Лабораторная работа №3

Типы переменных и доступные с ними методы и свойства.
Операторы циклов, switch, рекурсии, стеки в JavaScript

Цель:

Комментарий: http://learn.javascript.ru/play - сервис онлайн для обучения скрипту

Объекты различных типов и доступные с ними методы и свойства

В примере ниже показаны виды переменных, которые можно объявить как объекты определенного типа. Так как по сути они будут экземплярами, то они наследуют соответствующий набор методов и свойств, к которым можно обращаться и которые можно использовать.

```
<!DOCTYPE HTML>
<html>
  <head> </head>
  <body>

    <script>
        var i = 1;
        var j;
        var person = {
            firstname: "John",
            lastname: "Doe",
            id: 5566
        };
        var carname = new String;
        var x = new Number;
        var y = new Boolean;
        var cars = new Array;
        var anything = new Object;


    </script>

  </body>
</html>
```

Задание 1: Создать приведенный пример документа в Visual Studio. После объявления переменных по очереди напишите название переменной и поставьте точку, познакомьтесь в появившейся контекстной справке со свойствами и методами доступными при работе с переменными. Обратите внимание, что при выборе метода надо потом еще ставить скобки, например: x.toString(). Если метод принимает аргументы, то они указываются в скобках, например: carname.replace("ку","ре") – заменить «ку» на «ре» в текущем значении переменной carname.

Свойства и методы для String

Свойства:

- length
- prototype
- constructor

Методы:

- charAt()

- charCodeAt()
- concat()
- fromCharCode()
- indexOf()
- lastIndexOf()
- match()
- replace()
- search()
- slice()
- split()
- substr()
- substring()
- toLowerCase()
- toUpperCase()
- valueOf()

Если нужно обратиться к конкретному символу в строке, то обращение по принципу массива? например:

```
<!DOCTYPE HTML>
<html>
 <head> </head>
 <body>
  <script>
     var carname = new String;
     carname = "жигули";
     alert(carname[3]);  //выведет символ "у" (номер индекса с нуля)
  </script>
 </body>
</html>
```

Задание 2: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_2.html и открыть (запустить) его в любом браузере.

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">Нажмите на кнопку, чтобы определить, когда слово "определить" встретится
первый раз в данной строке.</p>
<p id="p2">0</p>
<button onclick="myFunction()">Попробуй это</button>

<script>
    function myFunction() {
        var str = document.getElementById("p1").innerHTML;
        var n = str.indexOf("определить");
        document.getElementById("p2").innerHTML = n + 1;
    }
</script>

</body>
</html>
```

Задание 3: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_3.html и открыть (запустить) его в любом браузере.

Свойства и методы для Number

Свойства:

- MAX_VALUE
- MIN_VALUE
- NEGATIVE_INFINITIVE
- POSITIVE_INFINITIVE
- NaN
- prototype
- constructor

Методы:

- toExponential()
- toFixed()
- toPrecision()
- toString()
- valueOf()

```
<!DOCTYPE html>
<html>
<body>

<script>

    var x;
    document.write("<p>Only 17 digits: ");
    x = 12345678901234567890;
    document.write(x + "</p>");

    document.write("<p>0.2 + 0.1 = ");
    x = 0.2 + 0.1;
    document.write(x + "</p>");

    document.write("<p>It helps multiplying and dividing by 10: ");
    x = (0.2 * 10 + 0.1 * 10) / 10;
    document.write(x + "</p>");

</script>

</body>
</html>
```

Задание 4: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_4.html и открыть (запустить) его в любом браузере.

Boolean

В нем может храниться одно из следующих значений в зависимости от ситуации:

- 0
- -0
- null
- ""
- false
- undefined

- NaN

Math Object

Math.E
Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
Math.PI
Math.sqrt
Math.round
Math.random
Math.max
Math.min

```
<!DOCTYPE html>
<html>
<body>

<p id="demo">Нажмите на кнопку, чтобы определить, что больше: 5 или 10.</p>

<button onclick="myFunction()">Попробуй это</button>

<script>
    function myFunction() {
        document.getElementById("demo").innerHTML = Math.max(5, 10);
    }
</script>

</body>
</html>
```

Задание 5: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_5.html и открыть (запустить) его в любом браузере.

Индивидуальное задание 1:

Вариант 1: Написать скрипт, в котором будет предлагаться ввести текст размером не менее случайного сгенерированного программно числа (генерировать в диапазоне от 10 до 100) символов. Определить количество символов во введенном тексте, наличие вхождений таких стоп-слов, как «был», «имеет», «есть». Вывести общим сообщением информацию о длине текста и вхождениях указанных слов.

Вариант 2: Написать скрипт, в котором будет предлагаться ввести текст размером не менее случайного сгенерированного программно числа (генерировать в диапазоне от 10 до 100) символов. Затем ввести текст размером в три раза меньше предыдущего. Если второй введенный текст встречается в первом введенном тексте, то вывести позицию вхождения второй строки в первую. Иначе сообщить, что совпадения нет.

Вариант 3: Написать скрипт, в котором будет предлагаться ввести текст размером не менее случайного сгенерированного программно числа (генерировать в диапазоне от 0 до 50) символов. Поменять местами в строке первое и последнее слово и вывести в сообщении полученный результат.

Вариант 4: Написать скрипт, в котором будет предлагаться ввести текст размером не менее случайного сгенерированного программно числа (генерировать в диапазоне от 1 до 45) символов. Все четные символы перевести в верхний регистр, все нечетные в нижний регистр. Вывести в сообщении полученный результат.

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++;
}
```

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.

Пример бесконечного цикла:

```
while (true) {
// ...
}
```

Условие в скобках интерпретируется как логическое значение, поэтому вместо while (i!=0) обычно пишут while (i):

```
var i = 3;
while (i) { // при i=0 значение в скобках будет false и цикл остановится
  alert(i);
  i--;
}
```

```
<!DOCTYPE HTML>
<html>
  <head> </head>
  <body>

    <script>
        var i = 1;
//Обратите внимание на создание объектного типа и способ обращения к нему в скрипте
        var phone_numbers = {
            firstNumber: "1",
            twoNumber: "2",
            threeNumber: "3"
        };
        while (i < 10) {
//обратите внимание – можно явно преобразовывать число к строке
            phone_numbers.firstNumber = phone_numbers.firstNumber + "-" + i.toString();
```

```
//но также работает и неявное преобразование
            phone_numbers.twoNumber = phone_numbers.twoNumber + "-" + i;
            i++;
        }
        alert(phone_numbers.firstNumber);
        alert(phone_numbers.twoNumber);
        alert(i);
    </script>

  </body>
</html>
```

Задание 6: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_6.html и открыть (запустить) его в любом браузере.

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 ...
}
```

Например, цикл ниже выводит значения от `0` до `3` (не включая `3`):

```
var i;

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

- *Начало* `i=0` выполняется при заходе в цикл.
- *Условие* `i<3` проверяется перед каждой итерацией.
- *Шаг* `i++` выполняется после каждой итерации, но перед проверкой условия.

В цикле также можно определить переменную:

```
for (var i=0; i<3; i++) {
  //...
}
```

Любая часть `for` может быть пропущена.

Например, можно убрать `начало`:

```
var i = 0;
for (; i<3; i++)
{ //...
}
```

Можно убрать и шаг:

```
var i = 0;
for (; i<3; ) {
 // цикл превратился в аналог while (i<3)
}
```

А можно и вообще убрать все, получив бесконечный цикл:

```
for (;;) {
  // будет бесконечный цикл
}
```

Задание 7: Для каждого цикла запишите, какие значения он выведет. Потом сравните с ответом.

1. Префиксный вариант

```
var i = 0;
while (++i < 5) alert(i);
```

2. Постфиксный вариант

```
var i = 0;
while (i++ < 5) alert(i);
```

Задание 8: Для каждого цикла запишите, какие значения он выведет. Потом сравните с ответом.

1. Постфиксная форма:

```
for(var i=0; i<5; i++) alert(i);
```

2. Префиксная форма:

```
for(var i=0; i<5; ++i) alert(i);
```

Индивидуальное задание 2:

Вариант 1: Напишите цикл, который предлагает через `prompt` ввести один из вариантов предлагаемого текста (например: укажите пол: мужской/женский). Если посетитель ввел другое значение, попросить ввести еще раз, и так далее. Цикл должен спрашивать, пока посетитель не введет нужное значение, либо не нажмет кнопку Cancel (ESC).

Вариант 2: Напишите цикл, который предлагает через `prompt` ввести число (в диапазоне от 15 до 90) и некоторый текст. Если посетитель ввел другое число или текст длиной меньше 10, то попросить ввести еще раз, и так далее. Цикл должен спрашивать число и текст пока либо посетитель не введет требуемые данные, либо не нажмет кнопку Cancel (ESC).

Вариант 3: Напишите цикл, который предлагает через `prompt` ввести число. Умножить в цикле число на случайное число. Если полученное число меньше 100, попросить ввести еще раз, и так далее.

Вариант 4: Напишите цикл, который предлагает через `prompt` ввести число больше 100 и некоторый текст. Если посетитель ввел другое число, то попросить ввести еще раз, и так далее. Цикл должен спрашивать число, пока посетитель не введет требуемые данные, либо не нажмет кнопку Cancel (ESC). При выходе из цикла вывести второе слово из введенного текста или сообщить, что текст состоит из одного слова.

Директивы break и continue

Для более гибкого управления циклом используются директивы `break` и `continue`.

Выход: break

Выйти из цикла можно не только при проверке условия но и, вообще, в любой момент.
Эту возможность обеспечивает директива `break`.
Например, бесконечный цикл в примере прекратит выполнение при `i==5`:

```
var i=0;

while(1) {
 i++;

 if (i==5) break;

 alert(i);
}

alert('Последняя i = '+ i ); // 5 (*)
```

Выполнение продолжится со строки `(*)`, следующей за циклом.

Следующая итерация: continue

Директива `continue` прекращает выполнение *текущей итерации* цикла. Например, цикл ниже не выводит четные значения:

```
for (var i = 0; i < 10; i++) {

 if (i % 2 == 0) continue;

 alert(i);
}
```

Для четных `i` срабатывает `continue`, выполнение блока прекращается и управление передается на `for`.

Нельзя использовать break/continue справа от оператора ‘?’
Обычно мы можем заменить `if` на оператор вопросительный знак `'?'`.
То есть, запись:

```
if (условие) {
 a();
} else {
 b();
}
```

..Аналогична записи:

```
условие ? a() : b();
```

В обоих случаях в зависимости от условия выполняется либо `a()` либо `b()`.
Но разница состоит в том, что оператор вопросительный знак `'?'`, использованный во второй записи, возвращает значение.

Синтаксические конструкции, которые не возвращают значений, нельзя использовать в операторе `'?'`. К таким относятся большинство конструкций и, в частности, `break/continue`.

Поэтому такой код приведёт к ошибке:

```
(i > 5) ? alert(i) : continue;
```

Метки

Бывает нужно выйти одновременно из нескольких уровней цикла.
Например, надо ввести значения в точках с координатами, вот так:

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

 for (var j = 0; j < 3; j++) {

  var input = prompt('Значение в координатах '+i+','+j, '');

  if (input == null) break; // (*)

 }
}
alert('Готово!');
```

Обычный вызов `break` в строке `(*)` не может прервать два цикла сразу. Для этого существуют метки.
Метка имеет вид `"имя:"`, имя должно быть уникальным. Она ставится перед циклом, вот так:

```
outer: for (var i = 0; i < 3; i++) { ... }
```

Можно также выносить ее на отдельную строку. Вызов `break outer` прерывает управление цикла с такой меткой, вот так:

```
outer:
for (var i = 0; i < 3; i++) {

 for (var j = 0; j < 3; j++) {

  var input = prompt('Значение в координатах '+i+','+j, '');

  if (input == null) break outer; // (*)

 }
}
alert('Готово!');
```

Директива `continue` также может быть использована с меткой. Управление перепрыгнет на следующую итерацию цикла с меткой.

Метки можно ставить в том числе на блок, без цикла:

```
my: {
```

```
 for (;;) {
  for (i=0; i<10; i++) {
    if (i>4) break my;
  }
 }

 some_code; // произвольный участок кода

}
alert("После my"); // (*)
```

В примере выше, `break` перепрыгнет через `some_code`, выполнение продолжится сразу после блока `my`, со строки `(*)`. Возможность ставить метку на блоке используется редко. Обычно метки ставятся перед циклом.

Индивидуальное задание 3:

Вариант 1: Создайте скрипт, который выводит все простые числа из интервала от **2** до **100**. В результате полученные выводить числа одной строкой через запятую.

Вариант 2: Создайте скрипт, который выводит все числа из интервала от 2 до 100, которые делятся на 3 и 17 с остатком 1. В результате полученные выводить числа одной строкой через запятую.

Вариант 3: Создайте скрипт, который выводит все числа из интервала от 2 до 100, которые делятся на 5 и 11 с остатком 2. В результате полученные выводить числа одной строкой через запятую.

Вариант 4: Создайте скрипт, который выводит все числа из интервала от 2 до 100, которые делятся на 13 и 7 с остатком 3. В результате полученные выводить числа одной строкой через запятую.

Конструкция switch

Конструкция `switch` заменяет собой сразу несколько `if`.
Это — более наглядный способ сравнить выражение сразу с несколькими вариантами.

Выглядит она так:

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

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
```

```
  [break]
}
```

- Переменная `x` проверяется на строгое равенство первому значению `value1`, затем второму `value2` и так далее.
- Если соответствие установлено — switch начинает выполняться от соответствующей директивы `case` и далее, *до ближайшего* `break` (или до конца `switch`).
  При этом `case` называют *вариантами* `switch`.
- Если ни один `case` не совпал — выполняется (если есть) вариант `default`.

Пример использования `switch` (сработавший код выделен):

```
<!DOCTYPE HTML>
<html>
  <head> </head>
  <body>

    <script>
        var a = 2 + 2;

        switch (a) {
            case 3:
                alert('Маловато');
                break;
            case 4:
                alert('В точку!');
                break;
            case 5:
                alert('Перебор');
                break;
            default:
                alert('Я таких значений не знаю');
        }

    </script>

  </body>
</html>
```

Будет выведено только одно значение, соответствующее `4`. После чего `break` прервёт выполнение.
Если его не прервать — оно пойдёт далее, при этом остальные проверки игнорируются.

Например:

```
var a = 2+2;

switch (a) {
  case 3:
    alert('Маловато');
  case 4:
    alert('В точку!');
  case 5:
    alert('Перебор');
  default:
    alert('Я таких значений не знаю');
}
```

В примере выше последовательно выполнятся три `alert`.

```
alert('В точку!');
alert('Перебор');
alert('Я таких значений не знаю');
```

В **case** могут быть любые выражения, в том числе включающие в себя переменные и функции.

```
var a = 1;
var b = 0;

switch(a) {
  case b+1:
    alert(1);
    break;

  default:
    alert('нет-нет, выполнится вариант выше')
}
```

Группировка: case

Несколько значений case можно группировать.

В примере ниже `case 3` и `case 5` выполняют один и тот же код:

```
var a = 2+2;

switch (a) {
  case 4:
    alert('Верно!');
    break;

  case 3:                // (*)
  case 5:                // (**)
    alert('Неверно!');
    break;

  default:
    alert('Я таких значений не знаю');
}
```

При `case 3` выполнение идёт со строки `(3)` и идёт вниз до ближайшего `break`, таким образом проходя и то, что предназначено для `case 5`.

```
switch (browser) {
  case 'IE':
    alert('О, да у вас IE!');
    break;

  case 'Chrome':
  case 'Firefox':
  case 'Safari':
  case 'Opera':
    alert('Да, и эти браузеры мы поддерживаем');
    break;
```

```
 default:
  alert('Мы надеемся, что и в вашем браузере все ок!');
}
```

Задание 9: Напишите скрипт с if..else, соответствующий предыдущему switch.

Тип имеет значение

Следующий пример принимает значение от посетителя.

```
<!DOCTYPE HTML>
<html>
  <head> </head>
  <body>

    <script>
        var arg = prompt("Введите arg?")
        switch (arg) {
            case '0':
            case '1':
                alert('Один или ноль');

            case '2':
                alert('Два');
                break;

            case 3:
                alert('Никогда не выполнится');

            case null:
                alert('Отмена');
                break;

            default:
                alert('Неизвестное значение: ' + arg)
        }

    </script>

  </body>
</html>
```

Задание 10: Выполнить пример, проверить разные значения ввода. Что код выведет при вводе чисел 0, 1, 2, 3?

- При вводе `0` или `1` выполнится первый `alert`, далее выполнение продолжится вниз до первого `break` и выведет второй `alert('Два')`.
- При вводе `2`, `switch` перейдет к `case '2'` и выведет `Два`.
- **При вводе `3`, `switch` перейдет на `default`.** Это потому, что `prompt` возвращает строку `'3'`, а не число. Типы разные. `Switch` использует строгое равенство `===`, так что совпадения не будет.
- При отмене сработает `case null`.

```
var a = +prompt('a?', '');

if (a == 0) {
  alert(0);
}
if (a == 1) {
  alert(1);
```

```
}

if (a == 2 ||  a == 3) {
  alert('2,3');
}
```

Задание 11: Перепишите код выше с использованием одной конструкции `switch`.

Индивидуальное задание 4:

Вариант 1: Написать скрипт, который будет запрашивать у пользователя ввод даты. Далее использовать оператор switch, который будет для 1,2 и 12 месяцев выводить в сообщении «зима», для 3,4,5 месяцев выводить «весна», для 6,7,8 месяцев выводить «лето», для 9,10,11 месяцев – «осень».

Вариант 2: Написать скрипт, который будет запрашивать у пользователя ввод времени в формате HH:MI (часы:минуты). Далее использовать оператор switch, который будет для времени с 07:00 до 12:00 выводить в сообщении «утро», с 12:00 до 17:00 выводить «день», с 17:00 до 21:00 выводить «вечер», с 21:00 до 07:00– «ночь».

Вариант 3: Написать скрипт, который будет запрашивать у пользователя ввод сведений о погоде (дождливо, ветренно, солнечно, снежно и т.п.). Далее использовать оператор switch, который будет для «дождливо» выводить в сообщении «захватите зонт и оденьте калоши», для «ветренно» выводить «укутайтесь потеплее», для «солнечно» выводить «возьмите солнечные очки-берегите глаза», для «снежно» – «пора лепить снеговиков», во всех остальных вариантов выводить «одевайтесь, как хотите, но погода непредсказуема».

Вариант 4: Написать скрипт, который будет запрашивать у пользователя ввод признаков описания грибов по цвету (белый, желтый, красный, коричневый), и наличию юбки (есть, нет). Далее использовать оператор switch, который будет для белого гриба без юбки выводить в сообщении «вы нашли белый гриб», для красного гриба с юбкой выводить «а не мухомор ли это», для красного гриба без юбки выводить «может это подосиновик», для остальных выводить «с грибами надо быть внимательными».

Функции

Объявление функции

Пример объявления функции:

```
function showMessage()
{
alert('Привет всем присутствующим!');
}
```

Вначале идет ключевое слово `function`, после него *имя функции*, затем *список параметров* в скобках (в примере выше он пустой) и *тело функции* — код, который вызывается при её вызове.

Объявленная функция доступна по имени, например:

```
<!DOCTYPE HTML>
<html>
  <head>
    <script>
        function showMessage() {
            alert('Привет всем присутствующим!');
        }
    </script>

  </head>
  <body>
    <script>
        showMessage();
        showMessage();
    </script>

  </body>
</html>
```

Задание 12: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_12.html и открыть (запустить) его в любом браузере.

Этот код выведет сообщение два раза. Уже здесь видна **главная цель создания функций: избавление от дублирования кода**.
Если понадобится поменять сообщение или способ его вывода — достаточно изменить его в одном месте: в функции, которая его выводит.

Локальные переменные

Функция может содержать *локальные* переменные, объявленные через `var`. Такие переменные видны только внутри функции:

```
<!DOCTYPE HTML>
<html>
  <head>   </head>
  <body>

    <script>
        function showMessage() {
            var message = 'Привет, я - Вася!'; // локальная переменная
            alert(message);
        }
        showMessage(); // 'Привет, я - Вася!'
        alert(message); // <-- не выведет значение, т.к. переменная видна только внутри
    </script>
```

```
  </body>
</html>
```

Задание 13: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_13.html и открыть (запустить) его в любом браузере.

Блоки `if/else`, `switch`, `for`, `while`, `do..while` не влияют на область видимости переменных.

При объявлении переменной в таких блоках, она всё равно будет видна во всей функции. Например:

```
function count() {
  for (var i=0; i<3; i++) {
    var j = i * 2;
  }

  alert(i); // i=3, на этом значении цикл остановился
  alert(j); // j=4, последнее значение, на котором цикл сработал, было i=2
}
```

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.

Объявления переменных в примере выше можно передвинуть вверх, это ни на что не повлияет:

```
function count() {
  var i, j; // передвинули объявления var в начало
  for (i=0; i<3; i++) {
    j = i * 2;
  }

  alert(i); // i=3
  alert(j); // j=4
}
```

Внешние переменные

Функция может обратиться ко внешней переменной, например:

```
var userName = 'Вася';

function showMessage() {
  var message = 'Привет, я ' + userName;
  alert(message);
}

showMessage(); // Привет, я Вася
```

Доступ возможен не только на чтение, но и на запись. При этом, так как переменная внешняя, то изменения будут видны и снаружи функции:

```
var userName = 'Вася';

function showMessage() {
  userName = 'Петя'; // (1) присвоение во внешнюю переменную
  var message = 'Привет, я ' + userName;
  alert(message);
}
```

```
showMessage();

alert(userName); // Петя, значение внешней переменной изменено функцией
```

Конечно, если бы внутри функции, в строке `(1)`, была бы объявлена своя локальная переменная `var userName`, то все обращения использовали бы её, и внешняя переменная осталась бы неизменной.

Внимание: неявное объявление глобальных переменных!

В старом стандарте JavaScript существовала возможность неявного объявления переменных присвоением значения.

Например:

```
function showMessage() {
  message = 'Привет'; // без var!
}

showMessage();

alert(message); // Привет
```

В коде выше переменная `message` нигде не объявлена, а сразу присваивается. Скорее всего, программист просто забыл поставить `var`.

В современном стандарте JavaScript такое присвоение запрещено, а в старом, который работает в браузерах по умолчанию, переменная будет создана автоматически, причём в примере выше она создаётся не в функции, а на уровне всего скрипта.

Параметры

При вызове функции ей можно передать данные, которые та использует по своему усмотрению.

Например, этот код выводит два сообщения:

```
function showMessage(from, text) { // параметры from, text

  from = "** " + from + " **"; // здесь может быть сложный код оформления
  alert(from + '\n\n' + text);
}

showMessage('Маша', 'Привет!');
showMessage('Маша', 'Как дела?');
```

Параметры копируются в локальные переменные функции.

В примере ниже изменение `from` в строке `(1)` не отразится на значении *внешней* переменной `from (2)`, т.к. изменена была *копия значения*:

```
function showMessage(from, text) {
  from = '**' + from + '**';  // (1), красиво оформили from
  alert(from + '\n\n' + text);
}

var from = 'Маша', msg = 'Привет!'; // (2)

showMessage(from, msg); // значения будут скопированы в параметры
alert(from); // перезапись в строке (1) не повлияет на внешнюю переменную
```

Аргументы по умолчанию

Функцию можно вызвать с любым количеством аргументов.

Например, функцию показа сообщения `showMessage(from, text)` можно вызвать с одним аргументом:

```
showMessage("Маша");
```

Если параметр не передан при вызове — он считается равным `undefined`.

Такую ситуацию можно отловить и назначить значение «по умолчанию»:

```
<!DOCTYPE HTML>
<html>
  <head>   </head>
  <body>

    <script>
        function showMessage(from, text) {
            if (text == undefined) {
                text = 'текст не передан';
            }

            alert(from + ": " + text);
        }

        showMessage("Маша", "Привет!"); // Маша: Привет!
        showMessage("Маша"); // Маша: текст не передан

    </script>

  </body>
</html>
```

Задание 14: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_14.html и открыть (запустить) его в любом браузере.

При объявлении функции необязательные аргументы, как правило, располагают в конце списка.

Для указания значения «по умолчанию», то есть, такого, которое используется, если аргумент не указан, используется два способа:

1. Можно проверить, равен ли аргумент `undefined`, и если да — то записать в него значение по умолчанию. Этот способ продемонстрирован в примере выше.
2. Использовать оператор `||`:

```
function showMessage(from, text) {
  text = text || 'текст не передан';

  ...
}
```

Второй способ считает, что аргумент отсутствует, если передана пустая строка, `0`, или вообще любое значение, которое в булевом виде является `false`.

Если аргументов передано больше, чем надо, например `showMessage("Маша", "привет", 1, 2, 3)`, то ошибки не будет. Но так как для «лишних» аргументов не предусмотрены параметры, то доступ к ним можно будет получить только через специальный объект `arguments`

Возврат значения

Функция может возвратить результат, который будет передан в вызвавший её код.

Например, создадим функцию `calcD`, которая будет возвращать дискриминант квадратного уравнения по формуле $b^2 - 4ac$:

```
<!DOCTYPE HTML>
<html>
  <head>   </head>
  <body>

    <script>
        function calcD(a, b, c) {
            return b * b - 4 * a * c;
        }

        var test = calcD(-4, 2, 1);
        alert(test); // 20


    </script>

  </body>
</html
```

Задание 15: Измените код так, чтобы он запрашивал три значения для переменных a, b, c. А потом выводил результат.

Для возврата значения используется директива `return`.
Она может находиться в любом месте функции. Как только до нее доходит управление — функция завершается и значение передается обратно.
Вызовов `return` может быть и несколько, например:

```
<!DOCTYPE HTML>
<html>
  <head>   </head>
  <body>

    <script>
        function checkAge(age) {
            if (age > 18) {
                return true;
            } else {
                return confirm('Родители разрешили?');
            }
        }

        var age = prompt('Ваш возраст?');

        if (checkAge(age)) {
            alert('Доступ разрешен');
        } else {
            alert('В доступе отказано');
        }



    </script>

  </body>
</html>
```

Задание 16: Создать приведенный пример документа в любом редакторе. Удобнее использовать редактор Visual Studio или любой другой, поддерживающий языки разметки. Сохранить файл с названием Пример3_15.html и открыть (запустить) его в любом браузере.

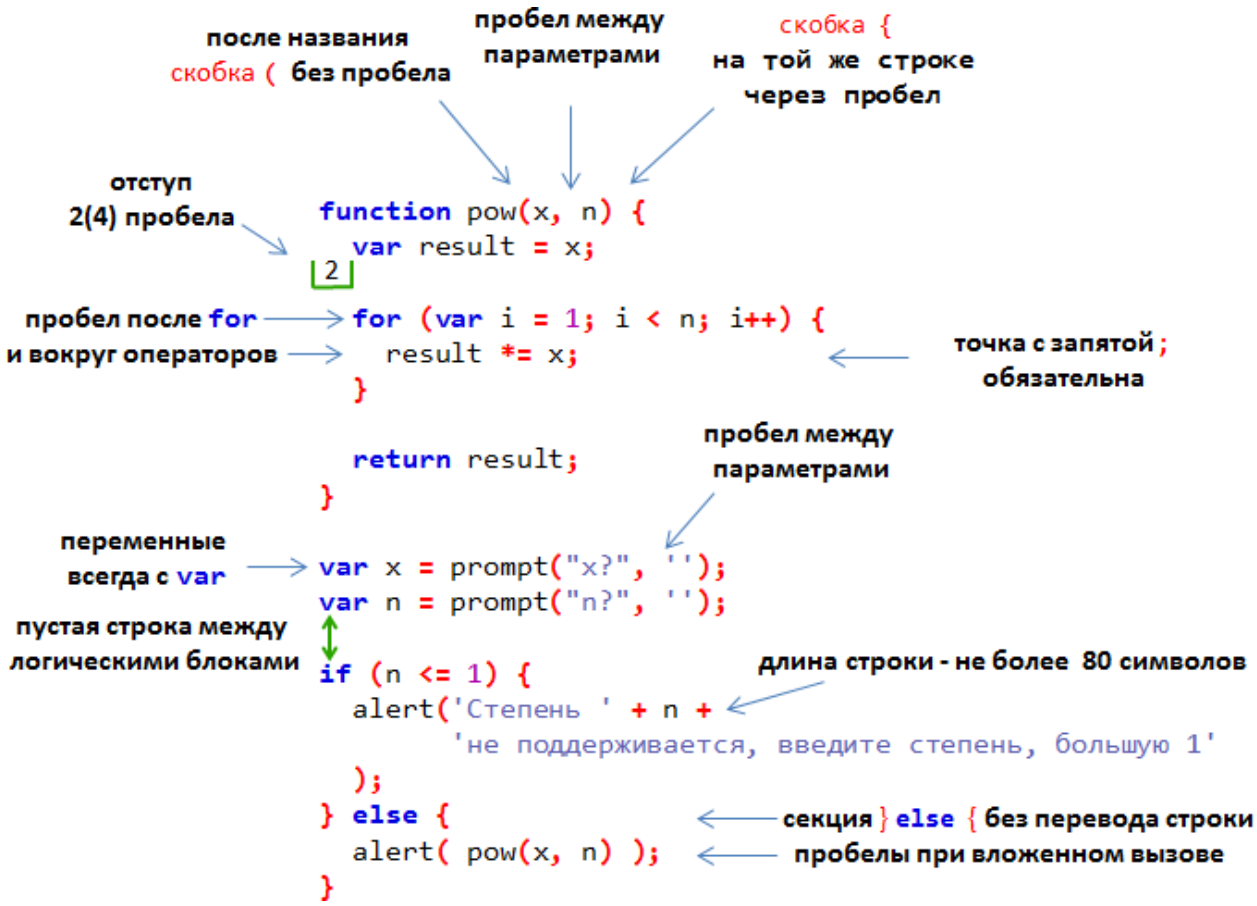
Индивидуальное задание 5:
Вариант 1: Скопируйте ваш скрипт из индивидуального задания 1 и измените его таким образом, чтобы все манипуляции с введенным текстом выполнялись в функции.

Вариант 2: Скопируйте ваш скрипт из индивидуального задания 1 и измените его таким образом, чтобы все манипуляции с введенными текстами выполнялись в функции.

Вариант 3: Скопируйте ваш скрипт из индивидуального задания 1 и измените его таким образом, чтобы все манипуляции с введенным текстом выполнялись в функции.

Вариант 4: Скопируйте ваш скрипт из индивидуального задания 1 и измените его таким образом, чтобы все манипуляции с введенным текстом выполнялись в функции.

Шпаргалка с правилами синтаксиса



Фигурные скобки

Пишутся на той же строке, так называемый «египетский» стиль. Перед скобкой — пробел.

Плохо! Скобки { } не нужны.

```
if (n <= 1) {alert('Степень ' + n + 'не поддерживается');}
```

В одну строку - приемлемо, если строка короткая

```
if (n <= 1) alert('Степень ' + n + 'не поддерживается');
```

Хорошо!

```
if (n <= 1) {
  alert('Степень ' + n + 'не поддерживается');
}
```

Автоматизированные средства проверки

Существуют онлайн-сервисы, проверяющие стиль кода, например:

- JSLint — проверяет код на соответствие стилю JSLint, в онлайн-интерфейсе вверху можно ввести код, а внизу различные настройки проверки, чтобы сделать её более мягкой.
- JSHint — ещё один вариант JSLint, ослабляющий требования в ряде мест.
- Closure Linter — проверка на соответствие Google JavaScript Style Guide.

Отчет по лабораторной работе

В соответствии со структурой заготовки отчета и примером оформления оформить в отчете все задания, выполняемые в ходе лабораторной работы, а также индивидуальные задания по вариантам. Файл с отчетом называть по шаблону: **Фамилия_лаб_раб_номер.**

Отчет предоставляется в электронном виде либо лично преподавателю, либо на электронную почту для проверки. Также по результатам лабораторной работы на следующем за ней занятии проводится выборочный опрос по командам языка.