Уровень 1: Введение

Бип-бип! JavaScript - это очень популярный язык сценариев для веб-разработки. Практически каждый веб-сайт (и, конечно же, этот тоже!) использует JavaScript для динамического отображения информации в вашем браузере.

Вкладка "Console" (Консоль)

Прежде чем мы начнем писать код, давайте познакомимся с основами: использованием консоли. Консоль - это инструмент, который разработчики используют для проверки правильной работы программы, ведения журнала вывода программы и взаимодействия с ней.

На вкладке **"Console"** введите 5 + 8 в текстовое поле, нажмите Enter и наблюдайте, что происходит. Первая строка (→ 5 + 8) - это ваш ввод, а вторая строка (= 13) - это результат после выполнения JavaScript вашего ввода.

Теперь для чего-то более интересного. Введите robot.info().x в консоли, результат = 80 - это положение робота по оси x в игре.

У робота есть и другие свойства, попробуйте следующее:

```
robot.info().y
robot.info().health
robot.info().energy
robot.info().width
```

Узнайте больше о консоли

Вкладка "Script" (Сценарий)

Хорошо. Теперь мы можем взаимодействовать с консолью. Давайте попробуем заставить робота разговаривать с консолью.

На вкладке "Script" вы увидите следующий код:

```
function init(robot) {
    // ваш код здесь
}

function loop(robot) {
    // ваш код здесь
}
```

Это функции JavaScript, с их помощью мы будем управлять роботом! Строки, начинающиеся с двойного слэша //, - это комментарии, они предназначены для нас (людей), чтобы лучше понимать код, который мы пишем. Программа их игнорирует.

Функция **init(robot)** является сокращением от initialize (инициализировать), эта функция выполняется один раз при запуске программы. Функция **loop(robot)** выполняется непрерывно в течение работы программы.

Измените функцию init(robot), чтобы код выглядел так:

```
function init(robot) {
    console.log("Инициализация робота...");
}
```

Функция **console.log(something)**; выводит то, что находится между скобками, в консоль. Нажмите кнопку "Run" в верхней части

страницы, затем откройте вкладку "Console". Вы увидите вывод = Инициализация робота....

Нажмите кнопку "Reset" в верхней части страницы, чтобы сбросить уровень и очистить консоль.

Теперь сделайте то же самое с функцией loop(robot), ваш код должен выглядеть так:

```
function init(robot) {
    console.log("Инициализация робота...");
}

function loop(robot) {
    console.log("Робот в цикле...");
}
```

Нажмите кнопку "Run" и снова откройте вкладку "Console". Вы увидите = Инициализация робота..., но теперь также будет писаться = Робот в цикле... снова и снова (три раза в секунду).

Нажмите кнопку **"Pause"**, чтобы остановить это. Нажмите кнопку **"Reset"**, чтобы очистить консоль.

Управление роботом

Цель игры - достигнуть флага в конце каждого уровня. Робот может получать повреждения и разряжаться, выполняя различные действия. Вы проигрываете, если у робота заканчивается энергия или он получает слишком много повреждений.

У робота есть несколько действий: **move** (движение), **jump** (прыжок), **shoot** (стрельба), **turn** (поворот) и **wait** (ожидание).

Действие **move** принимает значение от **-40 до 40**. Положительное значение перемещает робота вправо, отрицательное - влево.

Действие jump также принимает значение от -10 до 10, и как и раньше, положительное значение заставляет робота прыгнуть вправо, отрицательное - влево.

Чтобы управлять роботом, установите его действие, как показано в следующей функции loop:

```
function loop(robot) {
    robot.action = {type: 'move', amount: 40};
}
```

Это сообщит роботу двигаться на 40 пикселей вправо каждый раз при выполнении функции loop, которая выполняется три раза в секунду. Введите предоставленный код на вкладке "Script" и нажмите кнопку "Run".

Уровень 2: Продолжение введения и переменные

Вкладка HUD и свойств

В верхнем правом углу экрана вы увидите полезную информацию в интерфейсе Heads Up Display (HUD):

Здоровье и энергия робота, количество монет, собранных роботом, и положение мыши.
Это сделает игру более удобной.

Также есть вкладка **"Properties"**, где отображается более подробная информация о роботе. Вам это не понадобится прямо сейчас, но полезно знать, что она существует для будущего использования.

Режим практики
Управление с клавиатуры

В левом верхнем углу вы увидите кнопку, похожую на стрелки на клавиатуре. Это включает **"Keyboard Controls"** (Управление с клавиатуры) робота. Нажмите ее, и теперь вы можете управлять роботом с клавиатуры. Контроли по умолчанию:

Move Left (Двигаться влево): **A**
Move Right (Двигаться вправо): **D**
Jump (Прыжок): **W**
Jump Left (Прыжок влево): **Q**
Jump Right (Прыжок вправо): **E**
Turn (Поворот): **T**
Shoot Gun (Стрельба): **G**

После включения управления с клавиатуры появится сообщение **"Practice Mode**" (Режим практики). Режим практики активируется при использовании управления с клавиатуры, вкладки "Console" или при приостановке текущего сценария.

Когда вы закончите практику и захотите завершить уровень, запустите свой сценарий и избегайте использования консоли и управления с клавиатуры.

Управление роботом из консоли

Консоль может отправлять команды роботу. Попробуйте следующие:

robot.move(n): Где n - число **от -40 до 40**. Робот двигается вперед или назад на n пикселей.

robot.jump(n): Где n - число **от -10 до 10**. Робот делает прыжок вперед или назад со скоростью n.

robot.jump(): Робот делает вертикальный прыжок. Эквивалентно **robot.jump(0)**.

robot.turn(): Робот разворачивается. Это не влияет на направление предыдущих двух команд, а только на robot.shoot().

robot.shoot(): Робот стреляет из оружия!

Клавиатурные сочетания

Вкладка **"Instructions": Ctrl + 1**

Вкладка **"Script": Ctrl + 2**

Вкладка **"Console": Ctrl + 3**

Вкладка **"Properties": Ctrl + 4**

Управление с клавиатуры: **Ctrl + 5**

Запуск/Пауза сценария: **Ctrl + Enter**

Если вы используете **Mac**, используйте **Command** вместо **Ctrl**.

Типы данных

Типы данных центральны для работы языков программирования. Самые базовые встроенные типы данных называются примитивами.

Примеры примитивов в JavaScript:

Number: Любое число (целое или с плавающей точкой, положительное или отрицательное).

Примеры: 12, 7.25, 0, -10023

String: Любая последовательность символов, заключенная в одинарные или двойные кавычки.
Примеры: 'Смотрите, фейерверки!', "Итого: $25"

Boolean: Логическое значение true или false.

undefined: Специальное значение, которое указывает на то, что что-то не имеет значения. Часто свидетельствует об ошибке.

null: Значение, преднамеренно указывающее, что у чего-то нет значения. Отличие от undefined заключается в намерении.

Вы можете использовать typeof() для получения типа данных:

```
typeof(51.72);           // Выведет = "number"
typeof('I like jello'); // Выведет = "string"
typeof(true);            // Выведет = "boolean"
typeof(undefined);       // Выведет = "undefined"
typeof(null);            // Выведет = "object" (ошибочное
поведение языка)
```

Краткий курс по объектам:
Примитивы можно объединять для создания более сложных типов данных, называемых объектами.

Пример объекта:

```
robot.action = {type: 'move', amount: 40};
```

Объект представляет собой действие, состоящее из строки type и числового свойства amount.

Подробнее о объектах будет рассказано позже, так что не волнуйтесь, если вы пока не чувствуете себя комфортно с ними.

Учимся больше о переменных
Переменные
Переменная - это имя, которое ссылается на значение. Например:

```
favoriteFood = 'Лазанья';
```

Здесь **favoriteFood** - это переменная с значением в виде строки 'Лазанья'.

Переменная может иметь любой тип данных в качестве своего значения: число, строку, логическое значение, даже undefined и null:

```
favoriteColor = null; // Переменная с значением null
Существует правильный способ создания переменных в JavaScript: var variable = value;, где ключевое слово var означает "Сделайте мне переменную с именем: ".
```

Техническое название создания переменной (первого оператора) - это декларация, в то время как процесс присвоения значения переменной в первый раз (второго оператора) называется определением.

Перед определением переменной её значение - undefined.

Победный код для этого уровня такой же, как и для предыдущего:

```
function loop(robot) {
    robot.action = {type: 'move', amount: 40};
}
```

Уровни станут намного интереснее, когда мы изучим еще больше JavaScript. Я обещаю!

Уровень 3: Математика и Управление Потоком

Математика

Арифметические операторы

JavaScript поддерживает базовые арифметические операции: **+ - * / % ****.

Сложение, вычитание, умножение, деление, взятие остатка от деления и возведение в степень.

Операторы присваивания

В программировании часто используются следующие операции: i = i + 1;, m = m / 2;. Для них была разработана упрощенная запись: i += 1; и m /= 2;.

Две операции i += 1; и i -= 1; часто используются, поэтому они еще более сокращены до i++; и i--;.

Конкатенация строк

Сложение строк не является математикой, но это очень полезно. Вы можете объединять несколько строк с помощью оператора +:

"Роботы " + "крутые" + '!' дает строку "Роботы крутые!"

Числа можно добавлять к строкам. Явный способ - использовать конструктор String().

String(47) превращает число **47** в строку **"47"**. И затем мы можем выполнять конкатенацию со строкой.

"В этом пакете " + String(47) + " жевательных конфет."

Но часто достаточно очевидно, что мы пытаемся сделать, поэтому следующий вариант тоже работает в JavaScript:

"В этом пакете " + 47 + " жевательных конфет."

Правило: сначала преобразовывается число в строку при сложении числа и строки.

Но в некоторых случаях это может быть не так полезно, как хотелось бы. Например, `"7" + 3` дает строку `"73"`, что может не соответствовать вашим ожиданиям, поэтому будьте осторожны.

Преобразование строк в числа

Что произойдет, если мы попытаемся вычесть, например, `"7" - 3`? Попробуйте, и вы получите **4**.

JavaScript умел определить, что строка **"7"** на самом деле представляет число.

Но хотя это полезный ответ, и JavaScript - очень умный язык, мы хотим избегать использования чисел и строк вместе таким образом.

Если вы знаете, что у вас есть строка, содержащая число, преобразуйте ее в число с помощью одной из следующих функций:

Number.parseInt(): Преобразует строку, представляющую целое число, в число, игнорируя любые дополнительные символы. Number.parseInt() для "23", "7.25 литров", "1.995" и "3 яблока" дает результаты: = 23 = 7 = 1 = 3.

Number.parseFloat(): Преобразует строку, представляющую число с плавающей точкой, в число, игнорируя любые дополнительные символы. Number.parseFloat() для "23", "7.25 литров", "1.995" и "3 яблока" дает результаты: = 23 = 7.25 = 1.995 = 3.

NaN

Что происходит, когда мы пытаемся сделать что-то глупое с числами? Например, '**апельсин**' **- 4**, `Number.parseInt('футбол'), 0 / 0, (-1) ** 0.5`.

Мы получаем **NaN (Not a Number)**, что означает **"Не число"**.

NaN интересно, **typeof(NaN)** дает нам = **"number"**...

Так что "Не число" на самом деле является числом.

Есть встроенная функция для проверки наличия значений NaN: **Number.isNaN()**.

Infinity

Некоторые свойства Infinity:

1 / 0 = Infinity
1 / Infinity = 0
Infinity + 1 = Infinity
10 ** 1000 = Infinity
Узнать больше о объекте Number

Продвинутые математические функции

Math.round(number): Возвращает число, округленное до ближайшего целого.
Math.abs(number): Возвращает абсолютное значение числа.
Math.sin(number): Возвращает синус числа.
Math.cos(number): Возвращает косинус числа.
Math.sqrt(number): Возвращает квадратный корень числа.
Math.pow(base, exponent): Возвращает число, возведенное в степень.
Math.log(number): Возвращает натуральный логарифм ln() числа.
Math.max(x, y, ...): Возвращает наибольшее из переданных чисел.
Math.min(x, y, ...): Возвращает наименьшее из переданных чисел.

Узнать больше о математических функциях

Булевы значения

Следующие операторы возвращают true или false. Они будут необходимы для использования Управления Потоком, которое является основной концепцией этого уровня.

Операторы равенства

a == b: Возвращает **true**, если **a** и **b** равны по значению, и **false** в противном случае. Обратите внимание, что a и b могут иметь разные типы и все равно быть равными.

Примеры:

```
3.25 == 3.25 = true,
3 == "3" = true,
12 == 0.2 = false
```

a === b: Возвращает **true**, **если a и b равны по значению и имеют одинаковый тип**, и **false** в противном случае.

Примеры:

```
3.25 === 3.25 = true
3 === "3" = false
12 === 0.2 = false
```

a != b: Возвращает **true**, если a и b не равны по значению, независимо от типа, и false в противном случае.

Примеры:

```
3.25 != 3.25 = false
3 != "3" = false
12 != 0.2 = true
```

a !== b: Возвращает **true**, **если a и b либо не равны по значению, либо не равны по типу**, и false в противном случае.

Примеры:

```
3.25 !== 3.25 = false
3 !== "3" = true
12 !== 0.2 = true
```

Операторы сравнения

a < b: Возвращает true, если a меньше b, и false в противном случае.
Примеры:

```
1 < 2 = true
2 < 2 = false
3 < 2 = false
```

a <= b: Возвращает true, если a меньше или равно b, и false в противном случае.
Примеры:

```
 1 <= 2 = true
 2 <= 2 = true
3 <= 2 = false
```

a > b: Возвращает true, если a больше b, и false в противном случае.
Примеры: `1 > 2 = false, 2 > 2 = false, 3 > 2 = true`

a >= b: Возвращает true, если a больше или равно b, и false в противном случае.
Примеры: `1 >= 2 = false, 2 >= 2 = true, 3 >= 2 = true`

Логические операторы

!a: Оператор NOT, меняет true на false и наоборот.
Примеры: `!true = false, !false = true`

a && b: Оператор AND, возвращает true, если и a, и b - true, и false в противном случае.
Примеры:

```
true && true = true,
 true && false = false,
true && true = false
```

a || b: Оператор OR, возвращает true, если хотя бы одно из a и b - true, и false в противном случае.

Примеры:

```
true || true = true,
true || false = true,
true || true = false
```

Ложные значения

Ложное значение - это такое, которое ведет себя, как если бы оно было ложным. В JavaScript существует шесть **"ложных"** значений:

false: очевидно

0 и -0

"" или **''** - пустая строка

NaN (Not a Number)

undefined

null

Попробуйте следующие команды с ложными значениями, чтобы увидеть, как они ведут себя как ложные:

```
!(значение)
значение && true
значение || false
```

Затем попробуйте их с обычными числами и строками, и посмотрите, как меняется поведение.

Управление потоком

If-Else

Операторы управления потоком позволяют программе выполнять различный код в зависимости от условия. Если определенное условие истинно, выполняется один блок кода; если ложно, выполняется другой блок кода.

В следующем коде приведен пример оператора if-else:

```
var rollsOfToiletPaper = 5;
if (rollsOfToiletPaper <= 1) {
    console.log("Мне нужно купить больше туалетной
бумаги!");
} else if (rollsOfToiletPaper > 50){
    console.log("Почему я купил так много туалетной
бумаги??");
} else {
    console.log("Все в порядке.");
}
```

Эта программа должна быть легкой для понимания:

Если значение **rollsOfToiletPaper** меньше или равно 1, выполняется код в первой паре фигурных скобок.

Если оно больше 50, выполняется код между второй парой фигурных скобок.

В противном случае выполняется код в последней паре фигурных скобок.

Примечание: Код, написанный между фигурными скобками, называется блоком кода.

Формальное определение: после ключевого слова **if** всегда следует условие внутри пары круглых скобок **(condition)**, после условия мы помещаем блок кода **{ сделайте что-то здесь }**. По желанию, затем мы можем добавить else if, который работает

так же, как и if, или просто else, который не требует условия, а только блок кода.

Как только выполнено первое условие, блок, следующий за ним, выполняется, блоки для всех остальных операторов else игнорируются.

Switch

Иногда приходится писать программы вроде этой:

```
var letter = 'C';
if (letter === 'A') {
    console.log('A for Aligator');
} else if (letter === 'B') {
    console.log('B for Beaver');
} else if (letter === 'C') {
    console.log('C for Chameleon');
} else if (letter === 'D') {
    console.log('D for Dishwasher');
} else if (letter === 'E') {
    console.log('E for Elephant');
} else {
    console.log('Unsupported letter, please try again');
}
```

Это **утомительно** и **трудно читаемо**. Гораздо более удобный и привлекательный способ написания той же программы - с использованием оператора **switch**:

```
var letter = 'C';
switch (letter) {
    case 'A':
        console.log('A for Aligator');
        break;
    case 'B':
        console.log('B for Beaver');
        break
    case 'C':
        console.log('C for Chameleon');
        break;
    case 'D':
        console.log('D for Dishwasher');
        break;
    case 'E':
        console.log('E for Elephant');
}
```

Условный оператор (или тернарный оператор)

```
condition ? statement1 : statement2;
```

Если условие оценивается как **true**, то выполняется **statement1,** в противном случае выполняется **statement2.**

Рассмотрим следующий код:

```
var milkExpired = false;
if (milkExpired) {
    console.log("Сегодня я не завтракаю :(");
} else {
    console.log("Пора кушать кашу!");
}
```

Используя условный оператор, его можно сократить до:

```
var milkExpired = false;
milkExpired ? console.log("Сегодня я не завтракаю.") :
console.log("Пора кушать кашу!");
```

На самом деле его можно сократить еще больше:

```
var milkExpired = false;
console.log( milkExpired ? "Сегодня я не завтракаю." :
"Пора кушать кашу!");
```

Играть в уровень

Очевидно, что роботу придется прыгать. Попробуйте ввести следующую команду во вкладке **"Console":**

```
robot.jump(10);
```

Это самое дальний прыжок вперед, который может совершить робот.

Попробуйте чередовать между прыжками и движением вперед:

```
robot.move(40);
```

Когда текстовый ввод выбран в консоли, используйте клавиши вверх и вниз, чтобы перемещаться вперед и назад в истории выполненных вами команд.

Если можете, попробуйте написать свое решение во вкладке **"Script"**.

Одна вещь, о которой нужно помнить: если вы объявите переменную внутри функции **init()**, она не будет доступна в функции **loop()**.