



JAVASCRIPT

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми».

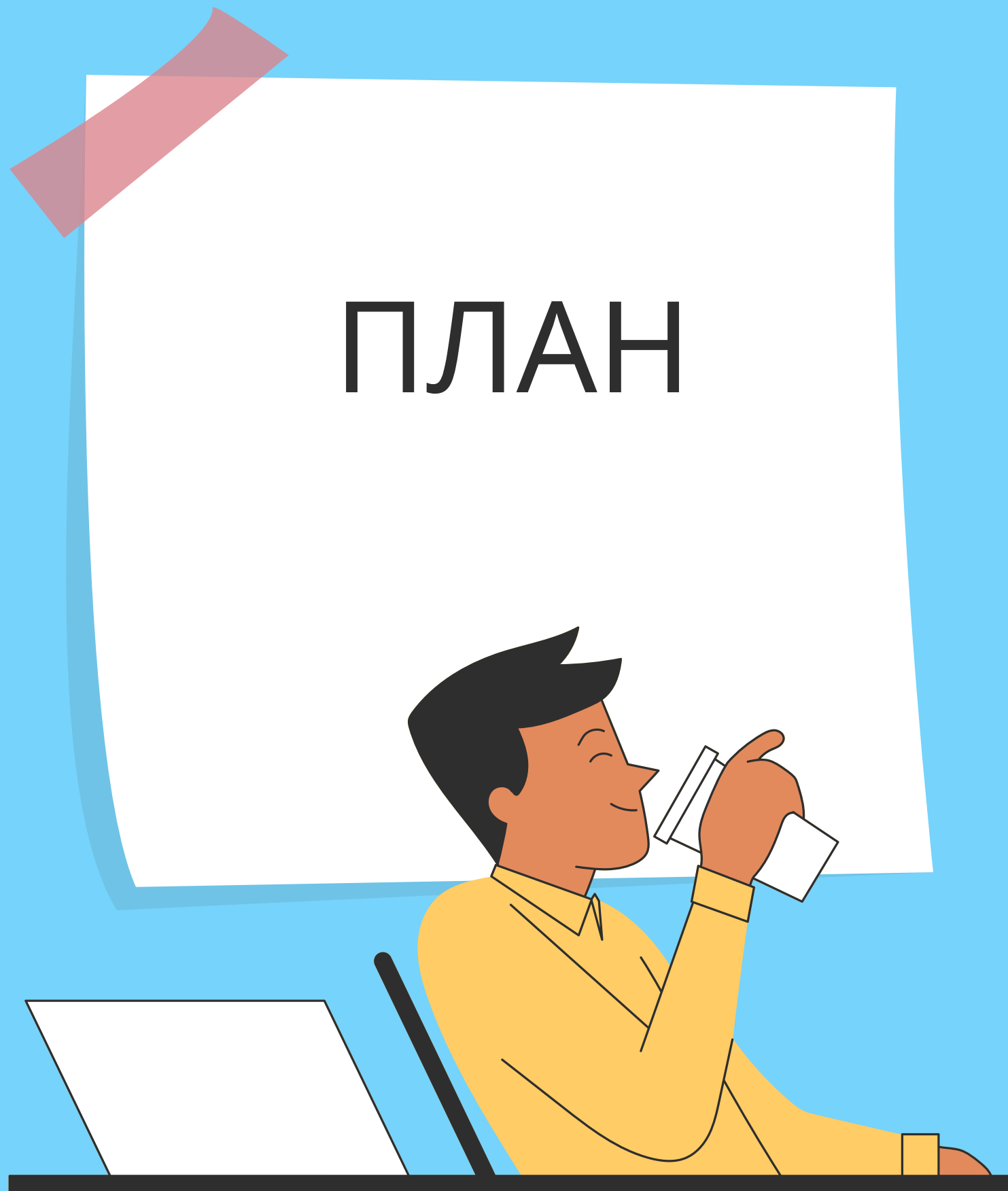
ПЛАН

1

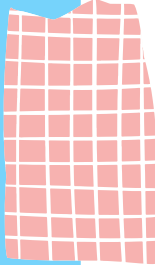
Введение в JavaScript.

2

Переменные.

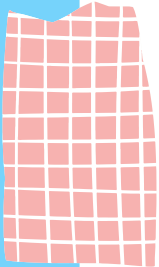


JAVASCRIPT

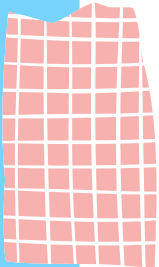


JavaScript - это язык программирования, который используется для создания интерактивных веб-страниц. Он позволяет вам манипулировать содержимым веб-страницы, обрабатывать события и создавать более сложное поведение, которое невозможно сделать только с помощью HTML и CSS.

Почему JavaScript?



Когда JavaScript создавался, у него было другое имя – «LiveScript». Однако, язык Java был очень популярен в то время, и было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно. Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся ECMAScript, и сейчас не имеет никакого отношения к Java.



Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Что может JavaScript в браузере?

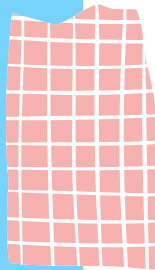
Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.js поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д. В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

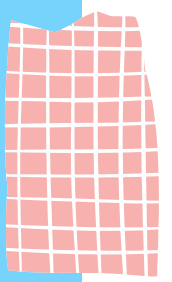
Например, в браузере JavaScript может:

- Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (AJAX).
- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

Чего НЕ может **JavaScript** в браузере?

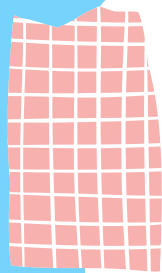


Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

- 
- JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.
 - Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега `<input>`.
 - Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в ФСБ.
 - Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).
 - Это называется «Политика одинакового источника» (Same Origin Policy). Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.
 - Это ограничение необходимо, опять же, для безопасности пользователя. Страница `https://anysite.com`, которую открыл пользователь, не должна иметь доступ к другой вкладке браузера с URL `https://gmail.com` и воровать информацию оттуда.
 - JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

ПОДКЛЮЧЕНИЕ JS

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
<script>
  console.log('Hello World!')
</script>
</body>
</html>
```

Для того чтобы подключить внешний JavaScript файл к HTML документу, нужно использовать тег `<script>` с атрибутом `src`, который указывает путь к JavaScript файлу.

```
index.html x index.js x
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport"
6      content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7    <meta http-equiv="X-UA-Compatible" content="ie=edge">
8    <title>Document</title>
9  </head>
10 <body>
11   <script src="index.js"></script>
12 </body>
13 </html>
```

В этом примере, тег `<script>` размещается в конце тела HTML документа, перед закрывающим тегом `</body>`. Это делается для того, чтобы сначала загрузить все элементы на странице, и только потом начать выполнение JavaScript кода. Это обеспечивает более быструю загрузку страницы и предотвращает возможные ошибки, которые могут возникнуть, если JavaScript попытается взаимодействовать с элементами, которые еще не загружены.



Также можно добавить атрибут `async` или `defer` к тегу `<script>`, чтобы управлять порядком загрузки и выполнения скриптов.

- `async` делает так, что скрипт загружается асинхронно в фоновом режиме и выполняется, как только загрузка завершена, без ожидания полной загрузки страницы.
- `defer` гарантирует, что скрипт, хоть и загружается асинхронно, будет выполнен только после того, как страница полностью загрузится, и в том порядке, в котором скрипты появляются в документе.

```
</head>
<body>
  <script src="index.js" async></script>
</body>
</html>
```

```
</head>
<body>
  <script src="index.js" defer></script>
</body>
</html>
```

По умолчанию, когда вы используете тег `<script>` без атрибутов `async` или `defer`, браузер будет загружать и выполнять JavaScript синхронно. Это означает, что процесс рендеринга веб-страницы приостановится, пока скрипт не будет загружен и выполнен. Это может приводить к заметным задержкам в загрузке страницы, особенно если скрипт большой или загружается с медленного сервера.

Давайте разберёмся подробнее:

- Без `async` и `defer`: Скрипт загружается синхронно и выполнение скрипта блокирует рендеринг страницы. Если у вас есть несколько скриптов, они будут загружены и выполнены по порядку, один за другим.
- С `async`: Скрипт загружается асинхронно в фоновом режиме. Это означает, что страница продолжает рендеринг, пока скрипт загружается. Как только скрипт загружен, выполнение страницы приостанавливается, и начинается выполнение скрипта.
- С `defer`: Скрипт также загружается асинхронно, но выполнение скрипта откладывается до тех пор, пока вся страница не будет загружена. Это полезно, когда скрипту нужен доступ к элементам DOM, и вы хотите быть уверены, что весь DOM загружен перед выполнением скрипта.

В современной веб-разработке часто рекомендуется использовать `async` или `defer`, чтобы улучшить производительность и ускорить загрузку страниц. Однако выбор между этими атрибутами зависит от конкретных требований вашего скрипта и того, как он взаимодействует с элементами на странице.

Структура кода

Начнём изучение языка с рассмотрения основных «строительных блоков» кода.

1. Выражения (Expressions)
2. Инструкции (Statements):

Основное различие между выражениями и инструкциями состоит в том, что выражения всегда вычисляются в значение, тогда как инструкции выполняют действия и не всегда имеют значение, которое можно присвоить или возвращать.

Также стоит отметить, что выражения могут быть частью инструкций. Например, в инструкции

`let x = 5 + 10;`

`5 + 10` является выражением,

а `let x = 5 + 10;` в целом является инструкцией.

Выражения (Expressions):

Выражение — это любой код, который вычисляется в единичное значение. В выражениях могут быть использованы литералы, операторы, переменные и вызовы функций. Вот несколько примеров выражений:

```
5 + 10 // выражение, которое вычисляется в 15  
  
"Hello" + " World" // выражение, которое вычисляется в "Hello World"  
  
console.log(5, 10) // выражение, которое вычисляется в результат вызова функции
```

Инструкции (Statements):

Инструкция — это команда, которая выполняет некоторое действие. Инструкции не обязательно должны вычисляться в единичное значение. Вот несколько примеров инструкций:

```
let x = 10;           // инструкция объявления переменной

if (x > 5) {           // инструкция условия (if statement)
  console.log(x);
}

for (let i = 0; i < 10; i++) { // инструкция цикла (for loop)
  console.log(i);
}
```

Комментарии

Со временем программы становятся всё сложнее и сложнее. Возникает необходимость добавлять комментарии, которые бы описывали, что делает код и почему.

Комментарии могут находиться в любом месте скрипта. Они не влияют на его выполнение, поскольку движок просто игнорирует их.

Однострочные комментарии начинаются с двойной косой чертой `//`.

Часть строки после `//` считается комментарием.

Такой комментарий может как занимать строку целиком, так и находиться после инструкции.

```
// Этот комментарий занимает всю строку
alert('Привет');

alert('Мир'); // Этот комментарий следует за инструкцией
```

Переменные

Переменные в программировании можно рассматривать как контейнеры для хранения данных. В JavaScript вы можете создать переменную и сохранить в ней значение, которое затем можно будет использовать или изменить.

В JavaScript есть три способа объявления переменных: с использованием ключевых слов `var`, `let` и `const`.

1. **var**: Это старый способ объявления переменных, который был доступен до ES6 (ECMAScript 2015). Переменные, объявленные с помощью **var**, имеют функциональную область видимости и подвержены "всплытию" (hoisting).

```
var name = "Alice";
```

1. **let**: Это новый способ объявления переменных, который был введен в ES6. В отличие от `var`, `let` имеет блочную область видимости. Это означает, что переменная существует только внутри блока кода, в котором она была объявлена.

```
let age = 25;
```

1. **const**: Также было введено в ES6. Переменные, объявленные с использованием `const`, имеют блочную область видимости, как и `let`. Основное отличие в том, что переменные, объявленные с помощью `const`, не могут быть изменены после их инициализации. Они являются константами.

```
const pi = 3.14;
```

Присваивание значений переменным

1. Когда вы объявляете переменную, вы можете присвоить ей значение сразу же или позже. Вы также можете изменить значение переменной позже, если она была объявлена с использованием `var` или `let`.

```
let a;      // Объявление переменной без присвоения значения
a = 10;     // Присвоение значения переменной

let b = 20; // Объявление переменной с присвоением значения

// Изменение значения переменной
b = 30;
```

Именованние переменных

При именовании переменных важно следовать некоторым правилам:

- Имена переменных могут содержать буквы, цифры, символы подчеркивания (`_`) и знаки доллара (`$`).
- Имена переменных должны начинаться с буквы, символа подчеркивания или знака доллара (не цифры).
- Имена переменных чувствительны к регистру (т.е., `age` и `Age` - это две разные переменные).
- Не следует использовать зарезервированные слова JavaScript в качестве имен переменных (например, `let`, `var`, `const`, `function` и так далее).

- Существуют также соглашения об именовании переменных, которые не являются строгими правилами, но широко приняты сообществом разработчиков. Например, предпочтение отдается camelCase стилю именования, когда первое слово начинается с маленькой буквы, а каждое последующее слово начинается с большой буквы:

```
let myVariable = "This is camelCase style";
```



**THANK
YOU!**

Have a
great day
ahead.