

Алгоритмизация и программирование

3. Рекурсии и циклы

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Два способа повторения однотипных действий

- ▶ Рекурсия: прямое или косвенное использование функцией самой себя

Два способа повторения однотипных действий

- ▶ Рекурсия: прямое или косвенное использование функцией самой себя
- ▶ Цикл: многократное повторение одного и того же блока в функции

Рекурсия: факториал

- ▶ $n! = n(n-1)!$

Рекурсия: факториал

- ▶ $n! = n(n-1)!$
- ▶ $0! = 1, 1! = 1$

Рекурсия: факториал

- ▶ $n! = n(n-1)!$
- ▶ $0! = 1, 1! = 1$ (база)

Рекурсия: факториал

- ▶ $n! = n(n-1)!$
- ▶ $0! = 1, 1! = 1$ (база)

```
fun factorial(n: Int): Double =  
    if (n < 2) 1.0 else n * factorial(n - 1)
```

Цикл: факториал

► $n! = 1 * 2 * \dots * (n-1) * n$

Цикл: факториал

► $n! = 1 * 2 * \dots * (n-1) * n$

```
fun factorial(n: Int): Double {  
    var result = 1.0  
    for (i in 1..n) {  
        result = result * i  
    }  
    return result  
}
```

Цикл: факториал

► $n! = 1 * 2 * \dots * (n-1) * n$

```
fun factorial(n: Int): Double {  
    var result = 1.0  
    for (i in 1..n) {  
        result = result * i  
    }  
    return result  
}
```

// for = цикл: для (каждого) i в (интервале) 1..n ...

Цикл: факториал

► $n! = 1 * 2 * \dots * (n-1) * n$

```
fun factorial(n: Int): Double {  
    var result = 1.0  
    for (i in 1..n) {  
        result = result * i  
    }  
    return result  
}
```

// for = цикл: для (каждого) i в (интервале) 1..n ...

// var = мутирующая переменная

Модифицирующие операторы

```
fun factorial(n: Int): Double {  
    var result = 1.0  
    for (i in 1..n) {  
        result *= i // result = result * i  
    }  
    return result  
}
```

Модифицирующие операторы

- ▶ $+=$ увеличить на ...
- ▶ $-=$ уменьшить на ...
- ▶ $*=$ домножить на ...
- ▶ $/=$ разделить и присвоить
- ▶ $\%=$ взять остаток от деления и присвоить

Операторы инкремента / декремента

- ▶ `a++` или `++a` – увеличить на 1
- ▶ `a--` или `--a` – уменьшить на 1

Операторы инкремента / декремента

- ▶ `a++` или `++a` – увеличить на 1
- ▶ `a--` или `--a` – уменьшить на 1

```
fun foo() {  
    var a = 3  
    val b = a++  
}
```

Операторы инкремента / декремена

- ▶ `a++` или `++a` – увеличить на 1
- ▶ `a--` или `--a` – уменьшить на 1

```
fun foo() {  
    var a = 3  
    val b = a++ // a = 4, b = 3  
}
```


Операторы инкремента / декремента

- ▶ `a++` или `++a` – увеличить на 1
- ▶ `a--` или `--a` – уменьшить на 1

```
fun foo() {  
    var a = 3  
    val b = ++a  
}
```

Операторы инкремента / декремена

- ▶ `a++` или `++a` – увеличить на 1
- ▶ `a--` или `--a` – уменьшить на 1

```
fun foo() {  
    var a = 3  
    val b = ++a // a = 4, b = 4  
}
```

Операторы инкремента / декремента

- ▶ $a++$ или $++a$ – увеличить на 1
- ▶ $a--$ или $--a$ – уменьшить на 1
- ▶ Значение $a++$ или $a--$ равно старому значению a

Операторы инкремента / декремента

- ▶ $a++$ или $++a$ – увеличить на 1
- ▶ $a--$ или $--a$ – уменьшить на 1
- ▶ Значение $a++$ или $a--$ равно старому значению a
- ▶ А значение $++a$ или $--a$ равно новому значению a

Содержимое цикла

// Заголовок

```
for (i in 1..10) {  
    // Тело цикла  
}
```

Содержимое цикла

// Заголовок

```
for (i in 1..10) {  
    // Тело цикла  
}
```

// Итерация = одно выполнение тела

Интервалы и прогрессии

```
for (i in 1..10) { ... } // интервал
```

Интервалы и прогрессии

```
for (i in 1..10) { ... } // интервал
```

```
for (i in 10 downTo 1) { ... } // прогрессия
```


Интервалы и прогрессии

```
for (i in 1..10) { ... } // интервал
```

```
for (i in 10 downTo 1) { ... } // прогрессия
```

```
for (i in 1..99 step 2) { ... } // прогрессия
```

Интервалы и прогрессии

```
for (i in 1..10) { ... } // интервал
```

```
for (i in 10 downTo 1) { ... } // прогрессия
```

```
for (i in 1..99 step 2) { ... } // прогрессия
```

```
for (i in 100 downTo 2 step 2) { ... } // прогрессия
```

Пример: простые числа

- ▶ Число N простое, если у него ровно 2 делителя: 1 и N

Пример: простые числа

- ▶ Число N простое, если у него ровно 2 делителя: 1 и N
- ▶ Проверка на простоту: убедиться, что в интервале $2..N-1$ нет ни одного делителя

Пример: простые числа

- ▶ Число N простое, если у него ровно 2 делителя: 1 и N
- ▶ Проверка на простоту: убедиться, что в интервале $2..N-1$ нет ни **одного** делителя

```
fun isPrime(n: Int): Boolean {  
    if (n < 2) return false  
    for (m in 2..n - 1) {  
        if (n % m == 0) return false  
        else return true  
    }  
}
```

Пример: простые числа

- ▶ Число N простое, если у него ровно 2 делителя: 1 и N
- ▶ Проверка на простоту: убедиться, что в интервале $2..N-1$ нет ни одного делителя

```
fun isPrime(n: Int): Boolean {  
    if (n < 2) return false  
    for (m in 2..n - 1) {  
        if (n % m == 0) return false  
        else return true  
    }  
    return true  
}
```

Пример: простые числа

- ▶ Число N простое, если у него ровно 2 делителя: 1 и N
- ▶ Проверка на простоту: убедиться, что в интервале $2..N-1$ нет ни одного делителя

```
fun isPrime(n: Int): Boolean {  
    if (n < 2) return false  
    for (m in 2..n - 1) {  
        if (n % m == 0) return false  
    }  
    return true  
}
```

Прерывание и продолжение цикла

- ▶ Оператор **break** используется, если необходимо прервать выполнение цикла

Прерывание и продолжение цикла

- ▶ Оператор **break** используется, если необходимо **прервать** выполнение цикла
- ▶ Оператор **continue** используется, если необходимо завершить текущую **итерацию** цикла и **продолжить** цикл со следующей итерации

Совершенные числа: break / continue

- ▶ Совершенное число равно сумме всех своих делителей, кроме него самого

Совершенные числа: break / continue

- ▶ Совершенное число равно сумме всех своих делителей, кроме него самого
- ▶ $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$

Совершенные числа: break / continue

- ▶ Совершенное число равно сумме всех своих делителей, кроме него самого
- ▶ $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$

```
fun isPerfect(n: Int): Boolean {  
    var sum = 1  
    for (m in 2..n/2) {  
        if (n % m == 0) {  
            sum += m  
            if (sum > n) break  
        }  
    }  
    return sum == n  
}
```

Совершенные числа: break / continue

- ▶ Совершенное число равно сумме всех своих делителей, кроме него самого
- ▶ $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$

```
fun isPerfect(n: Int): Boolean {  
    var sum = 1  
    for (m in 2..n/2) {  
        if (n % m > 0) continue  
        sum += m  
        if (sum > n) break  
    }  
    return sum == n  
}
```

Циклы while

- ▶ **for** = цикл с фиксированным количеством итераций (не считая break / return)
- ▶ **while** = цикл с неопределённым количеством итераций

Циклы `while`: пример

- ▶ Сколько раз цифра M входит в число $N \geq 0$?

Циклы while: пример

- ▶ Сколько раз цифра M входит в число $N \geq 0$?

```
fun digitCountInNumber(n: Int, m: Int): Int {  
    var count = 0  
    var number = n  
    while (number > 0) {  
        if (m == number % 10) {  
            count++  
        }  
        number /= 10  
    }  
    return count  
}
```


Циклы while: пример

- ▶ Сколько раз цифра M входит в число $N \geq 0$?

```
fun digitCountInNumber(n: Int, m: Int): Int {  
    var count = 0  
    var number = n  
    do {  
        if (m == number % 10) {  
            count++  
        }  
        number /= 10  
    } while (number > 0)  
    return count  
}
```

while VS do while

- ▶ **while** = цикл с предусловием (тело может не выполниться ни разу)
- ▶ **do ... while** = цикл с постусловием (тело обязано выполниться хотя бы раз)
- ▶ Условие = ВСЕГДА условие продолжения

Рекурсивное решение

- ▶ Сколько раз цифра M входит в число $N \geq 0$?

Рекурсивное решение

- ▶ Сколько раз цифра M входит в число $N \geq 0$?

```
fun digitCountInNumber(n: Int, m: Int): Int =  
    if (n == m) 1  
    else if (n < 10) 0  
    else digitCountInNumber(n / 10, m) +  
         digitCountInNumber(n % 10, m)
```

Упражнения к лекции

- ▶ См. lesson3/task1 в обучающем проекте
- ▶ Решите хотя бы одно из заданий
- ▶ Протестируйте решение с помощью готовых тестов
- ▶ Добавьте ещё хотя бы один тестовый случай
- ▶ Попробуйте придумать рекурсивное решение хотя бы одной задачи
- ▶ Добавьте коммит в свой репозиторий
- ▶ Создайте Pull Request и убедитесь в правильности решения