Алгоритмизация и программирование

1. Простые функции

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Алгоритм

- Последовательность действий (обычно записанная формально), необходимая для решения определённой задачи

Программа

- Запись алгоритма на языке программирования (или в машинных кодах)

Программа

- Запись алгоритма на языке программирования (или в машинных кодах)
- В более широком смысле – инструкция для компьютера, позволяющая ему решить определённую задачу

Функция в программировании

- Участок программы, также решающий определённую задачу (часть или всю задачу, решаемую программой)

Функция в программировании

- Участок программы, также решающий определённую задачу (часть или всю задачу, решаемую программой)

- Имеет входы (параметры) и выход (результат)

Функция в программировании

- Участок программы, также решающий определённую задачу (часть или всю задачу, решаемую программой)
- Имеет входы (параметры) и выход (результат)
- Похожа на функцию в математике (но не полностью)

Простая функция на Котлине

```
fun sqr(x: Int) = x * x
```

- fun = ключевое слово
- sqr = имя функции
- x: Int = параметр функции
  - Int = тип параметра функции
- “= x * x” = тело функции
  - x * x = результат функции
    - * = операция

Операции в программировании

- Элементарные арифметические, логические и прочие действия, обозначаемые знаком операции = **оператором**

Операции в Котлине (арифметические)

- Сложение a + b
- Вычитание a – b
- Умножение a * b
- Деление a / b
- Остаток от деления a % b
- Скобки (…)

Типы в программировании

- Тип подобен области допустимых значений в математике

Типы в программировании

- Тип подобен области допустимых значений в математике
- Например, множеству целых чисел в математике соответствует целый тип в программировании

Типы в программировании

- Тип подобен области допустимых значений в математике
- Например, множеству целых чисел в математике соответствует целый тип в программировании
- Тип есть у параметра функции, результата функции, переменной, …

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$
- Double = вещественное число (примерно) $-1.7 * 10^{308}$ … $1.7 * 10^{308}$

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$
- Double = вещественное число (примерно) $-1.7 * 10^{308}$ … $1.7 * 10^{308}$
- Boolean = true или false, истина или ложь

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$
- Double = вещественное число (примерно) $-1.7 * 10^{308}$ … $1.7 * 10^{308}$
- Boolean = true или false, истина или ложь
- Char = символ (из таблицы Unicode), с кодом 0 … $2^{16} - 1$

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$
- Double = вещественное число (примерно) $-1.7 * 10^{308}$ … $1.7 * 10^{308}$
- Boolean = true или false, истина или ложь
- Char = символ (из таблицы Unicode), с кодом 0 … $2^{16} - 1$

Стандартные типы в Котлине

- Int = целое число -2^{31} … $2^{31} - 1$
- Double = вещественное число (примерно) $-1.7 * 10^{308}$ … $1.7 * 10^{308}$
- Boolean = true или false, истина или ложь
- Char = символ (из таблицы Unicode), с кодом 0 … $2^{16} - 1$
- String = строка = любое количество Char

Литералы (константы)

- Int : 42, -256
- Double : 1.0, 3.1415, 6.67e-11
- Boolean : true, false
- Char : 'a', 'z'
- String : "Hello", ""

Int против Double

- Int : точный тип, Double : приближённый

Int против Double

- Int : точный тип, Double : приближённый
- Диапазон значений Double гораздо шире

Int против Double

- Int : точный тип, Double : приближённый
- Диапазон значений Double гораздо шире
- Операции над целыми числами дают целый результат: 5 / 2 = 2
- Операции над вещественными числами дают вещественный результат:
  5.0 / 2.0 = 2.5, 5 / 2.0 = 2.5

Int против Double

- Int : точный тип, Double : приближённый
- Диапазон значений Double гораздо шире
- Операции над целыми числами дают целый результат: 5 / 2 = 2
- Операции над вещественными числами дают вещественный результат:
  5.0 / 2.0 = 2.5, 5 / 2.0 = 2.5
- Преобразования: n.toDouble(), x.toInt()

Имена в программировании

- Используются, чтобы различать различные элементы программы
- Есть у функций, параметров, переменных, типов, …

Имена в Котлине

- Обязаны начинаться с буквы
  (или с символа _ что не рекомендуется)
- Состоят из букв, цифр, символа _
- Прописные и строчные буквы различаются

Рекомендуемые правила формирования имён (Java, Kotlin)

- Только латинские буквы, никаких **моё_имя_123**
- Нет транслитерации, никаких dlinaOtrezka

Рекомендуемые правила формирования имён (Java, Kotlin)

- Только латинские буквы, никаких **моё_имя_123**
- Нет транслитерации, никаких dlinaOtrezka

- Имена функций, параметров, переменных начинаются со строчной буквы: segmentLength или calculateArea
- Имена типов начинаются с прописной: Rectangle

Имена параметров и функций

- Параметр – это объект, существительное
- Функция – это действие, глагол

Готовые математические функции (Java, Kotlin)

- Пакет (package) Math
  - abs(x: Int), abs(x: Double) – модуль
  - sqrt(x: Double) – квадратный корень
  - pow(x: Double, y: Double) – x в степени y
  - sin/cos/tan(x: Double) – синус / косинус / тангенс, аргумент задаётся в радианах
  - exp(x: Double) – e в степени x
  - log / log10(x: Double) – натуральный и десятичный логарифмы
  - min / max(x: Int) или (x: Double) – минимум и максимум из двух чисел
  - PI = 3.14…, E = 2.72…

Использование готовых функций

```
// Комментарий: дискриминант
fun discriminant(a: Double, b: Double, c: Double) =
    sqr(b) - 4 * a * c
// sqr(b) = вызов функции sqr
//      b = аргумент функции sqr
```

Использование готовых функций

```
// Комментарий: дискриминант
fun discriminant(a: Double, b: Double, c: Double) =
    sqr(b) - 4 * a * c
// Комментарий: корень квадратного уравнения
fun sqRoot(a: Double, b: Double, c: Double) =
    (-b + Math.sqrt(discriminant(a, b, c))) / (2 * a)
// Math.sqrt(…) – вызов функции sqrt из пакета Math
// discriminant(a, b, c) – вызов функции discriminant
// и одновременно аргумент функции sqrt
```

Короткие и полные имена

- Math.sqrt = полное имя (с указанием пакета)
  - В IDEA можно сократить автоматически: Alt+Enter
- sqrt = короткое имя

```
// Директива импорта – в верхней части файла
import java.lang.Math.sqrt

// …
fun sqRoot(a: Double, b: Double, c: Double) =
    (-b + sqrt(discriminant(a, b, c))) / (2 * a)
```

Промежуточные переменные

```
fun quadraticRootProduct(
    a: Double, b: Double, c: Double
): Double /* тип результата */ {
    // Тело в виде блока
}
```

Промежуточные переменные

```
fun quadraticRootProduct(
    a: Double, b: Double, c: Double
): Double /* тип результата */ {
    // Тело в виде блока
    // val = промежуточная переменная
    val sd = sqrt(discriminant(a, b, c))
}
```

Промежуточные переменные

```
fun quadraticRootProduct(
    a: Double, b: Double, c: Double
): Double /* тип результата */ {
    // Тело в виде блока
    // val = промежуточная переменная
    val sd = sqrt(discriminant(a, b, c))
    // Ещё две переменных
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
}
```

Промежуточные переменные

```
fun quadraticRootProduct(
    a: Double, b: Double, c: Double
): Double /* тип результата */ {
    // Тело в виде блока
    // val = промежуточная переменная
    val sd = sqrt(discriminant(a, b, c))
    // Ещё две переменных
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
    // Чему равен результат?
    return x1 * x2
}
```

Новые ключевые слова

- val = определение промежуточной переменной
- return = **оператор** возврата (вычисления результата)

Вывод на консоль

```
fun solveQuadraticEquation(
    a: Double, b: Double, c: Double
) /* no result */ {
    val sd = sqrt(discriminant(a, b, c))
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
}
```

Вывод на консоль

```
fun solveQuadraticEquation(
    a: Double, b: Double, c: Double
) /* no result */ {
    val sd = sqrt(discriminant(a, b, c))
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
    // Вывод на экран значений x1 и x2
    println(x1)
    println(x2)
}
```

Вывод на консоль

```
fun solveQuadraticEquation(
    a: Double, b: Double, c: Double
) /* no result */ {
    val sd = sqrt(discriminant(a, b, c))
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
    // Вывод на экран значений x1 и x2
    println(x1)
    println(x2)
    // Вывод на экран строки вида x1 = 3.0 x2 = 2.0
    println("x1 = $x1 x2 = $x2")
}
```

Вывод на консоль

```
fun solveQuadraticEquation(
    a: Double, b: Double, c: Double
) /* no result */ {
    val sd = sqrt(discriminant(a, b, c))
    val x1 = (-b + sd) / (2 * a)
    val x2 = (-b - sd) / (2 * a)
    // Вывод на экран значений x1 и x2
    println(x1)
    println(x2)
    // Вывод на экран строки вида x1 = 3.0 x2 = 2.0
    println("x1 = $x1 x2 = $x2")
    // Вывод на экран произведения корней
    println("x1 * x2 = ${x1 * x2}")
}
```

Тестовые функции

- Особый вид функций
- Контролируют правильность работы других функций
- Обычно реализуются на основе специальных тестирующих библиотек (пример = JUnit)

Пример тестовой функции

```
// Test = аннотация
@Test
fun testSqr() {
    // Проверить, что квадрат нуля это 0
    assertEquals(0, sqr(0))
    // Проверить, что квадрат двух это 4
    assertEquals(4, sqr(2))
    // Проверить, что квадрат -3 это 9
    assertEquals(9, sqr(-3))
}
```

assertEquals

- Проверка на равенство
  - Ничего не делает, если аргументы равны
  - Прекращает тест с ошибкой, если аргументы не равны

Главная функция

- «Точка входа» в программу = отсюда программа начинает свою работу
- Во многих языках называется **main**

Пример главной функции

```
fun main(args: Array<String>) {
    // Решаем x^2 - 3*x + 2 = 0
    val x1x2 = quadraticRootProduct(1.0, -3.0, 2.0)
    println("Root product: $x1x2")
}
```

Упражнения к лекции

- См. lesson1/task1 в обучающем проекте
- Решите хотя бы одну из задач
- Протестируйте решение
- Добавьте коммит в свой репозиторий
- Создайте Pull Request и убедитесь в правильности решения
- Попробуйте написать главную функцию, использующую функцию, написанную вами