**Введение**

В предыдущем спринте вы впервые столкнулись [с ключевым словом this](https://praktikum.yandex.ru/trainer/web/lesson/d82a0f2f-032b-4793-844e-65bf44e80919/task/14a94970-726d-4bad-9dd6-4fbe1d4948af/). Продолжим изучать его, но сначала напомним, что вы уже знаете.

Если функцию вызывают как метод объекта, this содержит ссылку на объект, на котором она вызвана:

Скопировать кодJAVASCRIPT

const obj = {

prop: 'Свойство',

method: function () {

console.log(this.prop);

}

}

obj.method(); *// "Свойство"*

Вызов функции как метода объекта — не единственный способ установки this. В этой теме мы расскажем про четыре других способа, но подробнее остановимся только на двух.

# Способы привязки this

Ключевое слово this целиком зависит от того, где и как была вызвана функция. И не важно, в каком месте кода функция была написана — привязка this происходит при каждом вызове.

Чтобы определить, на что ссылается this, нужно исследовать точку вызова функции. Это то место в коде, где происходит вызов функции.

Установить значение this в функции в зависимости от точки вызова можно так:

1. По умолчанию.
2. При вызове функции как метода объекта.
3. Явной привязкой методами call, apply и bind.
4. При вызове функции с оператором new.

Также есть способ привязки this, который не зависит от точки вызова: this внутри стрелочной функции.

В этой теме мы разберём подробнее только 2 из них:

1. По умолчанию.
2. Явная привязка методом bind.

А в следующих спринтах вы узнаете про другие способы установки значения this.

# Привязка this по умолчанию и потеря контекста

В прошлом уроке мы говорили о 2-х случаях привязки this:

1. По умолчанию.
2. Явная привязка методом bind.

У каждого способа есть свой приоритет. Начнём с самого низкого — привязки по умолчанию. В этом случае вы просто вызываете функцию по её имени, то есть пишете имя функции и круглые скобки. Тогда this принимает значение window:

Скопировать кодJAVASCRIPT

function globalFunction() {

console.log(this);

}

globalFunction(); *// Window — this ссылается на глобальный объект window*

Функция globalFunction вызывается просто по имени. Поэтому this ссылается на объект window.

Ключевое слово this может принимать и другое значение. Вы изучите это в следующей теме о this.

## Потеря контекста

Привязку по умолчанию редко используют намеренно — обычно это следствие какой-то ошибки или потери контекста. Например, потерять контекст можно, передав функцию как колбэк:

Скопировать кодJAVASCRIPT

class SendButton {

constructor() {

this.buttonName = 'Send Button';

}

click() {

console.log('I am ' + this.buttonName);

}

setEventListeners() {

*// Передадим слушателю события метод SendButton.click как колбэк:*

document.querySelector('.btn').addEventListener('click', this.click); *// При клике мы надеемся увидеть сообщение "I am Send Button"*

}

}

const button = new SendButton();

button.setEventListeners();

Наша задача была –– выводить имя кнопки при каждом нажатии на неё. Но это не сработало, потому что this определяется в момент вызова функции, то есть в момент клика по элементу. Важно, что функция click не вызывается как метод класса — она передаётся методу addEventListener и вызывается из него. Механизм слушателя события такой, что this при этом ссылается на DOM-элемент, на котором событие сработало. У DOM-узла нет свойства buttonName, поэтому в консоли оказывается сообщение I am undefined.

Потери контекста можно избежать, если явным образом указать функции значение this. О том, как это сделать, расскажем в следующем уроке.

# Явная привязка. Метод bind

Как и все объекты, функции тоже имеют свои методы. Из этого урока вы узнаете о методе bind, который применяют, чтобы явно указать значения this в функции.

Метод bind создаёт обёртку с изначальной функцией и фиксированным контекстом. Другими словами, функцию можно представить как овощерезку, которая может резать разные овощи: 🥒 🍆 🥕. Эти овощи — контекст this. С помощью метода bind можно указать овощ 🥒, который и будет резать овощерезка. А так это выглядит на JS:

Скопировать кодJAVASCRIPT

function chopVegetables() {

console.log(`Chop ${this}`);

}

const boundChopVegetables = chopVegetables.bind(🥒);

boundChopVegetables();

*// chopVegetables - овощерезка*

*/\**

*Chop 🥒*

*\*/*

Разберём на более сложном примере:

Скопировать кодJAVASCRIPT

const car = {

registrationNumber: 'O287AE',

brand: 'Tesla'

};

function displayDetails(ownerName, greeting) {

console.log(`${greeting} ${ownerName}`);

console.log(`Car info: ${this.registrationNumber} ${this.brand}`);

}

*// создаём новую функцию с привязанным контекстом. Где бы мы ни вызвали функцию boundDisplayDetails, значением this внутри неё всегда будет объект car*

const boundDisplayDetails = displayDetails.bind(car);

*// теперь можно вызвать её просто по имени — к ней привязан контекст*

boundDisplayDetails('Matt', 'Hello');

*/\**

*Hello Matt*

*Car info: O287AE Tesla*

*\*/*

Метод bind удобно использовать, чтобы не потерять this при использовании коллбэк-функций:

Скопировать кодJAVASCRIPT

class SendButton {

constructor() {

this.buttonName = 'Send Button';

}

click() {

console.log('I am ' + this.buttonName);

}

setEventListeners() {

document.querySelector('.btn')

.addEventListener('click', this.click.bind(this));

*// жёстко привязываем контекст при передаче функции*

}

}

const button = new SendButton();

button.setEventListeners();

Метод bind не вызывает функцию. Он указывает значение this, с которым эта функция будет вызываться.

const sendButton = {

message: 'Меня нажали',

click: function () {

console.log(this.message);

}

};

const anotherButton = {

message: 'И меня нажали',

};

const button1 = document.querySelector('.btn-1');

const button2 = document.querySelector('.btn-2');

button1.addEventListener('click', sendButton.click.bind(sendButton));

button2.addEventListener('click', sendButton.click.bind(anotherButton));

**Заключение**

Это была капля в море this. Но вы успешно с ней справились. Повторим, что вы уже знаете.

Чтобы правильно определить значение this, нужно изучить точку вызова функции. Привязка this подчиняется пяти правилам. В этом разделе вы узнали про два из них:

1. Если функция вызвана после привязки методом bind, в this оказывается переданный в метод объект.
2. По умолчанию — глобальный объект window.

Умение определять привязку по умолчанию поможет быстрее находить участки в коде, где внутри функции потерялся контекст.

А явную привязку методом bind удобно использовать, чтобы избежать потери контекста в функциях.

Ничего страшного, если понимание того, где переопределяется контекст, не пришло сразу. Скорее всего, на земле нет ни одного JavaScript-разработчика, не терявшего контекст ни разу. Осознанно с ним работать сложно даже опытным разработчикам.

Впереди ещё много секретов, которые хранит в себе this. Но о них вы узнаете в следующих спринтах.