```go
package main

import (
    "errors"
    "math"
    "math/cmplx"
    "sort"
    "strconv"
    "strings"
    "unicode"
)


func convertingNumberSystem(num string, base int, toBase int) string {
    res, err := strconv.ParseInt(num, base, 64)
    if err != nil {
        return ""
    }
    return strconv.FormatInt(res, toBase)
}

func quadraticEquationSolver(a float64, b float64, c float64) (complex128,
complex128) {
    discriminant := b*b - 4*a*c
    if discriminant < 0 {
        complex_discriminant := cmplx.Sqrt(complex(discriminant, 0))
        complex_root1 := (complex(-b, 0) + complex_discriminant) / (2 *
complex(a, 0))
        complex_root2 := (complex(-b, 0) - complex_discriminant) / (2 *
complex(a, 0))
        return complex_root1, complex_root2
    }
    root1 := (-b + math.Sqrt(discriminant)) / (2 * a)
    root2 := (-b - math.Sqrt(discriminant)) / (2 * a)
    return complex(root1, 0), complex(root2, 0)
}

func sortNumsByModule(nums []float64) []float64 {
    sort.SliceStable(nums, func(i, j int) bool {
        return math.Abs(nums[i]) < math.Abs(nums[j])
    })
    return nums
}

func mergeTwoArrays(arr1 []float64, arr2 []float64) []float64 {
    merged := make([]float64, 0, len(arr1)+len(arr2))
    merged = append(merged, arr1...)
    merged = append(merged, arr2...)
    return merged
}
```

```go
func findSubstringInString(substring string, fullString string) (startIndex int)
{
    for i := 0; i <= len(fullString)-len(substring); i++ {
        if fullString[i] == substring[0] {
            match := true
            for j := 0; j < len(substring); j++ {
                if fullString[i+j] != substring[j] {
                    match = false
                    break
                }
            }
            if match {
                return i
            }
        }
    }
    return -1
}

func calculate(num1, num2 float64, operator string) (float64, error) {
    switch operator {
    case "+":
        return num1 + num2, nil
    case "-":
        return num1 - num2, nil
    case "*":
        return num1 * num2, nil
    case "/":
        if num2 == 0 {
            return 0, errors.New("деление на ноль невозможно")
        }
        return num1 / num2, nil
    case "^":
        return math.Pow(num1, num2), nil
    case "%":
        if num2 == 0 {
            return 0, errors.New("деление на ноль невозможно")
        }
        return float64(int(num1) % int(num2)), nil
    default:
        return 0, errors.New("недопустимая операция")
    }
}

func checkPalindrom(fullString string) bool {
    var cleanedString []rune
    for _, r := range fullString {
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
            cleanedString = append(cleanedString, unicode.ToLower(r))
        }
    }
```

```
    for i := 0; i < len(cleanedString)/2; i++ {
        if cleanedString[i] != cleanedString[len(cleanedString)-i-1] {
            return false
        }
    }
    return true
}

func hasIntersection(x1, y1, x2, y2, x3, y3 float64) bool {
    x_max := math.Max(x3, math.Max(x1, x2))
    y_min := math.Min(y3, math.Min(y1, y2))
    return x_max <= y_min
}

func maxLengthOfWords(s string) string {
    var indexOfMax int
    maxLength := -1
    words := strings.Fields(s)
    for i, word := range words {
        if len(word) > maxLength {
            maxLength = len(word)
            indexOfMax = i
        }
    }
    return words[indexOfMax]
}

func checkingTheHighYear(n int) bool {
    if n%4 != 0 || (n%100 == 0 && n%400 != 0) {
        return false
    }
    return true
}

func fibonacciNumbers(n int) (res []int) {
    if n > 0 {
        res = append(res, 0)
        if n == 1 {
            return res
        } else {
            res = append(res, 1)
            for i := 2; i < n; i++ {
                res = append(res, res[len(res)-1]+res[len(res)-2])
            }
        }
    }
    return res
}

func armstrongNumbers(start, end int) (res []int) {
    for i := start; i <= end; i++ {
```

```
        if isArmstrong(i) {
            res = append(res, i)
        }
    }
    return res
}

func isArmstrong(num int) bool {
    sum := 0
    n := num
    digits := int(math.Log10(float64(num))) + 1

    for n > 0 {
        digit := n % 10
        sum += int(math.Pow(float64(digit), float64(digits)))
        n /= 10
    }

    return sum == num
}

func primeNumbers(start, stop int) []int {
    isPrime := make([]bool, stop+1)
    for i := 2; i <= stop; i++ {
        isPrime[i] = true
    }

    for p := 2; p*p <= stop; p++ {
        if isPrime[p] {
            for i := p * p; i <= stop; i += p {
                isPrime[i] = false
            }
        }
    }

    var primes []int
    for p := start; p <= stop; p++ {
        if isPrime[p] {
            primes = append(primes, p)
        }
    }
    return primes
}

func reverseString(s string) string {
    n := len(s)
    reversed := make([]byte, n)

    for i := 0; i < n; i++ {
        reversed[i] = s[n-1-i]
    }
```

```
    return string(reversed)
}

func gcd(a, b int) int {
    for b != 0 {
        a, b = b, a%b
    }
    return a
}
```