**Новый синтаксис. Деструктуризация**

Прежде чем переходить к изучению материалов спринта, нужно познакомиться с новым синтаксисом. Он появился в ES6 и называется деструктуризацией.

Деструктуризация — это способ записи значений в переменные. Используя её, можно заметно сократить код.

**Деструктуризация объектов**

Прежде свойства объекта записывали в переменную так:

Скопировать кодJAVASCRIPT

const user = {

name: 'Василий',

age: 28,

lookingForJob: true

};

const name = user.name;

const age = user.age;

*/\* теперь имя и возраст хранятся в переменных,*

*с ними удобно работать \*/*

console.log(name); *// "Василий"*

console.log(age); *// 28*

Деструктуризация позволяет записать значения свойств объекта в одноимённые переменные. Для этого после декларации const или let записывают имена переменных в фигурных скобках, знак равенства и имя объекта, из которого нужно достать значения:

Скопировать кодJAVASCRIPT

const greekGods = { love: 'Афродита', war: 'Арес', trade: 'Гермес' };

*/\* перечисляем свойства, которые нужно достать, в фигурных скобках \*/*

const { love, war, trade } = greekGods; *// имена переменных совпадают с ключами объекта*

console.log(love, war, trade); *// Афродита Арес Гермес*

Если имена переменных и ключи объекта должны отличаться, можно дописывать имя каждой переменной после имени свойства через двоеточие:

Скопировать кодJAVASCRIPT

const greekGods = { love: 'Афродита', war: 'Арес', trade: 'Гермес' };

const {

love: goddessOfLove,

war: godOfWar,

trade: godOfTrade

} = greekGods;

console.log(goddessOfLove, godOfWar, godOfTrade); *// Афродита Арес Гермес*

**Деструктуризация массива**

С массивом тоже работает механизм деструктуризации. До ES6 получать элементы массива можно было только по индексам:

Скопировать кодJAVASCRIPT

const precipitation = ['дождь', 'морось', 'роса']; *// англ. precipitation, «осадки»*

const rain = precipitation[0];

const drizzle = precipitation[1];

const dew = precipitation[2];

console.log(rain, drizzle, dew); *// дождь морось роса*

С деструктуризацией код становится короче и изящнее. После декларации const или let можно перечислить имена переменных в квадратных скобках, указать оператор присваивания и имя массива, из которого нужно достать значения:

Скопировать кодJAVASCRIPT

const precipitation = ['дождь', 'морось', 'роса'];

const [rain, drizzle, dew] = precipitation;

console.log(rain, drizzle, dew); *// дождь морось роса*

Строки 'дождь', 'морось' и 'роса' запишутся в переменные с именами rain, drizzle и dew согласно их порядку.

**Параметры функции**

Деструктурировать массив или объект можно при передаче их функции. Без использования деструктуризации код выглядел бы так:

Скопировать кодJAVASCRIPT

const userData = {

name: 'Виктор',

patronymic: 'Семёнович',

age: 55

};

const printUserParams = (params) => {

console.log(params.name);

console.log(params.patronymic);

console.log(params.age);

};

printUserParams(userData);

*/\**

*Виктор*

*Семёнович*

*55*

*\*/*

Теперь можно указать параметры функции в фигурных скобках, а при вызове — передать нужный объект. Код будет выглядеть так:

Скопировать кодJAVASCRIPT

const userData = {

name: 'Виктор',

patronymic: 'Семёнович',

age: 55

};

const printUserParams = ({ name, patronymic, age }) => {

console.log(name);

console.log(patronymic);

console.log(age);

};

printUserParams(userData);

*/\**

*Виктор*

*Семёнович*

*55*

*\*/*

Если у функции несколько параметров, можно деструктурировать только один из них:

Скопировать кодJAVASCRIPT

const userData = {

name: 'Виктор',

patronymic: 'Семёнович',

age: 55

};

const printUserParams = ({ name, patronymic, age }, secondParameter) => {

console.log(name);

console.log(patronymic);

console.log(age);

console.log(secondParameter);

};

printUserParams(userData, 'второй параметр');

*/\**

*Виктор*

*Семёнович*

*55*

*второй параметр*

*\*/*

**Параметры метода или конструктора класса**

Методы и конструктор класса — это функции. Поэтому их параметры тоже можно деструктурировать:

Скопировать кодJAVASCRIPT

*// код без использования деструктуризации параметров*

class Card {

constructor(data) {

this.\_text = data.text;

this.\_image = data.image;

this.\_description = data.description;

}

}

*// код c использованием деструктуризации параметров*

class Card {

constructor({ text, image, description }) { *// достаём ключи объекта сразу*

*// теперь значения ключей объекта data*

*// лежат в одноимённых переменных*

this.\_text = text;

this.\_image = image;

this.\_description = description;

}

}

В этом спринте мы будем часто использовать синтаксис деструктуризации в конструкторе класса.

**Значения по умолчанию**

И последнее в этом уроке: деструктуризация позволяет задавать значения по умолчанию.

Этим часто пользуются, когда функция принимает объект в качестве параметра, но при вызове функции какие-то из свойств объекта могут отсутствовать. Тогда им можно присвоить значения по умолчанию через знак равенства после имени переменной:

Скопировать кодJAVASCRIPT

*// consoleUserInfo выводит данные пользователя в консоль*

function consoleUserInfo(user) {

*// если какого-то из свойств не будет в переданном объекте,*

*// в переменные запишутся значения после знаков равенства*

const { name = 'Дэвид', dateOfBirth = '8 января' } = user;

console.log(`${name}, ${dateOfBirth}`);

}

*// name и dateOfBirth будут взяты из переданного объекта*

consoleUserInfo({ name: 'Леонард', dateOfBirth: '21 сентября' }); *// "Леонард, 21 сентября"*

*// name будет взято из переданного объекта,*

*// а dateOfBirth примет значение по умолчанию*

consoleUserInfo({ name: 'Стивен' }); *// "Стивен, 8 января"*

*// всё как в прошлом вызове, но наоборот*

consoleUserInfo({ dateOfBirth: '2 мая' }); *// "Дэвид, 2 мая"*

*// и name и dateOfBirth примут значение по умолчанию*

consoleUserInfo({}); *// "Дэвид, 8 января"*

Код функции consoleUserInfo можно сократить, деструктурировав параметр прямо в круглых скобках функции:

Скопировать кодJAVASCRIPT

*// можно присвоить значения по умолчанию, деструктурировав параметр функции*

function consoleUserInfo({ name = 'Дэвид', dateOfBirth = '8 января' }) {

console.log(`${name}, ${dateOfBirth}`);

}

Значения по умолчанию можно присвоить и при деструктуризации массива:

Скопировать кодJAVASCRIPT

function getFirstThreeUsers(users) {

const [first = 'Дэвид', second = 'Леонард', third = 'Стивен'] = users;

console.log(first, second, third);

}

*// если не передать третьего пользователя, им станет Стивен*

*// так как переменная third в коде функции примет значение по умолчанию*

getFirstThreeUsers(['Элвис', 'Ларс']);

**Шпаргалка**

Код можно сделать короче и изящнее, используя деструктуризацию. Синтаксис запомнить несложно:

* При деструктуризации объекта мы перечисляем имена переменных в фигурных скобках и после знака равенства пишем объект, из которого нужно взять соответствующие свойства. Если имена свойств и переменных должны различаться, то названия переменных пишут через двоеточие после ключей:

Скопировать кодJAVASCRIPT

const { a: type, b: name } = { a: 'кресло', b: 'Василий' };

console.log(type, name); *// кресло Василий*

* При деструктуризации массива переменные нужно перечислять уже в квадратных скобках. Дальше после знака равенства также указываем массив, из которого нужно взять значения переменных:

Скопировать кодJAVASCRIPT

const [type, name] = ['кресло', 'Василий'];

console.log(type, name); *// кресло Василий*

* Деструктуризацию можно использовать при передаче объекта функции. Для этого при объявлении функции параметры пишут в фигурных скобках, а при вызове — передают нужный объект:

Скопировать кодJAVASCRIPT

const chair = { type: 'кресло', name: 'Василий' };

const whereToSit = ({ type, name }) => {

return `Садитесь в ${type} «${name}»`;

};

console.log(whereToSit(chair)); *// Садитесь в кресло «Василий»*

* И наконец, деструктуризация позволяет задавать значения по умолчанию:

Скопировать кодJAVASCRIPT

const { name = 'Василий' } = {};

const [type = 'кресло'] = [];

console.log(type, name); *// кресло Василий*