# **Понятия**

this – это ссылка на объект, который вызывает код в данный момент

*let* count = 0;

*function* f1() {

    console.log(count);

    console.log(this);

    this.textContent = count;

    count++;

}

document.querySelector('.b-1').addEventListener('click', f1);

В данном случае в this храниться ссылка на объект(элемент) .b-1 (это кнопка).

Можем просто вызвать функцию

В стрелочных функциях this – нет

# **Метод call**

Позволяет подменить запуск функции с нужным контекстом. Данный метод позволяет вызывать функцию и передавать ей необходимые аргументы. Позволяет подменять контекст функции при запуске. Первый параметр это содержимое this.

*let* count = 0;

*function* f1() {

    console.log(count);

    console.log(this);

    this.textContent = count;

    count++;

}

//Если вызвать просто: count: 0, this:Window {window: Window, self: Window,

f1();

//внутри пропишем, что будет лежать в this

f1.call(document.querySelector('.b-1'));

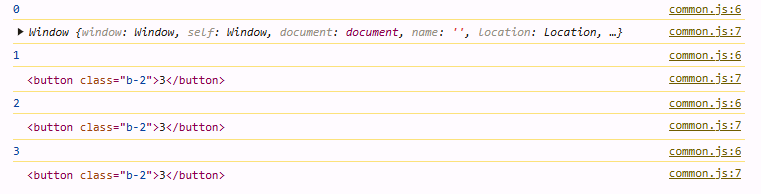
//внутри пропишем, что будет лежать в this

f1.call(document.querySelector('.b-1'));

//внутри пропишем, что будет лежать в this

f1.call(document.querySelector('.b-1'));

Результат



Также без всяких проблем можно прописать функции f1() вторую кнопку

Применение call с addEventListener

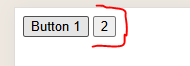
//Применяем функцию call с addEventListener

document.querySelector('.b-1').addEventListener('click', ()*=>*{

    f1.call(document.querySelector('.b-2'));

})

Результат



Привязали обработчик события к первой кнопке, но прописали для this вторую кнопку. По умолчанию в this должна была попасть первая кнопка. Но в call принудительно прописали вторую. В выделенном участке кода изменяется вторая кнопка при клике на первую.

*let* count = 0;

*function* f1() {

    console.log(count);

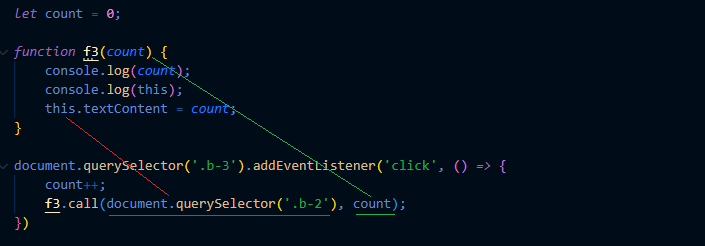
    console.log(this);

    this.textContent = count;

    count++;

}

При помощи второго параметра можно передавать аргументы функции



Пример

При клике на вторую кнопку, содержимое блока = 7

*function* sum1(*a*, *b*) {

    this.innerHTML = *a* + *b*;

}

document.querySelector('.b-2').addEventListener('click', () *=>* {

    sum1.call(document.querySelector('.out-3'), 3, 4);

});

Результат



# **Метод apply**

Выполняет ту же функцию что и call, за исключением того, что аргументы в функцию подаются в виде массива. В параметрах массив разбивается поэлементно.

*function* sum1(*a*, *b*) {

    this.innerHTML = *a* + *b*;

}

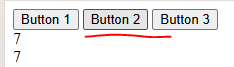
document.querySelector('.b-2').addEventListener('click', () *=>* {

    sum1.call(document.querySelector('.out-3'), 3, 4);

    sum1.apply(document.querySelector('.out-4'), [3, 4]);

});

Результат



# **Метод bind**

C помощью данного метода могу делает то же самое что и с call и apply, но если call и apply нужно указывать при каждом вызове функции, то bind позволят создать новую функцию и привязать контекст один раз, потом эту новую функцию выполнять. Т.е. создается обертка вокруг функции и можно с ней работать.

*function* f1() {

    console.log(count);

    console.log(this);

    this.textContent = count;

    count++;

}

*const* f4 = f1.bind(document.querySelector('.out-5'));

document.querySelector('.b-5').addEventListener('click', f4);

Передача аргументов

*function* sum1(*a*, *b*) {

    this.innerHTML = *a* + *b*;

}

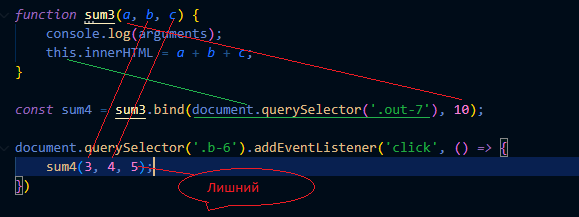
*const* sum2 = sum1.bind(document.querySelector('.out-6'));

sum2(4, 5);

sum2(5, 10);

# **Частичные функции с переопределенным элементом (частичные вычисления)**

Параметры метода bind перебивают аргументы переданные в функцию.



Можно и не передавать параметр this

*function* sum7(*a*, *b*, *c*) {

    return *a* + *b* + *c*;

}

*const* sum8 = sum7.bind(undefined, 100, 300);

document.querySelector('.b-8').addEventListener('click', () *=>* {

    document.querySelector('.out-8').textContent = sum8(5);

})

//Ответ 405

# **Вытягивание методов**

//Вытягивание методов

*const* validate = {

    password: 'himahai',

    email: 'pupkin@de',

    isValid: false,

    sayHi() {

        console.log(this);

        return(this.password.length > 6) ? true : false;

    }

}

console.log(validate.sayHi());

//Создал новый оъект

*const* obj = {password: 'hello'};

//Вытянул метод sayHi и перенес в obj

*const* validatePassword = validate.sayHi.bind(obj);

console.log(validatePassword());

Результат

