**Лабораторная работа №22**

**Тема работы:** Разработка программ с применением таймеров в JavaScript.

**Цель работы:** закрепить навыки по работе с операторами временной задержки.

**Теоретические сведения**

**Планирование: setTimeout и setInterval**

Мы можем вызвать функцию не в данный момент, а позже, через заданный интервал времени. Это называется «планирование вызова».

Для этого существуют два метода:

* setTimeout позволяет вызвать функцию **один раз** через определённый интервал времени.
* setInterval позволяет вызывать функцию **регулярно**, повторяя вызов через определённый интервал времени.

Эти методы не являются частью спецификации JavaScript. Но большинство сред выполнения JS-кода имеют внутренний планировщик и предоставляют доступ к этим методам. В частности, они поддерживаются во всех браузерах и Node.js.

**[setTimeout](https://learn.javascript.ru/settimeout-setinterval" \l "settimeout)**

Синтаксис:

let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)

Параметры:

func|code

Функция или строка кода для выполнения. Обычно это функция. По историческим причинам можно передать и строку кода, но это не рекомендуется.

delay

Задержка перед запуском в миллисекундах (1000 мс = 1 с). Значение по умолчанию – 0.

arg1, arg2…

Аргументы, передаваемые в функцию (не поддерживается в IE9-)

Например, данный код вызывает sayHi() спустя одну секунду:

function sayHi() {

alert('Привет');

}

*setTimeout(*sayHi*,* *1000);*

С аргументами:

function sayHi(phrase, who) {

alert( phrase + ', ' + who );

}

*setTimeout(*sayHi*,* *1000,* *"Привет",* *"Джон");* *// Привет, Джон*

Если первый аргумент является строкой, то JavaScript создаст из неё функцию.

Это также будет работать:

setTimeout("alert('Привет')", 1000);

Но использование строк не рекомендуется. Вместо этого используйте функции. Например, так:

setTimeout(() => alert('Привет'), 1000);

Передавайте функцию, но не запускайте её

Начинающие разработчики иногда ошибаются, добавляя скобки () после функции:

// не правильно!

setTimeout(sayHi(), 1000);

Это не работает, потому что setTimeout ожидает ссылку на функцию. Здесь sayHi() запускает выполнение функции, и результат выполнения отправляется в setTimeout. В нашем случае результатом выполнения sayHi() является undefined (так как функция ничего не возвращает), поэтому ничего не планируется.

**[Отмена через clearTimeout](https://learn.javascript.ru/settimeout-setinterval" \l "otmena-cherez-cleartimeout)**

Вызов setTimeout возвращает «идентификатор таймера» timerId, который можно использовать для отмены дальнейшего выполнения.

Синтаксис для отмены:

let timerId = setTimeout(...);

clearTimeout(timerId);

В коде ниже планируем вызов функции и затем отменяем его (просто передумали). В результате ничего не происходит:

let timerId = setTimeout(() => alert("ничего не происходит"), 1000);

alert(timerId); // идентификатор таймера

clearTimeout(timerId);

alert(timerId); // тот же идентификатор (не принимает значение null после отмены)

Как мы видим из вывода alert, в браузере идентификатором таймера является число. В других средах это может быть что-то ещё. Например, Node.js возвращает объект таймера с дополнительными методами.

Повторюсь, что нет единой спецификации на эти методы, поэтому такое поведение является нормальным.

Для браузеров таймеры описаны в [разделе таймеров](https://www.w3.org/TR/html5/webappapis.html#timers) стандарта HTML5.

**[setInterval](https://learn.javascript.ru/settimeout-setinterval" \l "setinterval)**

Метод setInterval имеет такой же синтаксис как setTimeout:

let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)

Все аргументы имеют такое же значение. Но отличие этого метода от setTimeout в том, что функция запускается не один раз, а периодически через указанный интервал времени.

Чтобы остановить дальнейшее выполнение функции, необходимо вызвать clearInterval(timerId).

Следующий пример выводит сообщение каждые 2 секунды. Через 5 секунд вывод прекращается:

// повторить с интервалом 2 секунды

let timerId = setInterval(() => alert('tick'), 2000);

// остановить вывод через 5 секунд

setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);

Во время показа alert время тоже идёт

В большинстве браузеров, включая Chrome и Firefox, внутренний счётчик продолжает тикать во время показа alert/confirm/prompt.

Так что если вы запустите код выше и подождёте с закрытием alert несколько секунд, то следующий alert будет показан сразу, как только вы закроете предыдущий. Интервал времени между сообщениями alert будет короче, чем 2 секунды.

**[Рекурсивный setTimeout](https://learn.javascript.ru/settimeout-setinterval" \l "rekursivnyy-settimeout)**

Есть два способа запускать что-то регулярно.

Один из них setInterval. Другим является рекурсивный setTimeout. Например:

/\*\* вместо:

let timerId = setInterval(() => alert('tick'), 2000);

\*/

let timerId = setTimeout(function tick() {

alert('tick');

timerId *=* *setTimeout(*tick*,* *2000);* *// (\*)*

}, 2000);

Метод setTimeout выше планирует следующий вызов прямо после окончания текущего (\*).

Рекурсивный setTimeout – более гибкий метод, чем setInterval. С его помощью последующий вызов может быть задан по-разному в зависимости от результатов предыдущего.

Например, необходимо написать сервис, который отправляет запрос для получения данных на сервер каждые 5 секунд, но если сервер перегружен, то необходимо увеличить интервал запросов до 10, 20, 40 секунд… Вот псевдокод:

let delay = 5000;

let timerId = setTimeout(function request() {

...отправить запрос...

if (ошибка запроса из-за перегрузки сервера) {

// увеличить интервал для следующего запроса

delay \*= 2;

}

timerId = setTimeout(request, delay);

}, delay);

А если функции, которые мы планируем, ресурсоёмкие и требуют времени, то мы можем измерить время, затраченное на выполнение, и спланировать следующий вызов раньше или позже.

**Рекурсивный setTimeout позволяет задать задержку между выполнениями более точно, чем setInterval.**

Сравним два фрагмента кода. Первый использует setInterval:

let i = 1;

setInterval(function() {

func(i);

}, 100);

Второй использует рекурсивный setTimeout:

let i = 1;

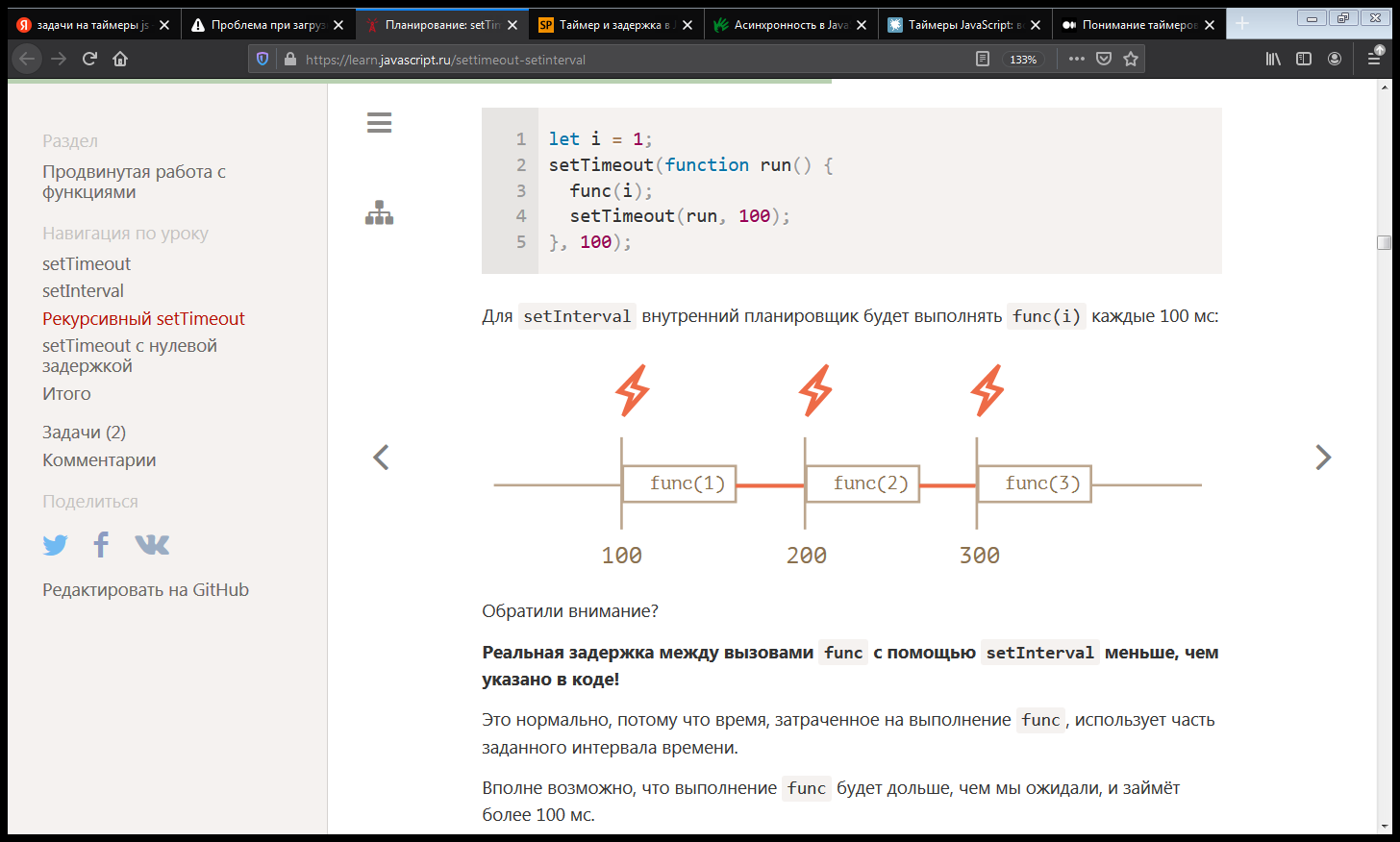
setTimeout(function run() {

func(i);

setTimeout(run, 100);

}, 100);

Для setInterval внутренний планировщик будет выполнять func(i) каждые 100 мс:



Обратили внимание?

**Реальная задержка между вызовами func с помощью setInterval меньше, чем указано в коде!**

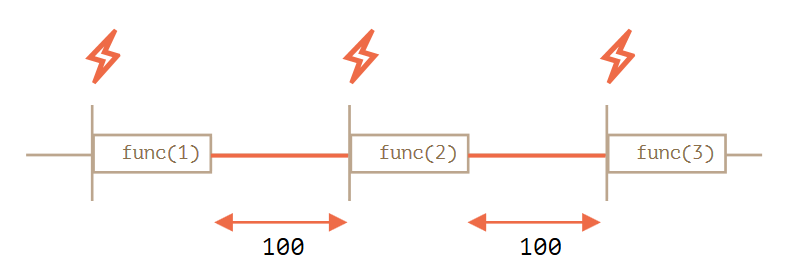
Это нормально, потому что время, затраченное на выполнение func, использует часть заданного интервала времени.

Вполне возможно, что выполнение func будет дольше, чем мы ожидали, и займёт более 100 мс.

В данном случае движок ждёт окончания выполнения func и затем проверяет планировщик и, если время истекло, немедленно запускает его снова.

В крайнем случае, если функция всегда выполняется дольше, чем задержка delay, то вызовы будут выполняться без задержек вообще.

Ниже представлено изображение, показывающее процесс работы рекурсивного setTimeout:



**РекурсивныйsetTimeout гарантирует фиксированную задержку (здесь 100 мс).**

Это потому, что новый вызов планируется в конце предыдущего.

Сборка мусора и колбэк setTimeout/setInterval

Когда функция передаётся в setInterval/setTimeout, на неё создаётся внутренняя ссылка и сохраняется в планировщике. Это предотвращает попадание функции в сборщик мусора, даже если на неё нет других ссылок.

// функция остаётся в памяти до тех пор, пока планировщик обращается к ней

setTimeout(function() {...}, 100);

Для setInterval функция остаётся в памяти до тех пор, пока не будет вызван clearInterval.

Есть и побочный эффект. Функция ссылается на внешнее лексическое окружение, поэтому пока она существует, внешние переменные существуют тоже. Они могут занимать больше памяти, чем сама функция. Поэтому, если регулярный вызов функции больше не нужен, то лучше отменить его, даже если функция очень маленькая.

**[setTimeout с нулевой задержкой](https://learn.javascript.ru/settimeout-setinterval" \l "settimeout-s-nulevoy-zaderzhkoy)**

Особый вариант использования: setTimeout(func, 0) или просто setTimeout(func).

Это планирует вызов func настолько быстро, насколько это возможно. Но планировщик будет вызывать функцию только после завершения выполнения текущего кода.

Так вызов функции будет запланирован сразу после выполнения текущего кода.

Например, этот код выводит «Привет» и затем сразу «Мир»:

setTimeout(() => alert("Мир"));

alert("Привет");

Первая строка помещает вызов в «календарь» через 0 мс. Но планировщик проверит «календарь» только после того, как текущий код завершится. Поэтому "Привет" выводится первым, а "Мир" – после него.

Есть и более продвинутые случаи использования нулевой задержки в браузерах, которые мы рассмотрим в главе [Событийный цикл: микрозадачи и макрозадачи](https://learn.javascript.ru/event-loop).

Минимальная задержка вложенных таймеров в браузере

В браузере есть ограничение на то, как часто внутренние счётчики могут выполняться. В [стандарте HTML5](https://www.w3.org/TR/html5/webappapis.html#timers) говорится: «после пяти вложенных таймеров интервал должен составлять не менее четырёх миллисекунд.».

Продемонстрируем в примере ниже, что это означает. Вызов setTimeout повторно вызывает себя через 0 мс. Каждый вызов запоминает реальное время от предыдущего вызова в массиве times. Какова реальная задержка? Посмотрим:

let start = Date.now();

let times = [];

setTimeout(function run() {

times.push(Date.now() - start); // запоминаем задержку от предыдущего вызова

if (start + 100 < Date.now()) alert(times); // показываем задержку через 100 мс

else setTimeout(run); // если нужно ещё запланировать

});

// пример вывода:

// 1,1,1,1,9,15,20,24,30,35,40,45,50,55,59,64,70,75,80,85,90,95,100

Первый таймер запускается сразу (как и указано в спецификации), а затем задержка вступает в игру, и мы видим 9, 15, 20, 24....

Аналогичное происходит при использовании setInterval вместо setTimeout: setInterval(f) запускает f несколько раз с нулевой задержкой, а затем с задержкой 4+ мс.

Это ограничение существует давно, многие скрипты полагаются на него, поэтому оно сохраняется по историческим причинам.

Этого ограничения нет в серверном JavaScript. Там есть и другие способы планирования асинхронных задач. Например, [setImmediate](https://nodejs.org/api/timers.html) для Node.js. Так что это ограничение относится только к браузерам.

**Порядок выполнения работы**

Задача 1

На странице разместить кнопку, при нажатии на которую запускается таймер, вывод сделать в документ. Так же добавить кнопки, которые будут останавливать таймер и отчищать его.

Задача 2

Реализовать функцию, которая будет отсчитывать время до полуночи. Так же дополнить функцию таким образом, чтобы можно было вводить дату, до которой производить отсчет, дата не может быть меньше текущей.