Практическое занятие №12 Объекты в JS

Задание №1

Проверьте запущен ли у Вас Open Server. Папка с материалами урока (js12.loc) должна быть размещена в директории /domains сервера. Заходим на http://js12.loc и начинаем урок.

Задание №2

Объект является фундаментальным типом данных в языке JS. Объект – это коллекция именованных свойств и методов. Существует несколько способов создания объектов в JS. Рассмотрим их на практике:

```
var obj1 = new Object();
var obj2 = Object();

var obj3 = {}; // предпочтительный способ

console.log(obj1);
console.log(obj2);
console.log(obj3);
```

Как Вы сами видите в консоли мы создали три «пустых» объекта в JS. В терминах JS мы создали пользовательский объект.

Пользовательский объект – это такой объект, который создаётся в результате выполнения программного кода JS.

Так же выделяют ***объекты базового языка*** – это объекты, определяемые спецификацией самого языка (массивы, функции, даты).

Объекты среды выполнения – это объекты, которые определяются в конкретной сборке JS для конкретных нужд. Например в браузерах существуют объекты отдельный HTML элементов.

Задание №3

Пары «***имя свойства/метода: значение свойства/метода***» разделяются запятой. После последней пары запятая не ставится (иначе возникнут проблемы в некоторых версиях IE). Если не писать ничего внутри фигурных скобок, будет создан пустой объект.

В отличие от массивов, где каждый элемент имеет индекс – номер, по которому можно его найти в любой момент, в объекте элементы представляют свойства (методы) с именем. При создании свойства (метода) мы должны дать ему имя. В результате созданное свойство (метод) хранится с данным именем.

В файле задания создаём объект person, как показано ниже:

```
var person = {};
```

Теперь добавляем ему свойства:

```
var person = {

    name: 'John',
    'last name': 'Smith', // название свойства состоит из нескольких слов
    sex:'male',
    age: 19,
    job: 'janitor'

};
```

Для того что бы обратиться к свойству объекта из вне необходимо обраться к нему следующим образом:

```
// <объект>.<свойство>
console.log( person.sex );
// или
console.log( person['sex'] );

// к этому свойству можно обратиться только так
console.log( person['last name'] );
```

Следует понимать, что свойства - это информация которой располагает объект, так же у объектов есть «поведение», то есть методы. Методы создаются как функции внутри объекта. Добавим нашему объекту метод:

```
var person = {

    name: 'John',
    'last name': 'Smith', // название свойства состоит из нескольких слов
    sex:'male',
    age: 19,
    job: 'janitor',

    getInfo: function() {
        console.log('вызван метод getInfo');
    }

};
```

Вызвать данный метод вне объекта можно следующим образом:

```
person.getInfo();
// или
person['getInfo']();
```

Сейчас мы научились создавать и вызывать методы объектов. Однако наш метод «getInfo» ужасно не функционален. Для того что бы улучшить его нам нужно ВНУТРИ метода получить доступ к свойствам объекта, сделать это можно с помощью ключевого слова «this», которое внутри объекта ссылается на сам объект. Перепишем метод «getInfo» с использованием «this»:

```
getInfo: function() {

    var string =  'My name - '+this.name+''+this['last name']+"\n\r"+
                  'I am '+this.age;

    console.log(string);
}
```

Задание №4

В JS так же существует возможность создавать методы и свойства объекта не только при его создании, а уже когда объект существует. Рассмотрим и протестируем пример кода:

```
// создаём "пустой" объект
var obj = {};

// создаём свойство "на лету"
obj.status = 'all done!';

// создаём метод "на лету"
obj.getStatus = function() {

  console.log( this.status );

};


// выводим информацию об объекте в консоль
console.log( obj );
```

Самостоятельно пересоздайте объект из задания №3 как показано в этом примере, то есть с использованием объявления свойств и методов «из вне» объекта (сначала создаём пустой объект, потом добавляем и описываем ему все нужные свойства и методы).

Задание №5

Для удаления свойств и методов объекта вызывается оператор delete. Это возможность используется довольно редко, но знать о ней необходимо. Рассмотрим и протестируем следующий пример кода:

```
var obj = {a: 5};

delete obj.a;

console.log(obj.a); //вернет undefined - свойство удалено
```

Задание №6

Все типы данных в JavaScript, кроме тривиальных типов (null, undefined) имеют собственные функции-конструкторы. Можно было заметить, что при их вызове используется оператор вызова функций “()”.

```
var number = new Number(18);
```

Что же представляют собой функции-конструкторы? Функции-конструкторы - это ФУНКЦИИ, которые возвращают экземпляр объекта определённого типа. Создадим простую функцию-конструктор:

```
// определяем класс-конструктор
function Person(name, family) {

    this.name = name;
    this.family = family;

}

// создаём обьект с помощью класса-конструктора
var $obj = new Person('John', 'Dou');

// выводим объект в консоль
console.log( $obj );
```

Как же можно узнать, какая функция-конструктор была использована при создании объекта? С помощью свойства ***constructor***. Выведем в консоль свойство ***constructor*** нашего класса и некоторый встроенных классов:

```
// посмотрим на конструктор нашего класса
console.log( $obj.constructor );

// попытаемся получить конструкторы встроенный объектов JS
var number = new Number(18);
console.log( number.constructor );

var arr = new Array(12,13);
console.log( arr.constructor );
```

Проверить принадлежность объекта к определённому конструктору можно и с помощью оператора ***instanseof***:

```
console.log($obj instanceof Person);
console.log($obj instanceof Number);

console.log($obj instanceof Object);
console.log($obj instanceof Object);
```

Иногда такая проверка «типа» объекта просто необходима.

ЭТО ВАЖНО! Тип данных Object является прототипом для абсолютно всех объектных типов данных в JavaScript. Все объекты (встроенные и пользовательские) получают методы этого типа. Фактически, вся структура объектов в JavaScript двухуровневая – во главе стоит единственный объект Object.

Задание №7

Самостоятельно создайте функцию-конструктор, который описывает объект «Employee». Этот объект должен включать в себя следующую информацию:

- имя
- фамилия
- пол
- возраст
- должность
- отдел
- стаж работы в месяцах
- размер оклада
- кол-во детей у сотрудника

А также методы:

- получаем полное имя сотрудника (имя + фамилия)
- получаем информацию о должности (должность + отдел + стаж работы)
- получаем оклада за год работы. Если известно, что если у работника стаж больше 12 мес. то он получает премию в размере одного оклада под Новый год.
- получаем размер медицинской страховки в размере 5% от годового оклада если стаж работы меньше 36 мес. и 0% если выше (в этом случае компания оплачивает страховку за свой счет).
- Получаем сумму затрат на новогодние подарки для детей сотрудника. Она рассчитывается по формуле: «кол-во детей * 200».

Создать несколько объектов «Employee», протестировать и вывести в консоль результаты выполнения методов этих объектов.

Задание №8

Мы отмечали, что все объекты получают свойства и методы от Object, а экземпляры этих типов данных наследуют свойства и методы от своих классов-конструкторов. Как же происходит «наследование» (с более классическим наследованием мы познакомимся в PHP), то есть передача свойств и методов от объекта к объекту, в JS?

Дело в том, что каждый объект в языке JS имеет второй объект, ассоциированный с ним. Этот второй объект называется прототипом, и первый объект получает от прототипа свойства и методы.

Атрибут ***prototype*** объекта определяет объект, от которого наследуются свойства и методы. Он устанавливается в момент создания объекта. Наберите код ниже и изучите информацию о прототипах из консоли:

```
function foo() {};

// прототип объекта функции
console.log( foo.prototype );

// прототип объекта Number
console.log( Number.prototype );
```

Теперь о том зачем нам нужны прототипы. Например, мы хотим расширить возможности работы с числами в нашей программе, мы хотим создать возможность инвертирования чисел. Для этого мы можем добавить новый метод в прототип объекта Number, как показано ниже:

```
// добавляем новый метод в прототип
Number.prototype.invert = function() {
  return - this.valueOf();
};


var number = 3;

// проверьте результат в консоли
console.log( number.invert() );
```

Расширение прототипов встроенных объектов считается не очень хорошей практикой, поскольку ухудшает переносимость кода, однако бывают ситуации когда это оправдано.

Задание №9

Создайте простой метод для объекта Number, который должен переводить значение числа-процента в доли единицы, при это если значение числа меньше 0, то нужно возвращать 0. Если больше 100, то нужно возвращать 1.

Задание №10

Цикл for / in используется для обхода свойств и методов объекта, про который нам ничего неизвестно. Создадим простой цикл for / in, как показано ниже:

```
var obj = {
    a: 1,
    b: 2,
    c: 3
};

for (var i in obj) {
    console.log( i + ': ' + obj[i] );
}
```

Следует помнить, что в переменную «i» попадает только название свойства, а не его значение!

Задание №11

Для объектов оператор присваивания работает несколько особенным образом, рассмотрим это на примере кода:

```
// создаем простой объект
var object = {
    a: 1,
    b: 2,
    c: 3
}

// пытаемся копировать объект
var object2 = object;

// проверяем это две разный копии объекта или один и тот же объект
object.a = 111;

console.log( object );
console.log( object2 );
```

Что мы видим в консоли?

Модификация переменной «object» привела к модификации объекта в переменной «object2». Отсюда делаем вывод, что обе эти переменные ссылаются на один и тот же объект и это действительно так. Объекты не клонируются, в отличии от простых значений, а создаются ссылки на один и тот же объект. Это нужно для быстродействия JS и в большинстве случаев это ожидаемое поведение.

Что бы создать полную копию объекта нужно перебрать его свойства и методы в цикле for in и последовательно присвоить другому объекту. Сделайте это самостоятельно и создайте клон объекта «object» с помощью цикла for in.

Задание №12

Очень часто нужно обратиться к определённому свойству объекта, даже когда точно не известно существует оно или нет, для проверки существования свойств и методов объекта используется оператор ***in.***

Рассмотрим несколько примеров использования оператора in:

```
// создаём объект
var obj = {
    prop: true,

    foo: function() {}
};

// эти свойства существуют, вернёт true
console.log( 'prop' in obj );
console.log( 'foo' in obj );

// такого свойства нет
console.log( 'val' in obj );
```

Теперь попытаемся вызвать существующее свойство «foo» и отсутствующее свойство «val»:

```
obj.foo();
obj.val();
```

Какой результат мы получили? Самостоятельно перед вызовом метода «val» добавьте проверку на его существование.