



Онлайн-образование

Проверить, идет ли запись!



Меня хорошо видно && слышно?

Ставьте +, если все хорошо

Напишите в чат, если есть проблемы

Парадигмы программирования



Дмитрий Кириллов

Технический директор

1С-Старт

@esteps_kirillov

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #php-2022-01 или #general



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара

01. Основы ООП



02. Основы ФП



03. Работа со списками



04. Реактивность

Цели вебинара | После занятия вы сможете

1

Выбрать подходящую парадигму
для решения конкретной задачи

2

Выполнять типовые операции
над списками

3

Разобраться с реактивным
программированием

Введение

Вопросы на засыпку:

Что такое парадигма?

Устоявшаяся, общепринятая система идей и понятий.

Упрощённо говоря, парадигма – это стиль, в котором написана программа.

Введение

Вопросы на засыпку:

О каких парадигмах программирования вы слышали?

- объектно-ориентированная
- функциональная
- процедурная
- логическая
- аспектно-ориентированная...

Зачем изучать разные парадигмы

- на большинстве языков можно писать в смешанном стиле
- есть ряд задач, которые лучше решаются средствами функционального, логического, реактивного и т.д. программирования

Введение

Учебная задача: формирование прайс-листа

- нам дали массив товаров
- берём из него только те товары, которые есть в наличии
- группируем их по категориям
- для каждой категории выводим табличку с товарами

Прайс-лист

Колбасные изделия

Товар	Кол-во	Цена	Сумма
Колбаса докторская	14	120,75 ₽	1 690,50 ₽
Сосиски молочные	53	245,50 ₽	13 011,50 ₽

Ликёро-водочные изделия

Товар	Кол-во	Цена	Сумма
Кагор №32	100	350,00 ₽	35 000,00 ₽
Пиво Бархатное	2	90,00 ₽	180,00 ₽
Водка Столичная	1	400,00 ₽	400,00 ₽

Структурная парадигма

- самая простая для понимания
- часто используется в начальном обучении
- разбиваем задачу на шаги
- реализуем шаги максимально простым образом

Введение

Три базовых конструкции структурной парадигмы

- последовательность
- ветвление
- ЦИКЛ

Практика: разработка кода

- 1** Я – ваш junior-разработчик, и мне нужно разработать прайс-лист в **структурном** стиле
- 2** Ваша задача: помочь мне написать нужный код
- 3** Критерии оценки: работоспособность и осмысленность кода



Тайминг: 5–10 минут

Введение

Вопросы на засыпку:

Какие недостатки вы заметили у структурной парадигмы?

01. Основы ООП

Минутка истории

- 1972: Си
- 1980: С++
- 1995: Java
- 2001: C#
- 2004: PHP 5

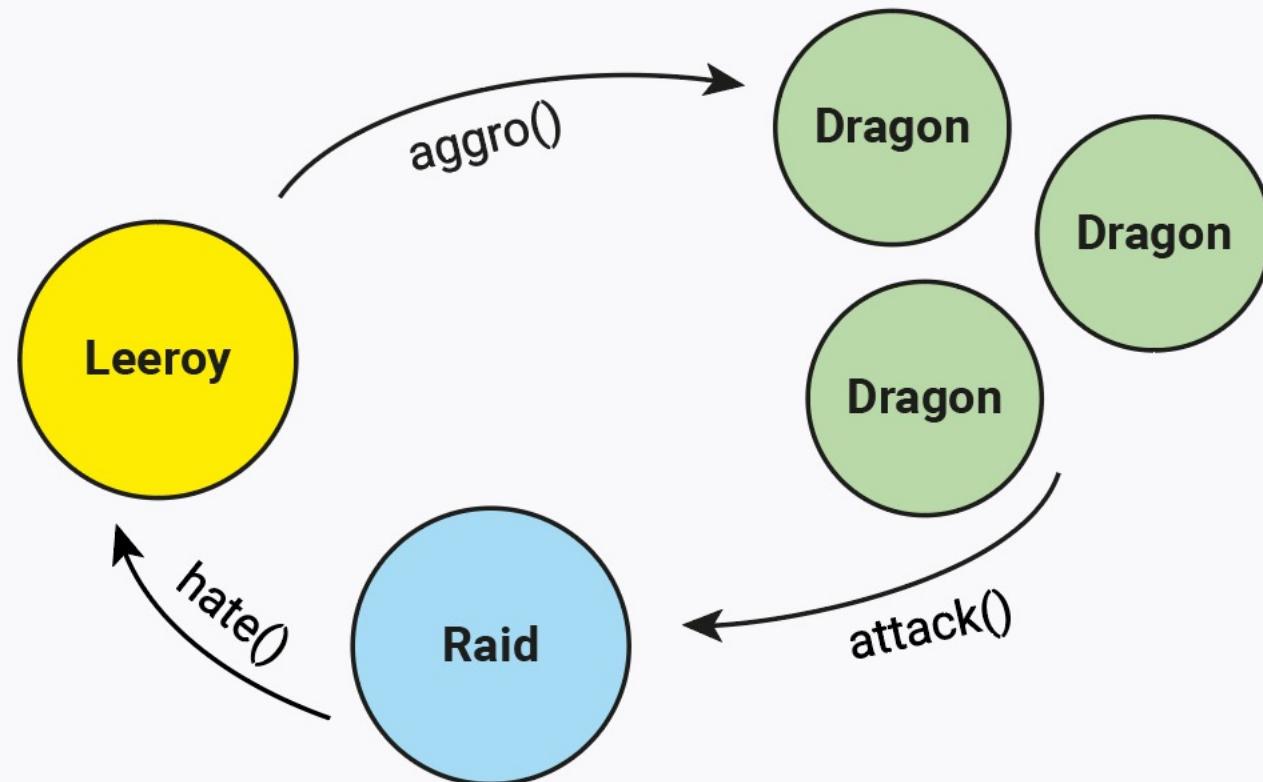
ООП: архитектура

- составные «кирпичики» программы — **объекты**
- каждый «кирпичик» можно описать существительным

ООП: взаимодействие

- у объектов есть внутреннее **состояние**
- объекты посылают друг другу **сообщения**
- реагируя на сообщение, объект
может изменить своё состояние

Схема взаимодействия элементов: ООП



Ключевые концепции ООП*

- абстракция
- инкапсуляция
- наследование
- полиморфизм

* не во всех ООП-языках

Строительные блоки ООП*

- классы
- интерфейсы
- объекты
- трейты

* тоже не во всех ООП-языках

Почему важны классы (и интерфейсы)

- у каждого объекта есть тип
- он всегда совпадает с именем класса
- у объекта может быть одновременно несколько типов
(наследование + интерфейсы)

На что это влияет:

- подсказки типов (type hints) снижают кол-во ошибок
- полиморфизм: можно **подменять одни объекты другими**, если их типы совпадают

Практика: разработка кода

- 1** Я – ваш junior-разработчик, и мне нужно разработать прайс-лист в ООП-стиле
- 2** Ваша задача: помочь мне написать нужный код
- 3** Критерии оценки: работоспособность и осмысленность кода



Тайминг: 10–15 минут

ООП: преимущества

- относительно невысокий порог входа
- огромное количество готовых решений
- отработанные методологии совместной разработки
- возможность говорить с бизнесом на одном языке (DDD)

ООП: недостатки

- абстракция: существительные плохо описывают задачу (нам важнее, "что сделать", а не "из чего сделать")*
- инкапсуляция: сложно отслеживать состояние всей системы в целом, оно может стать некорректным
- наследование: код "размазан" по куче классов, это провоцирует ошибки
- от программиста требуется высокая самодисциплина (как следствие — паттерны, принципы SOLID и т.д.)

* следствие: фабрики фабрик

02. Основы ФП

Минутка истории

- 1958: LISP
- 1986: Erlang
- 1990: Haskell
- 2004: Scala

Важное историческое отличие

- ООП-языки – в основном коммерческие
- ФП-языки – в основном академические

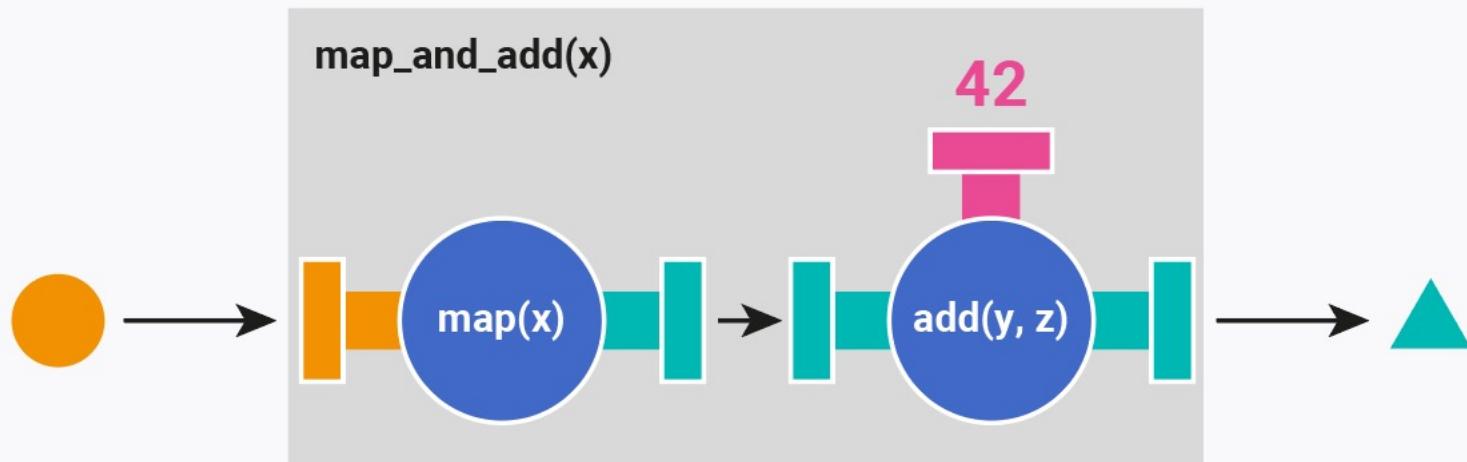
ФП: архитектура

- составные «кирпичики» программы — **функции**
- каждый «кирпичик» можно описать глаголом

ФП: взаимодействие

- функция просто обрабатывает данные и возвращает результат
- сложные функции комбинируются из более мелких
- вся программа — это цепочка вызовов функций,
в основном — **без состояния**

Схема взаимодействия элементов: ФП



Основные понятия ФП

- иммутабельность
- функции
- чистые функции
- функции первого класса
- функции высшего порядка
- анонимные функции

Иммутабельность

Иммутабельность

- если переменная создана, её значение поменять уже нельзя
- в PHP есть иммутабельные встроенные классы
- в PHP 8.1 появились `readonly`-свойства

Почему это важно:

- снижается количество ошибок
- упрощается конкурентное программирование

Функции

Анатомия функции

```
Имя функции →      Типы и имена параметров →      Тип возвр. значения →  
function sum (int $a, int $b): int {  
    return $a + $b;  
}  
↑  
Тело функции
```

Функции

Анатомия функции

- **параметры**: то, что объявлено в заголовке функции
- **аргументы**: фактические значения при вызове функции

```
function calc(int $a): int // Здесь $a - параметр
{
    if ($a === 1)...           // А здесь $a - аргумент
}
```

Функции

Анатомия функции

- унарная функция: 1 параметр
- бинарная функция: 2 параметра

```
// Унарная функция
function calc(int $a): int {...}
```

```
// Бинарная функция
function sum(int $a, int $b): int {...}
```

Чистые функции

Чистые функции

- не изменяют **внешние** переменные
- не занимаются вводом-выводом
- если вызывать их снова с теми же аргументами —
возвращают такое же значение

Почему это важно:

- эти функции легко тестировать
- эти функции легко кэшировать

Вопросы на засыпку:

Назовите любую **бинарную** функцию, "встроенную" в PHP?

Приведите пример **полезной** функции,
которая будет при каждом вызове возвращать новый результат?

Функции первого класса

Функции первого класса

- обычно мы передаём в функции строки, числа...
- в ФП можно передавать также **другие функции**
- по сути, функция просто упаковывается в переменную

Почему это важно:

- мы можем изменять поведение функций "на лету"
- это эквивалентно внедрению зависимостей в ООП

Функции первого класса

Функции первого класса в PHP

- функция может храниться в переменной

```
$compareByPrice = function(Product $p1, Product $p2): int {  
    return ($p1->getPrice() <=> $p2->getPrice());  
};
```

- тип такой переменной — **callable** (Closure)
- эту переменную можно передать в другую функцию

```
function sortProducts(array $list, callable $comparator): array {  
    ...  
}
```

Функции первого класса

Функции высшего порядка

- умеют **принимать** другие функции (как параметры)
- умеют **возвращать** другие функции
- в PHP нельзя указать сигнатуру параметров или возвращаемых значений

Почему это важно:

- можно создавать **абстрактные** функции
- это во многом эквивалентно принципу OCP из SOLID

Функции высшего порядка

Функции высшего порядка

```
function sortProducts(array $list, callable $comparator): array {  
    $copy = $list;  
    usort($copy, $comparator);  
    return $copy;  
}  
  
sortProducts($products, $compareByPrice);  
sortProducts($products, $compareByTitle);  
sortProducts($products, $compareByStockAndPrice);
```

Вопросы на засыпку:

Попробуйте вспомнить системные функции в PHP,
для работы которых требуются другие функции?

Анонимные функции

Анонимные функции

Есть 2 основных способа передать аргумент в функцию:

- как переменную:

```
$a = 2;  
$b = 3;  
echo add($a, $b); // 5
```

- или как литерал (т.е. фиксированное **анонимное** значение):

```
echo add(2, 3); // 5
```

Анонимные функции

Анонимные функции

Точно так же внутрь функции можно передавать и другую функцию:

- как переменную:

```
setCacheIfEmpty($product, $slowFunc);
```

- или как **анонимный** литерал:

```
setCacheIfEmpty(  
    $product,  
    function(Product $p): string {...}  
);
```

Анонимные функции

Анонимные функции: практика использования

- «одноразовые» операции (сортировка, фильтрация...)
- как правило, очень короткие по размеру
- рекомендуется объявлять их статическими (static)*

```
$drinks = array_filter(  
    $products,  
    static function(Product $p): bool {  
        return ($p instanceof AlkoholProduct);  
    }  
);
```

* если не нужен \$this

Промежуточные итоги

- функцию можно хранить в переменной
- эту переменную можно передать в другую функцию
- из функции может вернуться другая функция
- анонимные функции можно писать прямо в коде
(вместо имени переменной)

03. Работа со списками

Списки в ФП

- содержат элементы **одного** типа (например, только целые числа)
- обозначаются [A], где A – тип элемента списка:
[String], [Int]...

В PHP на практике используются обычные (индексированные) массивы, которые обрабатываются функциями:

- array_filter()
- array_map()
- array_reduce()

Фильтрация списка

Фильтрация списка: `array_filter()`

Какую задачу решает:

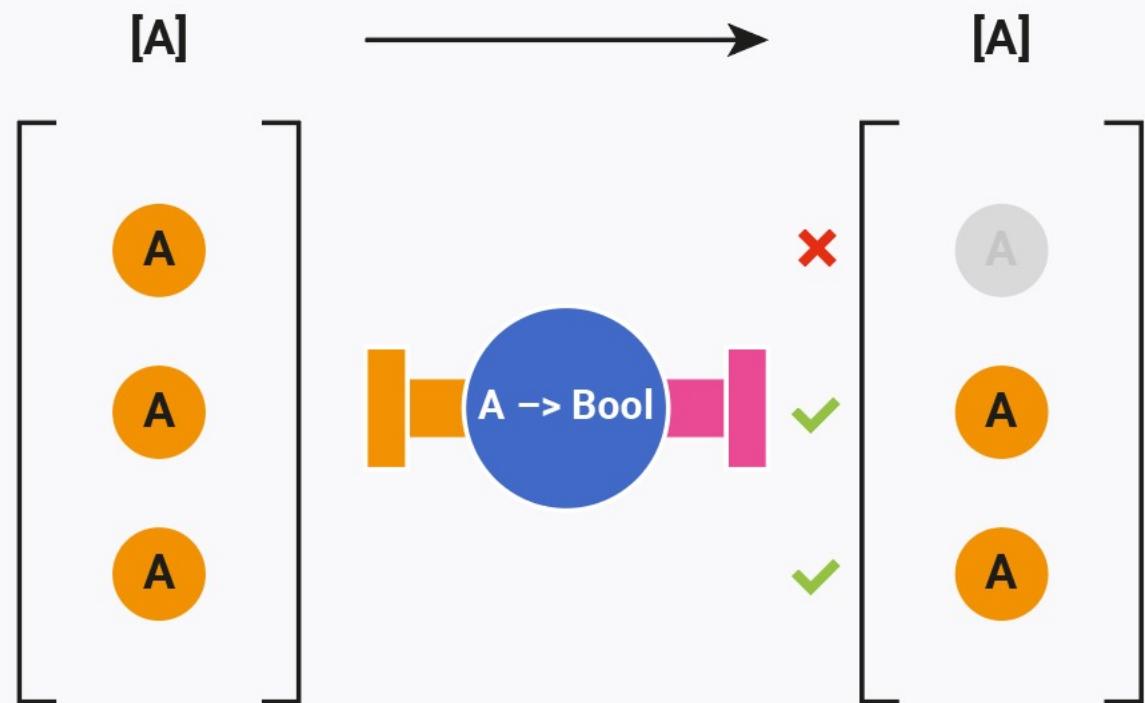
- у нас есть список
- мы хотим оставить в нём только часть значений по какому-то критерию (например, только чётные числа)

Как работает:

- отфильтровывает нужные значения из списка [A]
- требует для работы функцию-предикат ($A \rightarrow \text{Bool}$)
- возвращает **новый** список [A] из 0..N значений

Фильтрация списка

Фильтрация списка: array_filter()



Фильтрация списка

Фильтрация списка: array_filter()

[A]	A > 3	[A]
1	False	
4	True	4
5	True	5

Фильтрация списка

Фильтрация списка: array_filter()

Мини-практика

Дано: список товаров.

Задача: оставить только те товары,
которые есть в наличии.

Отображение списка

Отображение списка: array_map()

Какую задачу решает:

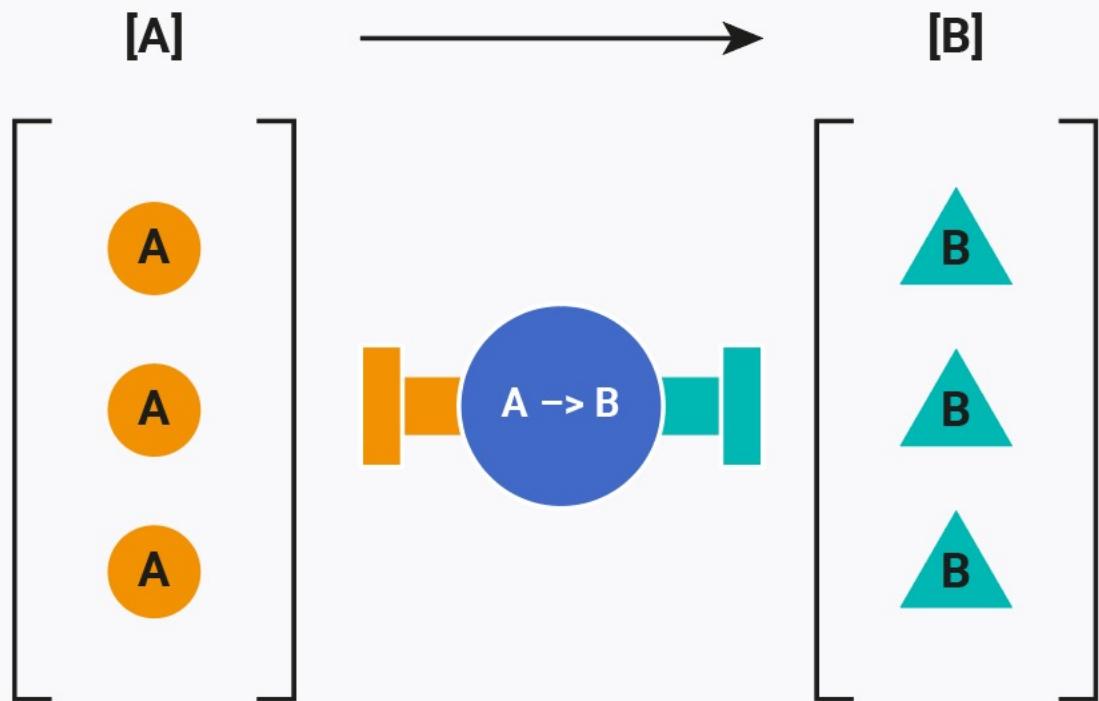
- у нас есть список
- мы хотим превратить **каждый** его элемент во что-то другое
(например, заменить все строки на кол-во символов в них)

Как работает:

- преобразует элементы списка [A] в [B]
- требует для работы унарную функцию ($A \rightarrow B$)
- возвращает **новый** список [B] строго из N значений

Отображение списка

Отображение списка: array_map()



Отображение списка

Отображение списка: array_map()

[A]	strlen(A)	[B]
"a"	1	1
"abc"	3	3
"qwerty"	6	6

Отображение списка

Отображение списка: array_map()

Мини-практика

Дано: список товаров.

Задача: вернуть список товаров с добавленным полем "Сумма".

Свёртка списка

Свёртка списка: array_reduce()

Какую задачу решает:

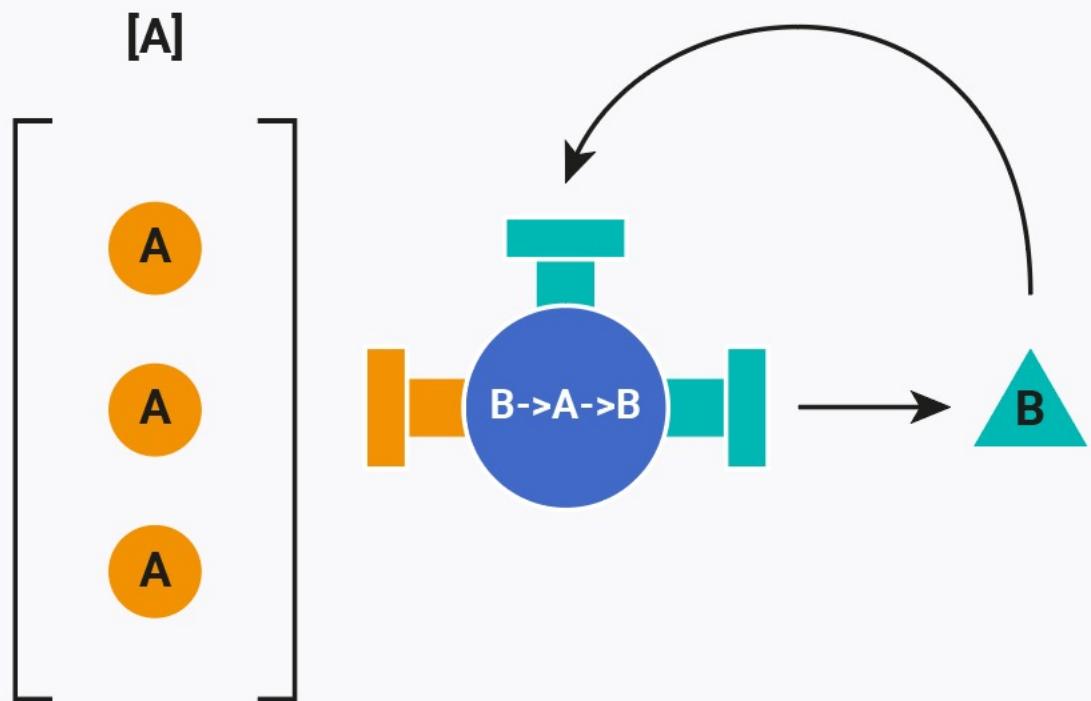
- у нас есть список
- мы хотим вычислить какое-то одно значение на его основе
(например, сумму всех его элементов)

Как работает:

- поочерёдно сводит **все** элементы списка [A] к **одному** значению B
- требует бинарную функцию ($B \rightarrow A \rightarrow B$) + начальное значение B
- возвращает значение **произвольного** типа B (число, строку, список...)

Свёртка списка

Свёртка списка: array_reduce()



Свёртка списка

Свёртка списка: array_reduce()

[A]	B + A	B
1	0+1	1
5	1+5	6
10	6+10	16

Свёртка списка

Свёртка списка: array_reduce()

Мини-практика

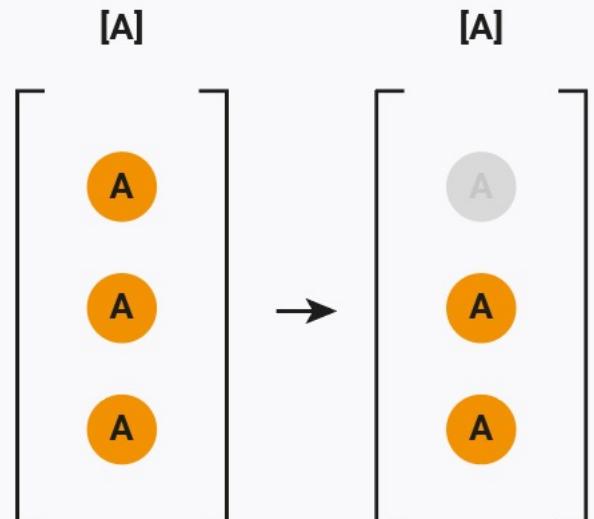
Дано: список товаров.

Задача: сгруппировать товары по категории.

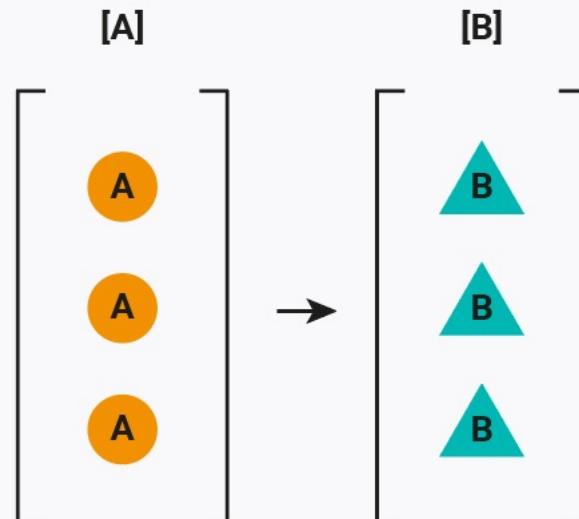
Работа со списками

Сравнение функций для работы со списками

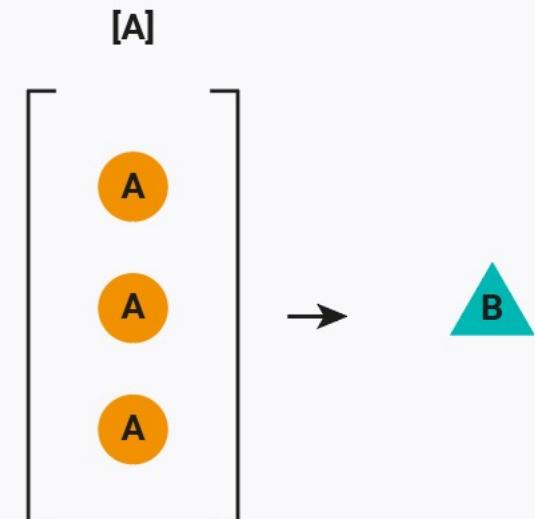
Фильтрация
(filter)



Отображение
(map)

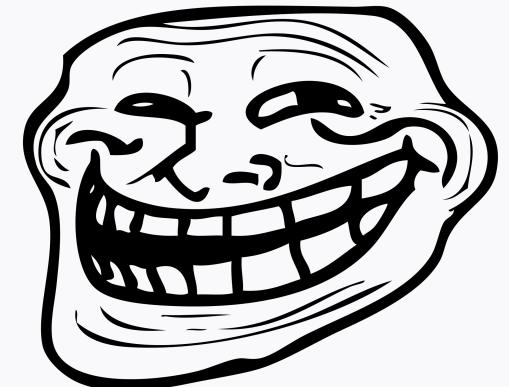


Свёртка
(fold / reduce)



Мнение «иксперда»

*Зачем вы понапридумывали
эти функции с функциями?
Foreach() forever!*



ФП: преимущества

- естественный подход к решению задач ("что делаем")
- более надёжный код
- не нужны тестовые двойники
- большинство задач отлично параллелится

ФП: недостатки

- высокие требования к квалификации разработчиков
- небольшое количество готовых коммерческих решений
- сложные компиляторы и сборщики мусора
- проблемы на стыке чистых и "грязных" функций

04. Реактивность

Реактивное программирование: мотивация

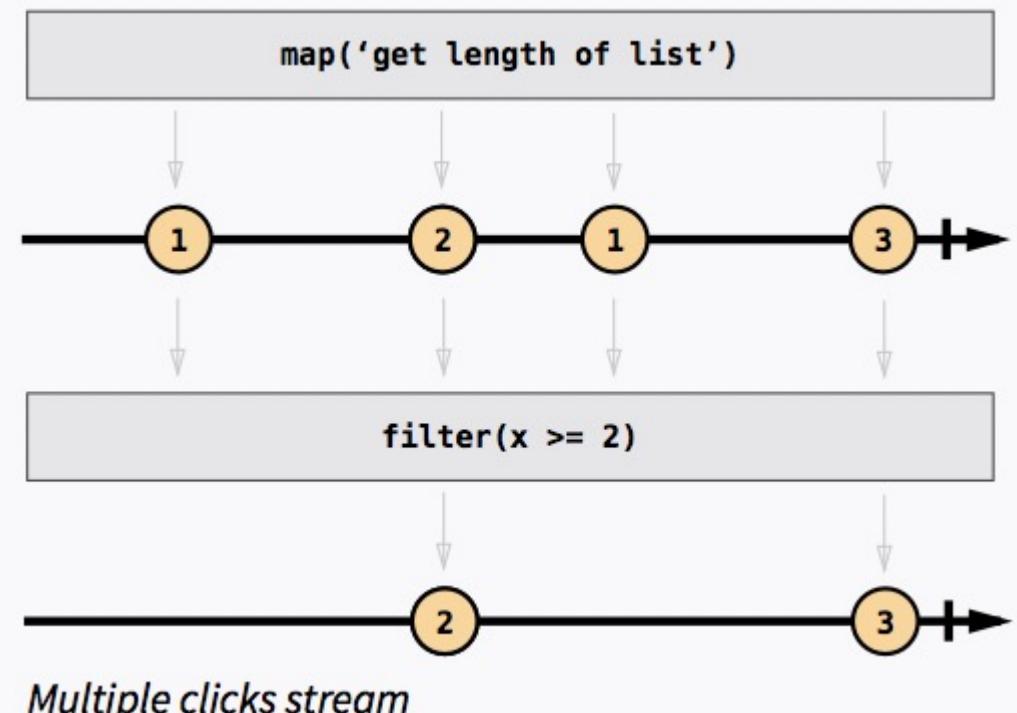
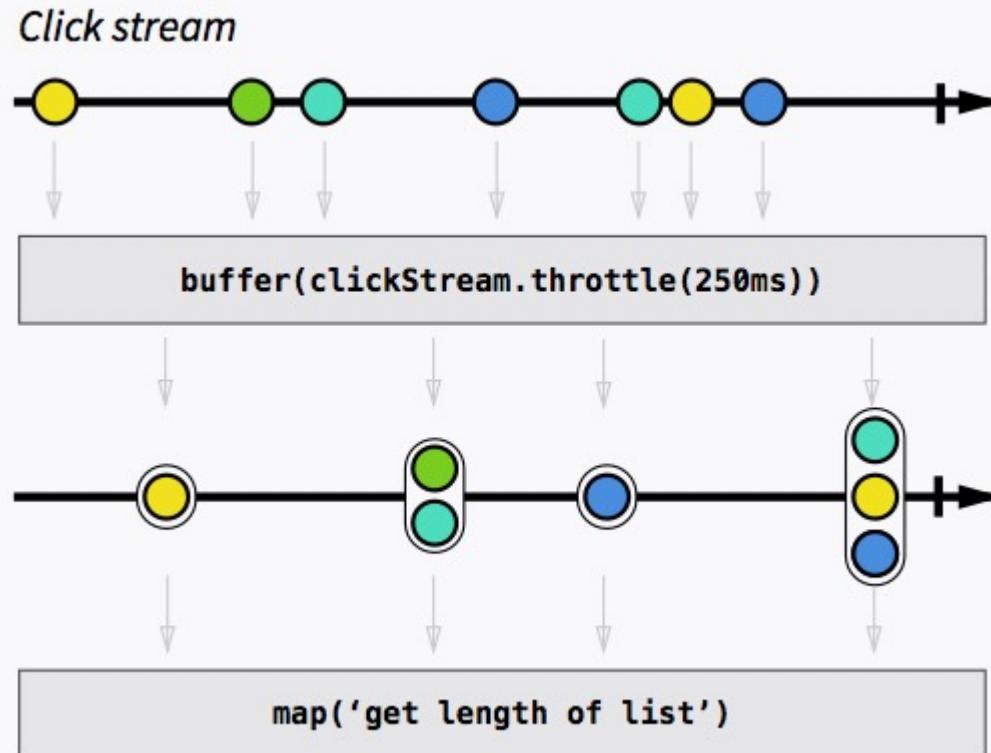
- у нас есть данные, которые динамически изменяются
- мы хотим автоматически реагировать на эти изменения
- для этого мы **подписываемся** на них
- пример: изменение значений в ячейках Excel

Реактивное программирование: архитектура

- во многом развивает идеи ФП
- данные представлены в виде **потоков**, похожих на списки
- над потоками можно выполнять операции map, filter, reduce...
- каждая такая операция возвращает новый поток
- программа реагирует на **изменения** потоков

Реактивность

Пример: реакция на двойные щелчки мышью



Практика: разработка кода

- 1 Я – ваш junior-разработчик, и мне нужно разработать прайс-лист в **реактивном** стиле
- 2 Ваша задача: помочь мне написать нужный код
- 3 Критерии оценки: работоспособность и осмысленность кода



Тайминг: 10–15 минут

Цели вебинара | Проверка достижения целей

1

Выбрать подходящую парадигму
для решения конкретной задачи

2

Выполнять типовые операции
над списками

3

Разобраться с реактивным
программированием

Домашнее задание

- 1 Это необязательное д/з на закрепление материала, в ЛК его нет
- 2 Попробуйте решить две задачи с реальных собеседований в компании уровня FAANG
- 3 Задачи будут выложены в материалах к вебинару; решения высыпайте в Slack



Срок: желательно сдать до 07.04

Следующий вебинар

Тема: Архитектура кода



Понедельник, 4 апреля, в 20:00



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию
в ЛК – можно
изучать



Обязательный
материал обозначен
красной лентой



Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие вебинары



Дмитрий Кириллов

Технический директор

1С-Старт

@esteps_kirillov