Erläutern Sie die Bedeutung und Funktionsweise dieser Datenstrukturen.

b) Was gibt ein Aufruf des folgenden Codes auf der Konsole aus?

```
    rd := ResData{}
    rd.AddRes("Mannheim", "Heidelberg")
    rd.AddRes("Heidelberg", "Stuttgart")

    fmt.Println(rd)
```

c) Warum schlägt der folgende Test fehl?

```
1  func ExampleResData_ChangeStart_failed() {
2      rd := ResData{}
3      rd.AddRes("Mannheim", "Heidelberg")
4      rd.AddRes("Heidelberg", "Stuttgart")
5      rd.ChangeStart(1, "München")
6
7      fmt.Println(rd)
8
9      // Output:
10     // Mannheim - Heidelberg
11     // München - Stuttgart
12 }
```

Aufgabe 8 (Funktionen) **[15 Punkte]**

Sie haben den Auftrag, eine Funktion `AppendDigitSum` zu entwerfen, die eine Zahl als String erwartet und die einen String zurückgibt, bei dem die Quersumme mit einem Unterstrich an den urprünglichen String angehängt ist. Die folgende Tabelle zeigt einige Beispiele:

Eingabe	Quersumme	Ergebnis
`"42"`	6	`"42-6"`
`"12345"`	15	`"12345-15"`
`"0"`	0	`"0-0"`

a) Entwerfen Sie die Funktion `AppendDigitSum`.

- Erläutern Sie, welche Schritte zur Lösung der Aufgabe notwendig sind.
- Geben Sie für jeden dieser Schritte eine Funktionssignatur und eine kurze Beschreibung an.
- Geben Sie auch eine Funktionssignatur für `AppendDigitSum` an.

b) Formulieren Sie einen Test für die Funktion `AppendDigitSum`.

Bemerkung: Sie müssen weder die Funktionen noch den Test implementieren. Es muss jedoch klar werden, was die notwendigen Schritte sind und was getestet werden soll.