Алгоритмизация и программирование

4.2. Функции высшего порядка

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>): List<Int> {
    val result = mutableListOf<Int>()
    for (element in list) {
        if (element < 0) {
            result.add(element)
        }
    }
    return result
}
```

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>) =
    list.filter { it < 0 }
```

Функция высшего порядка

- Функция, параметром которой является другая функция
- Пример: list.filter

Функция высшего порядка

- Функция, параметром которой является другая функция
- Пример: list.filter
  - Параметр: f: (T) -> Boolean (предикат)

Функция высшего порядка

- Функция, параметром которой является другая функция
- Пример: list.filter
  - Параметр: f: (T) -> Boolean (предикат)
  - Результат: список, в котором остаются только элементы, для которых предикат **истинен (true)**

Функция высшего порядка

- Функция, параметром которой является другая функция
- Пример: list.filter
  - Параметр: f: (T) -> Boolean (предикат)
  - Результат: список, в котором остаются только элементы, для которых предикат **истинен (true)**
- List<T>.filter(f: (T) -> Boolean): List<T>

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>) =
    list.filter(fun(it: Int) = it < 0)
```

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>) =
    list.filter({ it: Int -> it < 0 })
```

Лямбда

- Сокращённая форма записи функции
- Пример: `{ it: Int -> it < 0 }`
  - it: Int параметр функции
  - it < 0 тело функции

Лямбда

- Сокращённая форма записи функции
- Пример: `{ it: Int -> it < 0 }`
  - it: Int параметр функции
  - it < 0 тело функции
- Ещё короче
  - { it -> it < 0 }

Лямбда

- Сокращённая форма записи функции
- Пример: `{ it: Int -> it < 0 }`
  - it: Int параметр функции
  - it < 0 тело функции
- Ещё короче
  - { it -> it < 0 }
- И ещё короче
  - { it < 0 }

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>) =
    list.filter({ it < 0 })
```

Лямбда аргумент

- Если лямбда – последний аргумент, вместо
  - `list.filter({ it < 0 })`

Лямбда аргумент

- Если лямбда – последний аргумент, вместо
  - `list.filter({ it < 0 })`
- разрешается
  - `list.filter() { it < 0 }`

Лямбда аргумент

- Если лямбда – последний аргумент, вместо
  - `list.filter({ it < 0 })`
- разрешается
  - `list.filter() { it < 0 }`
- Или даже
  - `list.filter { it < 0 }`

Пример: фильтрация

- Получение всех отрицательных чисел из исходного списка

```
fun negativeList(list: List<Int>) =
    list.filter { it < 0 }
```

Пример: преобразование чисел в квадраты чисел

```
fun squares(list: List<Int>) =
    list.map { it * it }
```

Функция отображения

- List<T>.map(f: (T) -> R): List<R>
- Из List<T> делает List<R> того же размера

Пример: проверка на простоту

```
fun isPrime(n: Int) =
    n >= 2 && (2..n/2).all { n % it != 0 }
```

Функция «для всех»

- List<T>.all(f: (T) -> Boolean): Boolean
- Проверяет, что предикат f верен для всех элементов списка

Пример: проверка на не-простоту

```
fun isNotPrime(n: Int) =
    n < 2 || (2..n/2).any { n % it == 0 }
```

Функция «для какого-то»

- List<T>.any(f: (T) -> Boolean): Boolean
- Проверяет, что предикат f верен хотя бы для одного списка

Пример: произведение

```
fun multiplyAll(list: List<Double>) =
    list.fold(1.0) { prev, elem -> prev * elem }
```

Функция свёртки

- List<T>.fold(initial: R, f: (R, T) -> R): R
- «Сворачивает» список List<T> в одно значение типа R

Функция свёртки

- List<T>.fold(initial: R, f: (R, T) -> R): R
- «Сворачивает» список List<T> в одно значение типа R

- Берём initial, делаем var result = initial

Функция свёртки

- List<T>.fold(initial: R, f: (R, T) -> R): R
- «Сворачивает» список List<T> в одно значение типа R

- Берём initial, делаем var result = initial
- Для каждого элемента списка elem
  - Делаем result = f(result, elem)

В целом

- Функции высшего порядка помогают проще записывать некоторые преобразования (например, над списками)

В целом

- Функции высшего порядка помогают проще записывать некоторые преобразования (например, над списками)
- НО: требуют определённой привычки

Упражнения к лекции

- См. lesson4/task1 в обучающем проекте
- Попробуйте решить хотя бы одно из заданий с помощью функций высшего порядка