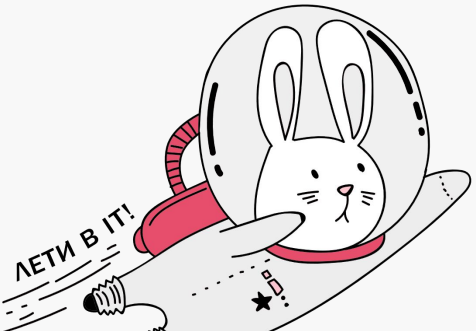
# Events in Javascript
# Capturing/Target/Bubbling
# SetTimeout, SetInterval
# Event Loop




dev {education}

# window.setTimeout, window.setInterval

Функции JavaScript setInterval() и setTimeout() позволяют вызывать код JavaScript через регулярные интервалы. Приложения, в которых требуются регулярные графические обновления, например аркадные игры, будет очень сложно (точнее, практически невозможно) написать без применения таких функций. Для многократного вызова функции ее можно передать setInterval() в качестве обратного вызова.

# Example

```
// Это функция обратного вызова.

const bigFunction = function() {
      // Делаем что-либо…
      // Этот код должен вызываться с регулярными интервалами.
      // На выполнение кода уходит 20 миллисекунд.
};

// setInterval будет пытаться вызвать bigFunction() каждые
   50 миллисекунд.

setInterval(bigFunction, 50);
```

# Особенности setInterval

На исполнение функции bigFunction() уходит 20 миллисекунд. Если задать более краткий интервал, setInterval(bigFunction, 15), новый обратный вызов будет поставлен в очередь и выполнен уже после того, как завершится первый обратный вызов.

Уменьшим эту задержку.

setInterval(bigFunction, 5);

Логично предположить, что за время, требуемое на выполнение первого обратного вызова, в очередь попадет несколько последующих вызовов setInterval().

Начнет ли обратный вызов, стоящий в очереди, выполняться сразу после того, как закончится предыдущий обратный вызов? Возможно, но гарантировать этого нельзя.

# Особенности setInterval

В браузере одновременно могут происходить другие события и выполняться другой код, они могут обусловить увеличение задержки при обработке обратных вызовов setInterval() либо вообще их сбросить. Более того, обратные вызовы вообще могут выполняться подряд, с интервалом меньше указанного. Это произойдет, если JavaScript обнаружит «свободное окно» и сбросит очередь.

При работе с setInterval() интервалы задаются в миллисекундах, поэтому нельзя гарантировать, что обратные вызовы будут выполняться точно с указанным интервалом.

# setTimeout

setTimeout() вызывает функцию, делает это только один раз и по истечении указанной задержки. Эту функцию можно считать более предсказуемым аналогом setInterval().

```
setTimeout(bigFunction, 50);
```

Таким образом, мы вызовем функцию bigFunction() только один раз, после задержки в 50 миллисекунд. Как и в случае с setInterval(), эту задержку следует считать просто ориентировочным значением. Можно использовать setTimeout() и для того, чтобы последовательно вызывать функцию несколько раз, но поведение кода получится менее предсказуемым, чем при применении setInterval().

# Example

```
// Это функция обратного вызова.

const bigFunction = function() {
    // Делаем что-либо...
    // Этот код должен вызываться регулярно.
    // На его выполнение уходит 20 миллисекунд.

    setTimeout(bigFunction, 10);
};
```

# Особенности setTimeout

Всякий раз, когда функция bigFunction() завершает работу, она устанавливает новую функцию setTimeount(), задавая для нее себя в качестве обратного вызова. В этом примере указанная задержка меньше, чем период, необходимый для выполнения bigFunction(). Тем не менее заданный обратный вызов setTimeount() выполнится только после того, как bigFunction() закончит работу. Фактически частота исполнения получится практически такой же, как и при выполнении альтернативного кода, использующего setInterval():

```
setInterval(bigFunction, 20+10);
```

# Event loop

https://www.youtube.com/watch?v=8cV4ZvHXQL4

# MVC

(model view controller)

Шаблон проектирования MVC родом из 1970-х. Он появился в научно-исследовательском центре Xerox PARC в ходе работы над языком программирования Smalltalk. Шаблон прошёл проверку временем в деле разработки графических пользовательских интерфейсов. Он пришёл в веб-программирование из настольных приложений и доказал свою эффективность в новой сфере применения.

# MVC

По сути, MVC — это способ чёткого разделения ответственностей. В результате конструкция решения, основанного на нём, оказывается понятной даже новому программисту, который по каким-то причинам присоединился к проекту. Как результат, даже тому, кто с проектом знаком не был, легко в нём разобраться, и, при необходимости, внести вклад в его разработку.

# MVC

Model - хранит текущее состояние, отвечает за работу с данными.
View - document.
Controller - взаимодействует с представлением(view) и моделью(model). Контроллер, основываясь на состоянии приложения и на произошедшем событии изменяет состояние model, обновляет view. Таким образом контроллер занимается обработкой событий и служит посредником между представлением и моделью. Он выясняет, что произошло, когда пользователь выполняет некое действие (например, щёлкает по кнопке или нажимает клавишу на клавиатуре). Логика клиентских приложений может быть реализована в контроллере. В более крупных системах, в которых нужно обрабатывать множество событий, этот элемент можно разбить на несколько модулей. Контроллер является входной точкой для событий и единственным посредником между представлением и данными.