Название

## **Цель урока**

Сделать эффектное меню для игр

## **Содержание**

* [Преобразим стилями](#_ttc2fnp69a6n)
* [Шаблонные строки](#_6by5qq9yief5)
* [Метод массива join](#_nzm234pl2uof)
* [Домашнее задание](#_9ph5bdf7wigu)

## **Что повторяем (закрепляем)**

CSS: базовые свойства, transition, :hover, flex

JS:Тернарный оператор, createElement, remove, setTimeout, setInterval, clearInterval

## **Что нового**

Шаблонные строки, метод массива join

## **Ссылки на материалы и личный кабинет**

Рабочие материалы (для преподавателя).

Материалы (сбрасываем студенту эту ссылку в начале урока).

Урок в личном кабинете (сбрасываем студенту эту ссылку в конце урока).

# Методический материал

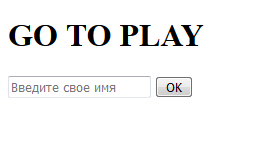
Мы будем делать программу - начало игры, в котором мы указываем имя игрока, затем программа пишет нам правила игры с эффектом печатной машинки, обращаясь к нам по имени. После появляется кнопка “Старт”, нажав на которую начинается игра.

Создаем шаблон html, подключаем css и js.

|  |
| --- |
| **<!DOCTYPE html>** <html lang="en" dir="ltr">  <head>  <meta charset="utf-8">  <title>Start game</title>  <link rel="stylesheet" href="style.css">  </head>  <body>   <script type="text/javascript" src="script.js">  </script>  </body> </html> |

Нам понадобится div, в которым мы все обернем. в теге h1 напишем что-то воодушевляющее вроде “go to play”, сразу создадим пустой <p></p> чтоб не делать лишний раз это через js, он нам мешать не будет. Также окно ввода и кнопку. В целом все будет выглядеть вот так:

|  |
| --- |
| **<!DOCTYPE html>** <html lang="en" dir="ltr">  <head>  <meta charset="utf-8">  <title></title>  <link rel="stylesheet" href="style.css">    </head>  <body>  <div>  <h1>GO TO PLAY</h1>  <p></p>  <input placeholder="Введите свое имя" type="text" name="" value="">  <button type="button" name="button">OK</button>  </div>   <script src="script.js">   </script>  </body> </html> |



## Преобразим стилями

Нам требуется выровнять его центру. Чтоб было проще регулировать размер элемента, лучше сделать дисплей flex. Сделаем выравнивание по центру и воспользуемся флекс свойством justify-content, чтоб выровнять элемент по центру.

|  |
| --- |
| **body**{  text-align: center;  display: flex;  justify-content: center;   } |

При помощи margin-top переместим наш div немного ниже, так как это flex элемент, мы можем регулировать его высоту. Добавим ширину и высоту.

|  |
| --- |
| **div**{  margin-top: 200px;  width: 30%;  height: 200px;   } |

Чтоб видеть, что у нас получилось, стоит добавить границу.

|  |
| --- |
| **div**{  margin-top: 200px;  width: 30%;  border: 5px solid white;  border-radius: 20px;  height: 200px;  } |

Для эффектности, добавим гифку на задний фон и так как она темная, цвет шрифта сделаем белым. С помощью свойств для заднего фона, сделаем ей нормальное расположение внутри элемента и размер.

|  |
| --- |
| **div**{  color: white;  margin-top: 200px;  width: 30%;  border: 5px solid white;  border-radius: 20px;  height: 200px;  background: url(giphy.gif);  background-repeat: no-repeat;  background-size: 100%;  background-position: center;  } |

Опционально можно добавить box-shadow, сменить цвет фона у body…

|  |
| --- |
| **body**{  text-align: center;  color: white;  display: flex;  justify-content: center;  background: linear-gradient(to right, black, rgb(4, 139, 208)); } **div**{  margin-top: 200px;  width: 30%;  border: 5px solid white;  border-radius: 20px;  box-shadow: 0 0 20px white;  height: 200px;  background: url(giphy.gif);  background-repeat: no-repeat;  background-size: 100%;  background-position: center;  } |

Теперь стиль для кнопок и инпута. Можно попросить ученика сделать самостоятельно, чтоб посмотреть его текущий уровень. Не забываем про hover и можно попросить ученика добавить transition. Если не помнит что это, то предложить ему загуглить и прочитать, как это используется.

Для примера:

|  |
| --- |
| **input**{  border-radius: 10px;  font-size: 20px;  padding: 5px;  border: none;  text-align: center; } **button**{  width: 50%;  margin: 20px;  padding: 10px;  background-color: black;  color: white;  border: none;  border-radius: 10px;  box-shadow: 0 0 5px white;  transition: box-shadow .5s; } **button**:hover{  box-shadow: 0 0 50px white; } |



На данном этапе этого достаточно. Разберемся с js.

Мы вводим свое имя и по нажатию на кнопку должна выполниться функция, которая с нами по здоровается и будет писать правила игры. Значит значение инпута нужно записать в переменную.

Создадим функцию

|  |
| --- |
| **let** input = document.querySelector('input') **function** **printRules**(){  **let** name = ' '  name = input.value  **let** rules = 'Привет, Neo. Тут будут правила игры'  } |

## Шаблонные строки

В данном случае нам удобнее использовать шаблонные строки, чтоб вставить переменную в строку. Это как конкатинация “привет, “ + name, но удобнее.

Для начала, нужно использовать обратные кавычки. ` `. И такой синтаксис.

|  |
| --- |
| **let** input = document.querySelector('input') **function** **print**(){  **let** name = ' '  name = input.value  **let** rules = `Привет, ${name}. Тут будут правила игры`  } |

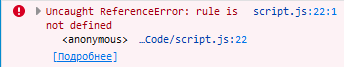
Давай попробуем вывести в консоль rules, по нажатию на кнопку.

|  |
| --- |
| **function** **printRules**(){  **let** name = ' '  name = input.value  **let** rules = `Привет, ${name}. Тут будут правила игры`  console.log(rules)  } |

И повесить атрибут onclick на нашу кнопку.



Вопрос, почему возникает такая ошибка?



Неплохая возможность в очередной раз рассказать про область видимости, если не догадался сам. Значит вынесем rules в глобальную область видимости и все заработает.

Теперь просто перенесем это данные из консоли в тег ‘p’.

|  |
| --- |
| **function** **printRules**(){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  document.querySelector('p').innerHTML = rules  } |

У нас текст появляется сразу, а хотелось бы сделать эффект печатной машинки, когда буква за буквой пишется с каким-то временным промежутком.

Создадим массив, в который будут записываться буквы из rules по очереди, а отрисовывать на страничке будет не rules, а этот массив.

Пишем функцию для записи одной буквы в массив

|  |
| --- |
| **let rules let arr = []   let input = document.querySelector('input') let button = document.querySelector('button')   function iteration() {  arr.push(rules[0])  document.querySelector('p').innerHTML = arr }  function printRules(){  let name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  iteration()  }** |

Но нам нужно потом записать rules[1], после 2, 3…

Для этого сделаем внешний счетчик, который станет на единицу больше после выполнения функции.

|  |
| --- |
| **let** counter = 0 **function** **iteration**() {  arr.push(rules[counter])  document.querySelector('p').innerHTML = arr  counter++ } |

И теперь нам нужно выполнять эту функцию много раз с промежутком во времени. Значит нам нужна функция setInterval. Чем она отличается от setTimeout?

|  |
| --- |
| **function** **printRules**(){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  setInterval(iteration, 100)   } |

И это будет выглядеть так



## Метод массива join

Это говорит о двух вещах. Первое - он печатает нам не строку, а массив. А второе - нам нужно прекратить повторять вызов функции, когда мы отрисуем все буквы.

Решаем первую ошибку. Чтоб все элементы массива превратить в строку используется метод массива join

|  |
| --- |
| **function** **iteration**() {  arr.push(rules[counter])  document.querySelector('p').innerHTML = arr.join(' ')  counter++   } |

В скобках передается символ, которым будет разделяться один элемент массива от другого. Мы там передаем пустую строку, то есть - ничем. Также можем заметить, что наша вторая проблема исчезла, это связано с тем, что в тег ‘p’ передается кучу раз пустая строка, а сколько бы мы пустых строк туда не добавили - они не отобразятся в браузере. Но все же js продолжает выполнять функцию, чтоб не заставлять его делать лишнюю работу, сделаем очистку интервала. Если счетчик больше длины строки - clearInterval

|  |
| --- |
| **let** rules **let** arr = [] **let** counter = 0  let interval   **let** input = document.querySelector('input') **let** button = document.querySelector('button')   **function** **iteration**() {  arr.push(rules[counter])  document.querySelector('p').innerHTML = arr.join('')  counter++  **if** (couter>rules.length) {  clearInterval(interval)  }   }  **function** **printRules**(){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  interval = setInterval(iteration, 100)   } |

Для еще большей красоты, удалим input и button, когда кнопка будет нажата.

|  |
| --- |
| **function** **printRules**(){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  interval = setInterval(iteration, 100)  input.remove()  button.remove()   } |

Теперь создадим ссылку, которая будет указывать на игру, вступление для которой мы делаем

|  |
| --- |
| let createA = document.createElement('a') |

Теперь запишем, что будет находиться внутри этого тега

|  |
| --- |
| createA.innerHTML = 'START GAME' |

А теперь сделаем href. Можно чтоб href был значением, которое мы передаем в функцию, чтоб пользователю нашей функцией нужно было только написать print(ссылка на игру).

|  |
| --- |
| **function** **printRules**(linkToYourGame){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  interval = setInterval(iteration, 100)  input.remove()  button.remove()  **let** createA = document.createElement('a')  createA.innerHTML = 'START GAME'  createA.href = linkToYourGame   } |



А теперь добавляем элемент в div

|  |
| --- |
| document.querySelector('div').append(createA) |

Не забудем сделать css для ‘a’

|  |
| --- |
| **a**{  text-decoration: none;  color: white;  background-color: black;  padding: 10px;  border-radius: 10px;  box-shadow: 0 0 5px white;  transition: box-shadow .5s; } **a**:hover{  box-shadow: 0 0 50px white; } |

Вот так выглядит конечный код

|  |
| --- |
| **let** rules **let** arr = [] **let** counter = 0   **let** input = document.querySelector('input') **let** button = document.querySelector('button')   **function** **iteration**() {  arr.push(rules[counter])  document.querySelector('p').innerHTML = arr.join('')  counter++  **if** (counter>rules.length) {  clearInterval(interval)    }  }  **function** **printRules**(linkToYourGame){  **let** name = ' '  name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  interval = setInterval(iteration, 100)  input.remove()  button.remove()  **let** createA = document.createElement('a')  createA.innerHTML = 'START GAME'  createA.href = linkToYourGame  document.querySelector('div').append(createA)   } |

## Домашнее задание

Читать

[про join](https://basicweb.ru/javascript/js_array_join.php)

[про let var const и область видимости](https://learn.javascript.ru/let-const)

Все мы знаем, что глобальные переменные - это зло. Нужно сделать так, чтоб все наши переменные были внутри функции и при этом все исправно работало.

Ответ

|  |
| --- |
| **function** **printRules**(linkToYourGame){  **let** rules  **let** arr = []  **let** counter = 0  **let** input = document.querySelector('input')  **let** button = document.querySelector('button')  **let** name = ' '   name = input.value  rules = `Привет, ${name}. Тут будут правила игры`  interval = setInterval(iteration, 100)  input.remove()  button.remove()  **let** createA = document.createElement('a')  createA.innerHTML = 'START GAME'  createA.href = linkToYourGame  document.querySelector('div').append(createA)    **function** **iteration**() {  arr.push(rules[counter])  document.querySelector('p').innerHTML = arr.join('')  counter++  **if** (counter>rules.length) {  clearInterval(interval)   }  } } |

Доп.задание( сделать так, чтоб ссылка появлялась после того, как отрисуется текст).

Ответ: Вставить все что касается ссылки в условие counter>rules.length