**Summe aller ungeraden Zahlen**

**func SumOdd(numbers []int) int {**

    if len(numbers) == 0 {

        return 0}

head := numbers[0]

tail := numbers[1:]

if head%2 != 0 {

    return head + SumOdd(tail)}

 return SumOdd(tail)}

**Produkt aller ungeraden Zahlen**

**func ProductOdd(numbers []int) int {**

if len(numbers) == 0{

    return 1

    head := numbers[0]

    tail := numbers[1:]

 if head%2 != 0 {

return head \* ProductOdd(tail)

  }return ProductOdd(tail)}

**Vereinigung der beiden Listen**

func Union(a, b []int) []int {

    result := []int{}

        if len(a) == 0 {

            return b }

        if len(a) == 0 {

            return b}

        for \_, el1 := range a {

            if !Contains(el1, result) {

                result = append(result, el1)}}

        for \_, el2 := range b {

            if !Contains(el2, result) {

                result = append(result, el2)}}

    return result}

func Contains(e int , l []int) bool {

    for \_, el := range l {

        if el == e {

            return true}}

    return false}

**Primzahl ja/nein**

func IsPrime(n int) bool {

return n > 1 && !DivisibleByAny(n, n/2)

}

func DivisibleByAny(n, c int) bool {

if c <= 1 {

return false }

if n%c == 0 {

return true}

return DivisibleByAny(n, c-1) }

**Potenz n^m rekursiv berechnen\***

func abc(n, m int) int {

    if m == 0 {

        return 1}

    return n \* abc(n, m-1)}

**Anzahl der Elemente zählt, die mit "abc**"

func CountAbc(list []string) int {

    result := 0

    for \_, val := range list {

 if len(val) >= 3 && val[:3] == "abc" {

       result++}}

return result}

**jeder Position das kleinere der beiden E**

func MinElements(a, b []int) []int {

if len(a) == 0{

    return b}

if len(b) == 0{

    return a}

    less := a[0]

    if less >= b[0]{

        less = b[0] }

    return append([]int{less}, MinElements(a[1:], b[1:])...)}

**m geteilt durch rekursiv\***

func abc(m, n int) int{

    if m < n {

        return 0 }

    return 1+ abc(m-n, n) }

**Ob gültige Dominokette**

func ValidDominoChain(dominos []Domino) bool {

    for i:= 0; i < len(dominos)-1; i++{

    if dominos[i].right != dominos[i+1].left {

        return false

    }return true}

type Domino struct{

    left int

    right int }

**func SumEven(numbers []int) int {**

    if len(numbers) == 0 {

        return 0}

    oddnumbers := numbers[0]

    tail := numbers[1:]

    if oddnumbers%2 == 0 {

        return oddnumbers + SumEven(tail)

    } else {

        return SumEven(tail) }}

**m geteilt n durch rekursiv\***

func abc(m, n int) int{

    if m < n {

        return 0 }

    return 1+ abc(m-n, n) }

**Modulo n%m rekursiv\***

func abc(n, m int) int{

    if n >= m{

        return Foo1(n-m, m)}

    if n < 0 {

      return Foo1(n+m, m) }}

**Slice umdrehen**

func Reverse(slice []int) []int {

for i, j := 0, len(slice)-1; i < j; i, j = i+1, j-1 {

slice[i], slice[j] = slice[j], slice[i] }

return slice }

**alle Elemente zurückgibt, die mit "abc"**

func AllAbc(list []string) []string {

result := []string{}

    for \_ , val := range list {

        if len(val) >= 3 && val[:3] == "abc" {

            result = append(result, val)}}

    return result}

**Funktion LongestAbc, die das längste E zurückgibt, das mit "abc" beginnt**

func LongestAbc(list []string) string {

    longestpos := -1

    longestlen := -1

    for pos , val := range list {

        currentlen := len(val)

        if currentlen >= 3 && val[:3] == "abc" && currentlen > longestlen{

            longestlen = currentlen

            longestpos = pos}}

if longestpos != -1 {

    return list[longestpos]

}   return "" }

**Schnittmenge/Dupli der beiden Listen zurückgibt**

func Intersection(a, b []int) []int {

    result := []int{}

        if len(a) == 0 {

            return b}

        if len(a) == 0 {

            return b}

        for \_, el1 := range a {

            if Contains(el1,b){

                result = append(result, el1)}}

    return result}

func Contains (e int , l []int) bool {

    for \_, el := range l {

        if el == e {

            return true}}

return false}

**Produkt aller ungeraden Zahlen**

func **ProductOdd**(numbers []int) int {

    result := 1

    for \_, i := range numbers {

        if i%2 != 0 {

            result = result \* i}}

return result

}

**alle Elemente zwischen first und last zurückgeben**

func **IncludeStngBetween**(list []string, start, end string) []string { firstpos := 0 lastpos := 0

    for pos , s := range list{

        if s == start { firstpos = pos}

        if s == end {

            lastpos = pos}}

    if firstpos >= lastpos {

        return []string{} }

return list[firstpos +1 : lastpos] }

**Summe geraderx Zahlen in der Liste**

func SumEven(numbers []int) int {

    if len(numbers) == 0 {

        return 0}

oddnumbers := numbers[0]

tail := numbers[1:]

if oddnumbers%2 == 0 {

return oddnumbers + SumEven(tail) } else { return SumEven(tail) }}

**Längestes E mit ende „def“**

func LongestDef(list []string) string {

longestpos := -1

longestlen := -1

for pos, val := range list {

currentlen := len(val)

if currentlen >= 3 && val[currentlen-3:] == "def" &&

currentlen > longestlen {

longestlen = currentlen

longestpos = pos} }

if longestpos != -1 {

return list[longestpos] }

return ""}

func **ExcludeBetween**(list []int, m, n int) []int {

 result := []int{}

 for i := 0; i < len (list); i++ {

   if list[i] > m && list[i] < n{

    result = append(result, list[i])}}

return result}

**jeder Position die Summe der beiden E**

func SumElements(a, b []int) []int {

    maxlenght := len(a)

    if maxlenght < len(b) {

        maxlenght = len(b)}

    result := make([]int, maxlenght)

    for i := 0; i < maxlenght; i++ {

        if i < len(a) {

            result[i] += a[i]}

        if i < len(b) {

            result[i] += b[i]}}

    return result}

func (c Card) **GreaterThan**(other Card) bool {

    return c.Suit==other.Rank && c.Rank > other.Rank}

type Card struct {

    Suit int

    Rank int}

**Elemente außerhalb von first und last zurückgeben**

func ExcludeStringsOutside(list []string, start string, end string) []string {

    firstpos := -1 lastpos := -1

    for pos, s := range list{

        if s == start {

            firstpos = pos}

        if s == end {

            lastpos = pos} }

        if firstpos >= lastpos {

            return []string{}}

        return append(list[:firstpos], list[lastpos +1:]...)}

**Stelle n, Summe der E aus list**

func **ArraySums**(list []int) []int {

result := []int{}

 for i := 0; i < len(list); i++ {

        result1 := 0

        for j := 0; j <= i; j++ {

            result1 += list[j]}

result = append(result, result1)}

  return result}

**Gemeinsame Vielfache**

func CommonMultiples(m, n, max int) []int {

    result := []int{}

    for i := 1 ; i <= max; i ++{

        if i%m == 0 && i%n == 0 {

            result = append(result,i)}}

    return result}

func **ElementProducts**(l1, l2 []int) []int {

    if len(l1) == 0 {

        return l2 }

        if len(l2) == 0 {

            return l1 }

 return append([]int{l1[0] \* l2[0]}, ElementProducts(l1[1:], l2[1:])...)

func **Power2**(x int) float64

if x == 0 {

return 1}

if x < 0 {

return Power2(x+1) \* 0.5}

return Power2(x-1) \*2}

Ein Bild, das Text, Schrift, Screenshot, Handschrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Handschrift, Schrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Schrift, Screenshot, Handschrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Schrift, Quittung, weiß enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Quittung, Schrift, weiß enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Zahl, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Quittung, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Quittung, Schrift, weiß enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Schrift, Quittung, weiß enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Quittung, Schrift, weiß enthält.

KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Schrift, Screenshot, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Schrift, Quittung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Schrift, Handschrift, Screenshot enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Schrift, Screenshot, Quittung enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Schrift, weiß, Werkzeug enthält.

KI-generierte Inhalte können fehlerhaft sein.