**Игра «Угадай число»**

Давайте представим, что ваш босс дал вам следующую информацию для создания этой игры:

Я хочу чтобы ты создал простую игру по принципу "Угадай число". Игра должна случайным образом генерировать число от 0 до 100, затем игрок должен отгадать это число за 10 попыток. После каждой попытки игроку сообщают угадал он число или не угадал и если он ошибся, то ему сообщается, что загаданное число больше или меньше того, которое он ввёл. Так же необходимо показывать игроку числа из его предыдущих попыток. Игра будет окончена, если игрок угадал число верно или если у него кончатся все попытки. После окончания игры игроку будет дана возможность сыграть в игру ещё раз.

Поглядев на это краткое изложение, первое, что мы можем сделать - это начать разбивать его на простые действия, максимально думая как программист:

1. Сгенерировать случайное число между 1 и 100.
2. Начать запись количества попыток игрока угадать число. Начать с 1.
3. Предоставить попытку угадать игроку загаданное число.
4. Как только попытка угадать была отправлена, сначала записать её где-нибудь, чтобы пользователь мог увидеть свои предыдущие попытки
5. Далее, проверить было ли это число верным.
6. Если число верное:
   1. Показать поздравительное сообщение.
   2. Оградить игрока от дальнейшей возможности ввода чисел (это испортит игру).
   3. Предоставить возможность для перезапуска игры.
7. Если число не верное и есть попытки:
   1. Сказать игроку, что он не угадал.
   2. Разрешить ему использовать ещё попытку.
   3. Повысить число попыток на 1.
8. Если число не верное и попыток нет:
   1. Сказать игроку, что игра окончена.
   2. Оградить игрока от дальнейшей возможности ввода чисел (это испортит игру).
   3. Предоставить возможность для перезапуска игры.
9. Во время перезапуска игры убедиться, что игровая логика и пользовательский интерфейс полностью сбросились на начальные значения и далее перейти обратно к пункту 1.

Давайте теперь перейдём к рассмотрению того, как мы можем превратить эти шаги в код, создавая примеры и исследуя возможности JavaScript по ходу.

**Подготовка**

Для начала выберите файл Шаблон.txt. Откройте его как в текстовом редакторе, так и в веб-браузере. На данный момент вы увидите простой заголовок, абзац с инструкцией и форму для ввода предположения, но форма в настоящее время ничего не сделает.

Место, где мы будем добавлять весь наш код, находится внутри элемента <script> в нижней части HTML:

<script>

// Your JavaScript goes here

</script>

**Добавление переменных для хранения данных**

Давайте начнём. Прежде всего добавьте следующие строки внутри элемента <script> :

var randomNumber = Math.floor(Math.random() \* 100) + 1;

var guesses = document.querySelector('.guesses');

var lastResult = document.querySelector('.lastResult');

var lowOrHi = document.querySelector('.lowOrHi');

var guessSubmit = document.querySelector('.guessSubmit');

var guessField = document.querySelector('.guessField');

var guessCount = 1;

var resetButton;

В этом разделе кода устанавливаются переменные, необходимые для хранения данных, которые будет использоваться нашей программой. Переменные - это в основном контейнеры для значений (например, числа или строки текста). Вы создаёте переменную с ключевым словом var, за которой следует имя для вашей переменной. Затем вы можете присвоить значение своей переменной знак равенства (=), за которым следует значение, которое вы хотите дать.

В нашем примере:

* Первой переменной - randomNumber - присваивается случайное число от 1 до 100, вычисленное с использованием математического алгоритма.
* Следующие три переменные сделаны для хранения ссылок на абзацы результатов в нашем HTML и используются для вставки значений в абзацы, приведённые далее в коде:

<p class="guesses"></p>

<p class="lastResult"></p>

<p class="lowOrHi"></p>

* Следующие две переменных хранят ссылки на форму ввода текста и кнопку отправки а позже используются для управления подачи догадки .

<label for="guessField">Enter a guess: </label>

<input type="text" id="guessField" class="guessField" />

<input type="submit" value="Submit guess" class="guessSubmit" />

* Наши последние две переменные сохраняют количество догадок 1 (используется для отслеживания того, сколько догадок у игрока было), и ссылку на кнопку сброса, которая ещё не существует (но позже появится).

**Примечание:** В дальнейшем вы узнаете намного больше о переменных.

**Функции (Functions)**

Затем добавьте следующие ниже предыдущего JavaScript:

function checkGuess() {

alert("I am a placeholder");

}

Функции представляют собой многократно используемые блоки кода, написав один раз вы можете запускать их снова и снова, сохраняя нужный постоянно повторяющийся код. Это действительно полезно. Существует несколько способов определить функцию, но пока мы сосредоточимся на одном простом варианте. Здесь мы определили функцию используя ключевое слово function, за ним идёт имя с двумя скобками после него. После этого мы добавляем две фигурные скобки ({ }). Внутри фигурных скобок содержится весь код, запускающийся всякий раз, когда вызываем функцию.

Код запускается вводом имени функции, за которым следуют две скобки.

Сейчас попробуйте сохранить код и обновить его в браузере.

Перейдите к консоли JavaScript в инструментах разработчика, и введите следующую строку:

checkGuess();

Вы должны увидеть предупреждение, в котором говорится "I am a placeholder"; в нашем коде мы определили функцию, которая создаёт предупреждение, когда её вызывают.

**Операторы (Operators)**

Операторы JavaScript позволяют нам проводить проверки, математические расчёты, объединять строки вместе и выполнять другие подобные действия.

Сохраните наш код и обновите страницу показанную в браузере. Откройте консоль JavaScript, если вы ещё её не открыли, чтобы попробовать ввести текст из приведённых ниже примеров — введите каждую строчку из столбца "Пример", нажимая Enter после каждого из них, и посмотрите какие результаты они возвращают. Если у вас нет доступа к инструментам разработчика в браузере, вы всегда можете использовать простую встроенную консоль, показанную ниже:

play

Сначала давайте посмотрим на арифметические операторы, например:

| **Оператор** | **Имя** | **Пример** |
| --- | --- | --- |
| + | Сложение | 6 + 9 |
| - | Вычитание | 20 - 15 |
| \* | Умножение | 3 \* 7 |
| / | Деление | 10 / 5 |

Вы также можете использовать оператор + для сложения строк текста (в программировании это называется конкатенацией). Попробуйте ввести следующие строки:

var name = "Bingo";

name;

var hello = " says hello!";

hello;

var greeting = name + hello;

greeting;

Также есть сокращённые операторы, называемые расширенными операторами присваивания. Например, если вы просто хотите добавить новую строку к существующей и вернуть результат, вы можете сделать так:

name += " says hello!";

Это эквивалентно этому:

name = name + " says hello!";

Когда мы запускаем проверку true/false (истина/ложь) (например, внутри условных выражений — смотри ниже), мы используем операторы сравнения, например:

| **Оператор** | **Имя** | **Пример** |
| --- | --- | --- |
| === | Строгое равенство (это точно одно и то же?) | 5 === 2 + 4 |
| !== | Строгое неравенство (это не одно и то же?) | 'Chris' !== 'Ch' + 'ris' |
| < | Меньше чем | 10 < 6 |
| > | Больше чем | 10 > 20 |

**Условные выражения (Conditionals)**

Вернёмся к нашей функции checkGuess(), я думаю, можно с уверенностью сказать, что мы не хотим, чтобы она просто выводила сообщение заполнитель. Мы хотим, чтобы она проверяла сделал игрок правильный выбор или нет, и соответствующие реагировала.

Теперь, заменим вашу текущую функцию checkGuess() на эту версию:

function checkGuess() {

var userGuess = Number(guessField.value);

if (guessCount === 1) {

guesses.textContent = "Previous guesses: ";

}

guesses.textContent += userGuess + " ";

if (userGuess === randomNumber) {

lastResult.textContent = "Congratulations! You got it right!";

lastResult.style.backgroundColor = "green";

lowOrHi.textContent = "";

setGameOver();

} else if (guessCount === 10) {

lastResult.textContent = "!!!GAME OVER!!!";

setGameOver();

} else {

lastResult.textContent = "Wrong!";

lastResult.style.backgroundColor = "red";

if (userGuess < randomNumber) {

lowOrHi.textContent = "Last guess was too low!";

} else if (userGuess > randomNumber) {

lowOrHi.textContent = "Last guess was too high!";

}

}

guessCount++;

guessField.value = "";

guessField.focus();

}

Как много кода — фу! Давайте отдельно рассмотрим каждый раздел и объясним, что он делает.

* Первая строка (строка под номером 2 в коде выше) объявляет переменную с именем userGuess и устанавливает её значение на то, что сейчас введено в текстовое поле. Мы также пропускаем это значение через встроенный метод Number(), чтобы убедится, что значение точно является числом.
* Затем мы сталкиваемся с нашим первым блоком условного кода (строки 3–5 в коде выше). Блок условного кода позволяет выборочно запускать код в зависимости от того, является определённое условие истинным или нет. Он немного похож на функцию, но это не так. Простейшая форма условного блока начинается с ключевого слова if, за ним круглые скобки, за ними ещё фигурные скобки. В круглые скобки мы добавляем проверку. Если проверка возвращает true, запускается код в фигурных скобках. Если нет, этот код пропускается и мы переходим к следующей части кода. В этом случае проверяется равна ли переменная guessCount числу 1 (то есть является ли это первой попыткой игрока или нет):

guessCount === 1;

Если это так, мы выводим параграф с содержанием "Previous guesses: ". Если нет, ничего не делаем.

* Строка 6 добавляет текущее значение userGuess в конец параграфа guesses, плюс пустое пространство поэтому между каждыми показанными предположениями будет пробел.
* Следующий блок (строки 8–24 ) делает несколько проверок:
  + Первая конструкция if(){ } проверяет, совпадает ли предположение пользователя с randomNumber установленному в верхней части нашего JavaScript. Если это так, игрок правильно догадался, и игра выиграна, поэтому мы показываем игроку поздравительное сообщение с приятным зелёным цветом, очищаем содержимое окна информации о минимуме / максимуме и запускаем функцию, называемую setGameOver (), которую мы обсудим позже.
  + Теперь мы добавили ещё одну проверку после предыдущей, используя конструкцию else if () {}. Эта конструкция проверяет, является ли этот ход последним ходом пользователя. Если это так, программа выполняет то же самое, что и в предыдущем блоке, но выведет сообщение с текстом GAME OVER.
  + Последний блок, в конце нашего кода (else {}), содержит код, который запускается только в том случае, если ни один из двух других тестов не возвращает true (т. е. Игрок не догадался правильно, но у него ещё остались догадки). В этом случае мы говорим игроку, что он ошибся, затем мы выполняем ещё один условный тест, чтобы проверить, было ли предположение больше или меньше ответа, показывая дополнительное сообщение.
* Последние три строки в функции (строки 26–28 ) готовят нас к следующей попытке. Мы добавляем 1 к переменной guessCount так как игрок использовал свой ход (++ оператор инкремента — увеличивает на 1), очищаем значение текстового поля и фокусируемся на нем снова, готовы для ввода следующего ответа.

**События (Events)**

На данный момент у нас есть хорошо реализованная функция checkGuess(), но она ничего не сделает, потому что мы ещё не вызвали её. В идеале мы хотим вызывать её во время нажатия кнопки « Submit guess », и для этого нам нужно использовать событие. События - это действия, которые происходят в браузере, например, нажатие кнопки или загрузка страницы или воспроизведение видео, в ответ на которые мы можем запускать блоки кода. Конструкции, которые прослушивают событие, называются **прослушивателями событий**, а блоки кода, выполняемые в ответ на срабатывание событий, называются **обработчиками событий**.

Добавьте следующую строку ниже закрывающей фигурной скобки функции checkGuess():

guessSubmit.addEventListener("click", checkGuess);

Здесь мы добавляем обработчик событий к кнопке guessSubmit. Это метод, который принимает два входных значения (называемые аргументами) - тип события, которое мы обработаем (в данном случае click) в виде строки, и код, который мы хотим запустить при возникновении события (в данном случае функция checkGuess() - обратите внимание, что нам не нужно указывать круглые скобки при записи внутри addEventListener()).

Попробуйте сохранить и обновить код сейчас, и ваш пример должен теперь работать, но до определённого момента. Единственная проблема в том, что если вы угадаете правильный ответ или исчерпаете догадки, игра сломается, потому что мы ещё не определили функцию setGameOver(), которая должна запускаться после завершения игры. Давайте добавим наш недостающий код и завершим пример функциональности.

**Завершение игры**

Давайте добавим функцию setGameOver() в конец нашего кода, а затем пройдём по ней. Добавьте это под нижней частью вашего JavaScript:

jsCopy to Clipboard

function setGameOver() {

guessField.disabled = true;

guessSubmit.disabled = true;

resetButton = document.createElement("button");

resetButton.textContent = "Start new game";

document.body.appendChild(resetButton);

resetButton.addEventListener("click", resetGame);

}

* Первые две строки отключают ввод текста и кнопку формы, устанавливая их отключённые свойства как true. Это необходимо, потому что, если бы мы этого не сделали, пользователь мог бы представить больше догадок после завершения игры, что испортит ситуацию.
* Следующие три строки генерируют новый элемент <button>, устанавливают его текстовую метку « Start new game» и добавляют её к нижней части нашего HTML.
* Последняя строка устанавливает обработчик событий на нашей новой кнопке, так что при нажатии на неё запускается функция resetGame().

Теперь нам нужно также определить эту функцию! Добавьте следующий код, снова в нижнюю часть вашего JavaScript:

function resetGame() {

guessCount = 1;

var resetParas = document.querySelectorAll(".resultParas p");

for (var i = 0; i < resetParas.length; i++) {

resetParas[i].textContent = "";

}

resetButton.parentNode.removeChild(resetButton);

guessField.disabled = false;

guessSubmit.disabled = false;

guessField.value = "";

guessField.focus();

lastResult.style.backgroundColor = "white";

randomNumber = Math.floor(Math.random() \* 100) + 1;

}

Этот довольно длинный блок кода полностью сбрасывает все на то, как это было в начале игры, поэтому у игрока может быть ещё один ход. Это:

* Устанавливает значение guessCount на 1.
* Удаляет все пункты информации.
* Удаляет кнопку сброса из нашего кода.
* Включает элементы формы, устанавливает фокус, делает поле доступным для следующих угадываний.
* Удаляет цвет фона из абзаца lastResult.
* Создаёт новое случайное число, чтобы вы не угадывали одно и тоже!

**С этого момента у вас есть полностью работающая (простая) игра - поздравляем!**

Все, что нам осталось сделать, - это поговорить о нескольких других важных функциях кода, которые вы уже видели, хотя вы, возможно, этого не осознали.

**Циклы (Loops)**

Одна часть вышеприведённого кода, которую мы должны рассмотреть более подробно, - это цикл for. Циклы - очень важная концепция программирования, которая позволяет вам снова и снова запускать кусок кода, пока не будет выполнено определённое условие.

Для начала перейдите в панель инструментов разработчика JavaScript-консоли и введите следующее:

for (var i = 1; i < 21; i++) {

console.log(i);

}

Что случилось? Номера с 1 по 20 были напечатаны в консоли. Это из-за цикла. Цикл for принимает три входных значения (аргументы): Начальное значение: в этом случае мы начинаем подсчёт c 1, но это может быть любое число которое вам нравится. Вы можете заменить i любым другим именем, которое вам нравится, но я использую его как условность, потому что оно короткое и легко запоминается. Условие выхода: Здесь мы указали i <21 - цикл будет продолжаться до тех пор, пока i будет меньше 21. Когда i достигнет 21, цикл больше не будет работать. Инкремент: мы указали i ++, что означает «увеличить i на 1». Цикл будет выполняться один раз для каждого значения i, пока оно не достигнет значения 21 (как обсуждалось выше). В этом случае мы просто печатаем значение i в консоли на каждой итерации с помощью console.log ().

Теперь давайте посмотрим на цикл в нашей игре угадывания чисел - в функции resetGame () можно найти следующее:

var resetParas = document.querySelectorAll(".resultParas p");

for (var i = 0; i < resetParas.length; i++) {

resetParas[i].textContent = "";

}

Этот код создаёт переменную, содержащую список всех абзацев внутри <div class = "resultParas">, используя метод querySelectorAll (), затем он проходит через каждый из них, удаляя текстовое содержимое каждого из них.

**Немного об объектах (Objects)**

Давайте добавим ещё одно окончательное улучшение, прежде чем перейти к обсуждению. Добавьте следующую строку чуть ниже var resetButton; в верхней части вашего JavaScript, затем сохраните файл:

guessField.focus();

Эта строка использует метод focus(), чтобы автоматически помещать текстовый курсор в текстовое поле <input>, как только загрузится страница. Пользователь сможет сразу набрать свою первую догадку, не нажимая поле формы. Это всего лишь небольшое дополнение, но оно улучшает удобство использования - даёт пользователю хорошую визуальную подсказку относительно того, что они должны делать в игре.

Давайте проанализируем, что произошло. В JavaScript все элементы являются объектами. Объект - это набор связанных функций, хранящихся в одной группе\*.\* Вы можете создавать собственные объекты, но это требует мастерства, и мы не хотели бы раскрывать эту тему в рамках данного курса. Будет достаточно обсудить встроенные объекты вашего браузера, которые позволяют реализовывать множество полезных вещей.

В нашем примере мы сначала создали переменную guessField, которая запоминает значение из поля ввода в нашем HTML - следующая строка находится среди первых в нашем коде:

var guessField = document.querySelector(".guessField");

Чтобы получить это значение, мы использовали метод querySelector() объекта document. querySelector() "берет" одну часть информации - CSS selector, который выбирает нужный элемент\*.\*

Поскольку guessField теперь содержит ссылку на элемент <input>, теперь он будет иметь доступ к ряду свойств (в основном к переменным, хранящимся внутри объектов, некоторые значения которых нельзя изменять) и методы (в основном функции, хранящиеся внутри объектов). Одним из методов, доступных для ввода элементов, является focus (), поэтому мы можем теперь использовать эту строку для фокусировки ввода текста:

guessField.focus();

Для переменных, которые не содержат ссылок на элементы формы, не будет доступен focus(). Например, переменная guesses содержит ссылку на элемент <p>, а guessCount содержит число.

**Поиграем с объектами браузера**

Давайте немного поиграем с некоторыми объектами браузера.

1. Для начала запустите свою программу в браузере.
2. Далее, откройте инструменты разработчика в вашем браузере, и убедитесь, что вы перешли во вкладку с консолью JavaScript.
3. Введите guessField и консоль покажет, что переменная содержит элемент <input>. Вы также можете заметить, что консоль автоматически заполняет имена объектов, которые существуют внутри исполняющей среды, включая ваши переменные!
4. Теперь введите следующее:

guessField.value = 2;

Свойство value представляет текущее значение, введённое в текстовое поле. Заметьте, что, введя эту команду, мы изменили его!

1. Попробуйте ввести guesses и нажать return. Консоль покажет, что в переменной содержится элемент <p>.
2. Теперь попробуйте ввести:

guesses.value;

Браузер вернёт вам undefined, потому что value не существует в параграфах.

1. Для изменения текста внутри параграфа, взамен используйте свойство textContent. Попробуйте:

guesses.textContent = "Where is my paragraph?";

1. Теперь немного повеселимся. Попробуйте ввести следующие строки, одну за другой:

guesses.style.backgroundColor = "yellow";

guesses.style.fontSize = "200%";

guesses.style.padding = "10px";

guesses.style.boxShadow = "3px 3px 6px black";

Каждый элемент на странице имеет свойство style, которое само по себе содержит объект, свойства которого содержат все встроенные стили CSS, применяемые к этому элементу. Это позволяет нам динамически задавать новые стили CSS для элементов с помощью JavaScript.