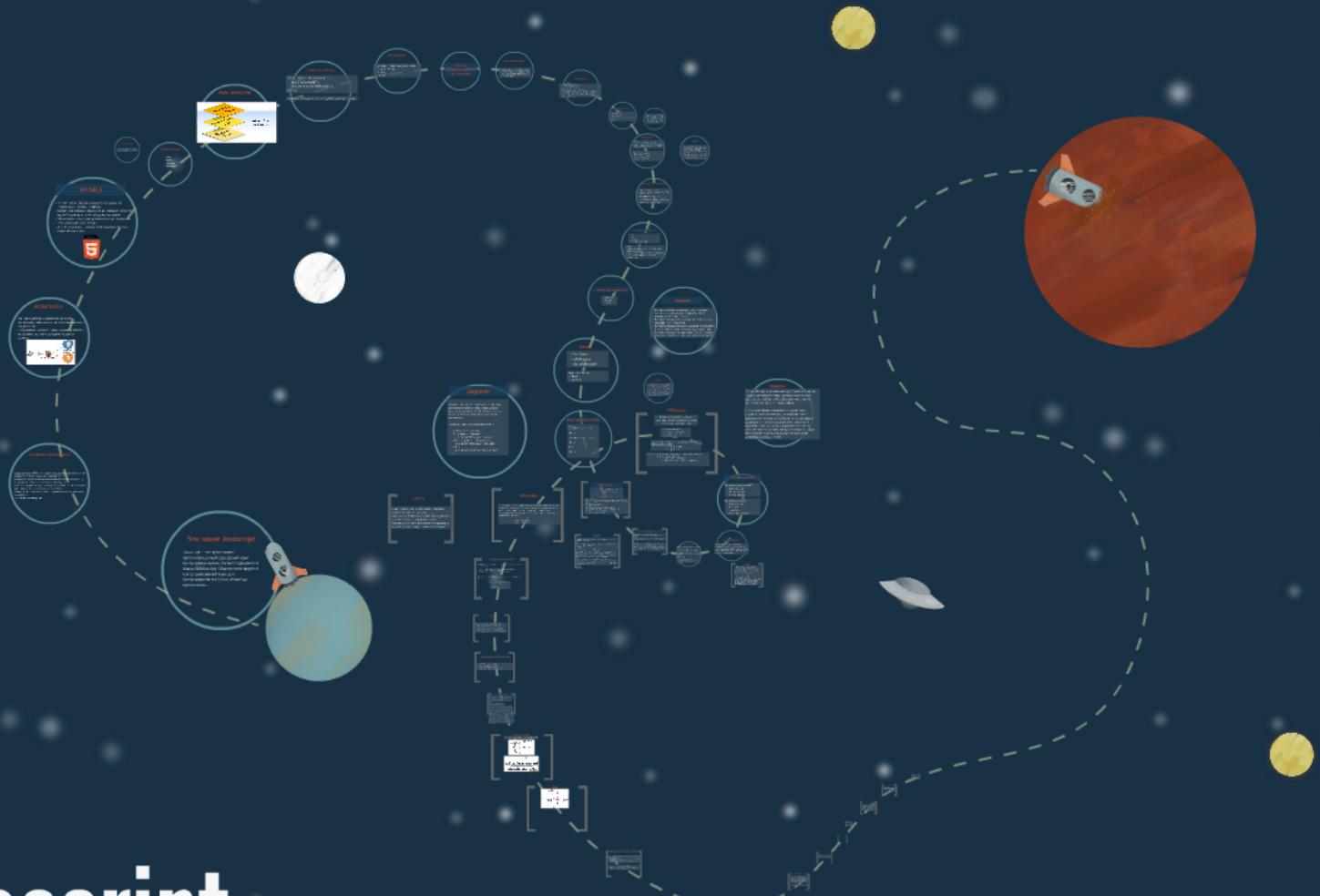




# Javascript

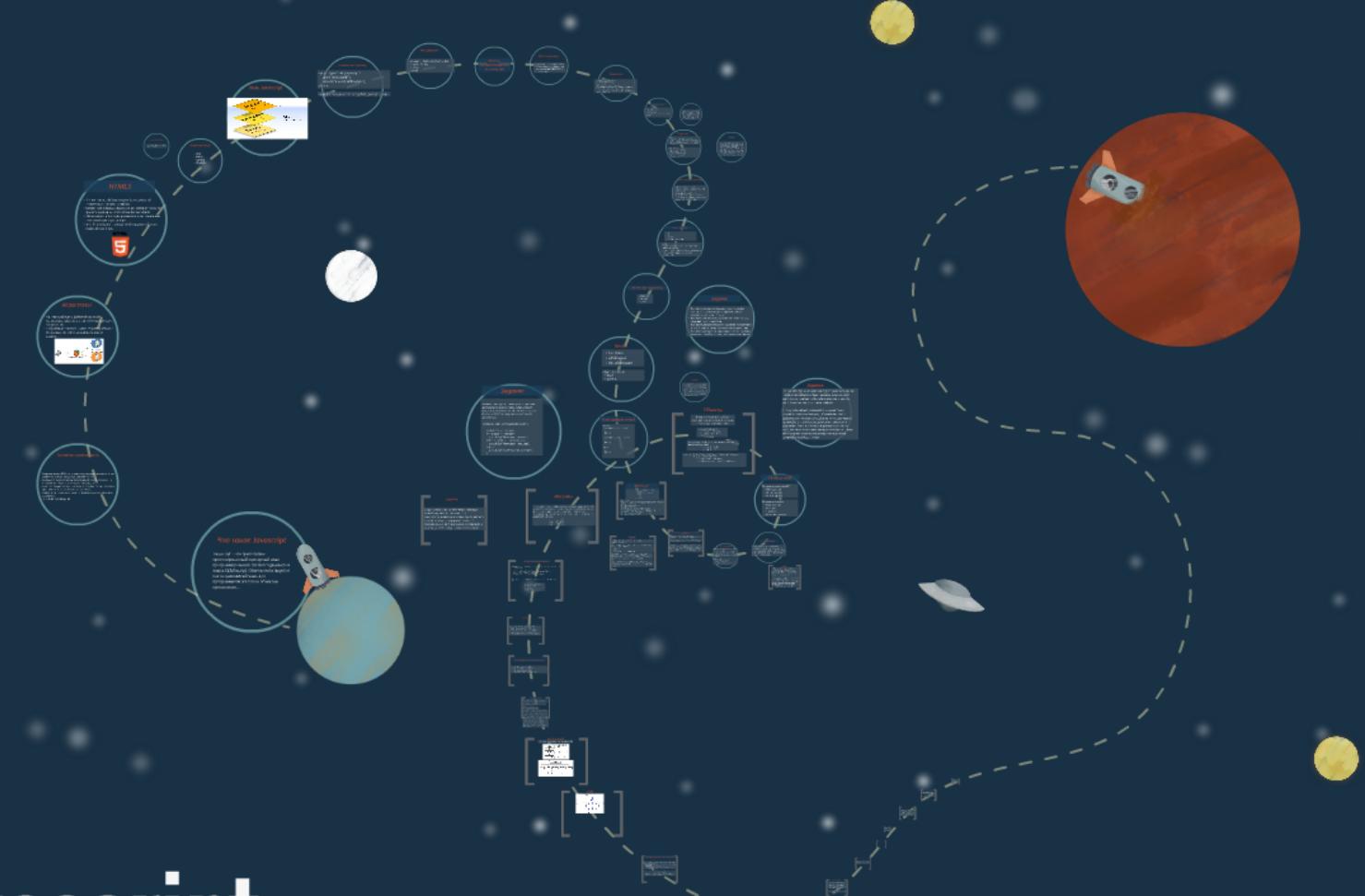
## введение





# Javascript

## введение



## *Что такое Javascript*

Javascript - это прототипно-ориентированный сценарный язык программирования. Являетсяialectом языка ECMAScript. Обычно используется как встраиваемый язык для программного доступа к объектам приложения....

## **Основные возможности**

- Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.
- Реагировать на действия посетителя, обрабатывать клики мыши, перемещение курсора, нажатие на клавиатуру и т.п.
- Посыпать запросы на сервер и загружать данные без перезагрузки страницы(эта технология называется "AJAX").
- Получать и устанавливать cookie, запрашивать данные, выводить сообщения...
- ...и многое, многое другое!

# НЕДОСТАТКИ

- Не умеет работать с файловой системой компьютера, работать с внешними программами, запускать их...
- Запущенный в одной вкладке, не умеет работать с остальными если они запущены на другом домене...



# **HTML5**

- Чтение/запись файлов на диск (в специальной «песочнице», то есть не любые).
- Встроенная в браузер база данных, которая позволяет хранить данные на компьютере пользователя.
- Многозадачность с одновременным использованием нескольких ядер процессора.
- 2d и 3d-рисование с аппаратной поддержкой, как в современных играх.



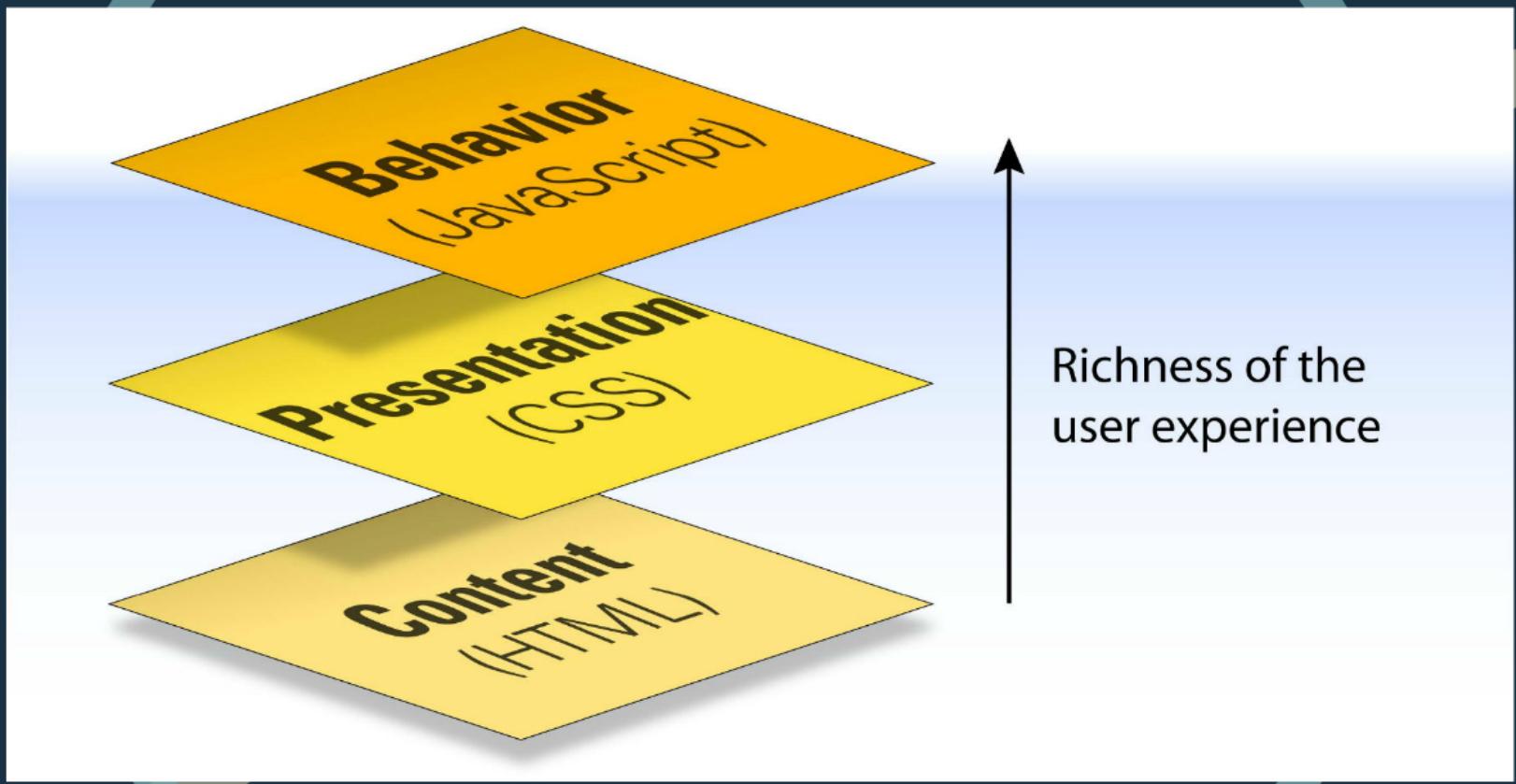
## *Среда исполнения*

- Браузер (Chrome / Firefox/ Opera)
- Сервер (node.js)

## **АЛЬТЕРНАТИВЫ**

- JAVA
- FLASH
- ACTIVEX
- Silverlight

# Роль Javascript



## *Вставка на страницу*

```
<script type="text/javascript">  
    alert("Hello world!");  
    document.write("Hello world");  
</script>
```

```
<script type="text/javascript" src=".js/hello_world.js"></script>
```

## *Инструменты*

- phpStorm / WebStorm /Sublime Text
- Chrome / Firefox
- Firebug
- GitHub

# *Основы программирования на Javascript*

## *Организация кода*

- Команды - строка кода которая создает переменную или изменяет ее, вызывает функцию и т.д...
- Комментарии

# *Переменные*

- Объявляются через директиву var
- Регистрозависимы
- Имеют тип данных
- Могут меняться в течении выполнения кода
- Имя переменной может состоять из: букв, цифр, символов \$ и \_
- Имя переменной не должно начинаться с цифры
- Нельзя использовать для имени зарезервированные слова var, class, return, implements и т.д

## Типы данных

- Числа (numbers)
- Строки (strings)
- Логические ( boolean)
- null ( означает "ничего")
- undefined ( означает что переменной не присвоено значение)
- Object

Создать переменные типа "string". В одной будет содержаться ваше имя, а в другой ваша фамилия. Также создать одну переменную типа "number" в которой будет содержаться ваш возраст и вывести значение этих переменных в консоль.

# *Операторы*

## Термины

- Операнд - цель применения оператора
- Унарный оператор - применяется к одному операнду
- Бинарный оператор - применяется к нескольким операндам

## Типы операторов

- Арифметические операторы
- Присваивание
- Инкремент/декремент ++ --
- Побитовые операторы (&
- Оператор запятая

## Задание

- Создать две переменные типа number с произвольным значением. Выполнить над ними несколько арифметических операций. Результаты всех операций выводить в консоль.
- Результат последней операции присвоить новой переменной.
- Ее значение увеличить на 2 с помощью функции инкремента и вывести результат в консоль в виде сообщения "Последнее значение переменной ..." (используя объединение строк)

# **Операторы сравнения**

- Больше/меньше:  $a > b$ ,  $a < b$ .
- Больше/меньше или равно:  $a \geq b$ ,  $a \leq b$ .
- Равно  $a == b$ .
- Оператор неравенства  $a != b$
- Оператор строгого сравнения  $a === b$
- При сравнении - результат возвращается в виде значение true или false

## Условные операторы

- If
- else
- If else if
- Тернарные операторы

Принципы:

- Вычисляет значение в скобках и приводит его к логическому значению
- 0, пустая строка "", null и undefined и NaN являются false в контексте логических значений, все остальное true.

## *Логические операторы*

- || (ИЛИ)
- && (И)
- ! (НЕ)

## Задание

- Напишите условие для проверки того что цифра находится в диапазоне двух цифры 10 и 100 и вывидите сообщение об этом.
- Напишите условие которые вычисляет какое из двух цифр больше и выводит его
- Напишите условие которое по значению переменной в которой записан номер месяца выводит время года
- Напишите условие в котором проверяется если цифра делится на 2 и 3 без остатка, то выводится сообщение.

# Циклы

- for Цикл
- while цикл
- do..while цикл

Директивы циклов

- break
- continue

## *Задание*

- Напишите цикл while и for для вывода всех цифр которые делятся на 2 без остатка в промежутке. Промежуток задается двумя переменными.
- Модифицировать цикл чтобы по достижению цифры 30 он прерывался
- Модифицировать цикл чтобы он выводил на экран результат умножения текущей цифры на 10 только в том случае если она равна 10 или 20 используя директиву continue.

# Конструкция *switch*

```
switch(x) {  
    case 'value1': // if (x === 'value1')  
        ...  
        [break]  
  
    case 'value2': // if (x === 'value2')  
        ...  
        [break]  
  
    default:  
        ...  
        [break]  
}
```

# Задание

Напишите конструкцию switch для определения континента по имени страны с использования проваливания и блока default. Учтите что одна из стран может быть одновременно и страной и континентом.

Переписать код с использованием switch:

```
var myName = 'Дмитрий';
if ( myName == 'Алексей' ) {
    console.log( 'Меня зовут Алексей' );
} else if ( myName == 'Александр' ) {
    console.log( 'Меня зовут Александр' );
} else {
    console.log( 'Значит все таки Дмитрий' );
}
```

# Массивы

Представляют собой структуру данных для хранения значений и других данных. Массивы могут содержать любой тип данных JavaScript, в том числе ссылки на другие массивы или на объекты или функции.

Каждое значение в массиве имеет свой номер - индекс. Индексация в массиве начинается с 0.

```
var a = new Array();  
var a = [1, "string", 3];
```

## *Задание*

Создать массив из 10 значений, с помощью способа перечисления вычислить максимальное число в массиве. Предусмотреть наличие в массиве строковых значений. Максимальное числовое значение сохранить в отдельную переменную и вывести на экран.



# ФУНКЦИИ

```
function myFirstFunction(param1, param2) {  
    var var1 = 10;  
    console.log('Моя первая функция');  
    return true;  
}
```

```
myFirstFunction()
```

Основное про функции:

- Объявление функции начинается со слова **function**, далее следует **имя функции**, круглые скобки в которых перечислены параметры через запятую и пара фигурных скобок, внутри которых размещено код тела функции.
- Функция имеет свою область видимости
- Внутри функции можно объявлять переменные
- Функция имеет свой контекст исполнения
- Параметры переданные в функцию становятся переменными внутри нее
- Функция может возвращать значения через конструкцию **return**. Функция которая не возвращает ничего - возвращает **undefined**. Функции можно передавать как параметры в другие функции.
- Функция может вызывать сама себя по имени (рекурсия).
- Инструкция **return** прерывает выполнение функции.
- Функция имеет доступ к внешним переменным



# Задание

1. Написать функцию которая возвращает максимальное значение из 2 параметров переданных ей

2. Напишите функцию калькулятор для сложения положительных чисел которая принимает 3 параметра

- первое число
- второе число
- имя операции (add, minus, multi, divide)

Функция должна проверять что каждое из 2 чисел больше 0 и числа не дробные (остаток от деления равен нулю) , а также имя операции одно из допустимых (switch) . В обратном случае должна возвращать 0 и выводить в консоль ошибку по которой можно понять что произошло.

3. Написать функцию которая вычисляет сумму элементов массива объявленного в глобальном областси видимости. Перед сложением элемент массива должен быть приведен к числовому значению и после приведения не равен NaN



# Типы функций

## Функция декларация (Function Declaration, сокращённо FD)

- Обязательно имеет имя
- Находится непосредственно в коде на глобальном уровне или внутри другой функции
- Создается на этапе входа в контекст
- Воздействует на объект переменным (Лексическое окружение)

## Функция-выражение (Function Expression)

- Всегда находится в зоне выражения
- Имя может быть optional
- Не воздействует на объект переменных
- Создается на этапе выполнения кода

## **Контекст исполнения. Область видимости. Лексическое окружение. Цепь областей видимости.**

Контекст выполнения - это абстрактное понятие, используемое спецификацией ЕСМА, для типизации и разграничения исполняемого кода.

Лексическое окружение - специальный объект который содержит все переменные и функции декларации текущего контекста исполнения

Область видимости - это область где происходит поиск вызванной переменной. Различают глобальную и локальную область видимости. Все локальные области видимости имеют доступ к переменным области видимости которая ее породила, а та в свою очередь имеет доступ к области видимости выше. Такая цепочка называется - цепью областей видимости.



## **Замыкания**

Функция которая сохраняет ссылки на переменные в своей области видимости , когда породившая их область видимости уже перестала существовать. Такие переменные называются свободными.

## *Задание*

1. Написать которая возвращает другую функцию. Возвращаемая функция должна при каждом вызове инкрементировать значение переменной обявленной в первой функции и выводить это значение на экран.
2. Объявить три переменные с вашими данными: Имя, Фамилия, Возраст. Написать функцию которая будет принимать три параметра ( ваши данные) и выводить их на экран. Функция должна проверять что переданы все три значения, иначе выводить текст с ошибкой.



# *Основы ООП*

Основные принципы ООП:

- Наследование
- Инкапсуляция
- Полиморфизм

Основные понятия:

- Конструктор
- Методы
- Свойства
- Екземпляры класса

## Набор именованных свойств и связанных с ними значений. Значение объекта хранится по ссылке

Пример объявления объекта:

```
var ourObject = new Object();
var ourObject = { field: 'value' };
```

Доступ к свойствам объекта происходит через точку или через квадратные скобки:

```
console.log(object.field);
console.log(object['field']);
```

Перебор свойств объекта происходит через конструкцию for (var i in object)

```
for (var fieldName in ourObject) {
    console.log('Имя поля: ' + fieldName);
    console.log('Значение поля: ' + ourObject[fieldName]);
}
```

## *Задание*

1. Создайте функцию которая будет принимать одним параметром объект и будет вычислять сумму всех свойств этого объекта. Не забывайте что значение свойств может быть не только цифры.
2. Создайте объект Автомобиль со свойствами аналогичными настоящему автомобилю. Также добавьте ему методы для добавления пассажиров в автомобиль и для начала движения и окончания движения. Список пассажиров должен хранится в свойстве объекта и иметь тип данных Массив. Также объект должен иметь индикатор того движется автомобиль сейчас или нет.

## Методы и свойства простых объектов

Большинство простых типов таких как строки и числа, а также массивы имеют встроенные свойства и методы.

- строки - свойство `.length`, методы `indexOf()`, `toUpperCase()` и другие
- числа - методы `.toString()`, `.toFixed(1)` и другие
- массивы - свойство `.length`, методы `pop()`, `push()` и другие

Вызов методов простых типов происходит через обрачивание их классами `String` и `Number`

```
var var1 = new String('my_string');
var var2 = "my_string";

var var3 = 10;
var var4 = new Number(10);
```

# *Задание*

Создать массив строк. Строки должны содержать себе буквы верхнего и нижнего регистра. Вывести на экран строки которые содержат в себе хотя бы одну букву "A". Перед выводом привести строку к верхнему регистру.

## *Взаимодействие с пользователем*

- alert - Вывод простого сообщения
- prompt - Запрос данных от пользователя
- confirm - Вопрос пользователю
- document.write - Вывод данных на страницу

# Домашнее задание

Написать функционал корзины интернет магазина. Требования - корзина должна быть описана как объект со своими свойствами и методами.

Свойства:

- список товаров которые уже в корзине
- количество товаров в корзине

Методы:

- добавление нового товара в корзину
- удаление товара в корзину
- подсчет суммы стоимости товаров в корзине

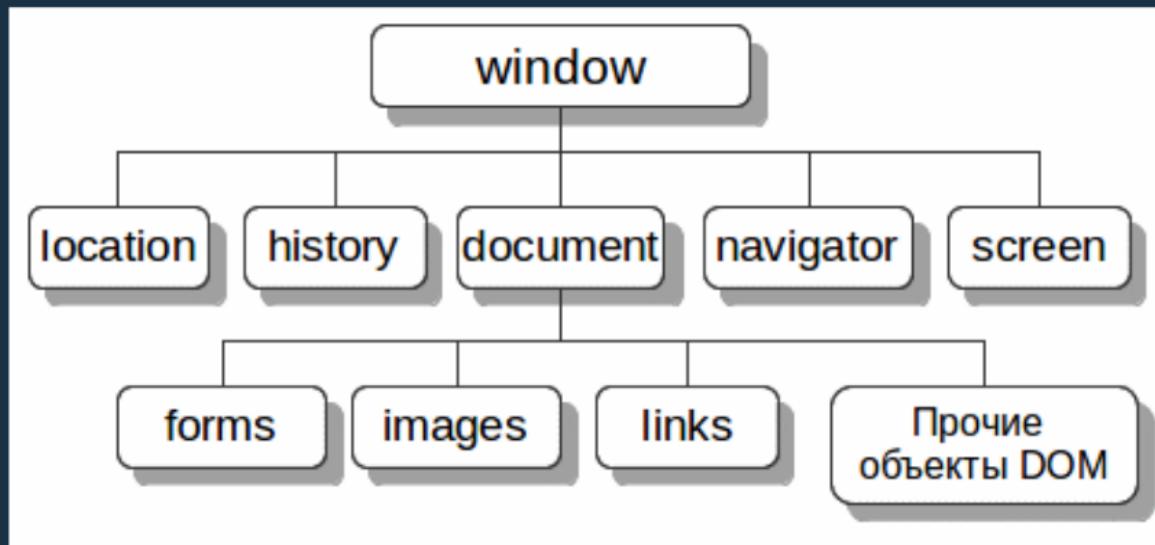
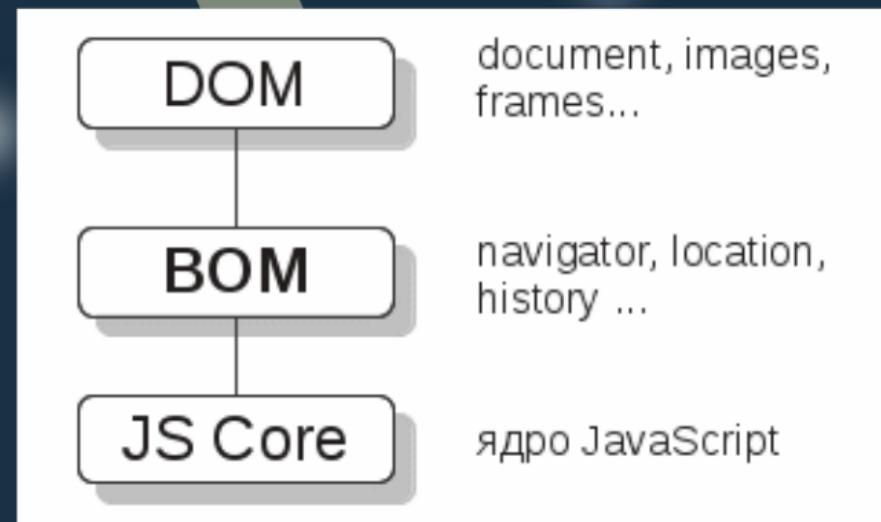
Список доступных товаров должен быть объявлен в глобальной переменной типа массив. В массиве должны хранится объекты с двумя полями ЦЕНА и ИМЯ

При добавлении товара в корзину - вы должны проверять доступен ли товар с таким именем в магазине, если нет с помощью alert выводить ошибку.

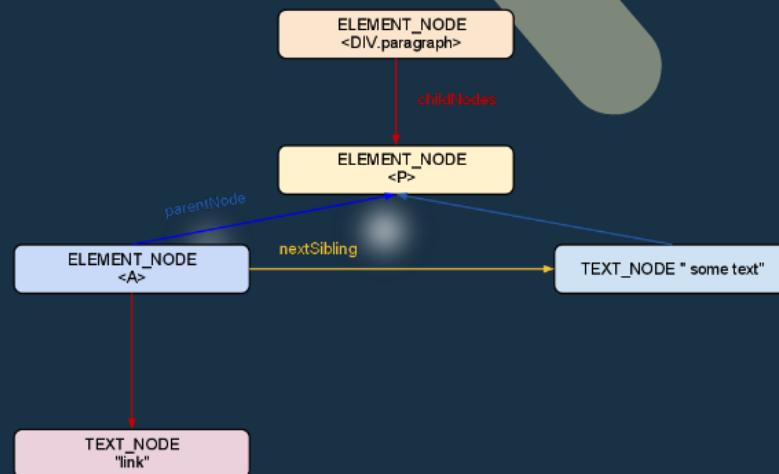
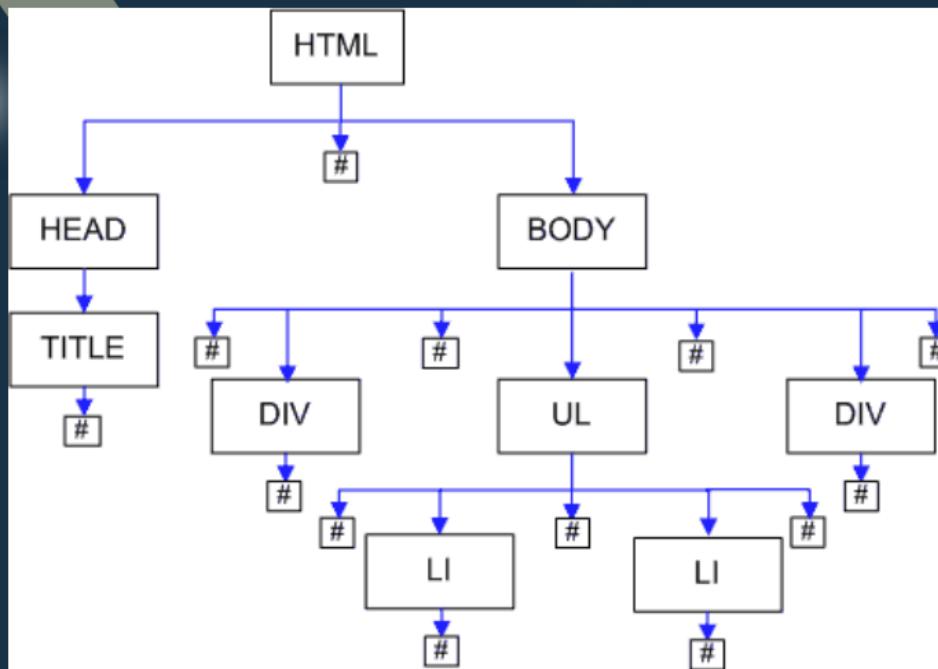
Логика работы программы: при загрузки страницы с помощью модального окна спрашивать пользователя хочет ли он добавить товар в корзину. При отрицательно ответе заканчивать выполнение программы, при

# **DOM и BOM**

Document Object Model и Browser Object Model



# DOM



# *Методы доступа к html элементам*

- `document.getElementById` - поиск элемента в DOM дереве по атрибуту `id`. Возвращает один элемент.
- `getElementsByClassName` - поиск элемента в DOM дереве по атрибуту `class`. Возвращает массив элементов.
- `getElementsByTagName` - поиск элемента в DOM дереве по имени тега. Возвращает массив элементов.

## Доступ к дочерним элементам

- свойство `childNodes` - список дочерних узлов с текстовыми узлами
- свойство `children` - список дочерних элементов без текстовых узлов

# Стили элементов

- Доступ к стилям элементов производится через свойство style. Доступны только те стили которые наложены с помощью атрибута style.
- Доступ к стилям наложенных с помощью CSS производится с помощью метода window.getComputedStyle(elem)
- Доступ к классу элементов производится через свойство className
- Работа со списком классов которые имеет элемент производится через свойство classList и его методы: contains(className), add/remove(cls), toggle (cls)

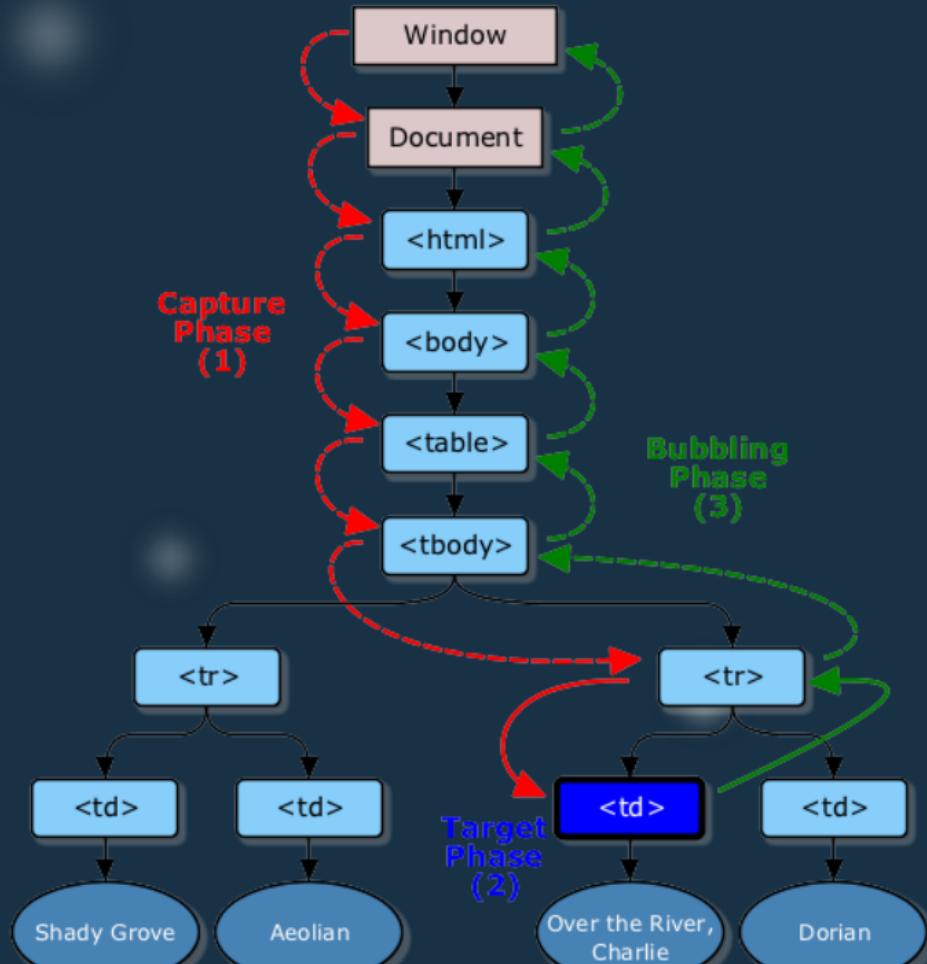
Изменение стилей элементов производится через изменение свойства style элементов.

```
elem.style.display = "block";
```

# *Вставка фрагментов*

- insertAdjacentElement - вставка элемента после текущего, перед ним или в нем
- insertAdjacentHTML - вставка чистого HTML после текущего, перед ним или в нем
- insertAdjacentText - вставка простого текста после текущего элемента, перед ним или в нем
- createDocumentFragment - Создание фрагмента вне DOM для вставки.

# DOM событий



# Работа с событиями

Способы добавить обработчик событий

- <input type="button" value="ClickHandler" onclick="simpleEventHendler();> - через атрибут HTML тега
- var button1 = document.getElementById('button1'); button1.onclick = function () {}; Через объект связанный с этим тегом, через установку свойства этого объекта.
- button2.addEventListener('click', function () {}, false); - Самый правильный и кросбраузерный способ. Позволяет присоединить несколько обработчиков в отличии от установки через свойство или атрибут.

Особенности обработчиков событий:

- this всегда ссылается на элемент на котором произошло событие
- обработчики выполняются в том же порядке в котором были добавлены
- функция обработчик всегда получает как параметр объект события с данными где и какое событие произошло.
- остановка других обработчиков осуществляется через event.stopPropagation();
- предотвращение действия по умолчанию (например для ссылки) в обработчике осуществляется методом event.preventDefault();
- Элемент на котором произошло событие хранится в свойстве target объекта события
- удаление обработчика осуществляется через метод removeEventListener и передачей ему имени функции которая использовалась как обработчик



# *События документа*

- onload - событие которое срабатывает когда загружено все ДОМ дерево и загружены картинки и остальные ресурсы
- DOMContentLoaded - событие которое срабатывает когда загружено все ДОМ дерево но еще не загружены картинки и остальные ресурсы
- onerror - событие которое срабатывает при любой JS ошибке вне блока try catch

# *setTimeout and setInterval*

- setTimeout - устанавливает интервал через который выполнить функцию или код переданные ей
- setInterval - устанавливает интервал через который выполнить функцию или код и выполнять его дальше каждый промежуток времени.

```
var intervalPointer = setTimeout("alert(1)", 1000);
```

Остановка отложенного выполнения осуществляется через clearTimeout для setTimeout и clearInterval для setInterval

```
clearInterval(intervalPointer)
```

# this

**this** - является определением контекста исполнения

- Внутри обычных функций **this** ссылается на **window**
- Внутри объектов **this** ссылается на объект в котором он используется
- Внутри конструкторов **this** ссылается на объект который будет создан в результате работы конструктора. Вызов конструктора осуществляется через вызов **new**

Изменение контекста выполнения осуществляется через методы функций **call** и **apply**. Первый параметром эти методы принимают контекст который должен применится к функции, вторым параметром аргументы с которыми эта функция должна быть вызвана.

```
someFunctionWithArguments.call(testObject, 'параметр', 'параметр');  
someFunctionWithArguments.apply(testObject, ['параметр', 'параметр']);
```





# Javascript

## введение

