# Front-end

JavaScript

# JS. debug

- debug

# JS. debug

## Отладка в браузере

**Отладка** – это процесс поиска и исправления ошибок в скрипте. Все современные браузеры и большинство других сред разработки поддерживают инструменты для отладки – специальный графический интерфейс, который сильно упрощает отладку. Он также позволяет по шагам отследить, что именно происходит в нашем коде.

Мы будем использовать браузер Chrome, так как у него достаточно возможностей, в большинстве других браузеров процесс будет схожим.

# JS. debug

## Панель «Исходный код» («Sources»)

Версия Chrome, установленная у вас, может выглядеть немного иначе, однако принципиальных отличий не будет.

Работая в Chrome, откройте тестовую страницу.

Включите инструменты разработчика, нажав F12 (Mac: Cmd+Opt+I).

Щёлкните по панели Sources («исходный код»).

# JS. debug

**Интерфейс состоит из трёх зон:**

- В зоне File Navigator (панель для навигации файлов) показаны файлы HTML, JavaScript, CSS, включая изображения, используемые на странице. Здесь также могут быть файлы различных расширений Chrome.
- Зона Code Editor (редактор кода) показывает исходный код.
- Зона JavaScript Debugging (панель отладки JavaScript) отведена для отладки.

# JS. debug

## Консоль

При нажатии на клавишу Esc в нижней части экрана вызывается консоль, где можно вводить команды и выполнять их клавишей Enter.

Результат выполнения инструкций сразу же отображается в консоли.

# JS. debug

## Точки останова (breakpoints)

Давайте разберёмся, как работает код нашей тестовой страницы. В файле *.js щёлкните на номере строки 4. Да-да, щёлкайте именно по самой цифре, не по коду.

Ура! Вы поставили точку останова. А теперь щёлкните по цифре 8 на восьмой линии.

# JS. debug

**Точка останова – это участок кода, где отладчик автоматически приостановит исполнение JavaScript.**

Пока исполнение поставлено «на паузу», мы можем просмотреть текущие значения переменных, выполнить команды в консоли, другими словами, выполнить отладку кода.

В правой части графического интерфейса мы видим список точек останова. А когда таких точек выставлено много, да ещё и в разных файлах, этот список поможет эффективно ими управлять:

Быстро перейдите к точке останова в коде (нажав на неё на правой панели).

Временно отключите точку останова, сняв с неё галочку.

Удалите точку останова, щёлкнув правой кнопкой мыши и выбрав Remove (Удалить).

…и так далее.

# JS. debug

## Команда debugger

Выполнение кода можно также приостановить с помощью команды debugger прямо изнутри самого кода:

```
function hello(name) {
  let phrase = `Привет, ${name}!`;

  debugger;  // <-- тут отладчик остановится

  say(phrase);
}
```

Такая команда сработает только если открыты инструменты разработки, иначе браузер ее проигнорирует.

# JS. debug

**Чтобы понять, что происходит в коде, щёлкните по стрелочкам справа:**

- Watch– показывает текущие значения для любых выражений.

Вы можете нажать на + и ввести выражение. Отладчик покажет его значение, автоматически пересчитывая его в процессе выполнения.

- Call Stack – показывает цепочку вложенных вызовов.

В текущий момент отладчик находится внутри вызова hello(), вызываемого скриптом в index.html (там нет функции, поэтому она называется “анонимной”).

Если вы нажмёте на элемент стека (например, «anonymous»), отладчик перейдёт к соответствующему коду, и нам представляется возможность его проанализировать.

- Scope показывает текущие переменные.

Local показывает локальные переменные функций, а их значения подсвечены прямо в исходном коде.

В Global перечисляются глобальные переменные (то есть вне каких-либо функций).

Там также есть ключевое слово this, которое мы ещё не изучали, но скоро изучим.

# JS. debug

## Пошаговое выполнение скрипта

А теперь давайте пошагаем по нашему скрипту.

Для этого есть кнопки в верхней части правой панели. Давайте рассмотрим их.

– «Resume»: продолжить выполнение, быстрая клавиша F8.

Возобновляет выполнение кода. Если больше нет точек останова, то выполнение просто продолжается, без контроля отладчиком.

# JS. debug

## Пошаговое выполнение скрипта

Выполнение кода возобновилось, дошло до другой точки останова внутри say(), и отладчик снова приостановил выполнение. Обратите внимание на пункт «Call stack» справа: в списке появился ещё один вызов. Сейчас мы внутри say().

– «Step»: выполнить следующую команду, быстрая клавиша F9.

Выполняет следующую инструкцию. Если мы нажмём на неё сейчас, появится alert.

Нажатие на эту кнопку снова и снова приведёт к пошаговому выполнению всех инструкций скрипта одного за другим.

– «Step over»: выполнить следующую команду, но не заходя внутрь функции, быстрая клавиша F10.

Работает аналогично предыдущей команде «Step», но ведёт себя по-другому, если следующая инструкция является вызовом функции (имеется ввиду: не встроенная, как alert, а объявленная нами функция).

Если сравнить, то команда «Step» переходит во вложенный вызов функцию и приостанавливает выполнение в первой строке, в то время как «Step over» выполняет вызов вложенной функции незаметно для нас, пропуская её внутренний код.

Затем выполнение приостанавливается сразу после вызова функции.

Это хорошо, если нам не интересно видеть, что происходит внутри вызова функции.

# JS. debug

## Пошаговое выполнение скрипта

«Step into», быстрая клавиша F11.

Это похоже на «Step», но ведёт себя по-другому в случае асинхронных вызовов функций. Если вы только начинаете изучать JavaScript, то можете не обращать внимания на разницу, так как у нас ещё нет асинхронных вызовов.

На будущее просто помните, что команда «Step» игнорирует асинхронные действия, такие как setTimeout (вызов функции по расписанию), которые выполняются позже. «Step into» входит в их код, ожидая их, если это необходимо. См. DevTools manual для получения более подробной информации.

– «Step out»: продолжить выполнение до завершения текущей функции, быстрая клавиша Shift+F11.

Продолжает выполнение и останавливает его в самой последней строке текущей функции. Это удобно, когда мы случайно вошли во вложенный вызов, используя , но это нас не интересует, и мы хотим продолжить его до конца как можно скорее.

# JS. debug

## Логирование

Чтобы вывести что-то на консоль из нашего кода, существует функция console.log.