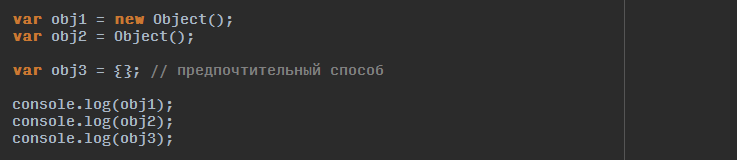
**Практическое занятие №12 Объекты в JS**

**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:



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**

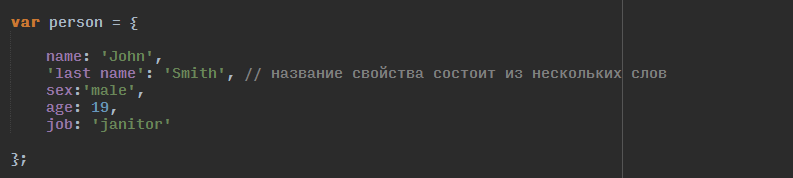
Пары «***имя свойства/метода: значение свойства/метода***» разделяются запятой. После последней пары запятая не ставится (иначе возникнут проблемы в некоторых версиях IE). Если не писать ничего внутри фигурных скобок, будет создан пустой объект.

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.

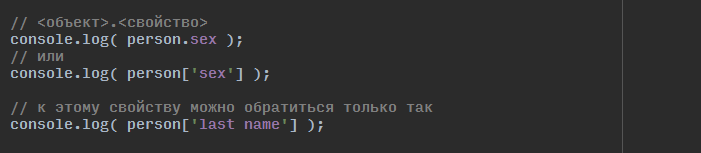
В файле задания создаём объект person, как показано ниже:



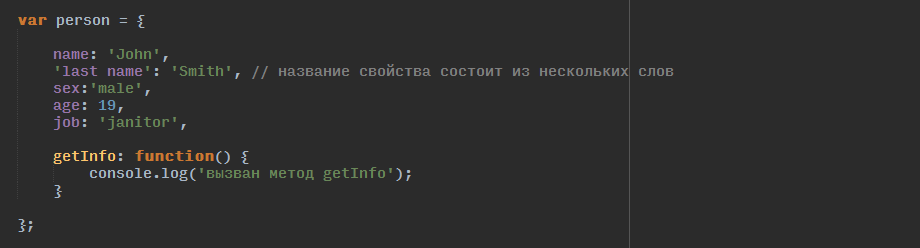
Теперь добавляем ему свойства:



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



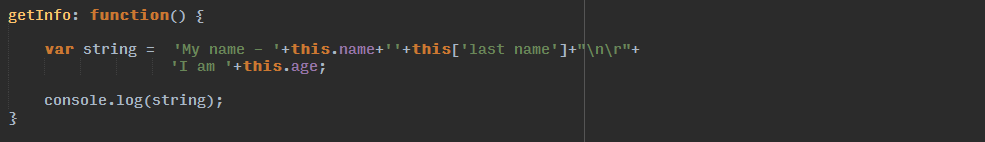
Следует понимать, что свойства - это информация которой располагает объект, так же у объектов есть «поведение», то есть методы. Методы создаются как функции внутри объекта. Добавим нашему объекту метод:



Вызвать данный метод вне объекта можно следующим образом:

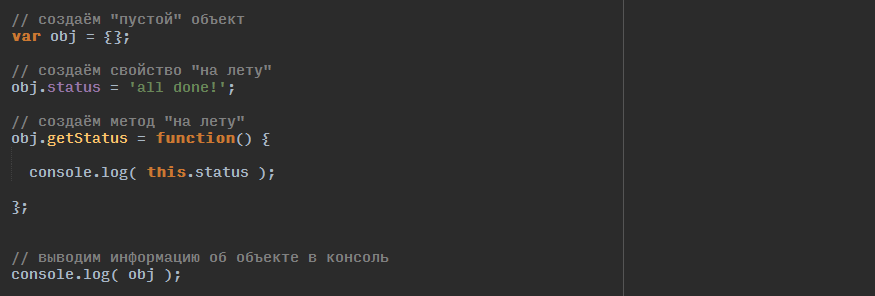


Сейчас мы научились создавать и вызывать методы объектов. Однако наш метод «getInfo» ужасно не функционален. Для того что бы улучшить его нам нужно ВНУТРИ метода получить доступ к свойствам объекта, сделать это можно с помощью ключевого слова «this», которое внутри объекта ссылается на сам объект. Перепишем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



**Задание №4**

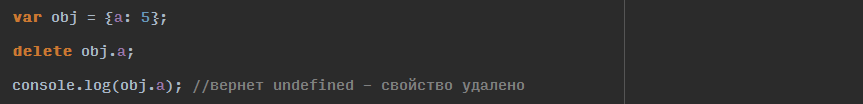
В JS так же существует возможность создавать методы и свойства объекта не только при его создании, а уже когда объект существует. Рассмотрим и протестируем пример кода:



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:

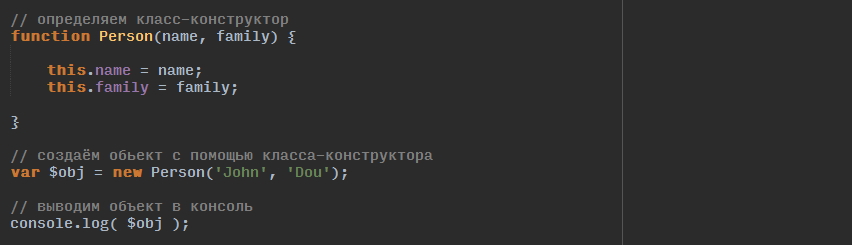


**Задание №6**

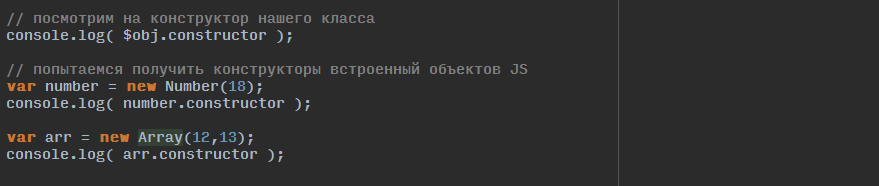
Все типы данных в JavaScript, кроме тривиальных типов (null, undefined) имеют собственные функции-конструкторы. Можно было заметить, что при их вызове используется оператор вызова функций “()”.



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



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:



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



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**

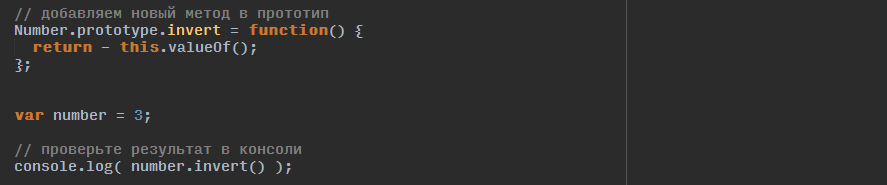
Мы отмечали, что все объекты получают свойства и методы от Object, а экземпляры этих типов данных наследуют свойства и методы от своих классов-конструкторов. Как же происходит «наследование» (с более классическим наследованием мы познакомимся в PHP), то есть передача свойств и методов от объекта к объекту,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



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:



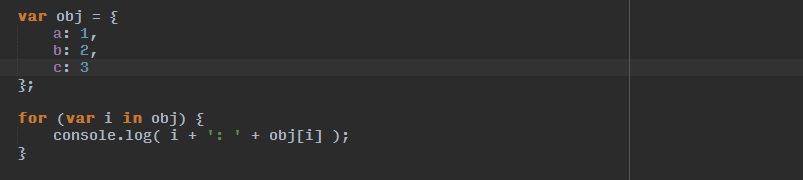
Расширение прототипов встроенных объектов считается не очень хорошей практикой, поскольку ухудшает переносимость кода, однако бывают ситуации когда это оправдано.

**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**

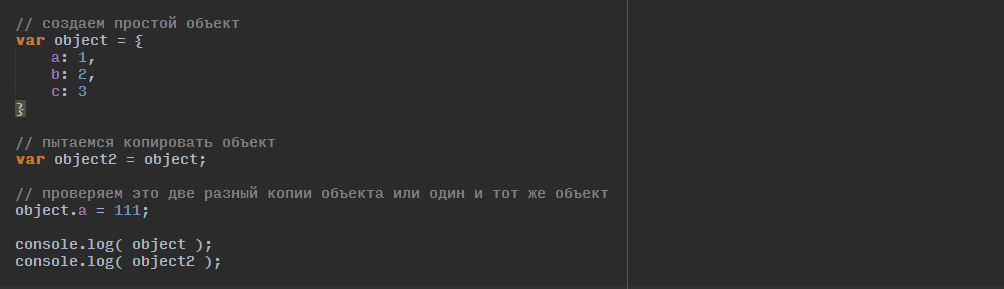
Цикл for / in используется для обхода свойств и методов объекта, про который нам ничего неизвестно. Создадим простой цикл for / in, как показано ниже:



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:



Что мы видим в консоли?

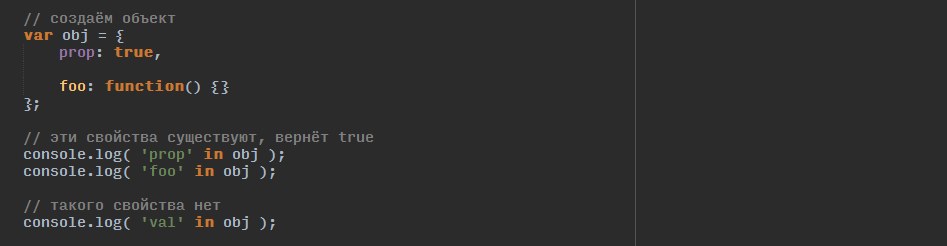
Модификация переменной «object» привела к модификации объекта в переменной «object2». Отсюда делаем вывод, что обе эти переменные ссылаются на один и тот же объект и это действительно так. Объекты не клонируются, в отличии от простых значений, а создаются ссылки на один и тот же объект. Это нужно для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



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



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