Пишем игру «Виселица»

В этой главе мы 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. И рисовать человечка наша программа не будет, поскольку мы пока не знаем, как это делается. 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 котором игрок может что-нибудь напечатать. Для начала создаем диалоговое окно, чтобы узнать кто с нами сегодня играет:

Задание

1. Объявите переменную `name`
2. Присвойте ей значение, спросив у пользователя `"Как вас зовут?"` с помощью функции `prompt()`
3. Поприветствуйте игрока по имени с помощью функции `alert()`

Используем confirm

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, хочется ли ему играть в целом. Если получен утвердительный ответ, переменная `doYouWannaPlayAGame` принимает значение `true` и мы печатаем: «`Тогда начнем!`» Если же играть пользователю не хочется, `doYouWannaPlayAGame` принимает значение `false`, и мы отвечаем: «`Приходи когда захочешь сыграть!`»

Задание

1. Объявите переменную `doYouWannaPlayAGame` и присвойте ей значение функции `confirm` с вопросом `"Сыграем в игру?"`
2. Добавьте проверку хочет ли пользователь стать игроком
3. Если пользователь согласен - выведите `alert` с фразой `"Тогда начнем!"`
4. Если пользователь не согласен - выведите `alert` с фразой `"Приходи когда захочешь сыграть!"`

Выбираем слово

Знакомство состоялось. Теперь нам необходимо начать саму игру. Если говорить кратко - структура игры должна быть следующей:

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.
Для начала создадим массив и выберем случайное слово из него, а потом выберем его элемент для игры.

Задание

1. Объявим массив угадываемых слов `words`;
2. Запишем в него 4 слова, не короче 5 букв каждое;
3. Вызови в консоль первый элемент массива методом `words[0]`
4. Вызови второй элемент
5. Вызови последний элемент

Выбираем случайное слово

Генерацией случайных чисел в JS занимается метод `Math.random()`, который возвращает случайное число от 0 до 1

```
Math.random(); //0.8945409457664937
Math.random(); //0.3697543195448816
Math.random(); //0.48314980138093233
```

Чтобы ограничить рандом до необходимого значения, используем умножение на верхний предел

```
Math.random() * 4; //0.6148658690895994
Math.random() * 4; //3.7103065797684804
Math.random() * 4; //2.9979994487671915
```

Такой формат нам не подходит, поскольку нам нужно выбрать элемент массива целым числом. Для округления используем функцию `Math.floor`

Задание

1. Удали строки с вызовами элементов
2. Объяви переменную `word`, в которую будет попадать случайным образом выбранное слово из массива `words`
3. Выполни округление до целого числа

Создаем итоговый массив и первый цикл

Чтобы хранить значения, введенные пользователем нужно выделить место под это. Далее создадим пустой массив под названием `answerArray` (итоговый массив) и заполним его символами подчеркивания (_), количество которых соответствует количеству букв в загаданном слове. Далее создадим цикл, который будет создавать строку из подчеркиваний, по количеству букв угадываемого слова. Наконец создадим переменную, которая 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

Задание

1. Объявим пустой массив `answerArray`
2. Создай цикл, который будет добавлять символы подчеркивания, по количеству букв загаданного слова
3. Объяви переменную `remainingLetters`, приравняв ее к длине загаданного слова.

Программируем игровой цикл

На данный момент мы знаем как зовут игрока, знаем какое слово он выбрал, но он к сожалению не знает об этом слове ничего. Согласно правил он должен знать количество букв, из которых состоит слово-загадка. Пора нам соблюсти правила, и показать их игроку.
Но для начала создадим цикл, которой не прервется до момента угадыванием игроком всех букв. Ранее мы объявили переменную `remainingLetters`, которая содержала число букв из выбранного слова `word.length`. Угадывание всех букв и будет условием победы, выхода из цикла. Основа игрового цикла будет выглядеть так:

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

Задание

1. Создадим цикл с условием `remainingLetters > 0`
2. Первым действием в нем отрисуем пользователю количество символов слова-загадки. Метод `.join` разделит подчеркивания для лучшей читаемости

Получаем первую букву

Теперь нужно запросить у игрока ответ и убедиться, что он ввел одиночную букву. Для этого в строке `prompt` запросим у игрока ответ и сохраним его в переменной `guess`.

Задание

1. Запросим у игрока ввод буквы и сохраним его в переменной `guess`
2. Добавим проверку полученной строки. Если вернулась пустая строка остановим выполнение программы
3. Проверим вернулся ли не один символ с помощью `guess.length`, если да - скажем игроку `"Пожалуйста, введите только одну букву."`

Проверяем ввод и обновляем состояние игры

Далее возможен один из трех вариантов развития событий.

1. Первый вариант — если игрок нажмет кнопку «Отмена», guess примет значение `null`. Этот вариант мы проверяем в строке командой `if (guess === null)`. Если это условие даст `true`, мы с помощью `break` выйдем из цикла.
2. Второй вариант — 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. В строке 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`».
3. Третий вариант — игрок, как и положено, ввел одну букву. Тогда мы должны обновить состояние игры — это происходит в строке , в секции `else`. Если игрок ввел корректный ответ, мы должны обновить `answerArray` согласно этому ответу.

Задание

1. Добавим последний вариант развития событий `else`
2. Создадим цикл с итератором j, который будет менять значение от 0 до `word.length`, не включая само значение `word.length`, с шагом 1
3. Проверим, равно ли введенное значение `guess` значению символа массива под номером j
4. Если условие выполняется присвоим `answerArray[j]` значение `guess` и, так как буква отгадана, декрементируем `remainingLetters`

Осматриваем наш проект еще раз

В предыдущем цикле мы проверяем каждую букву переменной `word`. Например, пусть в `word` находится строка "стул". Тогда при первом повторе цикла, когда j равно 0, `word[j]` вернет "с". При следующем повторе `word[j]` вернет "т", затем "у" и, наконец, "л". В строке 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`,

Например, представим, что мы только начали игру и дошли до цикла `for`. Пусть загадано слово «программа», в guess находится буква `"р"`, а `answerArray` имеет вид:

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

При первом повторе `for` в строке j равно 0, поэтому `word[j]` вернет "п". Наш ответ (`guess`) — это "р", поэтому мы пропускаем команду `if` в строке (ведь условие "п" === "р" дает `false`). При следующем повторе j равно 1, и `word[j]` вернет "р". Это значение совпадает с `guess`, и срабатывает оператор `if`. Команда `answerArray[j] = guess` присваивает элементу с индексом 1 (второй элемент) массива `answerArray` значение `guess`, и теперь `answerArray` имеет вид:

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

При следующих двух повторах цикла `word[j]` вернет "о", а затем "г", что не совпадает со значением `guess`. Однако когда j достигнет 4, `word[j]` снова вернет "р". И снова мы обновим `answerArray`, на этот раз присвоив значение `guess` элементу с индексом 4 (пятый элемент). Теперь `answerArray` выглядит так:

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

Оставшиеся буквы не совпадают с "р", так что при дальнейших повторах ничего не произойдет. Так или иначе, после завершения цикла в `answerArray` будут внесены все совпадения `guess` с соответствующими позициями `word`. Помимо обновления `answerArray` для каждого совпадения с `guess` требуется уменьшать `remainingLetters` на 1. Мы делаем это в строке 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. Для этого необходимо последний раз отобразить итоговый массив и поздравить игрока с победой

Задание

1. Добавим итоговый вывод сообщения, разделенного посимвольно с помощью метода `.join(" ")`
2. Поздравим игрока по имени и скажем: `"Отлично " + name + "! Было загадано слово " + word`

Виселица!

Осталось только собрать весь файл файл в кучу, не нарушая рабочей последовательности. И так как если пользователь не пожелает играть - нам нет необходимости трогать циклы, весь описанный код необходимо поместить в тело проверки истинности его желания играть

Конспект «Циклы».

Цикл for

Синтаксис

```
for (let i = 0; i < 10; i++) {
  // Повторяющиеся команды
}
```

В круглых скобках записывается код управления циклом. Он состоит из трёх частей, разделённых ;.

1. Первая часть — подготовительная. Команды отсюда запускаются один раз перед началом работы цикла. Обычно здесь задаётся исходное значение для переменной-счётчика. Обратите внимание, что в цикле мы создаём переменную-счётчик с помощью let, как в случае с любой другой переменной.
   for (let i = 0; i < 5; i = i + 1) { }
2. Вторая часть — проверочная. Она содержит условие и запускается перед каждым новым витком цикла. Если условие возвращает true, цикл делает ещё один виток, иначе цикл завершает свою работу.
   for (let i = 0; i < 5; i = i + 1) { }
3. Третья часть — дополняющая, или «закон изменения». Код третьей части запускается после каждого витка цикла. Обычно там изменяется переменная-счётчик.
   for (let i = 0; i < 5; i = i + 1) { }

Накопление значений в цикле:

Внутри циклов можно использовать обычные математические операции. Например, сложение:

```
let sum = 0;

for (let i = 1; i <= 5; i++) {
  sum += 2;
  console.log(sum);
}
```

Программа выведет:

```
LOG: 2 (number)
LOG: 4 (number)
LOG: 6 (number)
LOG: 8 (number)
LOG: 10 (number)
```

Проверки в теле цикла

Если добавить условие внутрь цикла, то оно будет проверяться на каждой итерации.

```
let sum = 0;
```

```
for (let i = 1; i <= 5; i++) {
  if (i > 2) {
    sum += 1;
  }
}
```

Поиск чётного числа

Оператор %, или «остаток от деления», возвращает остаток от деления.

```
10 % 5;  // Вернёт 0
12 % 5;  // Вернёт 2
7 % 3;   // Вернёт 1
5.5 % 2; // Вернёт 1.5
```

Если остаток от деления числа на 2 равен 0 — число чётное, иначе нечётное.

Сокращённые операторы

В JavaScript есть несколько удобных операторов, которые позволяют сократить код:

Название	Пример	Аналог
Инкремент (увеличение на единицу)	i++	i = i + 1
Декремент (уменьшение на единицу)	i--	i = i - 1
К-к-комбо!	i += 2	i = i + 2

Комбинировать можно не только сложение, но и остальные математические операции: вычитание -=, умножение *=, деление /= и нахождение остатка %=

Цикл while

Синтаксис

```
while (условие) {
  действия
}
```

Действия будут выполняться снова и снова, пока условие не вернёт false.
Чтобы цикл остановился, условие когда-нибудь должно стать ложным. Если условие выхода из цикла не срабатывает, то цикл не может остановиться. Это бесконечный цикл, одна из любимых ошибок программистов.

break и continue

Оператор break прерывает выполнение цикла.
Аналогично оператору прерывания цикла break существует оператор для быстрого перехода к следующей итерации цикла continue, но используют его крайне редко, так как он усложняет чтение кода и понимание работы цикла в целом. Использование continue без необходимости обычно является дурным тоном.

- Внутри while команда continue «перематывает» программу сразу к началу следующей итерации.
- Внутри for команда continue «перематывает» программу к дополнительной части текущей итерации, после выполнения которой начинается следующая итерация цикла.

Накопление значений в цикле

```
let sum = 0;
let i = 0;

while (i <= 5) {
  sum += 1;
  i++;
  console.log(i);
}
```

Программа выведет:
LOG: 1 (number)
LOG: 2 (number)
LOG: 3 (number)
LOG: 4 (number)
LOG: 5 (number)
LOG: 6 (number) // Код из тела цикла не выполнится, условие вернёт false

Поиск процента от числа

Самый простой способ найти процент от числа — разделить число на 100 и умножить на процент.

```
// Найдём 2 процента от 1000
1000 / 100 * 2 = 20;

// Найдём 7 процентов от 1200
1200 / 100 * 7 = 84;
```

«Диета v1»

Вернемся во вселенную, где вы все еще спортсмен и можете летать в другие страны. В такой жизни есть минус - необходимо постоянно контролировать свой вес. Данные, которыми вы оперируете в данной ситуации - вес в начале диеты и в конце. Основной постулат диеты - при ее соблюдении и ежедневных тренировках Вы теряете 5% своей массы в день. Остановиться можно при достижении желаемого веса.
Необходимо вычислить, сколько дней понадобится провести в таком режиме и записать количество в переменную `days`.
Ежедневно Вы теряете 5% от своего веса. При этом вес обновляется каждый день. Например, если сначала Вы весили 55кг (то есть 55000 грамм), то на следующий день вес станет 52250 грамм. И во второй день он уже потеряет 5% от нового веса, то есть от 52 с четвертью кг.

JS стартовый код

```
let initialWeight = 56000; // Исходный вес в граммах
let targetWeight = 45000; // Желанный вес в граммах
let b = 0;
let days = 0;
```

Тесты

1. Начальный вес 6000 гр., хотим похудеть до 5000 гр. Ожидаю ответ 4 дня.
2. Начальный вес 5000 гр., хотим похудеть до 5000 гр. Ожидаю ответ 0 дней.
3. Начальный вес 7000 гр., хотим похудеть до 5000 гр. Ожидаю ответ 7 дней.
4. Начальный вес 9000 гр., хотим похудеть до 5500 гр. Ожидаю ответ 10 дней.
5. Начальный вес 10000 гр., хотим похудеть до 5700 гр. Ожидаю ответ 11 дней.

Вычисление геометрическая прогрессия

Геометрическая прогрессия — последовательность чисел, где каждое следующее число — это предыдущее, увеличенное на множитель.
Например, нужно написать геометрическую прогрессию из пяти чисел, начиная с единицы. Множитель — двойка. Тогда числа будут такими: 1, 2, 4, 8, 16. Здесь каждое следующее число — произведение предыдущего числа и множителя (двойки).
Напиши программу, которая последовательно выводит в консоль числа в геометрической прогрессии.
Стартовое значение, с которого должна начаться последовательность, записано в переменную `startNumber`.
Множитель записан в переменную `multiplier`.
Количество чисел записано в переменную `quantity`.

JS стартовый код

```
let startNumber = 1;
let multiplier = 4;
let quantity = 7;
```

Тесты

1. Стартовое значение 3, множитель: 3, количество: 6. Ожидаю вывод в консоль: 3, 9, 27, 81, 243, 729.
2. Стартовое значение 5, множитель: 2, количество: 7. Ожидаю вывод в консоль: 5, 10, 20, 40, 80, 160, 320
3. Стартовое значение 2, множитель: 7, количество: 5. Ожидаю вывод в консоль: 2, 14, 98, 686, 4802.

Сумма чисел

Есть такая детская математическая головоломка, в которой нужно найти самый быстрый способ посчитать сумму чисел от 1 до 100. Мы можем написать универсальную программу, которая сможет находить суммы любых чисел.
Напишите универсальную программу, которая вычисляет сумму чисел от 1 до n.
Число, до которого нужно складывать числа (включительно), указано в переменной `lastNumber`.
Найдите сумму всех чисел и сохраните результат в переменную sum

JS стартовый код

```
let lastNumber = 10;
let sum = 0;
```

Тесты

1. Сумма чисел от 1 до 15. Ожидаю результат 120
2. Сумма чисел от 1 до 25. Ожидаю результат 325
3. Сумма чисел от 1 до 46. Ожидаю результат 1081

Произведение чётных чисел

Кроме суммы чисел от 1 до n можно ещё найти их произведение. Но в этот раз задача усложнилась — нужно найти произведение не всех чисел из последовательности, а только чётных.
Напишите универсальную программу, которая находит произведение всех чётных чисел из последовательности от 1 до n.
Число, до которого идёт последовательность (включительно), записано в переменную lastNumber
Найдите произведение всех чисел и сохраните результат в переменную multiplicationResult.

JS стартовый код

```
let lastNumber = 5;
let multiplicationResult = 1;
```

Тесты

1. Последовательность чисел от 1 до 12. Ожидаю результат 46080
2. Последовательность чисел от 1 до 8. Ожидаю результат 384
3. Последовательность чисел от 1 до 10. Ожидаю результат 3840

Делители числа

Напишите программу, которая находит все делители числа, кроме единицы и самого числа.
Число, делители которого нужно найти, записано в переменную `number`.
Выводите делители в консоль последовательно, друг за другом.
Делитель — число, на которое другое число делится без остатка. Например, у числа 119 четыре делителя, на которые это число делится без остатка: 1, 7, 17, 119. Для нашей задачи единица и само число не подходят. Поэтому в результате остаётся два числа: 7, 17.
Чтобы узнать есть ли остаток от деления двух чисел, нужно использовать оператор «остаток от деления». Он обозначается знаком процента (%) и возвращает остаток от деления чисел. Если остатка от деления нет, вернётся 0. Выглядит это так:

```
12 % 5;  // Вернёт 2
27 % 3;  // Вернёт 0
13 % 3;  // Вернёт 1
```

JS стартовый код

```
let number = 15;
```

Тесты

1. Стартовое значение 114. Ожидаю вывод в консоль делителей: 2, 3, 6, 19, 38, 57.
2. Стартовое значение 122. Ожидаю вывод в консоль делителей: 2, 61.
3. Стартовое значение 110. Ожидаю вывод в консоль делителей: 2, 5, 10, 11, 22, 55.
4. Стартовое значение 135. Ожидаю вывод в консоль делителей: 3, 5, 9, 15, 27, 45.

Количество цифр в числе

Напишите программу, которая умеет определять количество цифр в любом целом числе. Само число записано в переменную `number`.
Найди количество цифр в этом числе и запиши результат в переменную `quantity`.
Чтобы решить эту задачу, можно пойти разными путями. Например, математическим: делить число на 10 и округлять его вниз на каждой итерации.

JS стартовый код

```
let number = 123;
let quantity = 0;
```

Тесты

1. Число 15200. Ожидаю количество цифр: 5
2. Число 5678. Ожидаю количество цифр: 4
3. Число 17. Ожидаю количество цифр: 2
4. Число 1. Ожидаю количество цифр: 1