

# Именованя

Автор последнего обновления: Pavel.Tokarenko@evraz.com | 4 окт. 2023 г. в 12:11 GMT+3

Имена должны передавать намерения программиста. Имя переменной, функции, класса должно отвечать на все главные вопросы: почему существует, что делает и как используется.

*плохо*

```
let d: number; // Прошедшее время
```

*хорошо*

```
let elapsedTimeInDays: number;  
let daysSinceCreation: number;  
let daysSinceModification: number;  
let fileAgeInDays: number;
```

Содержательные имена упрощают понимание и модификацию кода. Например:

```
function getThem(list: List<number[]>): List<number[]> {  
    const list1: List<number[]> = new List<number[]>();  
    for (const x of list) {  
        if (x[0] === 4) list1.add(x);  
    }  
    return list1;  
}
```

1. Что за данные в `list`?
2. Чем важен элемент `list` с нулевым индексом?
3. Какой смысл несёт значение 4?
4. Как будет использоваться возвращаемый список?

Но выше представлен фрагмент игры в «Сапёр», где `list` – это игровое поле, элемент с нулевым индексом – это код состояния, код 4 – означает «Флаг установлен». Итог:

```
function getFlaggedCells(gameBoard: List<number[]>): List<number[]> {  
    const flaggedCells: List<number[]> = new List<number[]>();  
    for (const cell of gameBoard) {  
        if (cell[STATUS_CODE] === FLAGGED)  
            flaggedCells.add(cell);  
    }  
    return flaggedCells;  
}
```

## Базовые правила именования сущностей

1. *Избегайте дезинформации.* Например: `accountList` – плохо, так как есть структура данных `List`, если хранение в чём-то отличном (или вообще давно поменялось) вводит в заблуждение. Хорошо – `accounts`, `accountGroup`.
2. *Используйте осмысленные различия.* Например: `Product`, `ProductInfo`, `ProductData` – с точки зрения смысла абсолютно одинаковые. Такие сущности не должны существовать в рамках проекта. `UsersTable` – хуже чем `Users`.

3. *Используйте удобопроизносимые имена.* `genDts` – хуже чем `generatedTimestamp`
4. *Избегайте схем кодирования имён.* Не используйте аббревиатуры (например, венгерская запись) для имён без необходимости, не используйте дополнительные приставки, суффиксы, окончания, кроме тех случаев, когда это регламентировано библиотекой (например, эффектор регламентирует обозначения для юнитов: стор – приставка `$`, эффект - окончание `Fx`).

```
// плохо
const IDOF: number = 7 // номер дня недели
const s_id: string = 'afe...' // тип в приставке

// хорошо
const dayOfWeek: number = 7;
const $accounts: UnitStore<Account[]> = createStore();
const fetchAccountsFx: UnitEffect<Account[], void> = createEffect();
```

## camelCase

Названия функций, переменных, констант пишутся в нотации `camelCase`.

```
// плохо
let LastDayOfMonth: Date;
let last_day_of_month: Date;

// хорошо
let lastDayOfMonth: Date;
```

## PascalCase

Используется в названиях компонентов, классов, типов, интерфейсов, `enum`

```
// плохо
class user {}
type listItem = {};
interface use_fetch {};

// хорошо
class User {};
type ListItem = {};
interface AnimationProps {};
function UserBadge(props: UserBadgeProps) { ... }
```

## snake\_case

Используется в типах представляющих ответ API, в адаптерах при преобразовании в структуру соответствующую соглашениям выше. В названиях CSS селекторов, в том числе использующихся в CSS Modules.

```
// плохо
let user_index: number = 0;
<div className={styles.bodyDark} />

// хорошо
{
    fullName: origin.full_name,
}
<div className={styles.body_dark} />
```

## UPPER\_SNAKE\_CASE

Применяется для констант, которые используются как определения. Содержат не вычисленные значения.

```
// плохо
const initialDurationMs: number = 5;
const DELTA_TIME: number = lastFrameTime - currentTime;

// хорошо
const INITIAL_DURATION_MS: number = 5;
```

## Части речи

Для того чтобы код был само-документируемым названия ДОЛЖНЫ соответствовать определённым частям речи.

### Существительное, прилагательное, деепричастие

Названия **переменных, констант, enum** ДОЛЖНЫ отвечать на вопросы «Что?», «Какой?», «Что делающий?». Названия классов только на вопрос «Что?». Для прилагательных используется окончание `ed`, для деепричастий `able`. Множественное число ДОЛЖНО использоваться для массивов, списков, наборов (Set), словарей (Map), stack, queue.

```
// плохо
const fetchElement = ...; // глагол

// хорошо
const user = ...;
let account = ...;
const controllers: Controller[] = [];
const sortedUsers: User[] = []; // причастие
const moveableTask: Task = new Task(); // деепричастие
```

## Глагол

Названия **функций, методов классов** ДОЛЖНЫ отвечать на вопрос «Что делает?», «Что делать?».

```
// плохо
function accounts(): Account[] {} // существительное

// хорошо
function getAccounts(): Account[] {}
function sort<T>(sortedItems: T[]): T[] {}
```


## Вопросительное предложение

Названия переменных, констант, функций, имеющих, либо возвращающих, значение типа `boolean`, ДОЛЖНЫ записываться как вопрос, на который можно ответить только «Да» или «Нет». Для этого ДОЛЖНЫ использоваться глаголы `is`, `has`, `are`.

```
// плохо
let mounted: boolean = false;

// хорошо
let isMount: boolean = false;
function hasOvertime(): boolean {}
let areEqualName: boolean = false;
```

## CSS селекторы

Так как для компонентов ДОЛЖНЫ использоваться CSS Modules, название селектора должно относиться к элементу (понятие из БЭМ ([bem.info](https://bem.info)) ). Для указания модификатора используется разделитель `__` (Двойное подчеркивание).

```
/* плохо */
.card_title {}

/* хорошо */
.title {}
.title__primary {}
```

## Custom Properties

Согласно [требованиям к разработке компонента](#), для стилизации сложных компонентов используются Custom Properties. Названия имеют следующую структуру

```
--element_name_css-property[__state]
```

Например:

```
.root {
  --title_background-color: #3eaf22;
  --title_background_color__hover: #aa21e1;
}
```

## Обязательные именованя

### Обработчики событий

Функции обработки событий ДОЛЖНЫ начинаться с глагола `handle`, НЕЛЬЗЯ использовать `on`. Название ДОЛЖНО относиться к предметной области проекта.

```
// плохо
function selectMine(event: SelectEvent) {}
function handleOnSelect(event: SelectEvent) {}

// хорошо
function handleSelectMine(event: SelectEvent) {}
function handleOpenPathConfigurator(event: ClickEvent) {}
```

## Порождение событий

Как правило применяется к атрибутам функций, методам классов. Название атрибута ДОЛЖНО содержать приставку `on`.

```
// плохо
function Selector(change: () => {}) {}
function createSomething(e: () => {}) {}

// хорошо
function Selector(onChange: () => {}) {}
function createSomething(onCreated: () => {}) {}
```

## JSX / TSX

Именованя атрибутов компонента ДОЛЖНЫ быть прилагательными или существительными и отвечать на вопрос «Что?», «Какой?».

```
<!-- Плохо -->
<Component isDisabled />
<Component isPrimary />
<Component renderSlot={...} />

<!-- Хорошо -->
<Component disabled />
<Component primary />
<Component itemSlot={...} />
```