МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Воронежский государственный технический университет»

Кафедра графики, конструирования и информационных технологий
в промышленном дизайне

537-2021

WEB-ДИЗАЙН

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению самостоятельных работ для студентов, обучающихся
по направлению 54.03.01 «Дизайн» (профиль «Промышленный дизайн»),
всех форм обучения

Воронеж 2021

УДК 681.3(07)
ББК 30.18я7

Составители: А. В. Кузовкин, А. П. Суворов, Ю. С. Золототрубова

Web-дизайн: методические указания к выполнению самостоятельных работ для студентов, обучающихся по направлению 54.03.01 «Дизайн» (профиль «Промышленный дизайн»), всех форм обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: А. В. Кузовкин, А. П. Суворов, Ю. С. Золототрубова. – Воронеж: Изд-во ВГТУ, 2021. – 22 с.

В методических указаниях приводится описание выполнения самостоятельных работ по курсу «Web-дизайн».

Предназначены для студентов, обучающихся по направлению 54.03.01 «Дизайн» (профиль «Промышленный дизайн»), всех форм обучения.

Методические указания подготовлены в электронном виде и содержатся в файле МУ_СР_Веб-дизайн.pdf

Ил. 3. Библиогр.: 4 назв.

УДК 681.3(07)
ББК 30.18я7

Рецензент – Е. В. Смоленцев, д-р техн. наук, проф. кафедры технологии машиностроения

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Методические указания по дисциплине «Web-дизайн» предназначены помочь студентам овладеть первоначальными навыками и умениями в разработке и создание Web-сайтов. Материал приводится в виде практических пошаговых инструкции с описанием теоретического материала и самостоятельной частью, которую студент выполняет сам. Данное методическое пособие посвящено вопросам разработки дизайна и верстки Web-сайтов. В нем представлены основные методы создания Web-сайтов на основе языка гипертекстовой разметки HTML: форматирование текста, использование таблиц и графики и т.д. Проводятся основные теги языка HTML и способы их применения. При этом не требуется знаний в области программирования достаточно базовых знаний владения компьютером.

Web-дизайн — вид графического дизайна, направленный на разработку и оформление объектов информационной среды интернета, призванный обеспечить им высокие потребительские свойства и эстетические качества. Подобная трактовка отделяет веб-дизайн от веб-программирования, подчеркивает специфику предметной деятельности веб-дизайнера, позиционирует веб-дизайн как вид графического дизайна.

В настоящее время под термином *веб-дизайн* понимают именно проектирование структуры веб-ресурса, обеспечение удобства пользования ресурсом для пользователей.

ЛАБОРАТОРНАЯ РАБОТА № 1
ОСНОВЫ HTML

Цель: познакомиться с основными тегами html.

Задачи:
1. Научиться создавать html документ.
2. Научиться работать с таблицами в html.

Теоретические сведения

HTML-документ – это обычный текстовый документ, может быть создан как в обычном текстовом редакторе **(Блокнот)**, так и в специализированном, с подсветкой кода **(Notepad++).** HTML-документ имеет расширение .html.

Язык HTML – язык тегов. **Теги** описывают структуру HTML-документа. Теги оформляются угловыми скобками <имя тега>, между которыми прописывается имя тега.

Элемент <**html**> является корневым элементом документа. Все остальные элементы содержатся внутри тегов <html>...</html>. Все, что находится за пределами тегов, не воспринимается браузером как код HTML и никак им не обрабатывается.

Элемент <**head**>. Раздел <head>...</head> содержит техническую информацию о странице: заголовок, описание, ключевые слова для поисковых машин, кодировку и т.д.

Элемент <**title**>. Обязательным тегом раздела <head> является тег <title>. Текст, размещенный внутри этого тега, отображается в строке заголовка веб-браузера.

Элемент <**style**>. Внутри этого элемента задаются стили, которые используются на странице. Для задания стилей в html-документе используется язык CSS.

Элемент <**body**>. В этом разделе располагается все содержимое документа.

HTML таблицы состоят из ячеек, образующихся при пересечении строк и столбцов.

Ячейки таблиц могут содержать любые HTML-элементы, такие как заголовки, списки, текст, изображения, элементы форм, а также другие таблицы.

Основные теги для создания таблицы:

- caption - заголовок для таблицы
- col - указывает стилевую информацию для одной или более колонок таблицы
- colgroup - указывает общую стилевую информацию для группы колонок в таблице
- table - таблица

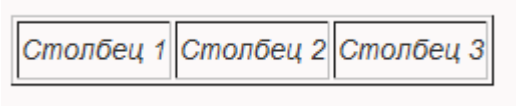
- tbody - основное содержимое (средняя часть) таблицы
- td - ячейка таблицы
- tfoot - нижняя часть таблицы
- th - заголовочная ячейка таблицы
- thead - верхняя часть таблицы
- tr - строка таблицы

Ход выполнения

Давайте создадим простую таблицу, которая состоит из 3 столбцов. HTML-код таблицы:

```
<table>
 <tr>
  <td>Столбец 1</td>
  <td>Столбец 2</td>
  <td>Столбец 3</td>
 </tr>
</table>
```

На странице в браузере таблица будет выглядеть вот так:

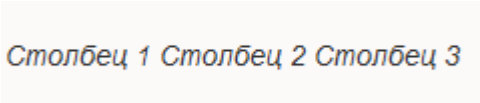


По умолчанию браузер выводит таблицу с рамкой. Чтобы скрыть рамку таблицы, вам понадобится атрибут тега TABLE border. Пример таблицы без рамки:

```
<table bor-
der="0">
 <tr>
  <td>Столбец
1</td>
  <td>Столбец
2</td>
  <td>Столбец
3</td>
 </tr>
```

```
</table>
```

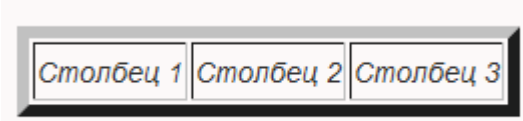
На странице мы увидим нашу таблицу без рамки:



Атрибут border изменяет величину лишь внешней рамки нашей таблицы. Значение 0, как вы поняли, делает её невидимой, а значение от 1 до 10 задает её толщину. Примедем пример таблицы с рамкой 6:

```
<table border="6">
 <tr>
  <td>Столбец
1</td>
  <td>Столбец
2</td>
  <td>Столбец
3</td>
 </tr>
</table>
```

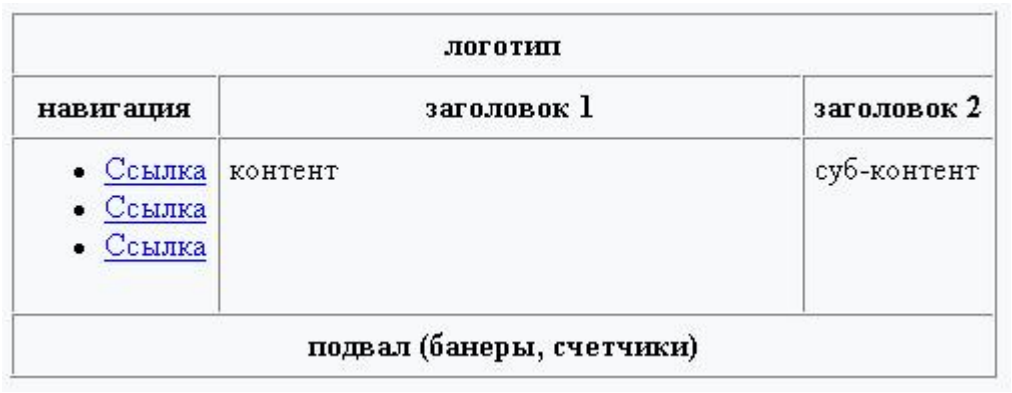
На выходе страницы получим такую таблицу:



Таким образом можно создавать таблицы и изменять их внешний вид.

Самостоятельная работа

Создать дизайн макет сайт на основе табличной верстки.



Контрольные вопросы

1. Теги для создания ссылок.
2. Теги для работы с таблицами.
3. Теги для работы с картинками.
4. Структура HTML документа.

ЛАБОРАТОРНАЯ РАБОТА № 2
БЛОЧНАЯ ВЕРСТКА

Краткие теоретические сведения

В HTML4 и XHTML слой — это элемент веб-страницы, созданный с помощью тега ** **, к которому применяется стилевое оформление.

Таким образом, выражение «блочная вёрстка» или вёрстка с помощью слоёв заключается в конструктивном использовании тегов ** ** и стилей. При этом придерживаются следующих принципов.

Разделение содержимого и оформления

Код HTML должен содержать только теги разметки и теги логического форматирования, а любое оформление выносится за пределы кода в стили. Такой подход позволяет независимо управлять видом элементов страницы и её содержимым. Благодаря этому над сайтом может работать несколько человек, при этом каждый выполняет свою функцию самостоятельно от других. Дизайнер, верстальщик и программист работают над своими задачами автономно, снижая время на разработку сайта.

**Активное применение тега **

При блочной вёрстке существенное значение уделяется универсальному тегу ** **, который выполняет множество функций. Фактически это основа, на которую «навешиваются» стили, превращая её то в игрушку, то в зверушку. Совершенно не значит, что применяется только один этот тег, нужно ведь и ри-

сунки вставлять и оформлять текст. Но при вёрстке с помощью слоёв тег ** ** является кирпичиком вёрстки, её базовым фундаментом.

Благодаря этому тегу HTML-код распадается на ряд чётких наглядных блоков, код при этом получается более компактным, чем при табличной вёрстке, к тому же поисковые системы его лучше индексируют.

Таблицы применяются только для представления табличных данных

При блочной вёрстке, конечно же, используются таблицы, но только в тех случаях, когда они нужны, например, для наглядного отображения чисел и других табличных данных. Вариант, когда от таблиц предлагается отказаться вообще, является нецелесообразным и, более того, вредным.

В HTML4 и XHTML слой это базовый элемент вёрстки веб-страниц, при которой активно применяются стили и придерживаются спецификаций HTML и CSS. При таком подходе важная роль уделяется тегу ** **, с которым у большинства людей и ассоциируются слои. В каком-то смысле это является верным, поэтому договоримся в дальнейшем употреблять термин «слой» к тегу ** ** для которого указан стилевой идентификатор или класс. Таким образом, выражение «слой с именем content» подразумевает, что используется тег или .

В HTML5 добавлено несколько новых тегов разметки для обозначения разных типовых блоков страницы. К примеру, **<header>** и **<footer>** используются для создания «шапки» и «подвала», **<nav>** для навигации, **<aside>**для боковой панели. Включение в спецификацию HTML подобных элементов призвано снизить доминирование тега ** ** и придать больше смысла разметке. Поэтому в вёрстке на HTML5 активно применяется термин «элемент», под которым подразумевается соответствующий тег и элемент который он создаёт.

Изложенные выше принципы блочной вёрстки при этом сохраняются за исключением того момента, что ** ** в некоторых случаях заменяется более осмысленными тегами.

Блочная модель

Любой блочный элемент состоит из набора свойств, подобно капустным листам накладываемых друг на друга. Основой блока выступает его контент (это может быть текст, изображение и др.), ширина которого задается свойством width, а высота через height; вокруг контента идут поля (padding). Они создают пустое пространство от контента до внутреннего края границ; затем идут собственно сами границы (border) и завершают блок отступы (margin), невидимое пустое пространство от внешнего края границ. Порядок влияния этих свойств на блок четко определён и не может быть нарушен. На рис. 1 показан блок в виде набора этих свойств.

Рис. 1. Поля (padding)

Полем будем называть расстояние от внутреннего края границы или края блока до воображаемого прямоугольника, ограничивающего содержимое блока. Из-за того, что значения полей могут различаться на каждой стороне, применяют выражения «верхнее поле» или «поле сверху», и им подобные для других сторон. Обозначение «поля» следует понимать как одинаковое значение полей для всех сторон. Основное предназначение полей — создать пустое пространство вокруг содержимого блочного элемента, например текста, чтобы он не прилегал плотно к краю элемента.

Границы — это линии вокруг полей элемента на одной, двух, трёх или всех четырёх его сторонах. У каждой линии есть толщина, стиль и цвет. Для создания рамки применяется универсальное свойство border одновременно задающее все эти параметры, а для создания линий на отдельных сторонах элемента можно воспользоваться свойствами border-left,border-top, border-right и border-bottom, соответственно устанавливающих границу слева, сверху, справа и снизу. В примере 3.2 показано добавление линии слева от элемента.

Отступом будем называть пустое пространство от внешнего края границы, полей или содержимого блока. Как уже упоминалось, границы с полями не обязательны и могут отсутствовать, так что способ формирования отступов зависит от ситуации. Как и в случае с полями, применяют выражения «верхний отступ» или «отступ сверху», и им подобные для других сторон. Обозначение «отступы» следует понимать как одинаковое значение отступов для всех сторон.

Для отступов характерны следующие особенности.

- Отступы прозрачны, на них не распространяется цвет фона или фоновая картинка, заданная для блока. Однако если фон установлен у родительского элемента, он будет заметен и на отступах.
- Отступы в отличие от полей могут принимать отрицательное значение, это приводит к сдвигу всего блока в указанную сторону. Так, если задано margin-left: -10px, это сдвинет блок на десять пикселей влево.
- Для отступов характерно явление под названием «схлопывание», когда отступы у близлежащих элементов не суммируются, а объединяются между собой.
- Отступы, заданные в процентах, вычисляются от ширины контента блока. Это касается как вертикальных, так и горизонтальных отступов.

Ширина блока — это комплексная величина и складывается из нескольких значений свойств:

- width — ширина контента, т.е. содержимого блока;
- padding-left и padding-right — поле слева и справа от контента;
- border-left и border-right — толщина границы слева и справа;
- margin-left и margin-right — отступ слева и справа.

Как уже упоминалось, какие-то свойства могут отсутствовать и в этом случае на ширину не оказывают влияние.

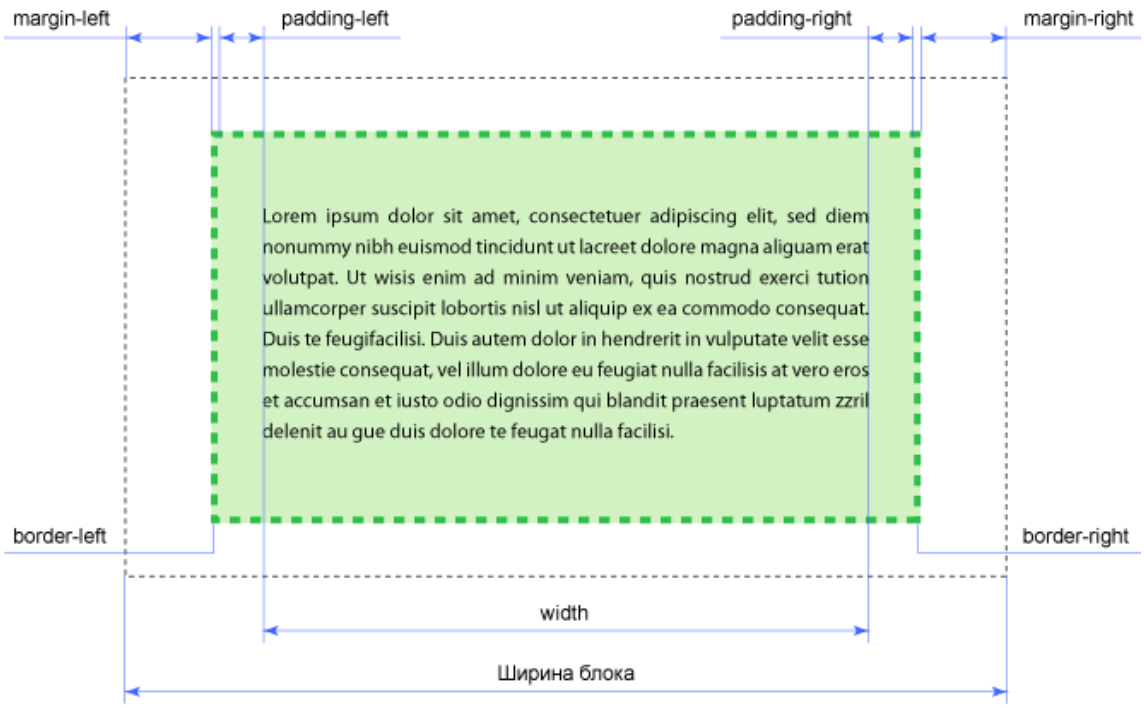


Рис. 2

На высоту блока действуют те же правила, что и на ширину. А именно, высота складывается из значений высоты контента (height), полей (padding), границ (border) и отступов (margin) (рис. 2). Если свойство height не указано, то

оно считается как auto, в этом случае высота контента вычисляется автоматически на основе содержимого (рис. 3).

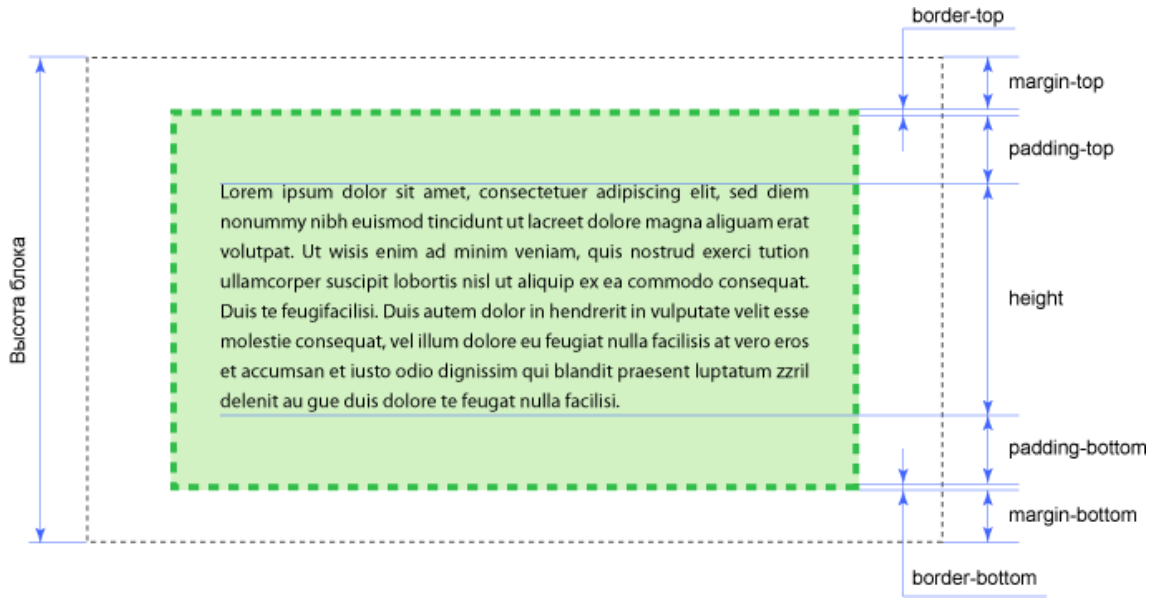


Рис. 3

Самостоятельная работа

Разработать дизайн сайта на основе блочной верстки.

Контрольные вопросы

1. Что такое ?
2. Что представляет собой блочная модель?
3. Какие существуют основные параметры настройки блока?

ЛАБОРАТОРНАЯ РАБОТА № 3
ЦИКЛЫ В JAVASCRIPT

Цель: научится работать с циклами в *Javascript.*
Задачи: познакомится с циклами в *Javascript.*

Теоретические сведения

При написании скриптов зачастую встает задача сделать однотипное действие много раз.

Например, вывести товары из списка один за другим. Или просто перебрать все числа от 1 до 10 и для каждого выполнить одинаковый код.

Для многократного повторения одного участка кода – предусмотрены *циклы*.

Цикл while

Цикл while имеет вид:

```
while (условие) {
  // код, тело цикла
}
```

Пока условие верно — выполняется код из тела цикла.

Например, цикл ниже выводит i пока i < 3:

```
var i = 0;
while (i < 3) {
  alert( i );
  i++;
}
```

Повторение цикла по-научному называется «итерация». Цикл в примере выше совершает три итерации.

Если бы i++ в коде выше не было, то цикл выполнялся бы (в теории) вечно. На практике, браузер выведет сообщение о «зависшем» скрипте и посетитель его остановит.

Цикл do..while

Проверку условия можно поставить под телом цикла, используя специальный синтаксис do..while:

```
do {
  // тело цикла
} while (условие);
```

Цикл, описанный, таким образом, сначала выполняет тело, а затем проверяет условие.

Например:

```
var i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Синтаксис do..while редко используется, т.к. обычный while нагляднее — в нём не приходится искать глазами условие и ломать голову, почему оно проверяется именно в конце.

Цикл for.

Чаще всего применяется цикл for. Выглядит он так:

```
for (начало; условие; шаг) {

  // ... тело цикла ...

}
```

Пример цикла, который выполняет alert (i) для i от 0 до 2 включительно (до 3):

```
var i;


for (i = 0; i < 3; i++) {

  alert( i );

}
```

Здесь:

- Начало: i=0.
- Условие: i<3.
- Шаг: i++.
- Тело: alert(i), т.е. код внутри фигурных скобок (они не обязательны, если только одна операция)

Цикл выполняется так:

1. Начало: i=0 выполняется один-единственный раз, при заходе в цикл.
2. Условие: i<3 проверяется перед каждой итерацией и при входе в цикл, если оно нарушено, то происходит выход.
3. Тело: alert(i).
4. Шаг: i++ выполняется после *тела* на каждой итерации, но перед проверкой условия.
5. Идти на шаг 2.

Самостоятельная работа

Написать скрипт выводящий на экран таблицу Пифагора.

Контрольные вопросы

1. Цикл For.
2. Цикл while.

ЛАБОРАТОРНАЯ РАБОТА № 4
МАССИВЫ В JAVASCRIPT

Цель: научится работать с массивами в Javascript.
Задачи: Создать массив в Javascript.

Теоретические сведения

Массив – это тип данных, хранящий пронумерованные значения. Каждое пронумерованное значение называется элементом массива, а число, с которым связывается элемент, называется его индексом. Массивы JavaScript нетипизированы, это значит, что элемент массива может иметь любой тип, причем разные элементы одного массива могут иметь разные типы. Помимо этого массивы JavaScript являются динамическими, это значит, что объявлять фиксированный размер не нужно и можно добавить новые элементы в любое время.

Создание массива

Массив можно создать двумя способами, первый: создать массив с помощью литерала массива - квадратные скобки, внутри которых расположен список элементов, разделенных запятыми.

```
var empty = [];  //пустой массив

var numbers = [4, 1, 2, 5]; //массив с 5 числовыми элементами

var diff = [1.5, false, "текст"];  //массив с 3 элементами различного типа
```

Значения не обязательно должны быть простыми (числа или строки) - это также могут быть и любые другие выражения, например: литералы объектов, другие массивы или функции.

```
var num = 700;

var tab = [function(a) { alert(a) }, { name: 'Петя' }, [1, 2, 3], num + 1];
```

Второй способ создания массива - вызов конструктора Array(). Вызвать конструктор Array() можно тремя способами.

Вызов конструктора без аргументов:

```
var b = new Array();
```

В этом случае создается пустой массив, эквивалентный пустому литералу [].

В конструкторе явно указываются значения n элементов массива:

```
var b = new Array(1, 3, 5, 8, "строка", true);
```

В этом случае конструктор получает список аргументов, которые становятся элементами нового массива. Аргументы записываются в массив в том порядке, в котором указаны.

Выделение места для последующего присваивания значений. Это делается путем указания при определении массива одного числа в круглых скобках:

```
var b = new  Array(5);
```

Этот способ определения массива предполагает выделение массиву определенного количества элементов (каждый из которых имеет значение undefined) с возможностью последующего присваивания значений по ходу сценария. Такая форма обычно используется для предварительного размещения массива, если его длина известна заранее.

ЛАБОРАТОРНАЯ РАБОТА № 5
ОБЪЕКТЫ JAVASCRIPT

Краткие теоретические сведения

Объекты (они же - ассоциативные массивы, хэши) и работа с ними в Javascript - реализованы не так, как в большинстве языков. С этим связано много ошибок.

Рассмотри базовые свойства объектов javascript, создание и изменение, перечисление свойств.

Объект в javascript представляет собой обычный ассоциативный массив или, иначе говоря, "хэш". Он хранит любые соответствия "ключ => значение" и имеет несколько стандартных методов.

Метод объекта в javascript - это просто функция, которая добавлена в ассоциативный массив. Далее - подробнее.

Создание и работа со свойствами

Создание объекта

Следующие два варианта создания объекта эквивалентны:

```
// эквивалентные записи
var o = new Object()

var o = {}
```

Добавление свойств

Есть два синтаксиса добавления свойств в объект. Первый - точка, второй - квадратные скобки:

// эквивалентные записи

```
o.test = 5
```

```
o["test"] = 5
```

Квадратные скобки используются в основном, когда название свойства находится в переменной:

```
var name
= 'test'
```

```
o[name] = 5
```

Здесь имя свойства "test" является ключом в ассоциативном массиве, по которому лежит значение 5.

Доступ к свойствам

Доступ к свойству осуществляется точно так же:

alert(o.test)

```
alert(o['test'])
```

Если у объекта нет такого свойства, то результат будет 'undefined'

```
var o = {}
```

```
alert(o.nosuchkey)   // =>
undefined
```

Никакой ошибки при обращении по несуществующему свойству не будет, просто вернется специальное значение undefined.

Проверка глобальной переменной

В javascript нельзя проверить существование глобальной переменной простым if:

```
if (x) { ...
}
```

Если x не определен, то конструкция if (x) вызовет ошибку javascript. Распространенное решение – использовать typeof:

```
if (typeof x != 'undefined') { ... }  // или
typeof(x)
```

Однако зная, что глобальная переменная в javascript - всего лишь свойство объекта window - мы можем записать проще:

```
     if (window.x) { ... }   // правильный аналог
if(x)

     // или

     if (window.x !== undefined) // аналог typeof
x .
```

Все свойства объектов - public, т.е при определении свойства никак нельзя ограничить доступ к свойству. В javascript есть специальные выверты для создания private свойств, связанные с замыканиями. Они рассмотрены вместе с наследованием объектов.

Удаление свойств:

Удаляет свойство оператор delete:

```
o.test = 5

delete o.test

o['bla'] = true
```

Расширенное создание:

Свойства можно указывать непосредственно при создании объекта, через список в фигурных скобках вида {..., ключ : значение, ...}:

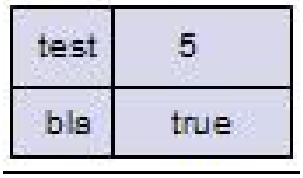
```
Var 0 = {

     Test: 5,

     Bla: true

     }
```

Получившийся объект можно изобразить так:



Методы объектов

Добавление методов

Как и в других языках, у объектов javascript есть методы.
Например, создадим объект rabbit с методом run

1) var rabbit = {}

2)rabbit.run = function(n) {

3)alert("Пробежал "+n+" метров!")}

Добавление метода в объект - просто присвоение функции function(n) { ... }свойству rabbit.run.

Теперь можно запускать

```
var rabbit = {}

rabbit.run = function(n) {

    alert("Пробежал "+n+" метров!")

}

rabbit.run(5)  // Пробежал 5 метров

rabbit.run(7)  // Пробежал 7 метров
```

Здесь не идет речь о классах, создании экземпляров и тому подобном. Просто - в любой объект в любое время можно добавить новый метод или удалить существующий.

Доступ к объекту из метода

Обычно хочется, чтобы метод не просто вызывался из объекта, но имел доступ к самому объекту, мог менять находящиеся в нем данные.

Для этого используется ключевое слово this:

```
for(var key in obj) {

… obj[key] …

}
```

В отличие от многих языков, this никак не привязано к объекту, а обозначает просто **объект, вызвавший функцию**.

Например,

```
1   function this1() {

        var vasya = { name:'Вася' }
2
                                15
        var petya = { ame:'Пет..  ,
3
```

```
4
5           sayName  = function() {
6               alert("Я - "+ (this.name ? this.name : 'безымянный') )
7           }
8
9           vasya.sayName = sayName
0
1           // один и тот же метод в двух объектах
2           petya.sayName = vasya.sayName
3
4           // тут - this будет petya
5           petya.sayName()  // Я - Петя
6
7           // тут - this будет vasya
8           vasya.sayName()  // Я - Вася
9
0           // а тут - вызывается метод глобального объекта window, у ко-
  торого нет имени
1           sayName() // Я - безымянный
```

```
2       }
```

Перебор свойств объекта

Для перебора всех свойств объекта используется специальный вид конструкции for,for..in:

```
for(var key in object) {

 // key - название свойства

 // object[key] - значение свойства

   ...

}
```

Например,

```
var o = {a:5, b:true}

for (var key in o) {

    alert(key+':'+o[key])
```

Самостоятельная работа

Написать скрипт, формирующий галерею на основе массивов в JavaScript.

Контрольные вопросы

1. Двухмерные массивы в JavaScript.
2. Определение массивов в JavaScript.
3. Присваивание значений элементам массива.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алексеев, А. П. Введение в WEB-дизайн: учеб. пособие для вузов [Текст] / А. П. Алексеев. – М.: СОЛОН-ПРЕСС, 2008. – 184 с.

2. Евсеев, Д. А. Web-дизайн в примерах и задачах: учеб. пособие для вузов [Текст] / Д. А. Евсеев, В. Р. Трофимов; под. ред. В. В. Трофимова. – М.: КНОРУС, 2010. – 272 с.

3. Рейсинг, Джон. JavaScript. Профессиональные приемы программирования: учеб. пособие [Текст] / Джон Рейсинг. – СПб.: Питер, 2008. – 351 с.

4. Сырых, Ю. А. Современный веб-дизайн. Рисуем сайт, который продает [Текст] / Ю. А. Сырых. – М.: И.Д. Вильямс, 2008. – 304 с.

ОГЛАВЛЕНИЕ

WEB-ДИЗАЙН

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению самостоятельных работ
для студентов, обучающихся по направлению 54.03.01 «Дизайн»
(профиль «Промышленный дизайн»), всех форм обучения

Составители:
Кузовкин Алексей Викторович
Суворов Александр Петрович
Золототрубова Юлия Сергеевна

Издается в авторской редакции

Подписано к изданию 08.11.2021.
Уч.-изд. л. 1,4.

ФГБОУ ВО «Воронежский государственный технический университет»
396026 Воронеж, Московский просп., 14