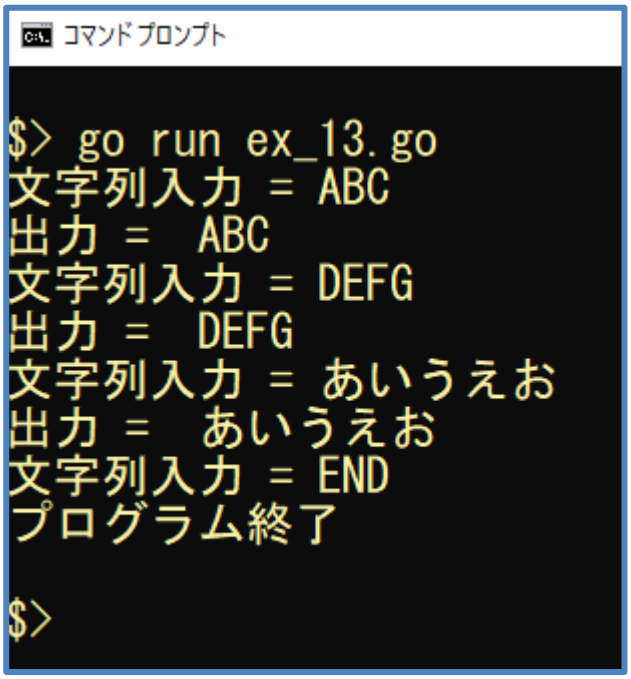
Go 言語　演習問題　その2

問題1

　下記のプログラムは、キーボードから文字列 “END” が入力されるまで、文字列を入力し、表示するプログラムです。
　下記の未完成プログラムの空欄部分を埋めて、実行結果となるように、プログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_13.go

```
/**
  無限ループ
  キーボードから"END"が入力されるまで
  文字列を入力し、表示し続ける
*/
package main

import "fmt"

func main() {
  // 変数 inputStr 宣言(型 string)
  var inputStr string

  // 無限ループ
  _____(1)_____ {
    // 入力促進
    fmt.Print("文字列入力 = ")
    // キーボードから文字列を入力し、変数 inputStr に保存
    fmt.Scanln(&inputStr)
    // 入力文字列が "END" か否かを判断
    _____(2)_____ {
      // 入力文字列が "END" であれば無限ループを抜ける
```

```
        _____(3)_____
    }
    // 表示
    fmt.Println("出力 = ", inputStr)
  }
  // プログラム終了表示
  fmt.Println("プログラム終了")
}
```

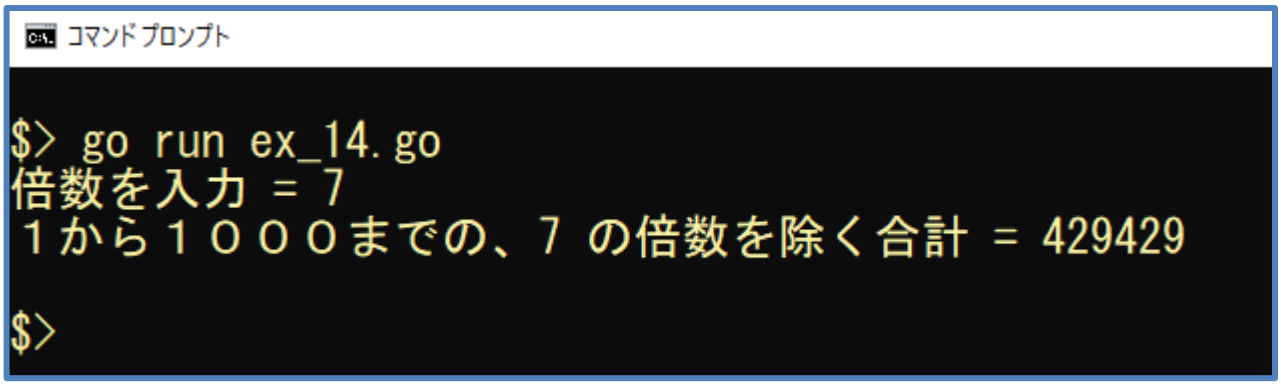
Go 言語　演習問題　その2

問題2

　下記のプログラムは、1から1000までの整数値の合計をもとめますが、キーボードから入力された整数値の倍数を除く、整数値の合計を求めて表示するものです。

　下記の未完成プログラムの空欄部分を埋めて、実行結果となるようにプログラムを完成させてください。

実行結果



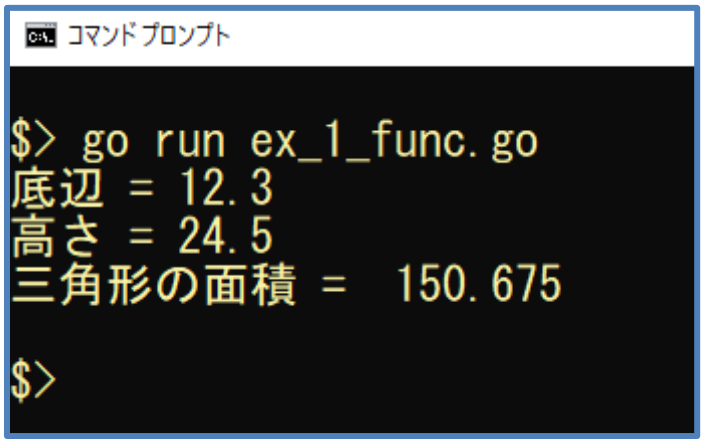
未完成プログラム　ファイル名：ex_14.go

```
/**
  continue 文
  1から1000までの合計を求めるが
  キーボードから入力された倍数を除く合計を求める
*/
package main

import "fmt"

func main() {
  // 変数 total（合計値）の初期化、型 int
  var total int = 0
  // 変数 mod（倍数）の宣言、型 int
  var mod int

  // キーボードから倍数を入力し変数 mod に保存
  // 入力促進
  fmt.Print("倍数を入力 = ")
  // 入力
  fmt.Scanln(&mod)

  // 1から1000までの入力された倍数を除く合計を求める
  _____(1)_____ {
    // 入力された倍数か否かを判断
    _____(2)_____ {
      // 入力された倍数だったら処理をスキップ
      _____(3)_____
    }
    // 合計を求める
    total += i
  }

  // 入力された倍数を除く合計を表示
  fmt.Printf("1から1000までの、%d の倍数を除く合計 = %d¥n", mod, total)
}
```

Go 言語　演習問題　その2

問題3
「Go 言語　演習問題　その1」の問題1において、三角形の面積を求める部分を関数にしてください。
下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_1_func.go

```
/**
  ユーザ定義関数を使用し、
  三角形の面積を求めて表示する
*/
package main

import "fmt"

// 関数 areaCalc の定義
/**
  関数シグニチャ
  関数名:
    areaCalc
  引数:
    base:底辺(型 float64)
    height:高さ(型 float64)
  処理:
    三角形の面積を求める
  戻り値:
    三角形の面積(型 float64)

*/
```

関数 areaCalc の定義
プログラム作成部分

```
func main() {
  // 底辺(bottom)、高さ(height)を宣言、型 float64
  var bottom, height float64
  // 面積(area)を宣言、型 float64
  var area float64
```

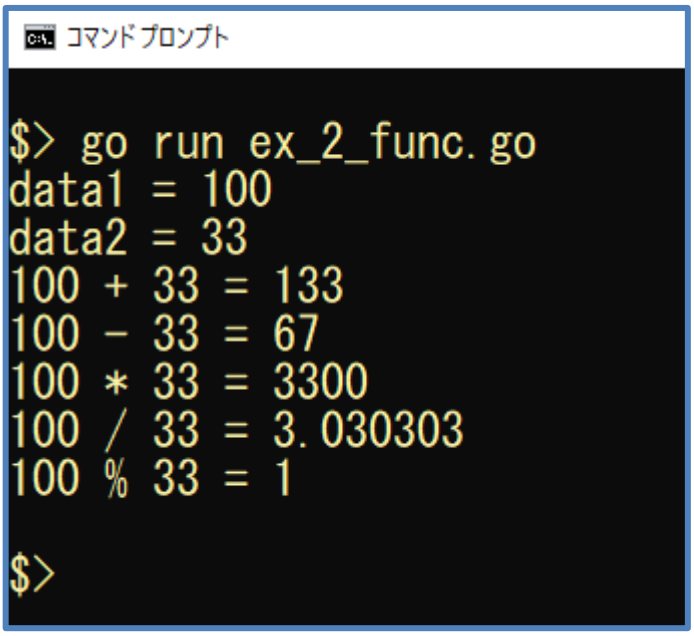
```
    // キーボードから底辺と高さを入力
    // 入力促進
    fmt.Print("底辺 = ")
    // キーボードから底辺を入力し、変数 bottom に保存
    fmt.Scanln(&bottom)
    // 入力促進
    fmt.Print("高さ = ")
    // キーボードから高さを入力
    fmt.Scanln(&height)

    // 三角形の面積を求め area に代入
    _____(1)_____

    // 三角形の面積を表示
    fmt.Println("三角形の面積 = ", area)
}
```

問題4

　問題2において、四則演算（たし算、ひき算、かけ算、わり算、剰余）を求める部分をユーザ定義関数とします。

　下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_2_func.go

```
/**
   ユーザ定義関数を使用し
   キーボードから二つの整数値を入力し
   四則演算を行い結果を表示
*/
package main

import "fmt"

// 関数 add の定義
/**
   関数シグニチャ
   関数名:
      add
   引数:
      number1:整数値1(型 int)
      number2:整数値2(型 int)
   処理:
      二つの整数値のたし算を行う
   戻り値:
      たし算の結果(型 int)
*/
```

プログラム作成部分

```
// 関数 sub の定義
/**
  関数シグニチャ
  関数名:
    sub
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のひき算を行う
  戻り値:
    ひき算の結果(型 int)
*/
```

プログラム作成部分

```
// 関数 mul の定義
/**
  関数シグニチャ
  関数名:
    mul
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のかけ算を行う
  戻り値:
    かけ算の結果(型 int)
*/
```

プログラム作成部分

```
// 関数 div の定義
/**
  関数シグニチャ
  関数名:
    div
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のわり算を行う
  戻り値:
    わり算の結果(型 float64)
*/
```

プログラム作成部分

```
// 関数 mod の定義
/**
  関数シグニチャ
  関数名:
    mod
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値の剰余算を行う
  戻り値:
    剰余算の結果(型 int)
*/
```

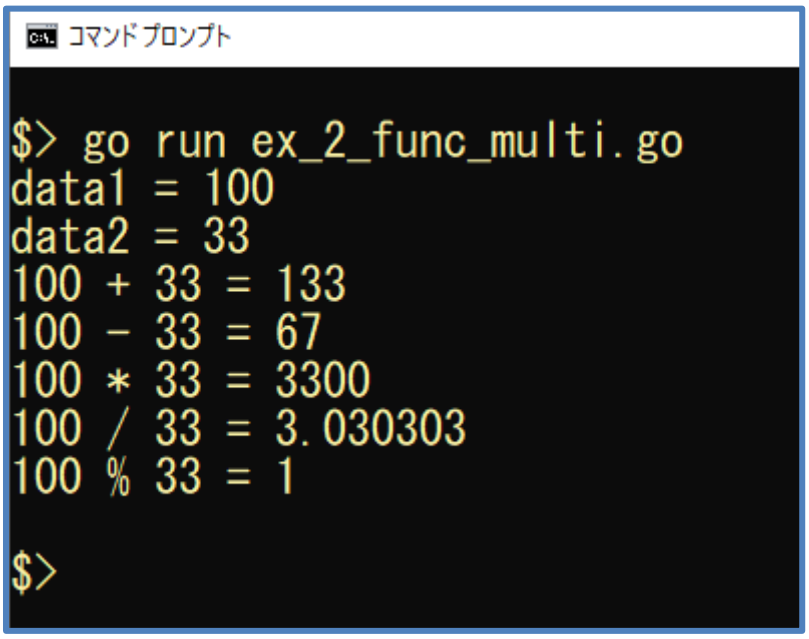
プログラム作成部分

```
func main() {
  // 変数 data1, data2 を宣言(型 int)
  var data1, data2 int
  // 変数 addAns, subAns, mulAns, modAns を宣言(型 int)
  var addAns, subAns, mulAns, modAns int
  // 変数 divAns を宣言(型 float64)
  var divAns float64

  // 入力促進
  fmt.Print("data1 = ")
  // キーボードから1つ目の整数値を入力
  fmt.Scanln(&data1)
  // 入力促進
  fmt.Print("data2 = ")
  // キーボードから2つ目の整数値を入力
  fmt.Scanln(&data2)

  // 四則演算
  // 加算
  _____(1)_____
  // 減算
  _____(2)_____
  // 乗算
  _____(3)_____
  // 除算
  _____(4)_____
  // 剰余
  _____(5)_____

  // 四則演算結果を表示
  // 加算
  fmt.Printf("%d + %d = %d¥n", data1, data2, addAns)
  // 減算
  fmt.Printf("%d - %d = %d¥n", data1, data2, subAns)
  // 乗算
  fmt.Printf("%d * %d = %d¥n", data1, data2, mulAns)
  // 徐算
  fmt.Printf("%d / %d = %f¥n", data1, data2, divAns)
```

```
    // 剰余
    fmt.Printf("%d %% %d = %d¥n", data1, data2, modAns)
}
```

問題5
　問題4のプログラムにおける、四則演算を求める関数を1つにまとめます。
　下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_2_func_multi.go

```
/**
   ユーザ定義関数を使用し
   キーボードから二つの整数値を入力し
   四則演算を行い結果を表示
*/
package main

import "fmt"

// 関数 arithmeticCalc の定義
/**
   関数シグニチャ
   関数名:
     arithmeticCalc
   引数:
     number1:整数値1(型 int)
     number2:整数値2(型 int)
   処理:
     2つの整数の加減乗除および剰余の計算を行いそれぞれの結果を返す
   戻り値:
     たし算の結果、引き算の結果、かけ算の結果(型 int)、わり算の結果(型 float64)
     剰余の結果(型 int)
*/
```

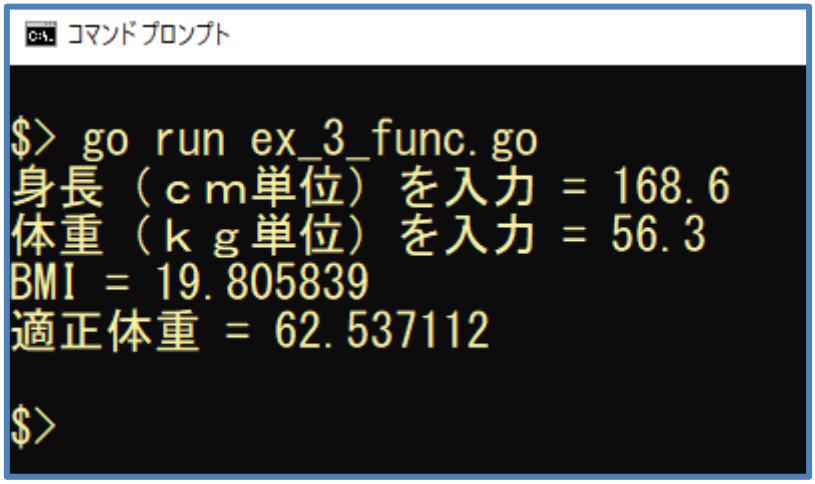
Go 言語　演習問題　その2


```
func main() {
  // 変数 data1, data2 を宣言(型 int)
  var data1, data2 int
  // 変数 addAns, subAns, mulAns, modAns を宣言(型 int)
  var addAns, subAns, mulAns, modAns int
  // 変数 divAns を宣言(型 float64)
  var divAns float64

  // 2つの整数を入力
  // 入力促進
  fmt.Print("data1 = ")
  // キーボードから1つ目の整数値を入力
  fmt.Scanln(&data1)
  // 入力促進
  fmt.Print("data2 = ")
  // キーボードから2つ目の整数値を入力
  fmt.Scanln(&data2)

  // 関数 arithmeticCalc を呼び出し四則演算し結果を受け取る
  _____(1)_____

  // 四則演算結果を表示
  // 加算
  fmt.Printf("%d + %d = %d¥n", data1, data2, addAns)
  // 減算
  fmt.Printf("%d - %d = %d¥n", data1, data2, subAns)
  // 乗算
  fmt.Printf("%d * %d = %d¥n", data1, data2, mulAns)
  // 徐算
  fmt.Printf("%d / %d = %f¥n", data1, data2, divAns)
  // 剰余
  fmt.Printf("%d %% %d = %d¥n", data1, data2, modAns)
}
```

問題6

「Go 言語　演習問題　その1」の問題3において、BMI 値を計算する機能と、適正体重を研鑽する機能をそれぞれ関数として定義し、実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_3_func.go

```
/**
   ユーザ定義関数を使用して
   キーボードから身長(cm単位)、体重(Kg 単位)
   を入力し BMI 値と適正体重を求めて表示する
   但し、BMI 値は以下の計算を行う
     BMI = 体重kg／(身長m)2
     適正体重 = 22×(身長m)2
*/
package main

import "fmt"

/**
   関数 bmiCalc の定義
   関数シグニチャ
   関数名:
     bmiCalc
   引数:
     height:身長(cm単位)(型 float64)
     weight:体重(kg単位)(型 float64)
   処理
     身長(m単位)と体重(kg単位)から BMI 値を求める
   戻り値:
     BMI 値(型 float64)
*/
```

プログラム作成部分

Go 言語　演習問題　その2

```
/**
  関数 properWeightCalc の定義
  関数シグニチャ
  関数名:
    properWeightCalc
  引数
    height:身長(cm単位)(型 float64)
  処理
    適正体重を計算し、返す
  戻り値
    適正体重(型 float64)
*/
```

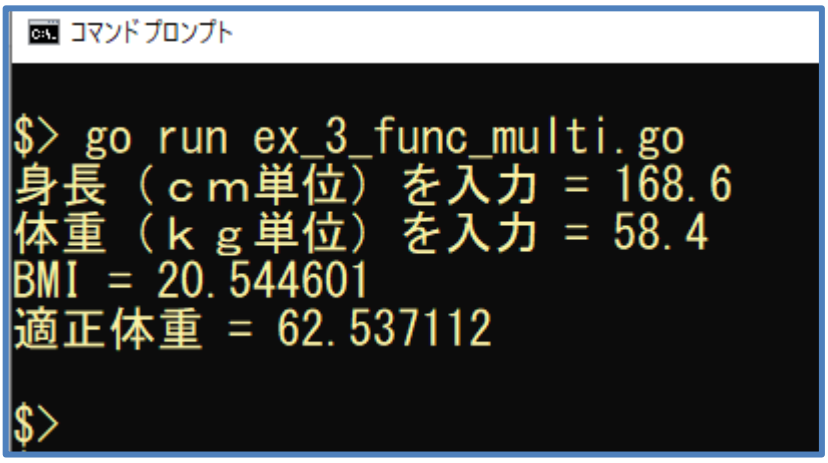
プログラム作成部分

```
func main() {
  // 変数 height, weight の宣言(型 float64)
  var height, weight float64

  // 身長(m 単位)を入力
  // 入力促進
  fmt.Print("身長(cm単位)を入力 = ")
  // 身長(cm単位)を入力し、変数 height に保存
  fmt.Scanln(&height)

  // 体重(kg単位)を入力
  // 入力促進
  fmt.Print("体重(kg単位)を入力 = ")
  // 体重(kg単位)を入力し、変数 weight に保存
  fmt.Scanln(&weight)

  // BMI 値を求めて、変数 bmi に保存
  _____(1)_____
  // 適正体重を求めて、変数 sutableWeight に保存
  _____(2)_____

  // BMI 値を表示
  fmt.Printf("BMI = %f¥n", bmi)
  fmt.Printf("適正体重 = %f¥n", sutableWeight)
}
```

問題7

問題6において、BMI 値と適正体重を求める機能を1つの機能にまとめて、関数化します。
下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを作成していください。

実行結果



未完成プログラム　ファイル名：ex_3_func_multi.go

```
/**
   ユーザ定義関数を使用して
   キーボードから身長(cm単位)、体重(Kg 単位)
   を入力し BMI 値と適正体重を求めて表示する
   但し、BMI 値は以下の計算を行う
     BMI = 体重kg／(身長m)2
     適正体重 = 22×(身長m)2
*/
package main

import "fmt"

/**
   関数 obesityInfo の定義
   関数シグニチャ
   関数名:
     obesityInfo
   引数:
     height:身長(cm単位)(型 float64)
     weight:体重(kg単位)(型 float64)
   処理
     身長(m単位)と体重(kg単位)から BMI 値と適正体重を求めて返す
   戻り値:
     BMI 値(型 float64)
     適正体重(型 float64)
*/
```

プログラム作成部分

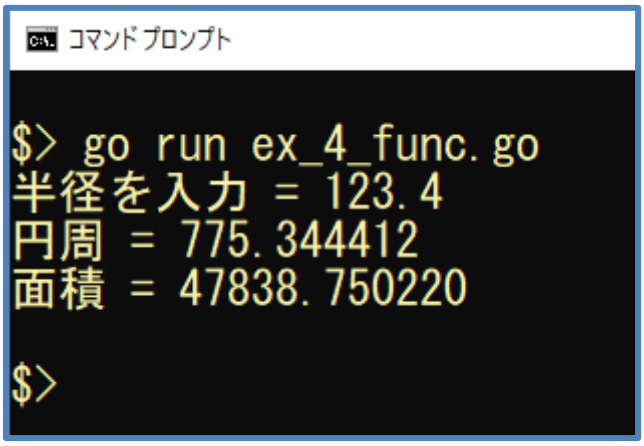
```
func main() {
  // 変数 height, weight の宣言(型 float64)
  var height, weight float64

  // 身長(m 単位)を入力
  // 入力促進
  fmt.Print("身長(cm単位)を入力 = ")
  // 身長(cm単位)を入力し、変数 height に保存
  fmt.Scanln(&height)

  // 体重(kg単位)を入力
  // 入力促進
  fmt.Print("体重(kg単位)を入力 = ")
  // 体重(kg単位)を入力し、変数 weight に保存
  fmt.Scanln(&weight)

  // 関数 obesityInfo を呼び出し、BMI 値と適正体重を求めて
  // 変数 bmi, sutableWeight に代入
  ______(1)______
  // BMI 値を表示
  fmt.Printf("BMI = %f¥n", bmi)
  fmt.Printf("適正体重 = %f¥n", sutableWeight)
}
```

問題8

「Go 言語　演習問題　その1」の問題4において、円周を計算する機能と、円の面積を求める機能を関数にします。

下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム

```
/**
   ユーザ定義関数を使用して
   キーボードから円の半径を入力し
   円周と面積を求めて表示する
*/
package main

import "fmt"

// 円周率 PAI(定数)3.14159 の宣言
const PAI float64 = 3.14159

/**
   関数 circumCalc の定義
   関数シグニチャ
   関数名:
     circumCalc
   引数:
     radius:半径(型 float64)
   処理:
     円周を求めて返す
   戻り値
     円周(型 float64)
*/
```

プログラム作成部分

```
/**
  関数 areaCircleCalc の定義
  関数シグニチャ
  関数名:
    areaCircleCalc
  引数:
    raidus:半径(型 float64)
  処理:
    円の面積を計算して返す
  戻り値
    円の面積(型 float64)
*/
```

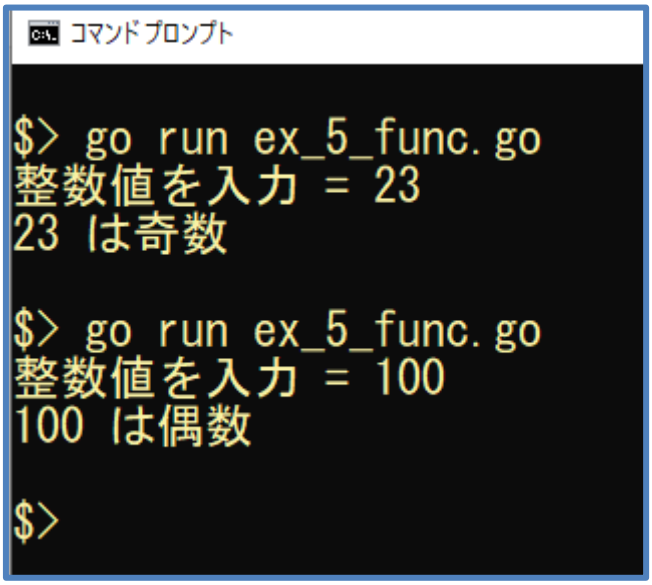
プログラム作成部分

```
func main() {
  // 変数 radius(半径)の宣言(型 float64)
  var radius float64

  // キーボードから半径を入力
  // 入力促進
  fmt.Print("半径を入力 = ")
  // キーボードから半径を入力し、変数 radius に保存
  fmt.Scanln(&radius)

  // 円周を計算し、変数 circumference に保存
  _____(1)_____
  // 面積を計算し、変数 area に保存
  _____(2)_____

  // 円周を表示
  fmt.Printf("円周 = %f¥n", circumference)
  // 面積を表示
  fmt.Printf("面積 = %f¥n", area)
}
```

問題9

「Go 言語　演習問題　その1」の問題5において、偶数か奇数かを判断する機能を関数とします。
下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名:ex_5_func.go

```
/**
   ユーザ定義関数を使用して
   キーボードから整数値を入力し
   偶数か奇数かを判断する
*/
package main

import "fmt"

/**
   関数 evenJudg
   関数シグニチャ
   関数名:
      evenJudg
   引数:
      number:調べる整数(型 int)
   処理:
      整数値が偶数か奇数かを判断する
   戻り値:
      true:整数値が偶数の時
      false:整数値が奇数の時
*/
```

Go 言語　演習問題　その2

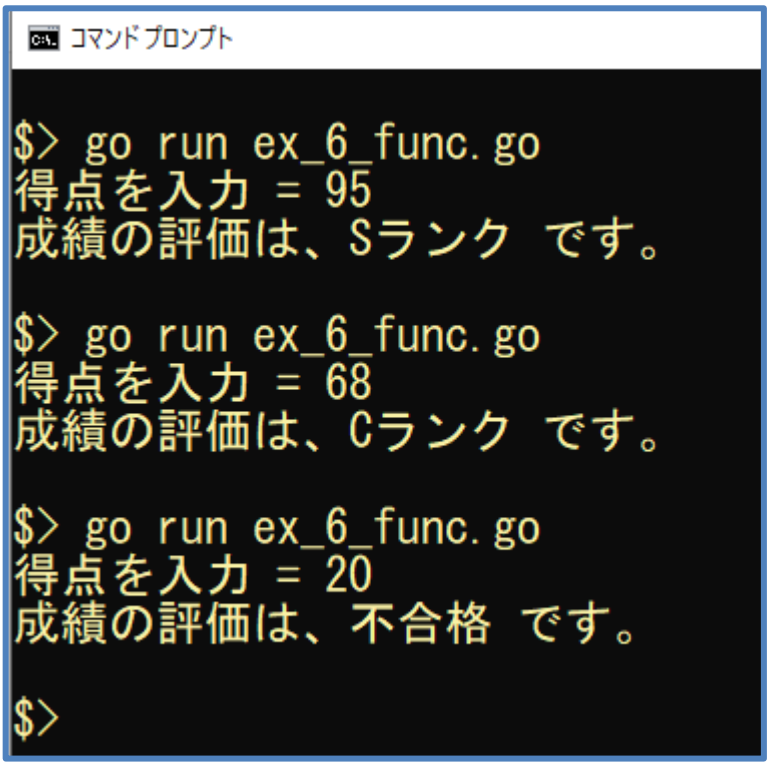
プログラム作成部分

```
func main() {
  // 変数 data の宣言(型 int)
  var data int

  // 入力促進
  fmt.Print("整数値を入力 = ")
  // キーボードから整数値を入力
  fmt.Scanln(&data)

  // 整数値が偶数化奇数かを判断
  if _____(1)_____ {
    // 偶数だったら
    fmt.Printf("%d は偶数¥n", data)
  } _____(2)_____ {
    // 奇数だったら
    fmt.Printf("%d は奇数¥n", data)
  }
}
```

問題10

「Go 言語　演習問題　その1」の問題6において、成績を評価する機能を関数とします。
下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_6_func.go

```
/**
   ユーザ定義関数を使用して
   キーボードから得点を入力し
   成績の評価を表示する
   但し、得点の評価は以下のとおりとする
     90 点以上 S ランク
     80 点以上 A ランク
     70 点以上 B ランク
     50 点以上 C ランク
     40 点以上 D ランク
     40 点未満 不合格
*/
package main

import "fmt"

/**
   関数 scoreEval の定義
   関数シグニチャ
   関数名:scoreEval
   引数:
     score:得点(型 int)
```

```
   処理:
      得点によりランクを判断する
   戻り値
      ランクを表す文字列
*/
```

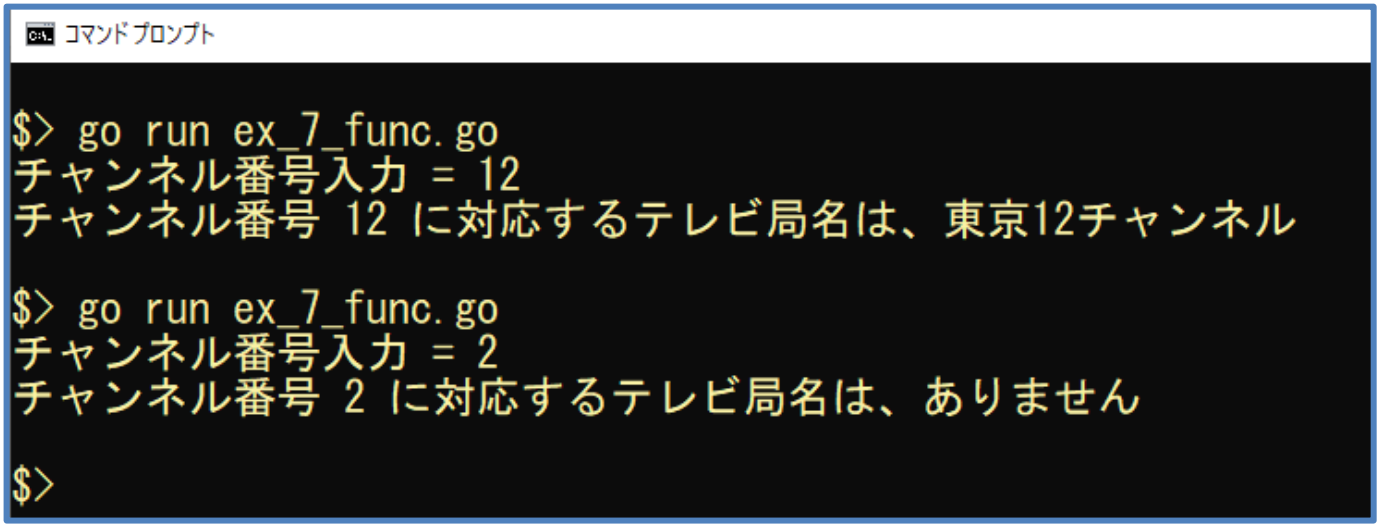

```
func main() {
   // 変数 score(得点)を宣言(型 int)
   var score int

   // キーボードから得点を入力
   // 入力促進
   fmt.Print("得点を入力 = ")
   // キーボードから得点を入力し変数 score に保存
   fmt.Scanln(&score)

   // 成績評価を表示
   fmt.Printf("成績の評価は、%s です。¥n", _____(1)_____)
}
```

問題11

「Go 言語　演習問題　その1」の問題7において、チャンネル番号から TV 局名を求める機能を関数とします。

下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_7_func.go

```
/**
   ユーザ定義関数を使用し
   キーボードからチャンネル番号を入力し
   対応するテレビ局名を表示する
*/
package main

import "fmt"

/**
   関数 tvStationName の定義
   関数シグニチャ
   関数名:tvStationName
   引数:
      channelName チャンネル番号(型 uint)
      )
   処理:
      チャンネル番号に対応するテレビ局名を判断し返す
   戻り値:
      テレビ局名(型 文字列)
*/
```

Go 言語　演習問題　その2

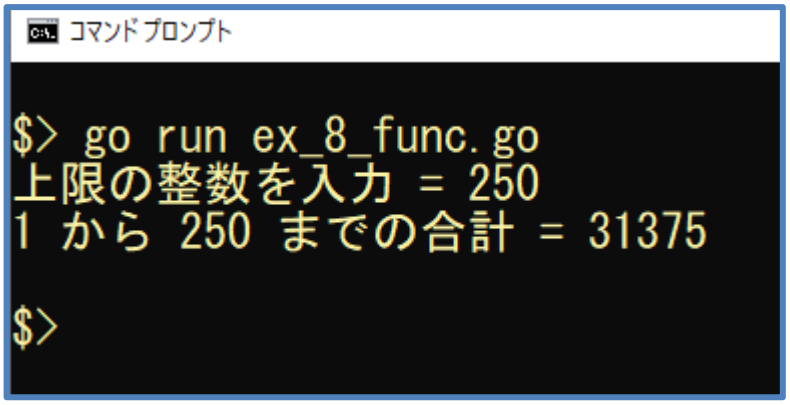

```
func main() {
  // 変数 channelNumber（チャンネル番号）の宣言（型 unit）
  var channelNumber uint

  // キーボードからチャンネル番号を入力
  // 入力促進
  fmt.Print("チャンネル番号入力 = ")
  // キーボードからチャンネル番号を入力し、変数 channelNumber に保存
  fmt.Scanln(&channelNumber)

  // チャンネル番号に対応するテレビ局名を表示
  fmt.Printf("チャンネル番号 %d に対応するテレビ局名は、%s¥n",
       _____(1)_____, _____(2)_____)
}
```

問題12

「Go 言語　演習問題　その1」の問題8において、1から指定された整数値までの合計を計算する機能を関数とします。

下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム

```
/**
   ユーザ定義関数を使用して
   キーボードから整数を入力し
   1からその入力した整数までの整数値の合計を求めて
   表示する
*/
package main

import "fmt"

/**
   関数 sumCalc の定義
   関数シグニチャ
   関数名:sumCalc
   引数:
      limit:上限となる整数(型 int)
   処理
      1から上限となる整数までの合計を計算して返す
   戻り値
      合計(型 int)
*/
```

プログラム作成部分

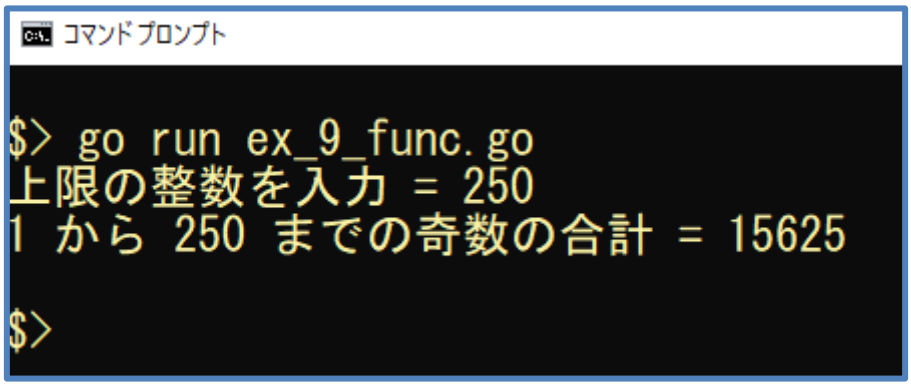
```
func main() {
  // 変数 limti（上限の整数）を宣言(型 int)
  var limit int

  // 上限の整数を入力
  // 入力促進
  fmt.Print("上限の整数を入力 = ")
  // キーボードから整数を入力し、変数 limit に保存
  fmt.Scanln(&limit)

  // 合計値を表示
  fmt.Printf("1 から %d までの合計 = %d¥n", _____(1)_____, _____(2)_____)
}
```

問題13

「Go 言語　演習問題　その1」の問題9において、1から指定された整数値までの奇数の合計を計算する機能を関数とします。

下記の実行結果となるように未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム

```
/**
   ex_8.go において
   ユーザ定義関数を使用して
   1から上限の整数までの奇数の合計を求めて表示する
*/
package main

import "fmt"

/**
   関数 sumOddCalc の定義
   関数シグニチャ
      関数名:sumOddCalc
      引数:
         limit:合計の上限となる整数(型 int)
      処理:
         1から上限数までの奇数の合計を求めて返す
      戻り値:
         奇数の合計値(型 int)
*/
```

プログラム作成部分

```
func main() {
  // 変数 limti(上限の整数) を宣言(型 int)
  var limit int

  // 上限の整数を入力
  // 入力促進
  fmt.Print("上限の整数を入力 = ")
  // キーボードから整数を入力し、変数 limit に保存
  fmt.Scanln(&limit)

  // 合計値を表示
  fmt.Printf("1 から %d までの奇数の合計 = %d¥n", ____(1)____, ____(2)____)
}
```

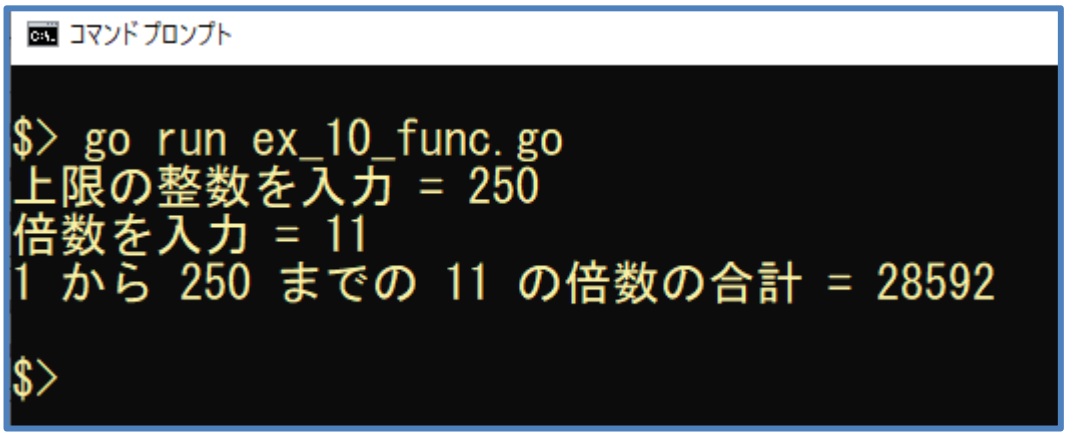
問題14

「Go 言語　演習問題　その1」の問題10において、1から指定された整数値までの指定された倍数の合計を計算する機能を関数とします。
下記の実行結果となるように作成してください。
関数シグニチャは以下のようにします

関数シグニチャ
　関数名:sumModCalc
　引数:
　　limit:上限数(型 int)
　　mod:倍数(型 int)
　処理:
　　1から上限数までの倍数の合計をもとめて返す
　戻り値:
　　1から上限数までの倍数の合計

完成プログラムファイル名:ex_10_func.go

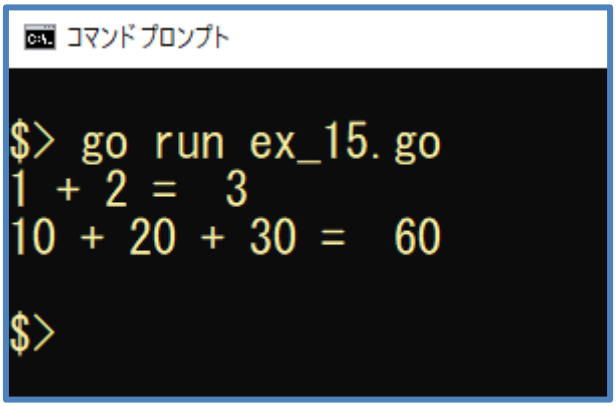
実行結果



問題15

可変長引数関数をしようして、複数整数値の合計を求めます。
下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_15.go

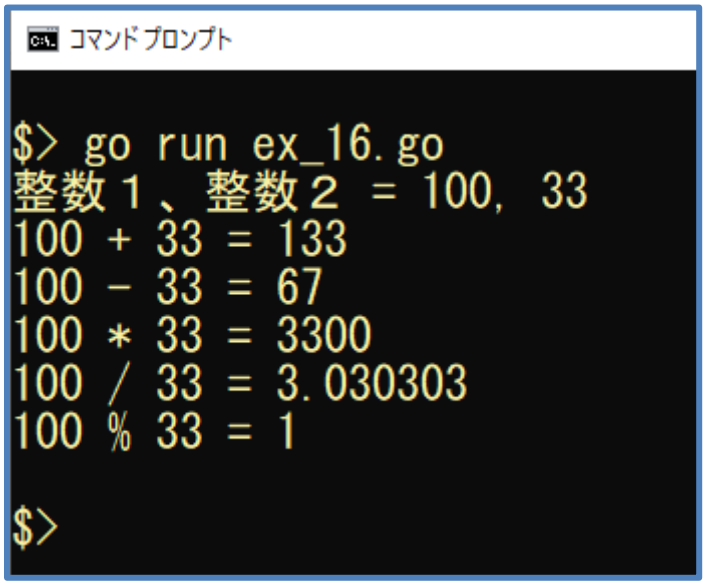
```
/**
   可変長パラメータ
   複数の数値の合計をもとめる
*/
package main

import "fmt"

/**
   関数 variableAddCalc の定義
   関数シグニチャ
      関数名:variableAddCalc
      引数:
         data：可変の整数値(型 int)
      処理:
         可変個整数値の合計をもとめる
      戻り値:
         合計値(型 int)
*/
```

プログラム作成部分

```
func main() {

  // 可変個整数値の合計を表示
  fmt.Println("1 + 2 = ", _____(1)_____)
  fmt.Println("10 + 20 + 30 = ", _____(1)_____)
}
```

問題16
　無名関数を定義して、四則演算を行います。
　下記の実行結果となるように、未完成プログラム空白部分を埋めてプログラムを完成させてください。

実行結果



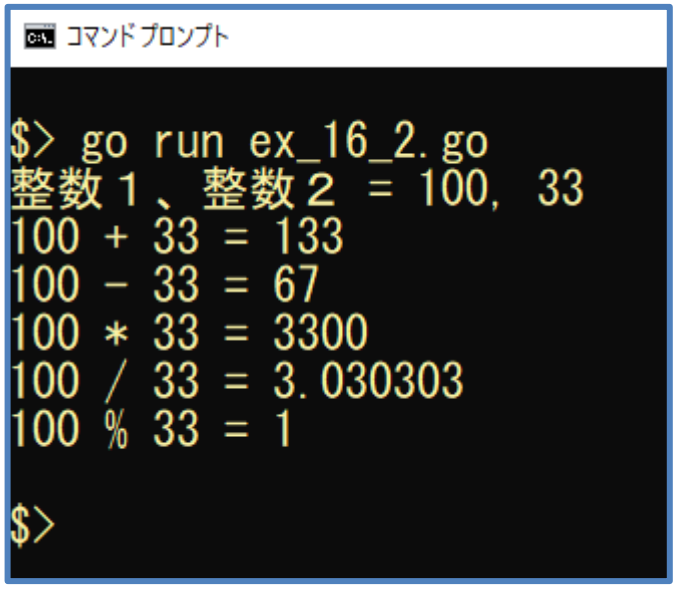
未完成プログラム　ファイル名：ex_16.go

```
/**
   無名関数
   無名関数を使用して四則演算を行う
*/
package main

import "fmt"

func main() {
  // 変数 num1, num2 の宣言(型 int)
  var num1, num2 int

  // キーボードから2つの整数を入力する
  // 入力促進
  fmt.Print("整数1、整数2 = ")
  // 2つの整数を入力し、変数 num1, num2 に保存
  fmt.Scanf("%d,%d", &num1, &num2)

  // 無名関数によるたし算
  プログラム作成部分

  // 無名関数によるひき算
  プログラム作成部分
```

```
    // 無名関数によるかけ算
    プログラム作成部分

    // 無名関数によるわり算
    プログラム作成部分

    // 無名関数による剰余
    プログラム作成部分


}
```

問題17
　問題16で定義した無名関数を「関数リテラル」として変数に代入して、実行します。
　下記の実行結果となるように、未完成プログラム作成部分を埋めてプログラムを完成させてください。

実行結果



未完成プログラム　ファイル名：ex_16_2.go

```
/**
   無名関数
   無名関数を使用して四則演算を行う
*/
package main

import "fmt"

func main() {
  // 変数 num1, num2 の宣言(型 int)
  var num1, num2 int

  // キーボードから2つの整数を入力する
  // 入力促進
  fmt.Print("整数1、整数2 = ")
  // 2つの整数を入力し、変数 num1, num2 に保存
  fmt.Scanf("%d,%d", &num1, &num2)

  // 無名関数によるたし算(関数リテラルを変数 add に代入)
  プログラム作成部分


  // 無名関数によるひき算(関数リテラルを変数 sub に代入)
  プログラム作成部分
```

```
    // 無名関数によるかけ算(関数リテラルを変数 mul に代入)
    プログラム作成部分

    // 無名関数によるわり算(関数リテラルを変数 div に代入)
    プログラム作成部分

    // 無名関数による剰余(関数リテラルを変数 mod に代入)
    プログラム作成部分


    // 四則演算の実行と結果の表示
    add(num1, num2) // たし算
    sub(num1, num2) // ひき算
    mul(num1, num2) // かけ算
    div(num1, num2) // わり算
    mod(num1, num2) // 剰余
}
```

問題18

　問題2のプログラムにおいて、関数型変数を使用して、下記の実行結果となるように、未完成プログラムの空欄部分を埋めてプログラムを完成させてください。

実行結果

```
$> go run ex_18.go
data1 = 100
data2 = 33
100 + 33 = 133
100 - 33 = 67
100 * 33 = 3300
100 / 33 = 3.030303
100 % 33 = 1

$>
```

未完成プログラム　ファイル名：ex_18.go

```
/**
   ユーザ定義関数を使用し
   キーボードから二つの整数値を入力し
   四則演算を行い結果を表示
   （関数型変数を使用して実行）
*/
package main

import "fmt"

// 関数 add の定義
/**
   関数シグニチャ
   関数名：
      add
   引数：
      number1：整数値1（型 int）
      number2：整数値2（型 int）
   処理：
      二つの整数値のたし算を行う
   戻り値：
      たし算の結果（型 int）
*/
```

プログラム作成部分

```
// 関数 sub の定義
/**
  関数シグニチャ
  関数名:
    sub
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のひき算を行う
  戻り値:
    ひき算の結果(型 int)
*/
```

プログラム作成部分

```
// 関数 mul の定義
/**
  関数シグニチャ
  関数名:
    mul
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のかけ算を行う
  戻り値:
    かけ算の結果(型 int)
*/
```

プログラム作成部分

```
// 関数 div の定義
/**
  関数シグニチャ
  関数名:
    div
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値のわり算を行う
  戻り値:
    わり算の結果(型 float64)
*/
```

プログラム作成部分

```
// 関数 mod の定義
/**
  関数シグニチャ
  関数名:
    mod
  引数:
    number1:整数値1(型 int)
    number2:整数値2(型 int)
  処理:
    二つの整数値の剰余算を行う
  戻り値:
    剰余算の結果(型 int)
*/
```

プログラム作成部分

```
func main() {
  // 変数 data1, data2 を宣言(型 int)
  var data1, data2 int
  // 変数 addAns, subAns, mulAns, modAns を宣言(型 int)
  var addAns, subAns, mulAns, modAns int
  // 変数 divAns を宣言(型 float64)
  var divAns float64

  // 関数型変数の宣言
  var fInt _____(1)_____
  var fFloat _____(2)_____

  // 入力促進
  fmt.Print("data1 = ")
  // キーボードから1つ目の整数値を入力
  fmt.Scanln(&data1)
  // 入力促進
  fmt.Print("data2 = ")
  // キーボードから2つ目の整数値を入力
  fmt.Scanln(&data2)

  // 四則演算
  // 加算
  _____(3)_____
  addAns = _____(4)_____
  // 減算
  _____(5)_____
```

```
    subAns = _____(6)_____
    // 乗算
    _____(7)_____
    mulAns = _____(8)_____
    // 除算
    _____(9)_____
    divAns = _____(10)_____
    // 剰余
    _____(11)_____
    modAns = _____(12)_____

    // 四則演算結果を表示
    // 加算
    fmt.Printf("%d + %d = %d¥n", data1, data2, addAns)
    // 減算
    fmt.Printf("%d - %d = %d¥n", data1, data2, subAns)
    // 乗算
    fmt.Printf("%d * %d = %d¥n", data1, data2, mulAns)
    // 徐算
    fmt.Printf("%d / %d = %f¥n", data1, data2, divAns)
    // 剰余
    fmt.Printf("%d %% %d = %d¥n", data1, data2, modAns)
}
```