Основы JavaScript

Татаринова А.Г., каф. ПМИ

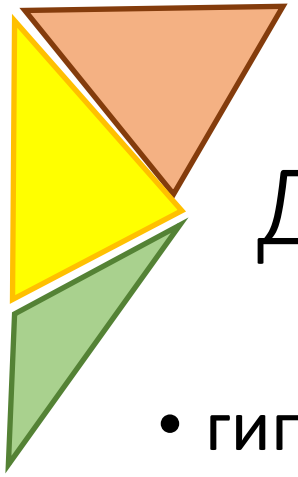
2025

Введение

- Интерактивное взаимодействие с веб-страницей добавляется посредством JavaScript («сделать веб-страницы живыми»)
- Могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы
- Код JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу - «движок» JavaScript
- У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript» Примеры движков JavaScript:
  - V8 – в Chrome, Opera и Edge
  - SpiderMonkey – в Firefox
  - «Chakra» для IE
  - «JavaScriptCore», «Nitro» и «SquirrelFish» для Safari
  - и т.д.

JavaScript

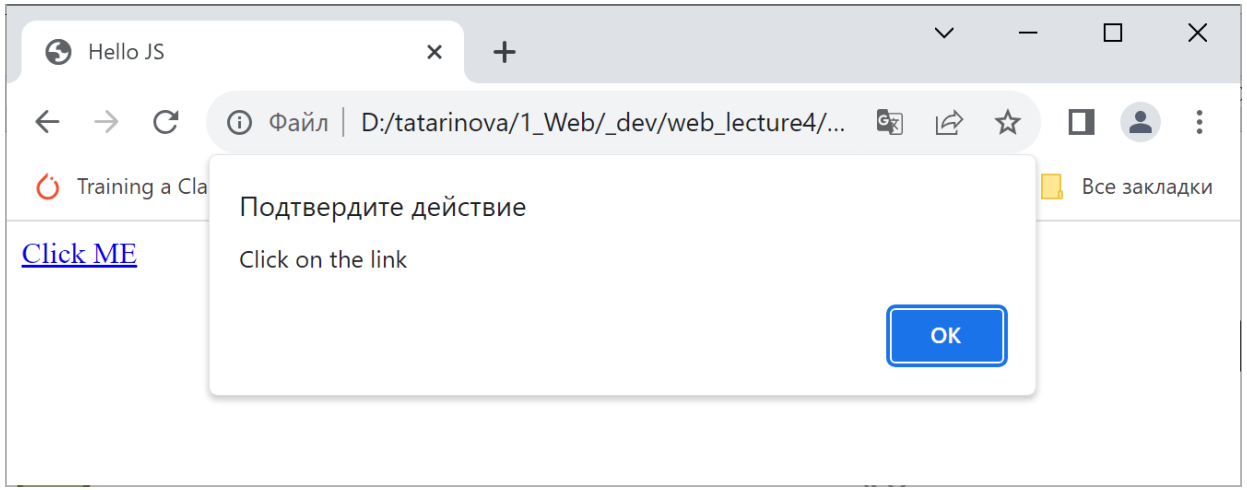
- **объектно-ориентированный** язык разработки встраиваемых приложений, выполняющихся на стороне клиента (или на стороне сервера)
- поддерживает **объектно-ориентированный**, **функциональный** стили программирования
- **чувствителен к регистру**
- браузер обеспечивает среду, в которой JavaScript имеет **доступ к объектам**, которые представляют собой окна, меню, диалоги, текстовые области и т. д. (набор объектов известен под названием Document Object Model - DOM)
- **динамическое создания содержимого страницы** во время ее загрузки или уже после того, как она полностью загружена
- браузер позволяет подключить **выполнение кода к событиям** таким как загрузка и выгрузка страниц, нажатие клавиш и движение мыши, выбор текста и пересылка форм и т.д.
- **отображение диалоговых панелей и сообщений**



Добавление JavaSript на страницу

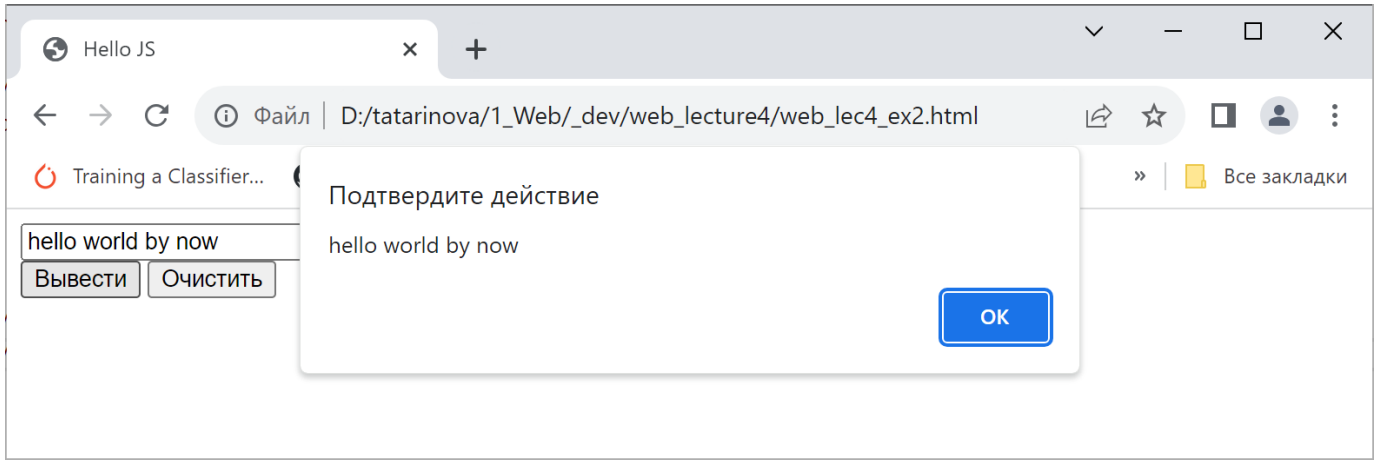
- гипертекстовая ссылка (схема URL)
- обработчик события (в атрибутах, отвечающих событиям)
- вставка через тег <SCRIPT>

Добавление JavaSript на страницу




```
<!DOCTYPE HTML>
<html>
<head>
    <title>Hello JS</title>
</head>
<body>
    <a href="JavaScript: alert('Click on the link')">
                Click ME
    </a>
</body>
</html>
```

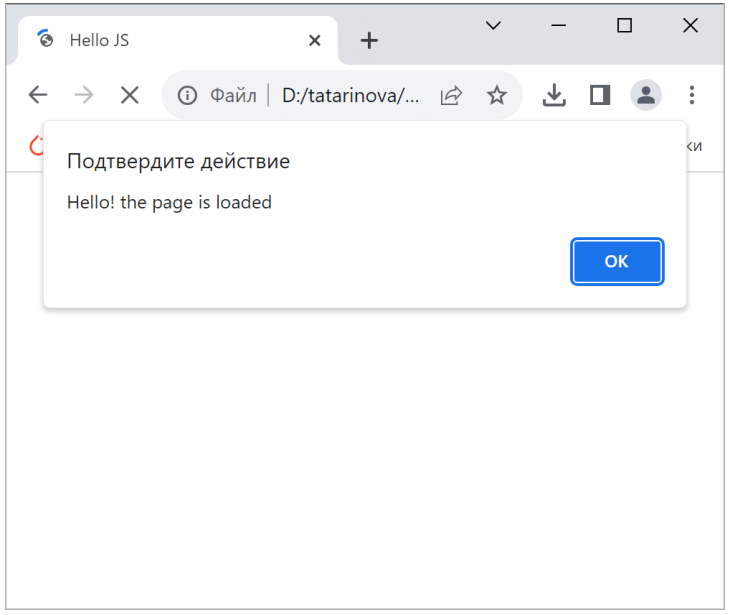
Добавление JavaSript на страницу

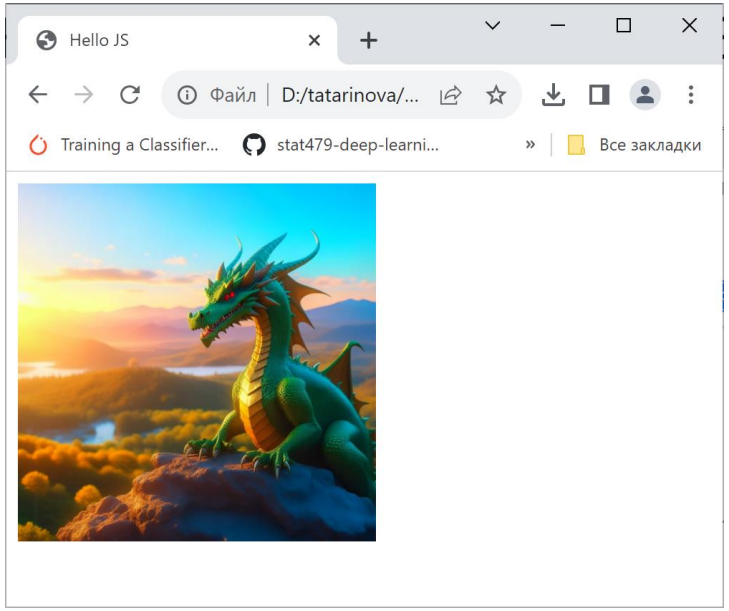


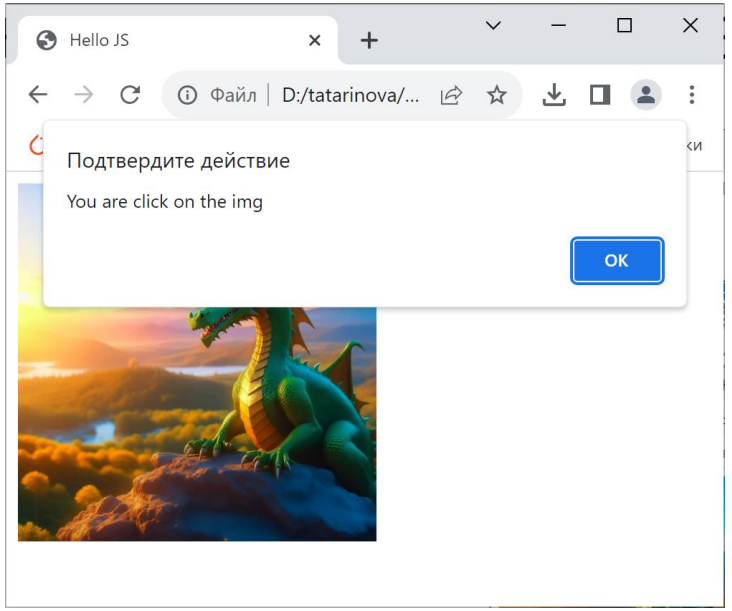

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Hello JS</title>
</head>
<body>
    <FORM METHOD=post NAME="form" ACTION="javascript:alert(form.e.value);">
        <INPUT TYPE=text NAME=e SIZE=30 VALUE="hello world by now"><BR>
        <INPUT TYPE=submit VALUE="Вывести">
        <INPUT TYPE=reset VALUE="Очистить">
    </FORM>
</body>
</html>
```

Добавление JavaSript на страницу

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Hello JS</title>
</head>
<body onLoad="alert('Hello! the page is loaded');">
    <img src="dragon.png" onclick="alert('You are click on the img')" height="250px"/>
</body>
</html>
```







Добавление JavaSript на страницу

```
<SCRIPT>alert('Hello');</SCRIPT>

<SCRIPT SRC="myscript.js"></SCRIPT>
```

- Блоки кода обрамляются фигурными скобками (функции, цикл, условия)
  Пример

```
function f()
{
  …
}
```

- Разделение инструкций через точку с запятой
  Пример:

```
let i = 5, k = 7;
let i = 5; k = 7;
```

Добавление JavaSript на страницу

```
<SCRIPT>alert(‘Hello');</SCRIPT>

<SCRIPT SRC="myscript.js"></SCRIPT>

Результат в браузере?
<SCRIPT>alert('</script>');</SCRIPT>
```

- Блоки кода обрамляются фигурными скобками (функции, цикл, условия)
  Пример

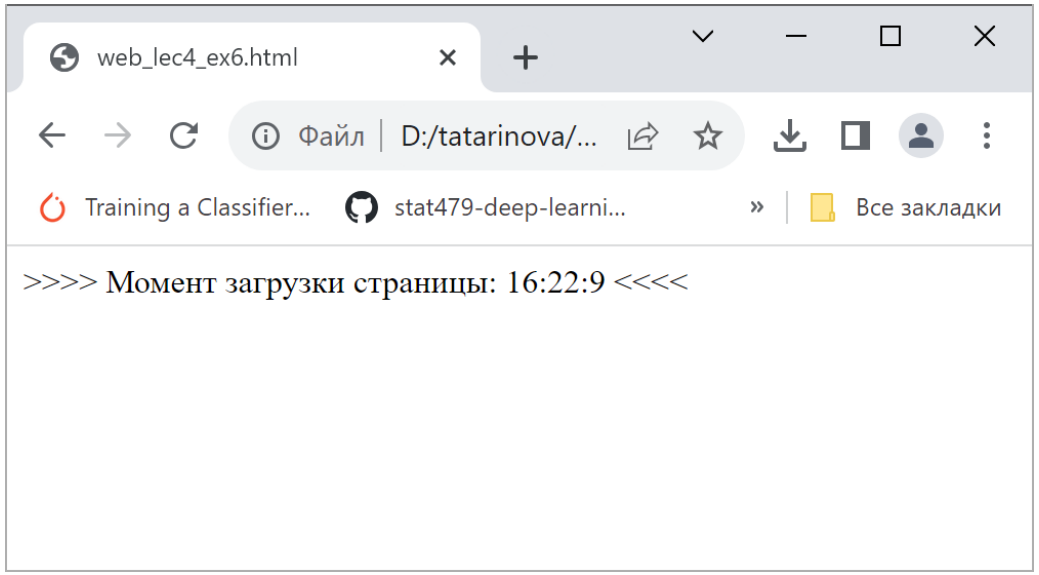
```
function f()
{
  …
}
```

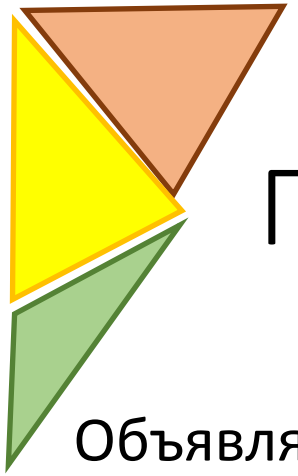
- Разделение инструкций через точку с запятой
  Пример:

```
let i = 5, k = 7;
let i = 5; k = 7;
```

Добавление JavaSript на страницу

```
<html><head></head>
<BODY>
    >>>>
    <SCRIPT>
        d = new Date();
        document.write('Момент загрузки страницы: '
            + d.getHours() + ':'
            + d.getMinutes() + ':'
            + d.getSeconds());
    </SCRIPT>
    <<<<
</BODY></html>
```





Переменные

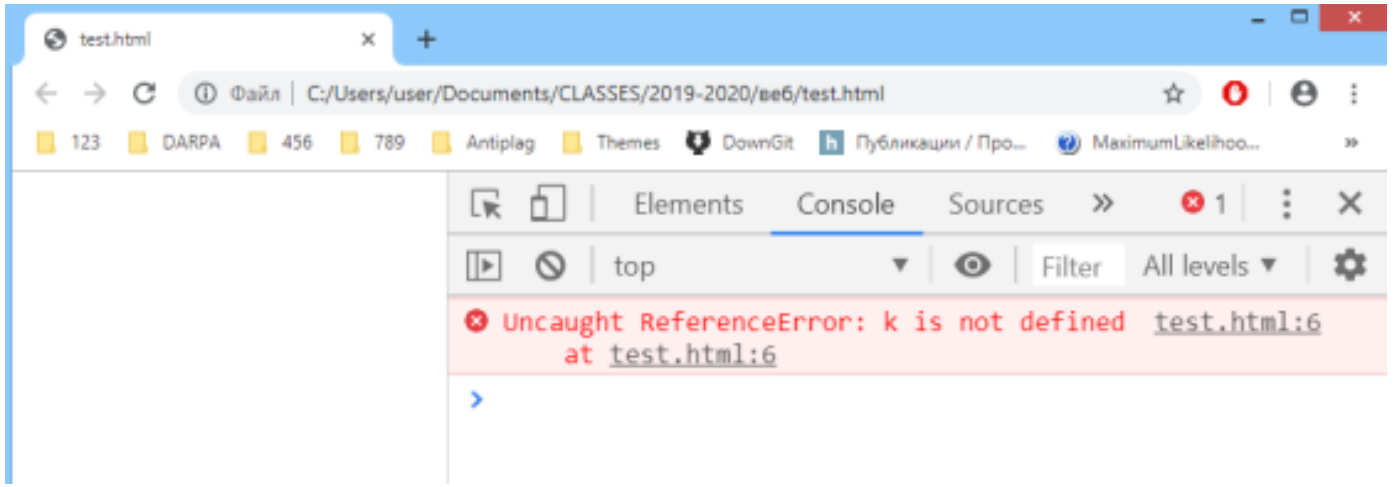
Объявлять переменные для хранения данных с помощью ключевых слов **`var`**, **`let`**, **`const`** и без служебного слова

- **`let`** – это современный способ объявления. Переменные имеют блочную область видимости, т.е. доступны только внутри блока, в котором были объявлены
- **`var`** – считается устаревшим способом объявления, но *его надо использовать по назначению*. Область видимости ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока
- без служебного слова – глобальная переменная
- **`const`** – похоже на **`let`**, но значение переменной не может изменяться после присваивания, *только модифицироваться*

Значение переменной относится к данным определённого типа

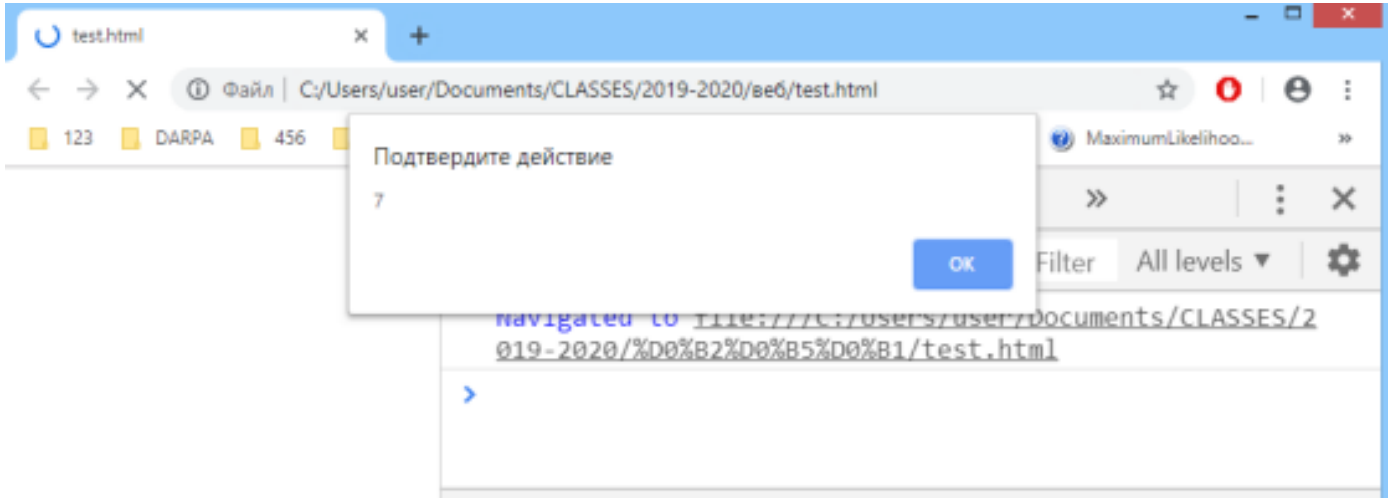
Пример 1

```
function f()
    { var k=5; }
f(); alert(k);
```



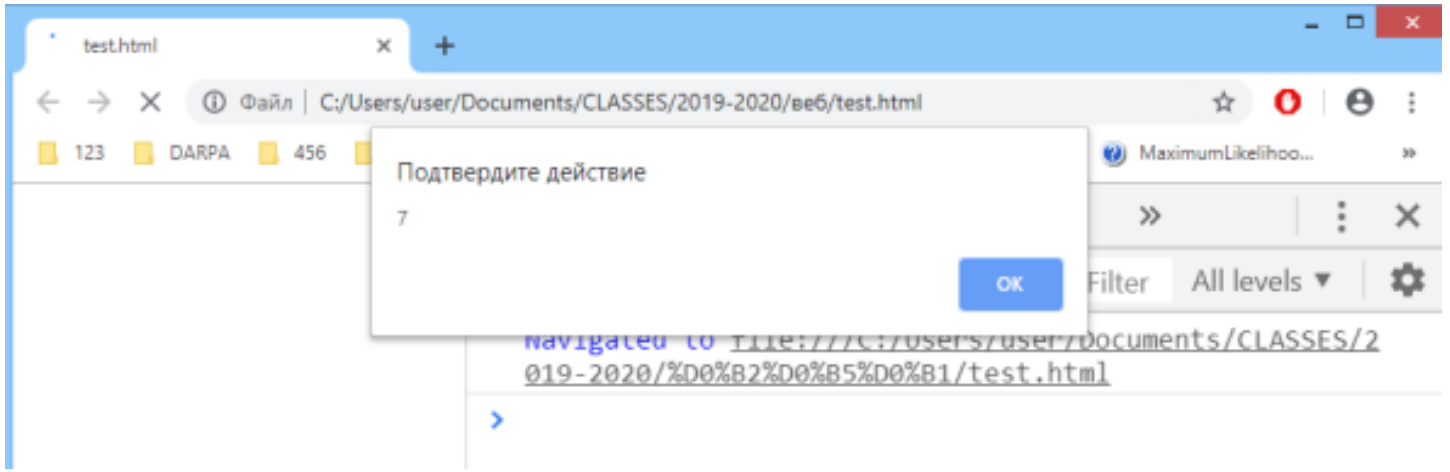
Пример 2

```
var k=7;
function f()
    { var k=5; }
f(); alert(k);
```



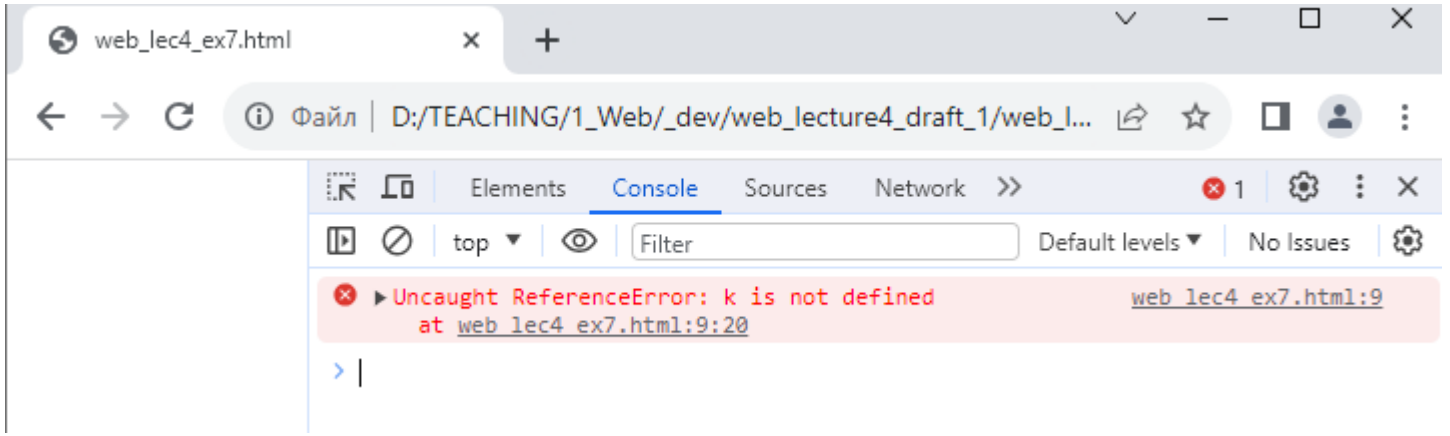
Пример 3

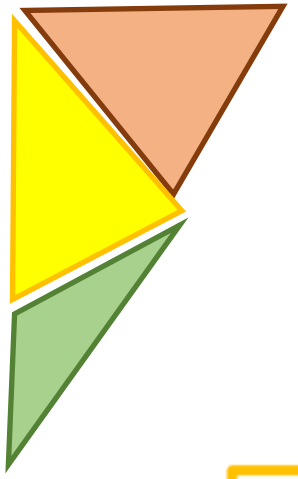
```
function f()
{ var i=5; k=7; }

f(); alert(k);
```



Пример 3(2)

```
function f()
{ var i = 5, k = 7; }

f(); alert(k)
```



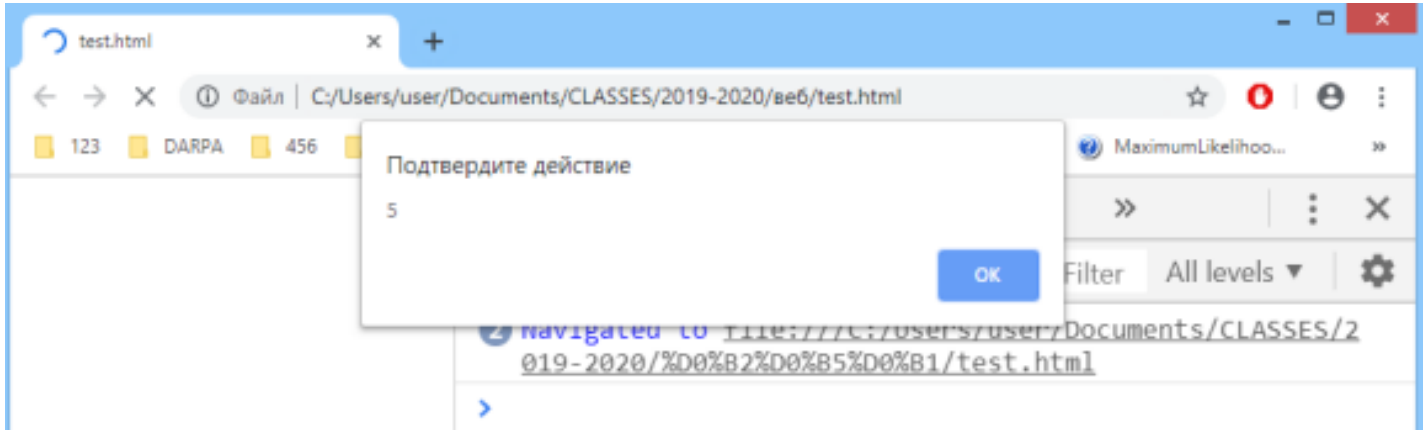


Пример 4

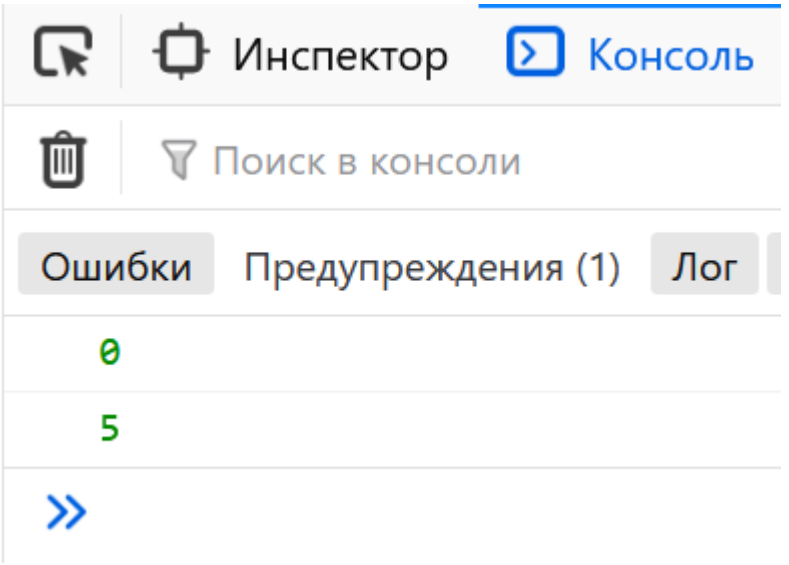
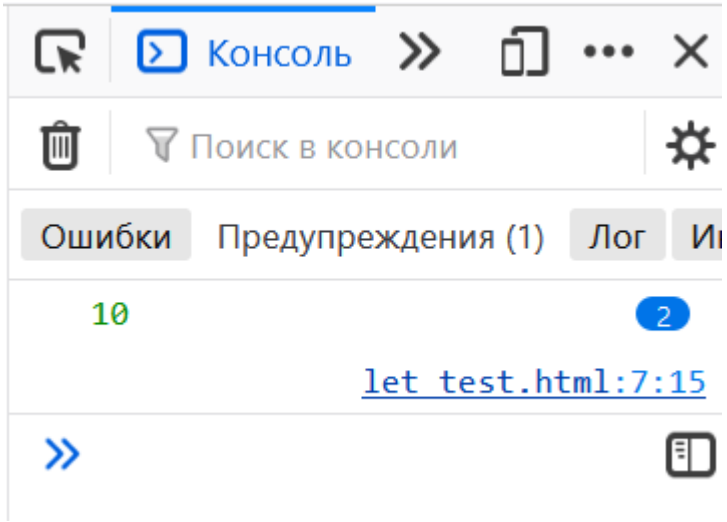
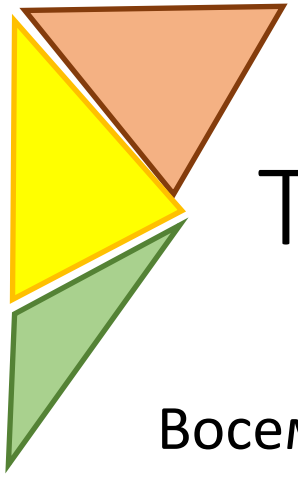
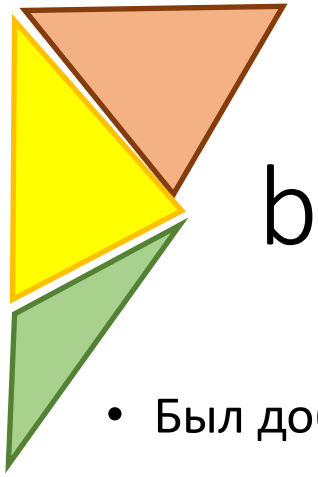
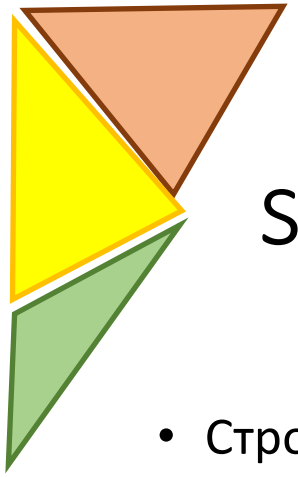
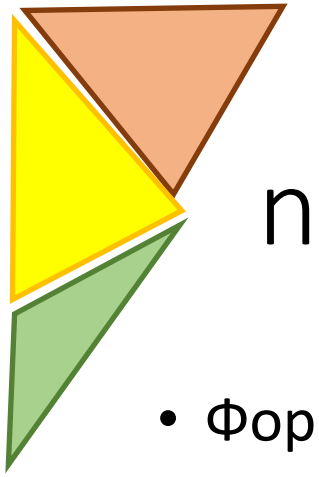
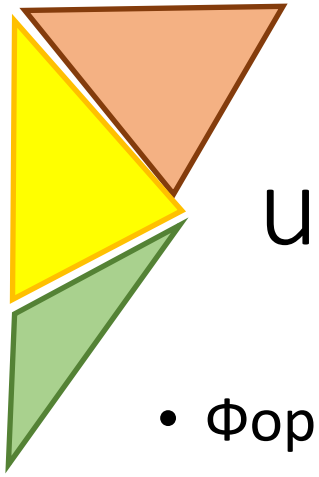
```
function f(i)
    { k=7;
      if(i==3) k=5;
      else { f(3); alert(k); }
    }
f(0);
```

Пример 4

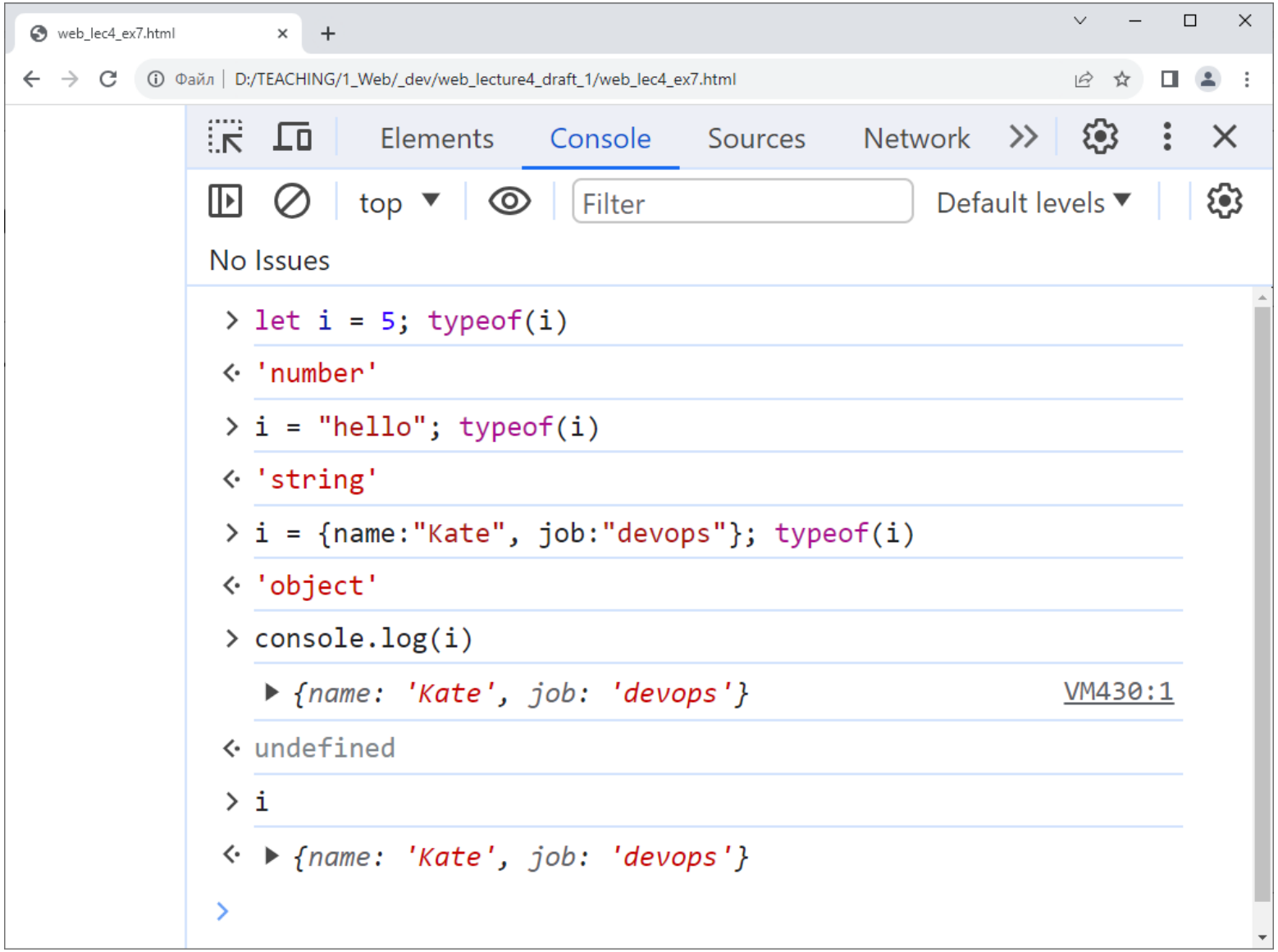
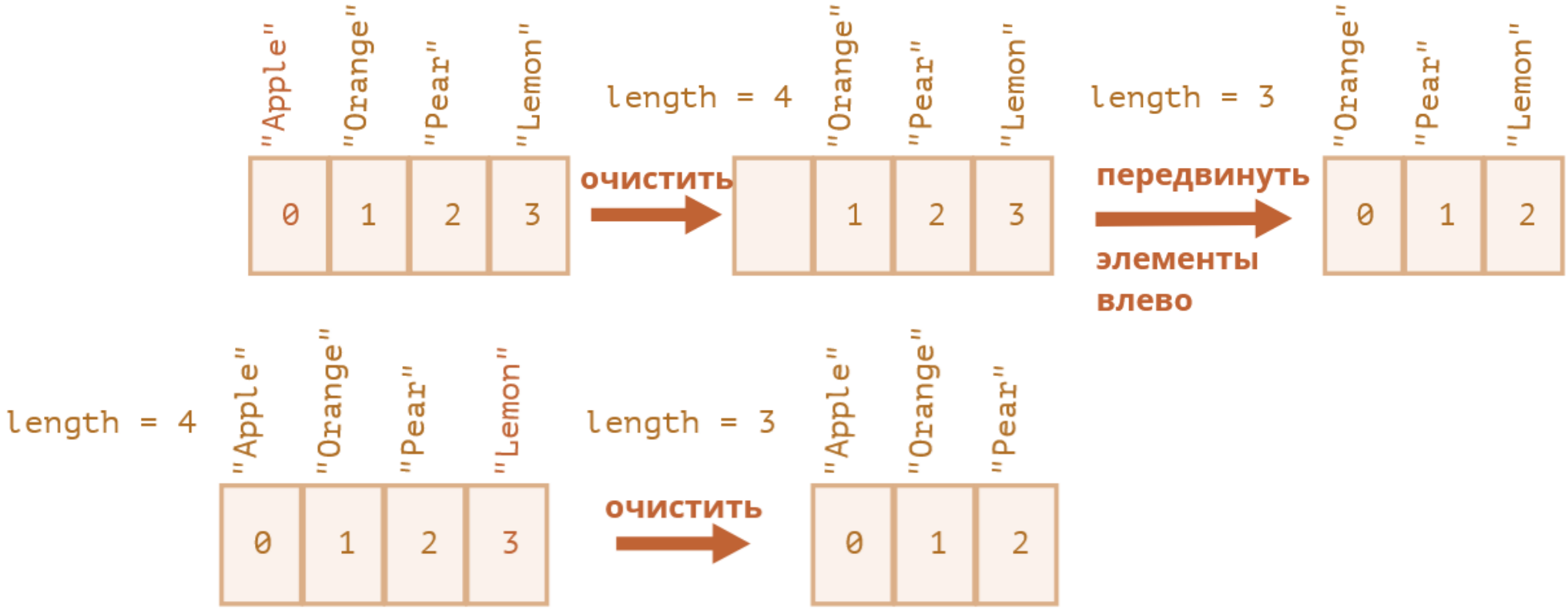
```
function f(i)
    { k=7;
      if(i==3) k=5;
      else { f(3); alert(k); }
    }
f(0);
```



let

- Область видимости переменной `let` – блок {...}
- Переменная `let` видна только после объявления
- Переменные `let` нельзя повторно объявлять
- При использовании в цикле, для каждой итерации создаётся своя переменная
- Примеры:
  - ```
    let message;
    message = 'Hello!';
    alert(message);
    ```
  - ```
    let message = 'Hello!';
    alert(message);
    ```
  - ```
    let user = 'John';
    let age = 25;
    let message = 'Hello';
    ```
  - ```
    let user = 'John', age = 25, message = 'Hello';
    ```
  - ```
    let user = 'John',
          age = 25,
          message = 'Hello';
    ```
    ```

Пример1 с let и var

```
<html><head>
<script>
function makeContainer() {
  let container = [];
  for (let i = 0; i < 10; i++) {
    container.push(function() {
      console.log(i);
    });
  }
  return container;
}
let x = makeContainer();
x[0]();
x[5]();
</script>
</head><body></body></html>
```



Пример1 с let и var

```
<html><head>
<script>
function makeContainer() {
  let container = [];
  for (var i = 0; i < 10; i++) {
    container.push(function() {
      console.log(i);
    });
  }
  return container;
}
let x = makeContainer();
x[0]();
x[5]();
</script>
</head><body></body></html>
```



Пример2 с let и var

- red
- green

```
<html><head><script>
function makeContainer() {
  let container = [];
  for (let i of ["red", "black", "green", "orange", "pink"]) {
    container.push(function() {
        style_li = `\"color: ${i}; font-size:2em\"`
        document.write("<li style="+style_li+">" + i + "</li>")
    });
  }
  return container;
}
document.write("<ul type='circle' >")
let x = makeContainer();
x[0]()
x[2]()
document.write("</ul>")
</script></head><body></body></html>
```

Пример2 с let и var

- pink
- pink

```
<html><head><script>
function makeContainer() {
  let container = [];
  for (var i of ["red", "black", "green", "orange", "pink"]) {
    container.push(function() {
        style_li = `\"color: ${i}; font-size:2em\"`
        document.write("<li style="+style_li+">" + i + "</li>")
    });
  }
  return container;
}
document.write("<ul type='circle' >")
let x = makeContainer();
x[0]()
x[2]()
document.write("</ul>")
</script></head><body></body></html>
```



Типы данных

Восемь основных типов данных в JavaScript:

- number
- bigInt
- string
- boolean
- null
- undefined
- object
- symbol

number

- Числовой тип данных (**number**) - целочисленные значения, так и числа с плавающей точкой
- Кроме обычных чисел, существуют «специальные числовые значения»: Infinity, -Infinity и NaN
- Безопасный диапазон чисел от $-(2^{53}-1)$ до $(2^{53}-1)$, вне диапазона – проблемы

Пример:

```
> console.log(9007199254740991 + 1);
> 9007199254740992
> console.log(9007199254740991 + 2);
> 9007199254740992
> console.log(9007199254740991 + 3);
> 9007199254740994
> console.log(9007199254740991 + 4);
> 9007199254740996
```



bigInt

- Был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины
- Чтобы создать значение типа **`bigInt`**, необходимо добавить n в конец числового литерала:

```
const bigInt = 1234567890123456789012345678901234567890n;
```

- Поддерживается не всеми браузерами (есть в Firefox, Chrome, Edge и Safari)



string

- Строка (**`string`**) в JavaScript должна быть заключена в кавычки
- В JavaScript существует три типа кавычек:
  - Двойные кавычки: "Привет“
  - Одинарные кавычки: 'Привет’
  - Обратные кавычки: `Привет`
- Обратные кавычки имеют расширенную функциональность
  Позволяют встраивать выражения в строку, заключая их в ${…}
  Например:

```
let name = "Иван";
alert( `Привет, ${name}!` ); // Привет, Иван!
alert( `результат: ${1 + 2}` ); // результат: 3
```

- Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда

boolean

- Булевый тип может принимать только два значения:
  true (истина) и false (ложь)
- Булевые значения также могут быть результатом сравнений:

```
let isGreater = 4 > 1;
alert( isGreater );
```



null

- Формирует отдельный тип, который содержит только значение **`null`**
- Значение **`null`** не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках
- Просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно»



undefined

- Формирует тип из самого себя (как и **`null`**)
- Означает, что «значение не было присвоено»
- Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет **`undefined`**
- Технически можно присвоить значение **`undefined`** любой переменной. Но так делать не рекомендуется. Обычно **`null`** используется для присвоения переменной «пустого» или «неизвестного» значения, а **`undefined`** – для проверок, была ли переменная назначена

object

- Хранят коллекции данных или более сложные структуры
- Коллекция, в которой каждый элемент является парой

«свойство/ключ»-«значение»

- Можно представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо
- Создание объекта:
  - `let user = new Object(); // синтаксис «конструктор объекта»`
  - `let user = {};  // синтаксис «литерал объекта»`
  - ```
    let user = {
            name: "John",
            age: 30
    };
    ```
    ```

object

- Обращение к значению свойства
  - `user.name`
  - `user["name"]` //удобно использовать когда свойство состоит из нескольких слов
- Добавление свойства
  - `user.isAdmin = true;`
- Удаление свойства
  - `delete user.age;`
- Ещё пример
  - ```
    let user = {
                name: "John",
                age: 30,
                "likes birds": true
      };
    ```
    ```

symbol

- используется для создания уникальных идентификаторов в объектах

```
<html><head><script>
const id = Symbol("id"); // Создаем символ с описанием "id"
const anotherId = Symbol("id");
// Проверка уникальности символов
console.log(id === anotherId);  // false, так как каждый символ уникален
// Использование символа как ключа в объекте
const user = {
    name: "Alice",
    age: 25,
    [id]: 123 // Используем символ как ключ
};
// Доступ к значению по символу
console.log(user[id]);  // 123
// Символы игнорируются циклом for...in
for (let key in user) {
    console.log(key); // Выведет только "name" и "age", символ не будет выведен
}
// Получение описания символа
console.log(id.description); // "id", описание доступно через свойство description
</script></head></html>
```


web_lec4_ex7.html
Файл | D:/TEACHING/1_Web/_dev/web_lecture4_draft_1/web_lec4_ex7.html
Elements
Console
Sources
Network
top
Filter
Default levels
No Issues
> let i = 5; typeof(i)
'number'
> i = "hello"; typeof(i)
'string'
> i = {name:"Kate", job:"devops"}; typeof(i)
'object'
> console.log(i)
▶ {name: 'Kate', job: 'devops'}
VM430:1
undefined
> i
▶ {name: 'Kate', job: 'devops'}

Массивы

- Упорядоченная коллекция данных
- Создание массива
  - `let arr = new Array();`
  - `let arr = new Array(10); // массив длины 10`
  - `let arr = new Array(10, 'Привет');`
  - `let arr = [];`
  - `let arr = ["Яблоко", "Апельсин", "Слива"];`
  - `let arr = [5, 'Тест', 2.71828, 'Число e'];`
- Доступ к элементу
  - ```
    let fruits = ["Apple", "Orange", "Plum"];
    alert( fruits[1] );
    ```
  - ```
    let fruits = ["Apple", "Orange", "Plum"];
    alert( fruits.at(1) );
    ```
- Доступ к последнему элементу
  - ```
    let fruits = ["Apple", "Orange", "Plum"];
    alert( fruits[fruits.length-1] );
    ```
  - ```
    let fruits = ["Apple", "Orange", "Plum"];
    alert( fruits.at(-1) );
    ```

Многомерные массивы

- Массивы могут содержать элементы, которые тоже являются массивами. Это можно использовать для создания многомерных массивов, например, для хранения матриц
  Пример:

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
alert( matrix[1][1] );
```

Массивы

- Массив - это объект
  - Добавление свойств
  - Копирование по ссылке

```
> let fruits=[];
< undefined
> fruits[9999]=5;
< 5
> fruits.age=30
< 30
> fruits
< ▶ (10000) [empty × 9999, 5, age: 30]
> arr = fruits
< ▶ (10000) [empty × 9999, 5, age: 30]
> arr[10]=100
< 100
> fruits
< ▶ (10000) [empty × 10, 100, empty × 9988, 5, age: 30]
```

Массивы

- Методы **pop**/**push**, **shift**/**unshift**
- Методы **join** и **split**
- Методы **push**/**pop** выполняются быстро, а методы **shift**/**unshift** – медленно
  Т.к. первые оперируют только концом массива, изменяя его длину, и не перемещая все элементы



Циклы

- Цикл с условием
```
while (condition) {
  // тело цикла
}
```
- Цикл с постусловием
```
do {
  // тело цикла
} while (condition);
```
- Цикл с параметром
```
for (начало; условие; шаг) {
  // тело цикла
}
```
- `break` – прерывание цикла
- `continue` – переход к следующий итерации

Перебор элементов

- ```
  let fruits = ["Яблоко", "Апельсин", "Груша"];
  for (let i = 0; i < fruits.length; i++) {
       alert(fruits[i] );
  }
  ```
- ```
  let fruits = ["Яблоко", "Апельсин", "Слива"];
  for (let fruit of fruits) {
    alert( fruit );
  }
  ```
- ```
  let fruits = ["Яблоко", "Апельсин", "Груша"];
  for (let key in fruits) {
    alert( fruits[key] );
  }
  ```
  ```

Перебор элементов

- Цикл **`for..in`** выполняет перебор всех свойств объекта
  В браузере и других программных средах также существуют так называемые «псевдомассивы» – объекты, которые выглядят, как массив. То есть, у них есть свойство **`length`** и индексы, но они также могут иметь дополнительные нечисловые свойства и методы, которые нам обычно не нужны. Тем не менее, цикл **`for..in`** выведет и их. Поэтому, если нам приходится иметь дело с объектами, похожими на массив, такие «лишние» свойства могут стать проблемой
- Цикл **`for..in`** оптимизирован под произвольные объекты, не массивы, и поэтому может быть в 10-100 раз медленнее

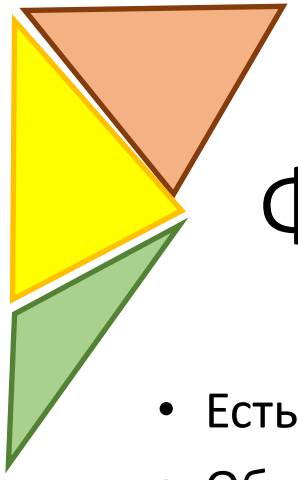
Условный оператор

- Инструкция `if(…)` вычисляет условие в скобках и, если результат `true`, то выполняет блок кода. При выполнении более одной инструкции, следует заключить блок кода в фигурные скобки
- Инструкция `if(…)` может содержать необязательный блок «`else`», который выполняется в случае, когда условие ложно
- Иногда нужно проверить несколько вариантов условия. Для этого используется блок `else if`
- Пример:

```
let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?', '');
if (year < 2015) {
  alert( 'Это слишком рано...' );
} else if (year > 2015) {
  alert( 'Это поздновато' );
} else {
  alert( 'Верно!' );
}
```

- Тернарный оператор

```
let result = условие ? значение1 : значение2;
```



Функции

- Есть встроенные функции
- Объявление функции

```
function имя(параметры) {
  // тело функции
}
```

- Функция обладает полным доступом к внешним переменным и может изменять их значение
- Можно передать внутрь функции данные, используя параметры
- Можно определять значения по-умолчанию для параметров
  - В параметрах функции
  - Оператор `||` (*возвращает правый операнд, если в левом операнде хранится любое ложноподобное значение*)
  - Оператор нулевого слияния `??` (возвращает значение правого операнда, если значение левого операнда содержит `null` или `undefined`, иначе возвращает значение левого операнда)
- Функция может вернуть результат, который будет передан в вызвавший её код

Пример таймера

```
<html>
<head>
    <script type="text/javascript">
        function startTime() {
            var tm = new Date(); var h = tm.getHours(); var m = tm.getMinutes();
            var s = tm.getSeconds();
            m = checkTime(m); s = checkTime(s);
            document.getElementById('txt').innerHTML = h + ":" + m + ":" + s;
            t = setTimeout('startTime()', 500);
        }
        function checkTime(i) {
            if (i < 10) { i = "0" + i; }
            return i;
        }
    </script>
</head>
<body onload="startTime()">
    <p id="txt"></p>
</body>
</html>
```