**Лабораторная работа №15**

**Тема работы:** Разработка программ с определением функция как объект. Методы call и apply

**Цель работы:** закрепить навыки по определению функции как объект, работа с методами call и apply.

**Теоретические сведения**

**Работа с this**

Пусть у нас есть какая-то функция **func**, внутри которой используется **this**:

function func() {

alert(this.value);

}

На что указывает **this** в этой функции? А мы не знаем. И JavaScript не знает. И сама функция не знает. То есть: в момент создания функции на что именно указывает **this** не определено. И определится это только тогда, когда эта функция будет вызвана.

Давайте теперь сделаем инпут и привяжем к нему нашу функцию **func**. Теперь this указывает на наш инпут:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

elem.addEventListener('blur', func);

function func() {

alert(this.value); //выведет 'привет'

}

Но ведь у нас может быть не один инпут, а несколько, и каждому мы можем привязать нашу функцию **func**. В этом случае для первого элемента **this** в функции будет указывать на него, а для второго - на него:

<input id="elem1">

<input id="elem2">

var elem1 **=** document.getElementById('elem1');

elem1.addEventListener('blur', func); //тут this - это первый элемент

var elem2 **=** document.getElementById('elem2');

elem2.addEventListener('blur', func); //тут this - это второй элемент

function func() {

alert(this.value); //а тут this не еще определен

}

А что будет, если в функции указать **this**, но не привязать ее ни к какому элементу, вот так:

function func() {

console.log(this);

}

func();

В этом случае в новом стандарте JavaScript там будет лежать **undefined**, а в старом - объект **window**.

**Давайте подведем итог**

В любой функции можно написать **this**, это не приведет к каким-либо ошибкам JavaScript. Но вот что именно будет лежать в этом **this** не определено до момента вызова функции. Причем при разных вызовах функции **this** может принимать разное значение. Все зависит от контекста, в котором была вызвана функция.

**Функция в функции**

Пусть у нас есть функция в функции. Пусть внешняя функция называется **func**, а внутренняя - **child**. Если задать какую-либо переменную во внешней функции - она будет доступна во внутренней:

function func() {

var test **=** 'привет';

function child() {

alert(test); //выведет 'привет'

}

child();

}

Пусть теперь наша функция **func** привязана к инпуту и this в ней указывает на инпут:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

elem.addEventListener('blur', func);

function func() {

alert(this.value); //выведет 'привет'

function child() {

alert(this.value); //выдаст ошибку!

}

child();

}

Внутри функции **child** мы пытаем обратиться к инпуту через this, но у нас это не получится, потому что контекст у этой функции другой и this не указывает на инпут.

Получается, что переменные функции **func** видны в функции **child**, но this у них разный.

Популярный способ решения данной проблемы такой: во внешней функции запишем this в любую переменную (например, в **self**) и эта переменная будет доступна в child, как и все переменные. Таким образом мы передадим this из внешней функции во внутреннюю:

function func() {

alert(this.value);

var self **=** this; //запишем this в любую переменную

function child() {

alert(self.value); //все работает, так как self тут доступна

}

child();

}

Проблемы такого плана поджидают нас в самых неожиданных местах. Пусть сейчас функция func сработает по клику на инпут и внутри функции запустится таймер, который каждую секунду будет алертом выводить value нашего инпута:

function func() {

window.setInterval(function() {

alert(this.value); //не будет работать

}, 1000);

}

Это не будет работать, так как мы потеряли контекст. Ведь this размещен во внутренней анонимной функции и поэтому он не указывает на наш инпут.

Поправим проблему введением **self**:

function func() {

var self **=** this;

window.setInterval(function() {

alert(self.value);

}, 1000);

}

Проблему также можно поправить через стрелочные функции ES6. В этом случае this внутри стрелочной функции такой же, как и this снаружи:

function func() {

window.setInterval(() **=>** alert(this.value), 1000);

}

**Привязывание контекста**

Итак, мы разобрали, как на самом деле работает this. Давайте теперь рассмотрим методы, которые позволяют принудительно указать, в каком контексте вызывается функция (*то есть принудительно сказать, чему равен this*).

Таких методов существует три: метод **call**, метод **apply** и метод **bind**. Давайте разбираться с ними.

**Метод call**

Пусть у нас есть инпут **#elem**. Давайте получим ссылку на него с помощью **getElementById** и запишем ее в переменную **elem**:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

Давайте теперь сделаем функцию **func**, внутри которой алертом выведем **this.value**:

function func() {

alert(this.value);

}

Пока наша функция не знает, на что ссылается this. Вот, если бы мы ее привязали через **addEventListener**, тогда да. Но мы не будем этого делать. В замен мы просто вызовем нашу функцию, сказав ей, что *this должен быть равен elem*.

Это делается вот так: **func.call(elem)**. Этот код эквивалентен простому вызову функции func вот так: **func()**, только с условием, что this равен elem.

Итак, синтаксис метода call такой: *функция.call(объект, который записать в this)*.

Давайте соберем все вместе:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

function func() {

alert(this.value);

}

func.call(elem); //выведет value инпута

**Метод call с параметрами**

Пусть теперь функция func принимает некоторые параметры, назовем их param1 и param2:

function func(param1, param2) {

alert(this.value **+** param1 **+** param2);

}

При вызове функции через call можно передать эти параметры вот так:

func.call(elem, param1, param2);

То есть полный синтаксис метода call такой: *функция.call(контекст, параметр1, параметр2 и так далее)*.

**Метод apply**

Кроме метода call, существует очень похожий метод apply - разница в способе передачи параметров. Следующие две записи эквивалентны:

func.call(elem, param1, param2);

func.apply(elem, [param1, param2]);

То есть в apply параметры передаются в виде массива, а не перечисляются через запятую, как в методе call. Вот и все отличие. В зависимости от задачи бывает удобен то один, то другой метод.

**Метод bind**

Следующий метод **bind** позволяет привязать контекст к функции навсегда. Он вызывается вот так: func.bind(elem), но результатом работы будет не результат функции func, а новая функция, которая такая же, как и func, но у нее this всегда указывает на elem.

Давайте посмотрим на примере. Пусть у нас есть следующая функция func:

function func(param1, param2) {

alert(this.value **+** param1 **+** param2);

}

С помощью bind скажем, что this в ней всегда равен elem и запишем в новую переменную:

var newFunc **=** func.bind(elem); //обратите внимание - параметры не передаем

Теперь в переменной newFunc лежит функция. Давайте вызовем ее, передав в первый параметр '!', а во второй '?' (напоминаю, что в elem лежит инпут с value, равным 'привет'):

newFunc('!', '?'); //выведет 'привет!?'

Давайте соберем все вместе:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

function func(param1, param2) {

alert(this.value **+** param1 **+** param2);

}

var newFunc **=** func.bind(elem);

newFunc('!', '?'); //выведет 'привет!?'

Не обязательно записывать результат работы bind в новую функцию newFunc, можно просто перезаписать func:

var func **=** func.bind(elem);

Теперь func - такая же функция, как и была, но с жестко связанным this.

Определение количества записей в хранилище: alert(localStorage.length).

Определение названия ключа по его номеру: alert(localStorage.key(номер)).

При выполнении операций с хранилищем, срабатывает событие **onstorage**. Если к этому событию привязать функцию - то в ней будет доступен объект Event со следующими свойствами:

<body onstorage="func()">

function func(event) {

var key **=** event.key; //ключ изменяемых данных

var oldValue **=** event.oldValue; //старое значение

var newValue **=** event.newValue; //новое значение

var storageArea **=** event.storageArea; //storageArea

}

**Порядок выполнения работы**

**Работа с call**

1. Пусть дан следующий код:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

function func() {

alert(this.value);

}

func(); //тут должно вывести value инпута

1. Добавьте в последнюю строчку метод **call()** так, чтобы на экран вывелось value инпута, лежащего в переменной elem.

Пусть дан следующий код:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

function func(surname, name) {

alert(this.value **+** ', ' **+** surname **+** ' ' **+** name);

}

func(); //тут должно вывести 'привет, Иванов Иван'

Добавьте в последнюю строчку метод **call()** так, чтобы на экран вывелось 'привет, Иванов Иван'. Слово 'привет' должно взяться из value инпута, а 'Иванов' и 'Иван' должны быть параметрами функциями.

**Работа с apply**

1. Переделайте решение предыдущей задачи так, чтобы место метода **call()** был метод **apply()**.

**Работа с bind**

1. Пусть дан следующий код:

<input id="elem" value="привет">

var elem **=** document.getElementById('elem');

function func(surname, name) {

alert(this.value **+** ', ' **+** surname **+** ' ' **+** name);

}

//Тут напишите конструкцию с bind()

func('Иванов', 'Иван'); //тут должно вывести 'привет, Иванов Иван'

func('Петров', 'Петр'); //тут должно вывести 'привет, Петров Петр'

Напишите в указанном месте конструкцию с методом **bind()** так, чтобы this внутри функции **func** всегда указывал на инпут из переменной elem.