Циклы

Как мы теперь знаем, условные конструкции позволяют запускать фрагмент кода, если условие дает true. Циклы, с другой стороны, позволяют выполнять фрагмент кода многократно — до тех пор, пока некое условие дает true. Примеры: до тех пор, пока есть ненаписанные зачётные программы писать программы, пока продолжается пара, следует держать смартфон выключенным.

Цикл while

Самый простой из циклов — цикл while. Этот цикл снова и снова выполняет код своего тела, до тех пор, пока заданное условие не перестанет давать true. Используя цикл while, мы имеем в виду следующее: «Продолжай делать это, пока условие дает true. Но если оно даст false, остановись».

Предположим, у вас проблемы со сном и вы решили посчитать овец. Но раз уж вы программист, почему бы не написать программу, которая будет считать овец за вас?

```
var sheepCounted = 0;
while(sheepCounted < 10)
{
console.log("Посчитано овец: " + sheepCounted + "!");
sheepCounted++;
}
console.log("Хрррррррррр-псссс");
```

Мы создали переменную sheepCounted и задали ей значение 0. Дойдя до цикла while, мы проверяем, правда ли, что sheepCounted меньше 10. Поскольку 0 меньше 10, выполняется код в фигурных скобках (тело цикла) и выражение "Посчитано овец: " + sheepCounted + "!" выводится в консоль как «Посчитано овец: 0!». Далее команда sheepCounted++ увеличивает значение sheepCounted на 1, мы возвращаемся к началу цикла, и все повторяется снова:

```
Посчитано овец: 0!
Посчитано овец: 1!
Посчитано овец: 2!
Посчитано овец: 3!
Посчитано овец: 4!
Посчитано овец: 5!
Посчитано овец: 6!
Посчитано овец: 7!
Посчитано овец: 8!
Посчитано овец: 9!
Хрррррррррр-псссс
```

Тело цикла повторяется, пока sheepCounted не примет значение 10, после чего условие становится ложным (false), ведь 10 не меньше 10. И тогда программа переходит к строке, идущей после цикла, — в данном случае на консоль выводится "Хрррррррррр-псссс".

Бесконечный цикл

Имея дело с циклами, помните: если условие никогда не даст false, цикл будет повторяться бесконечно (по крайней мере до тех пор, пока вы не закроете страницу в браузере). Например, не будь в теле цикла строчки sheepCounted++;, в sheepCounted всегда был бы 0 и программа печатала бы:

```
Посчитано овец: 0!
Посчитано овец: 0!
Посчитано овец: 0!
...
```

Поскольку повторения цикла ничем не ограничены, программа будет печатать эту строку снова и снова, без конца. Это называется бесконечным циклом.

Цикл for

Оператор for упрощает создание циклов, устроенных следующим образом: сначала создается переменная, а затем тело цикла выполняется снова и снова до тех пор, пока условие дает true,

причем в конце каждого повтора значение переменной обновляется. Программируя цикл for, мы создаем переменную, задаем условие, указываем, как должна меняться переменная после каждого повтора, — и лишь затем переходим к написанию тела цикла. Например, вот как можно считать овец с помощью for:

```
for (var sheepCounted = 0; sheepCounted < 10; sheepCounted++)
{
console.log("Посчитано овец: " + sheepCounted + "!");
}
```

В составе цикла for есть три выражения, разделенные точками с запятой: это настройка, проверка условия и приращение.

Настройка (var sheepCounted = 0) выполняется до запуска цикла. Как правило, здесь создают переменную для отслеживания количества повторов. В нашем случае это переменная sheepCounted с начальным значением 0.

Условие (sheepCounted < 10) проверяется перед каждым повтором тела цикла. Если условие дает true, тело выполняется, иначе цикл заканчивает работу. В нашем случае цикл остановится, когда значение sheepCounted достигнет 10.

Приращение (sheepCounted++) выполняется после каждого повтора тела цикла. Как правило, здесь изменяют значение переменной цикла. В этом примере мы после каждого повтора увеличиваем sheepCounted на 1.

Циклы for удобны, когда нужно сделать что-то определенное количество раз. Например, эта программа три раза выведет слово «Привет!».

```
var timesToSayHello = 3;
for(var i = 0; i < timesToSayHello; i++)
console.log("Привет!");
```

Вообразите, что вы интерпретатор JavaScript, который выполняет этот код. Сначала вы создадите переменную timesToSayHello, задав ей значение 3. Дойдя до цикла for, вы выполните настройку, то есть создадите переменную i и присвоите ей значение 0. Далее вы проверите условие. Поскольку в переменной i сейчас 0, а в timesToSayHello — 3, условие даст true и вы запустите тело цикла, где печатается строка "Привет!". А затем выполните приращение, то есть увеличите i на 1.

Теперь снова проверьте условие. Оно по-прежнему даст true, и вы опять перейдете к телу цикла, а затем к приращению. И так будет происходить до тех пор, пока i не примет значение 3. После этого условие даст false (3 не меньше, чем 3) — таким образом, вы завершите цикл.

Цикл for, массивы и строки

Очень часто цикл for используют для перебора всех элементов массива или всех символов строки. Например, вот цикл, который печатает названия всех животных, которые есть в зоопарке:

```
animals = ["лев", "фламинго", "белый медведь", "удав"];
for (var i = 0; i < animals.length; i++)
console.log("В этом зоопарке есть " + animals[i] + ".");
```

В этом цикле i сначала равняется 0, а затем возрастает до значения animals.length - 1, то есть 3. Числа 0, 1, 2 и 3 — индексы элементов в массиве animals. Это значит, что при каждом повторе цикла i принимает значение очередного индекса, а animals[i] соответствует очередному животному из массива animals. Когда в i число 0, animals[i] даст нам строку "лев". Когда в i число 1, animals[i] даст "фламинго" и т. д.

Запустив эту программу, мы увидим:

```
В этом зоопарке есть лев.
В этом зоопарке есть фламинго.
В этом зоопарке есть белый медведь.
В этом зоопарке есть удав.
```

Как мы уже знаем, к отдельным символам строки можно обращаться тем же способом, что и к элементам массива, — с помощью квадратных скобок. В следующем примере цикл for используется для вывода символов имени:

```
var name = "Анастасия";
```

```
for(var i = 0; i < name.length; i++)
console.log("В моём имени есть буква " + name[i] + ".");
```

Вот что выдаст эта программа:

```
В моём имени есть буква А.
В моём имени есть буква н.
В моём имени есть буква а.
В моём имени есть буква с.
В моём имени есть буква т.
В моём имени есть буква а.
В моём имени есть буква с.
В моём имени есть буква и.
В моём имени есть буква я.
```

Другие варианты применения for

Как вы, может быть, догадываетесь, не обязательно сначала задавать переменной цикла значение 0, а затем каждый раз увеличивать ее на 1. Например, вот как можно напечатать все степени двойки, не превышающие числа 10 000:

```
for(var x = 2; x < 10000; x *= 2)
console.log(x);
```

Здесь мы присваиваем x значение 2 и увеличиваем его командой x *= 2, то есть, удваиваем значение x при каждом повторе цикла. В результате x очень быстро возрастает:

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
```

Пишем игру «ВИСЕЛИЦА»

Сейчас давайте разработаем игру «Виселица» и разберемся, как с помощью диалоговых окон сделать ее интерактивной, запрашивая у игрока данные.

«Виселица» — игра на угадывание слов. Один игрок выбирает слово, а второй пытается его отгадать. Например, если первый игрок загадал слово КАПУСТА, он изобразит семь «пустых мест», по одному на каждую букву слова:

_ _ _ _ _ _ _

Второй игрок старается отгадать это слово, называя буквы. Каждый раз, когда он угадывает букву, первый игрок заполняет пустоты, вписывая ее везде, где она встречается. Например, если второй игрок назвал букву «А», первый должен вписать все «А» для слова КАПУСТА, вот так:

_ А _ _ _ _ А

Если второй игрок назовет букву, которой нет в слове, у него отнимается очко, а первый игрок рисует руку, ногу или другую часть тела человечка. Если первый игрок закончит рисовать человечка раньше, чем второй угадает все буквы, второй игрок проиграл.

В нашем варианте «Виселицы» JavaScript будет выбирать слово, а игрок-человек — отгадывать буквы. Но рисовать человечка наша программа не будет, поскольку мы пока не знаем, как это делается.

Взаимодействие с игроком

Для этой игры нам нужно, чтобы игрок (человек) мог каким-то образом вводить в программу свои ответы. Один из способов это сделать — открывать диалоговое окно (в JavaScript оно называется prompt), в котором игрок может что-нибудь напечатать.

Создаем диалоговое окно

Сначала создадим новый HTML-документ. Выбрав в меню Файл ► Сохранить как..., сохраните файл page.html из предыдущего материала под новым именем — prompt.html Чтобы создать диалоговое окно, введите следующий код между тегов <script>, а затем откройте файл prompt.html в браузере:

```
var name = prompt("Как вас зовут?");
console.log("Привет, " + name);
```

Здесь мы создали новую переменную name и присвоили ей значение, которое вернул вызов prompt ("Как вас зовут?"). При вызове prompt открывается маленькое диалоговое окно (часто его называют просто диалог).

Вызов prompt("Как вас зовут?") создает окно с запросом «Как вас зовут?» и строкой для ввода текста. В нижней части этого диалога есть две кнопки — «ОК» и «Отмена».

Если вы введете какой-нибудь текст и нажмете «ОК», этот текст станет значением, которое вернет в программу prompt. Например, если я введу имя «Адам» и нажму «ОК», JavaScript напечатает в консоли:

Привет, Адам

Поскольку я ввел «Адам» и нажал на «ОК», строка "Адам" попала в переменную name, а вызов console.log напечатал: "Привет, " + "Адам", то есть "Привет, Адам".

А что если вы нажмёте «Отмена»?

Если вы нажмете кнопку «Отмена», prompt вернет значение null. Как нам уже известно, null используется для обозначения чего-либо, что намеренно оставлено пустым.

После нажатия «Отмена» в консоли должно появиться:

Привет, null

В данном случае console.log печатает null как строку. Вообще-то null строкой не является, но, поскольку в консоль можно выводить только строки и вы попросили JavaScript напечатать "Привет, " + null, JavaScript преобразовал null в строку "null", чтобы напечатать это значение. Ситуация, когда JavaScript автоматически преобразует значение к другому типу, называется неявным приведением типа.

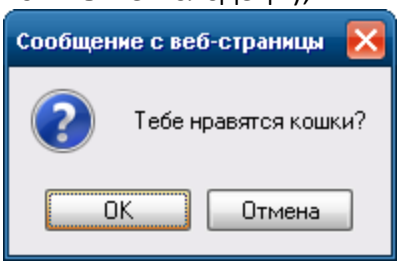
Неявное приведение типа — пример того, как JavaScript старается быть умным. Способа объединить строку и null не существует, и JavaScript делает лучшее, на что он способен. В данном случае он знает, что для успешного выполнения операции нужны две строки. Строковая версия значения null — это "null", и в результате мы видим в консоли "Привет, null".

Используем confirm, чтобы получить ответ «да» или «нет»

Функция confirm позволяет задать пользователю вопрос, на который он может ответить «да» или «нет» (что соответствует булеву значению). В следующем примере мы используем confirm, чтобы спросить у пользователя, нравятся ли ему кошки.

Если получен утвердительный ответ, переменная likesCats принимает значение true и мы печатаем: «Ты такой классный юзер!» Если же кошки пользователю не нравятся, likesCats принимает значение false, и мы отвечаем: «Что ж, не проблема. Хотя ты не молодец.»

```
var likesCat = confirm("Тебе нравятся кошки?");
if(likesCat)
console.log("Ты такой классный юзер!");
else
console.log("Что ж, не проблема. Хотя ты не молодец.");
```



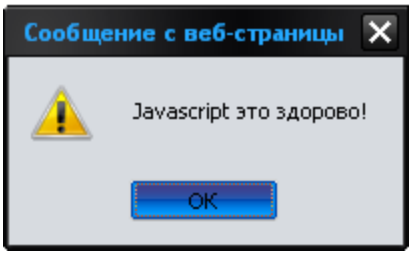
Ответ на вопрос, заданный с помощью confirm, возвращается в программу как булево значение. Если в окне пользователь нажмет «ОК», confirm вернет true. Если же пользователь нажмет «Отмена», confirm вернет false.

Используем alert для выдачи информации

Если требуется просто показать что-то пользователю, можно воспользоваться диалогом alert, который отображает сообщение с кнопкой «ОК». Например, если вы считаете, что JavaScript — это здорово, вы можете использовать alert так:

```
alert ("JavaScript стс здоров:-!" );
```

На рисунке показано, как выглядит этот диалог.



Диалог alert просто отображает сообщение до тех пор, пока пользователь не нажмет «ОК».

Чем alert лучше console.log?

Зачем нужен диалог alert, когда есть console.log? Во-первых, если необходимо просто сообщить о чем-то игроку, alert позволяет сделать именно это — не требуя, чтобы игрок открыл консоль и прочитал сообщение. Во-вторых, вызов alert (а также prompt и confirm) приостанавливает работу интерпретатора JavaScript до нажатия кнопки «ОК» (или «Отмена», в случае prompt) и — значит, у пользователя будет достаточно времени, чтобы прочитать сообщение. А при использовании console.log текст отображается в консоли, а интерпретатор тут же переходит к следующей строке программы.

Проектирование игры

Прежде чем перейти к созданию игры «Виселица», давайте подумаем о ее структуре. Нам нужно, чтобы программа умела выполнять следующие действия:

1. Случайным образом выбирать слово.
2. Запрашивать у игрока вариант ответа (букву).
3. Завершать игру по желанию игрока.
4. Проверять, является ли введенный ответ буквой.
5. Вести учет угаданных букв.
6. Показывать игроку, сколько букв он угадал и сколько еще предстоит угадать.
7 Завершать игру, если слово отгадано.

Все эти действия, кроме первого и последнего (выбор слова и завершение игры), нужно выполнять многократно, причем заранее неизвестно, сколько раз (это зависит от ответов игрока). И, как мы теперь знаем, если требуется повторять какие-то действия, значит, в программе нужен цикл.

Однако в нашем списке действий ничего не говорится о том, что и когда должно происходить. Чтобы выяснить этот вопрос и лучше представить себе структуру будущей программы, мы можем воспользоваться псевдокодом.

Используем псевдокод для проектирования игры

Псевдокод — удобный инструмент, который программисты часто используют при проектировании программ. Слово «псевдокод» означает «ненастоящий код». Хотя в псевдокоде есть циклы и условия, в целом программа описывается обычным человеческим языком. Чтобы разобраться, что это значит, давайте посмотрим на описание нашей игры в псевдокоде:

```
Выбрать случайное слово
Пока слово не угадано
{
    Показать игроку текущее состояние игры
    Запросить у игрока вариант ответа
    Если игрок хочет выйти из игры
    {
            Выйти из игры
    }
    Иначе Если вариант ответа — не одиночная буква
    {
            Сообщить игроку, что он должен ввести букву
    }
    Иначе
    {
            Если такая буква есть в слове
            {
                    Обновить состояние игры, подставив новую букву
            }
    }
}
Поздравить игрока с победой — слово угадано
```

Как видите, это не программный код, который может выполнить компьютер. Однако такая запись дает нам представление о структуре программы прежде, чем мы перейдем к написанию кода и выяснению мелких деталей, например, как именно выбирать случайное слово.

Отображение состояния игры

Одна из строк нашего псевдокода гласит: «Показать игроку текущее состояние игры». Для игры «Виселица» это означает подставить в слово угаданные игроком буквы, а также показать, какие буквы осталось угадать. Как мы будем это делать? В сущности, можно хранить состояние игры тем же способом, что и в обычной «Виселице»: в виде последовательности «пустых мест», которые мы будем заполнять по мере того, как игрок угадывает буквы.

Мы сделаем это с помощью массива «пустых мест» — по одному элементу для каждой буквы в слове. Назовем этот массив «итоговым массивом» и будем по ходу игры заполнять его угаданными буквами. А каждое из «пустых мест» представим в виде строки со знаком подчеркивания: "_".

Сначала наш итоговый массив будет просто набором «пустых мест», количество которых равно количеству букв в загаданном слове. Например, если загадано слово «рыба», массив будет выглядеть так:

["_", "_", "_", "_"]

Если игрок угадает букву «ы», мы заменим второй элемент на «ы»:

["_", "ы", "_", "_"]

А когда игрок угадает все буквы, массив примет вид:

["р", "ы", "б", "а"]

Также нам понадобится переменная для хранения количества букв, которые осталось угадать. Для каждого вхождения верно угаданной буквы эта переменная будет уменьшаться на 1, и, когда она примет значение 0, мы поймем, что игрок победил.

Проектируем игровой цикл

Основная часть игры будет располагаться внутри цикла while (в нашем псевдокоде этот цикл начинается со строки «Пока слово не угадано»), В цикле мы будем отображать текущее состояние игры (то есть слово, поначалу представленное одними знаками подчеркивания), запрашивать у игрока вариант ответа (и проверять, действительно ли тот ввел одиночную букву), а также обновлять итоговый массив, подставляя введенную букву, если она действительно есть в слове.

Практически все компьютерные игры организованы в виде того или иного цикла, нередко структурно похожего на цикл нашей «Виселицы». В целом игровой цикл выполняет следующие задачи:

1. Принимает ввод от игрока.
2. Обновляет состояние игры.
3. Показывает игроку текущее состояние игры.

Такого рода цикл применяется даже в играх, где непрерывно что-то меняется, — просто он выполняется очень быстро. В случае нашей «Виселицы» программа запрашивает у игрока вариант ответа, обновляет итоговый массив (если ответ верный) и отображает новое состояние итогового массива.

Если игрок угадает все буквы в слове, мы должны показать ему законченное слово, а также вывести сообщение, поздравляющее с победой.

Программируем игру

Теперь, когда у нас есть представление о структуре игры, можно переходить к написанию кода. Сначала мы рассмотрим его по частям. После этого вы увидите весь код целиком, чтобы с удобством ввести его и поиграть.

Выбираем случайное слово

Первым делом нам нужно выбрать случайное слово. Вот как это делается:

```
var words = [
"девушка", "селфи", "труд", "губернатор", "урожай", "молитва", "поворот",
"каталог", "пончик", "прокладка", "ассасин", "учпочмак", "хионофобия",
"кураре", "эвтаназия", "бурбон", "варвар"];
var word = words[Math.floor(Math.random() * words.length)];
```

Наша игра начинается со строки, где мы создаем массив со словами, из которого затем будем выбирать слово для отгадывания (все слова должны быть записаны строчными буквами). Сохраним этот массив в переменной words. В второй строке мы используем Math.random и Math.floor, чтобы выбрать из массива случайное слово — так же как выбирали слова для генератора дразнилок.

Создаем итоговый массив

Далее создадим пустой массив под названием answerArray (итоговый массив) и заполним его символами подчеркивания (_), количество которых соответствует количеству букв в загаданном слове.

```
var answerArray = [];
for(var i = 0; i < word.length; i++)
  answerArray[i] = "_";
var remainingLetters = word.length;
```

В начале цикла for создается переменная цикла i, которая сначала равна 0, а затем возрастает до word,length (не включая, однако, само значение word.length). При каждом повторе цикла мы добавляем в массив новый элемент — answerArray[i]. Когда цикл завершится, длина answerArray будет такой же, как длина слова. Например, если было выбрано слово «учпочмак» (в котором восемь букв), answerArray примет вид ["_", "_", "_", "_", "_", "_", "_", "_"] (восемь знаков подчеркивания).

Наконец, создадим переменную remainingLetters, приравняв ее к длине загаданного слова. Эта переменная понадобится, чтобы отслеживать количество букв, которые осталось угадать. Каждый раз, когда игрок угадает букву, мы будем декрементировать (то есть уменьшать) значение этой переменной: на 1 для каждого вхождения буквы в слово.

Программируем игровой цикл

Основа игрового цикла выглядит так:

```
while (remainingLetters >0)
{
  // Основной код
  // Показываем состояние игры
  // Запрашиваем вариант ответа
  // Обновляем answerArray и remainingLetters для каждого вхождения угаданной буквы
}
```

Мы используем цикл while, который будет повторяться до тех пор, пока условие remainingLetters > 0 дает true. В теле цикла надо будет обновлять remainingLetters для каждого правильного ответа игрока; когда игрок угадает все буквы, remainingLetters примет значение 0, и цикл завершится.

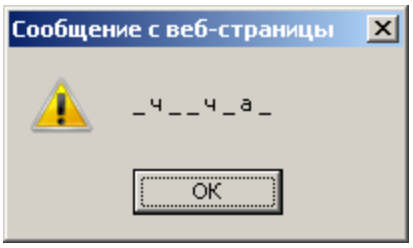
Далее мы рассмотрим код, составляющий тело игрового цикла.

Отображение состояния игры

Первым делом в теле игрового цикла нужно показать игроку текущее состояние игры:

```
alert(answerArray.join(" "));
```

Мы делаем это, объединяя элементы answerArray в строку с пробелом в качестве разделителя, а затем с помощью alert показываем эту строку. Например, пусть загадано слово «учпочмак» и игрок угадал буквы «ч» и «а». Тогда итоговый массив примет вид: ["_", "ч", "_", "_", "ч", "_", "а", "_"] и answerArray.join(" ") вернет строку "_ ч _ _ ч _ а _". Диалог alert в этом случае будет выглядеть так:



Обработка введенного ответа

Теперь нужно запросить у игрока ответ и убедиться, что он ввел одиночную букву.

```
var guess = prompt("Угадайте букву или нажмите Отмена для выхода из игры.");
if (guess === null)
{
  break;
}
else
  if(guess.length !== 1)
  {
    alert("Пожалуйста, введите только одну букву.");
```

```
}
else
{
  // Обновляем состояние игры
}
```

В первой строке prompt запрашивает у игрока ответ и сохраняет его в переменной guess. Далее возможен один из четырех вариантов развития событий.

Первый вариант — если игрок нажмет кнопку «Отмена», guess примет значение null. Этот вариант мы проверяем командой if (guess === null). Если это условие даст true, мы с помощью break выйдем из цикла.

Второй и третий варианты — игрок не ввел ничего либо ввел несколько букв. Если он просто нажал «ОК», ничего не вводя, в guess окажется пустая строка а guess.length вернет 0. Если же игрок ввел больше одной буквы, guess.length вернет число больше 1.

Мы с помощью else if (guess.length !== 1) обрабатываем эти варианты, то есть проверяем, что guess содержит в точности одну букву. В противном случае мы отображаем диалог alert, гласящий: «Пожалуйста, введите только одну букву».

Четвертый вариант — игрок, как и положено, ввел одну букву. Тогда мы должны обновить состояние игры — это происходит в секции else. Об этом пойдет речь ниже.

Обновление состояния игры

Если игрок ввел корректный ответ, мы должны обновить answerArray согласно этому ответу. Для этого добавим в тело else такой код:

```
for (var j = 0; j < word.length; j++)
  {
   if(word[j] === guess)
   {
    answerArray[j] = guess;
    remainingLetters--;
   }
  }
```

Мы задали цикл for с новой переменной j, которая будет менять значение от 0 до word.length, не включая само значение word,length. (Мы назвали переменную j, поскольку имя i уже использовано в предыдущем цикле for.) В этом цикле мы проверяем каждую букву переменной word.

В цикле мы с помощью if(word[j] === guess) проверяем, совпадает ли текущая буква (word[j]) с ответом игрока. Если это так, мы обновляем итоговый массив, добавляя туда букву командой answerArray[j] = guess. Для каждой буквы, совпадающей с ответом, мы обновляем соответствующую позицию итогового массива. Этот код работает, поскольку переменную цикла j можно использовать одновременно в качестве индекса в строке word и индекса в массиве answerArray.

Помимо обновления answerArray для каждого совпадения с guess требуется уменьшать remainingLetters на 1. Мы делаем это командой remainingLetters--;. Каждый раз, когда guess совпадает с буквой из word, remainingLetters уменьшается на 1, и, когда игрок угадает все буквы, remainingLetters примет значение 0.

Конец игры

Как мы знаем, игровой цикл while выполняется при условии remainingLetters > 0, поэтому его тело будет повторяться до тех пор, пока еще остаются неотгаданные буквы. Когда же remainingLetters уменьшится до 0, цикл завершится. После цикла нам остается лишь закончить игру — это позволяет сделать такой код:

```
alert(answerArray.join(" "));
alert("Отлично! Было загадано слово " + word);
```

В первой строке мы последний раз отображаем итоговый массив. Во второй строке, опять же с помощью alert, мы поздравляем игрока с победой.

Код игры

Итак, мы разобрали по частям весь код игры, осталось лишь соединить все вместе. Ниже он приведен целиком, от начала до конца. Я добавил в него комментарии, поясняющие, что происходит в том или ином месте программы. Обязательно вручную введите код в компьютер — это поможет вам поскорее набить руку в JavaScript. Создайте новый файл под названием hangman.html и введите в него следующее:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Виселица!</title>
</head>
<body>
<h1>Виселица!</h1>
<script>
var words = [
"девушка", "селфи", "труд", "губернатор", "урожай", "молитва", "поворот",
"каталог", "пончик", "прокладка", "ассасин", "учпочмак", "хионофобия",
"кураре", "эвтаназия", "бурбон", "варвар"];
var word = words[Math.floor(Math.random() * words.length)];
var answerArray = [];
for(var i = 0; i < word.length; i++)
  answerArray[i] = "_";
var remainingLetters = word.length;
// Игровой цикл
while(remainingLetters > 0)
{
// Показываем состояние игры
alert(answerArray.join(" "));
// Запрашиваем вариант ответа
var guess = prompt("Угадайте букву или нажмите Отмена для выхода из игры.");
if (guess === null)
{
  break; // Выходим из игрового цикла
}
else
  if(guess.length !== 1)
  {
    alert("Пожалуйста, введите только одну букву.");
  }
  else
  { // Обновляем состояние игры
    for (var j = 0; j < word.length; j++)
    {
      if(word[j] === guess)
      {
        answerArray[j] = guess;
        remainingLetters--;
      }
    }
  }
} // Конец игрового цикла
// Отображаем ответ и поздравляем игрока
alert(answerArray.join(" "));
```

```
alert("Отлично! Было загадано слово " + word);
</script>
</body>
</html>
```

Если игра не запускается, проверьте, все ли вы ввели правильно. Обнаружить ошибки вам поможет JavaScript-консоль. Например, если вы сделали опечатку в имени переменной, в консоли появится сообщение с указанием, в какой строке ошибка.

Поиграйте в свою «Виселицу» некоторое время. Работает ли программа так, как вы этого ожидаете? Можете ли вы представить, как выполняется во время игры JavaScript-код? В игре есть несколько ошибок — постарайтесь это исправить!