36. Функциональное программирование

Цель:

Познакомиться со следующими понятиями:

- функциональное программирование
- чистые функции
- иммутабельность
- map, reduce
- Big O notation

Парадигмы программирования:

Императивное программирование — это описание того, **как** ты делаешь что-то.

Декларативное программирование — это описание того, **что** ты делаешь.

Парадигмы программирования:

Императивный подход (как): Сначала необходимо купить продуктов в магазине, затем очистить картофель, порезать лук и картофель, посолить и поперчить мясо и ……

Декларативный подход (что): Хочу мясо с жареной картошкой

Функциональное программирование (ФП):

По распространенности ФП занимает 2 место после ООП.

Языки программирования, которые чаще всего выбираются для изучения:

- Clojure
- Haskell
- Scala
- Elixir
- Elm
- F#
- Idris и др

Основные принципы ФП:

1. Все функции — чистые
2. Все функции — первого класса и высшего порядка
3. Иммутабельность
4. Прозрачность функций
5. ФП основано на лямбда-исчислении

Чистая функция:

Все функции являются чистыми, если они удовлетворяют двум условиям:

1. Функция, вызываемая от одних и тех же аргументов, всегда возвращает одинаковое значение.
2. Во время выполнения функции не возникают побочные эффекты.

Пример “нечистой” функции

Побочные эффекты: переменной вне функции, вывод в консоль, вызов исключения, чтение данных из файла и тд.

```
const generateID = () => Math.floor(Math.random() * 10000);
```

Пример “нечистой” функции

```
const MAX_RATE = 25;

const totalPrice = quantity => MAX_RATE * quantity;
```

Пример “нечистой” функции

```
const createUser = username => ({
  id: ++id,
  username
});
```

Пример “нечистой” функции

```
const showMessage = msg => {
  console.log(msg);
};

showMessage("Hello TMS!");
```

Все функции — первого класса и высшего порядка

Для того, чтобы **функция** была **первого класса**, у нее должна быть возможность быть объявленной в виде переменной. Это позволяет управлять функцией как обычным типом данных и в то же время исполнять ее.

```
const sum = (a, b) => a + b;

sum(2, 4);
```

Все функции — первого класса и высшего порядка

Функции высшего порядка определяются как функции, которые принимают другую функцию как аргумент или возвращают функцию функцию.

Например: map и filter.

Иммутабельность

В ФП нельзя изменять переменную после ее инициализации. Можно создавать только новые, но не нельзя изменять существующие.

Прозрачность функций

Эта особенность означает, что мы можем заменить вызов функции соответствующим значением и наша программа(состояние) не изменится.

ФП основано на лямбда-исчислении

ФП сильно опирается на математическую систему, называющуюся лямбда-исчислением.

Есть 2 ключевых момента:

- В лямбда-исчислении все функции могут быть анонимными, поскольку единственная значимая часть заголовка функции — это список аргументов.
- При вызове все функции проходят процесс каррирования.

Каррирование

```
const curry = (f) => {
  return (a) => {
    return (b) => {
      return f(a, b);
    };
  };
}

const sum = (a, b) => a + b;

const curriedSum = curry(sum);

alert( curriedSum(5)(2) ); // 7
```

Преимущества ФП

- применение принципов ФП помогает снизить сложность кода
- упрощает тестирование за счет использования чистых функций
- повышение читаемости кода. Применение чистых значений предполагает их неизменное состояние до самого конца

Map

Маппинг — фундаментальная методика в ФП. Она применяется для оперирования всеми элементами массива с целью создания другого массива той же длины, но с преобразованным содержимым.

```
const numbers = [1, 4, 9];

const doubles = numbers.map((num) => num * 2);
```

Map

Маппинг — фундаментальная методика в ФП. Она применяется для оперирования всеми элементами массива с целью создания другого массива той же длины, но с преобразованным содержимым.

```
const numbers = [1, 4, 9];

const doubles = numbers.map((num) => num * 2);
```

Reduce

Метод **reduce()** применяет функцию к каждому элементу массива (слева-направо), возвращая одно результирующее значение. Результатом reduce может быть любой тип данных.

```
const numbers = [1, 4, 9];

const sum = numbers.reduce(
  (previousValue, currentValue) => previousValue + currentValue, 0);
```

Big O notation

Big O нотация нужна для описания сложности алгоритмов.