Тема 1 Вступ

Про мову JavaScript і робоче середовище для розробки.

Вступ до JavaScript

Давайте розглянемо, що такого особливого в JavaScript, чого ми можемо досягти за її допомогою та які ще технології пов'язані з нею.

Що таке JavaScript?

JavaScript було створено для того, щоб "оживити вебсторінки".

Програми цією мовою називаються *скриптами*. Їх можна писати прямо на сторінці в коді HTML і вони автоматично виконуватимуться при завантаженні сторінки.

Скрипти надаються та виконуються як простий текст. Для запуску їм не потрібна спеціальна підготовка чи компілятор.

У цьому плані JavaScript дуже відрізняється від іншої мови програмування — Java .

> **Чому цю мову називають JavaScript?**
>
> Коли мову JavaScript було створено, спочатку вона мала іншу назву: "LiveScript". Але тоді була дуже популярна мова програмування Java, тому було вирішено, що позиціонування нової мови як "молодшої сестри" Java допоможе в її популяризації.
>
> Але згодом JavaScript стала повністю незалежною мовою програмування зі своєю специфікацією ECMAScript і зараз не має нічого спільного з Java.

Сьогодні JavaScript може виконуватися не тільки в браузері, але й на сервері або на будь-якому пристрої, який має спеціальну програму — рушій JavaScript .

Браузер має вбудований рушій, який інколи називають "віртуальною машиною JavaScript".

Різні рушії мають різні "кодові назви". Наприклад:

- V8 – в Chrome, Opera та Edge.
- SpiderMonkey – в Firefox.
- …Є також інші кодові назви як "Chakra" для IE, "JavaScriptCore", "Nitro" і "SquirrelFish" для Safari, та інші.

Терміни вище добре було б запам'ятати, оскільки вони використовуються в статтях розробників на просторах інтернету. Ми також будемо їх використовувати. Наприклад, якщо "можливість X підтримується в V8", то, імовірно, вона працюватиме в Chrome, Opera та Edge.

Як рушії працюють?

Рушії складні. Але принцип роботи простий.

1. Рушій (вбудований, якщо це браузер) читає (“розбирає”) скрипт.
2. Потім він перетворює (“компілює”) скрипт у машинний код.
3. І потім цей машинний код виконується, причому досить швидко.

Рушій застосовує оптимізації на кожному етапі процесу. Він навіть слідкує за скомпільованим скриптом під час його виконання, аналізуючи дані, що проходять через нього, і оптимізує машинний код, зважаючи на ці знання.

Що може вбудований у браузер JavaScript?

Сучасний JavaScript – це “безпечна” мова програмування. Вона не надає низькорівневого доступу до пам’яті чи процесора, оскільки була створена для браузерів, які цього не потребують.

Можливості JavaScript значно залежать від середовища, у якому виконується скрипт. Наприклад, Node.js підтримує функції, які дозволяють JavaScript читати/записувати довільні файли, здійснювати мережеві запити тощо.

Вбудована в браузер JavaScript може робити все, що пов’язано з управлінням вебсторінками, взаємодією з користувачем та вебсервером.

Наприклад, JavaScript може:

- Додавати новий HTML-код на сторінку, змінювати наявний вміст, змінювати стилі.
- Реагувати на дії користувача, опрацьовувати натискання миші, переміщення вказівника, натискання на клавіші клавіатури.
- Відправляти запити через мережу до віддалених серверів, завантажувати та відвантажувати файли (так звані технології AJAX і COMET).
- Отримувати і надсилати куки , ставити запитання відвідувачам, показувати повідомлення.
- Запам’ятовувати дані на стороні клієнта (“local storage ”), які будуть доступні в майбутніх сесіях на цьому вебсайті.

Що НЕ може JavaScript?

Можливості JavaScript у браузері обмежені для безпеки користувача. Мета полягає в тому, щоб небезпечні вебсторінки не мали доступу до приватної інформації та не могли пошкодити інформацію на комп’ютері користувача.

Приклади таких обмежень:

- JavaScript на вебсторінці не може читати/записувати довільні файли на жорсткому диску, копіювати їх чи виконувати програми. Скрипт не має прямого доступу до функцій ОС.

  Сучасні браузери дозволяють працювати з файлами, але доступ до них обмежений і надається тільки тоді, коли користувач виконав відповідні дії, наприклад, перетягнув файл у вікно браузера чи вибрав його через тег `<input>`.

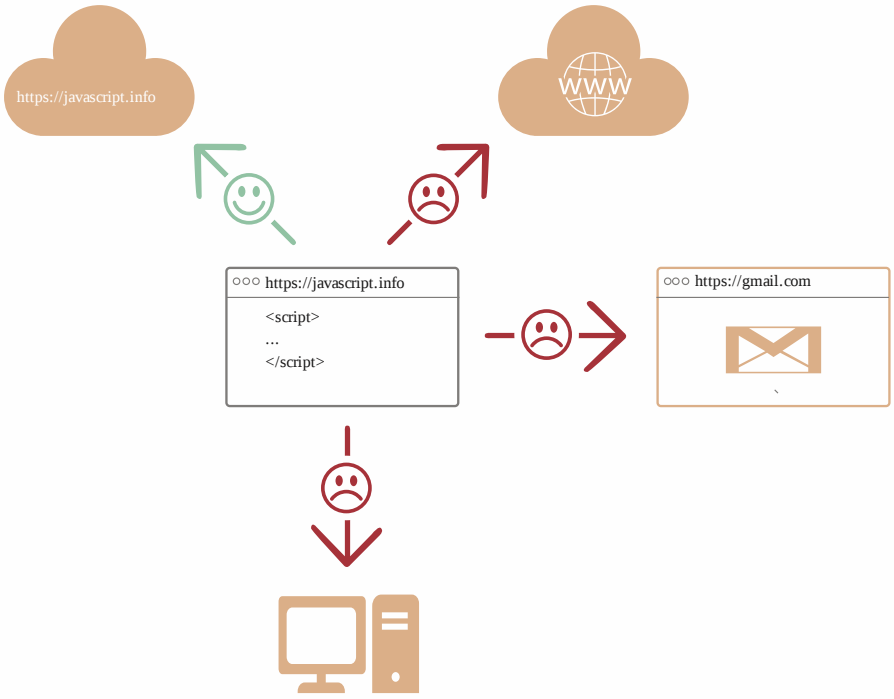
Є шляхи взаємодії з камерою/мікрофоном та іншими пристроями, але для цього потрібен явний дозвіл користувача. Тому сторінка з JavaScript не може нишком увімкнути веб-камеру, спостерігати за оточенням і відсилати інформацію до СБУ .

- Різні вкладки/вікна зазвичай не знають одне про одного. Іноді це можливо, наприклад, коли одне вікно використовує JavaScript, щоб відкрити інше. Але навіть у такому випадку JavaScript з однієї сторінки не має доступу до іншої, якщо вони з різних сайтів (мають різні домени, протоколи чи порти).

  Це називається “Політикою того ж походження (Same Origin Policy) ”. Щоб обійти це обмеження, *обидві сторінки* мають погодитися на обмін даними та містити спеціальний JavaScript-код, який здійснюватиме це. Ми розглянемо таку тему в посібнику.

  Знову-таки, це обмеження існує задля безпеки користувача. Сторінка за адресою `http://anysite.com`, яку відкрив користувач, не повинна мати доступ до іншої вкладки браузера з URL-адресою `http://gmail.com` і викрадати звідти інформацію.

- JavaScript може легко спілкуватися через мережу з сервером, від якого отримана поточна сторінка. Але здатність скрипта отримувати дані з інших сайтів/доменів обмежена. Такі запити можливі, але потребують спеціальної згоди (вираженої в HTTP-заголовках) від віддаленого сервера. Це також зроблено з метою безпеки.



Таких обмежень немає, якщо JavaScript використовується за межами браузера, наприклад, на сервері. Сучасні браузери дозволяють установлювати плаґіни/розширення, які мають розширені можливості, проте вимагають розширених прав.

Що робить JavaScript унікальною?

Є принаймні *три* чудові особливості JavaScript:

- Цілковита інтеграція з HTML/CSS.

- Прості речі робляться просто.
- Підтримується всіма сучасними браузерами й увімкнена усталено.

JavaScript – це єдина браузерна технологія, яка суміщає ці три речі.

Це й робить її унікальною. Ось чому JavaScript – найбільш поширений засіб створення браузерних інтерфейсів.

До слова, JavaScript також дозволяє створювати сервери, мобільні застосунки тощо.

Мови “над” JavaScript

Синтаксис JavaScript не задовольняє потреби кожного. Різні люди хочуть різних функцій.

Цього слід очікувати, тому що проєкти та вимоги різні для кожного.

Останнім часом з’явилося безліч нових мов, які *транспілюються* (конвертуються) в JavaScript перед виконанням у браузері.

Сучасні інструменти роблять транспіляцію дуже швидкою та прозорою, дозволяючи розробникам писати код іншою мовою й автоматично конвертувати його “під капотом”.

Приклади таких мов:

- CoffeeScript — це “синтаксичний цукор” для JavaScript. Вона вводить коротший синтаксис, дозволяючи нам писати більш чіткий і точний код. Зазвичай це до вподоби програмістам на Ruby.
- TypeScript зосереджена на додаванні “строгої типізації даних” для спрощення розробки та підтримки складних систем. Розробляється Microsoft.
- Flow також додає типізацію даних, але іншим способом. Розробляється Facebook.
- Dart – це автономна мова, яка має власний рушій, що працює в небраузерних середовищах (як-от мобільні застосунки), але також може транспілюватися в JavaScript. Розробляється Google.
- Brython – це транспілятор коду Python у JavaScript, що дозволяє писати застосунки на чистому Python без використання JavaScript.
- Kotlin — це сучасна, лаконічна та безпечна мова програмування, яку можна компілювати для браузера або NodeJS.

Існують й інші мови. Звісно, навіть якщо ми використовуємо одну з цих транспілюючих мов, ми також повинні знати JavaScript, щоб дійсно розуміти, що робимо.

Підсумки

- Мова JavaScript спочатку була створена лише як мова для браузера, але зараз її також використовують в інших середовищах.
- Сьогодні JavaScript позиціонується як найбільш поширена мова для браузера, яка повністю інтегрована з HTML/CSS.
- Існує багато мов, які “транспілюються” в JavaScript і надають певні функції. Рекомендується переглянути їх, хоча б мигцем, після освоєння JavaScript.

Довідники й специфікації

Цей сайт – *підручник*. Він спрямований на те, щоб допомогти вам поступово вивчити мову. Проте, як тільки ви познайомитеся з основами, вам знадобляться й інші джерела.

Специфікація

Специфікація ECMA-262 містить найглибшу, найдетальнішу й найбільш формалізовану інформацію про JavaScript. Фактично, ця специфікація визначає мову.

Але саме через формалізований стиль її важко зрозуміти з першого разу. Тому, якщо вам потрібне найнадійніше джерело інформації про деталі мови, специфікація – правильне місце. Однак, це джерело не для повсякденного використання.

Щороку випускається нова версія специфікації. Між цими випусками, остання “чернетка” доступна на сайті https://tc39.es/ecma262/ .

Щоб прочитати про найновіші функції, включно з тими, які “майже входять в стандарт” (так звана “стадія 3”), перегляньте пропозиції на https://github.com/tc39/proposals .

Також, якщо ви розробляєте для браузерів, вам буде цікаво прочитати про інші специфікації, які описано в другій частині цього підручника.

Довідники

- **MDN (Mozilla) JavaScript Reference** – це головний довідник з прикладами та іншою інформацією. Він чудово підходить для детального вивчення окремих функцій, методів тощо.

  Його можна знайти за цим посиланням https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference .

Хоча, замість пошуку на сайті, краще використовувати пошукові системи. Просто напишіть “MDN [термін]” в пошуковому запиті. Наприклад, запит “MDN parseInt ” знайде інформацію про функцію `parseInt`.

Таблиці сумісності

Мова JavaScript активно розвивається – до неї регулярно додаються нові функції.

Щоб дізнатися, чи підтримує браузер або інший рушій певну можливість JavaScript, дивіться на сайтах:

- https://caniuse.com/ – для кожної технології приведено таблицю сумісності з усіма браузерами; тобто, щоб побачити, які браузери підтримують сучасні криптографічні функції, слід ввести в пошуку “Cryptography ”.
- https://kangax.github.io/compat-table – таблиця з усіма можливостями мови та рушіями, які підтримують або не підтримують відповідні технології.

Всі ці ресурси корисні в повсякденній розробці, тому що вони містять корисну інформацію про деталі мови, їхню підтримку тощо.

Будь ласка, збережіть собі ці сайти (або цю сторінку); вони вам знадобляться, якщо буде потреба детальніше розібратися в конкретному функціоналі мови.

Редактори коду

Редактор коду – це місце, де програмісти проводять найбільше часу.

Є два основні види редакторів коду: IDE і легкі редактори. Багато людей використовують декілька таких редакторів для різних потреб.

IDE

Термін IDE (Інтегроване середовище розробки) означає потужний редактор з багатьма можливостями, що зазвичай працює з “цілим проектом”. Як зрозуміло з назви, це не тільки редактор коду, а повноцінне “середовище розробки”.

IDE завантажує проект (який може мати багато файлів), дозволяє переключатися між файлами, надає можливість автозаповнення, яке базується на цілому проекті (не лише на відкритому файлі), інтегрується із системою контролю версій (наприклад, git), надає можливість розгортання вашого проекту на тестове середовище та багато інших функцій “на рівні проекту”.

Якщо Ви досі не вибрали IDE, розгляньте наступні варіанти:

- Visual Studio Code (багатоплатформний, безкоштовний).
- WebStorm (багатоплатформний, платний).

Для Windows, також може бути “Visual Studio”, не плутайте з “Visual Studio Code”. “Visual Studio” – потужний платний редактор, який працює лише на Windows, добре підходить для програмування на платформі .NET. Також хороший для програмування на JavaScript. Також існує його безкоштовна версія: Visual Studio Community.

Багато IDE платні, проте мають пробний період. Їхня вартість зазвичай незначна в порівнянні із зарплатою кваліфікованого розробника. Правильний вибір редактора дозволить зберегти найцінніший ресурс – ваш час. Тому просто виберіть найкращий варіант, який задовольнятиме усім вашим потребам.

Легкі редактори

“Легкі редактори” не такі потужні, як IDE, проте вони прості, доступні і швидко запускаються.

Їх зазвичай використовують, щоб швидко відкрити і відредагувати один або декілька файлів.

Головна їхня відмінність від IDE в тому, що IDE працює на рівні проекту, тому він завантажує набагато більше даних під час запуску, і якщо потрібно, аналізує його структуру. Легкий редактор набагато швидший, якщо нам необхідно відредагували лише один файл.

На практиці, легкі редактори можуть мати багато плаґінів, включаючи аналізатори синтаксису на рівні проекту, автозаповнення і т. д. Через те, що це значно розширює їх можливості, немає чіткої межі між легкими редакторами та IDE.

Ось ці варіанти заслуговують вашої уваги:

- Sublime Text (багатоплатформний, безкоштовний на час випробувального терміну).
- Notepad++ (Windows, безкоштовний).
- Vim та Emacs також хороші, якщо знати, як ними користуватися.

Не будемо сперечатися

Я, та мої хороші друзі-розробники, вже давно користуємося цими редакторами, і вони цілком задовольняють усім нашим потребам.

У нашому великому світі є й інші редактори. Будь ласка, приділіть трохи часу на перегляд декількох редакторів, і виберіть той, який вам найбільш до вподоби.

Вибір редактора, як і будь-якого іншого інструменту, індивідуальний, і залежить від ваших проєктів, звичок і персональних вподобань.

Інструменти розробника

Будь-який код схильний до помилок. Швидше за все, ви будете робити помилки… Хоча, про що я говорю? Ви *точно* будете робити помилки, принаймні, якщо ви людина, а не робот .

Зазвичай, користувачі не бачать помилок у браузері. Тому, якщо в скрипті трапиться щось хибне, ми не побачимо помилки і не зможемо її виправити.

Щоб побачити помилки і отримати додаткову інформацію про виконання скриптів, було створено і вбудовано в браузери “інструменти розробника”.

Більшість розробників надають перевагу Chrome чи Firefox, тому що ці браузери мають найкращі інструменти розробника. Інші браузери теж мають інструменти розробника, деколи навіть зі спеціальними функціями, проте вони не такі популярні, як Chrome чи Firefox. Тому більшість розробників мають “улюблений” браузер і переключаються на інші, якщо проблема специфічна для браузера.

Інструменти розробника потужні; вони мають багато функцій. Для початку, ми вивчимо, як їх відкрити, як переглядати помилки і як виконувати команди JavaScript.

Google Chrome

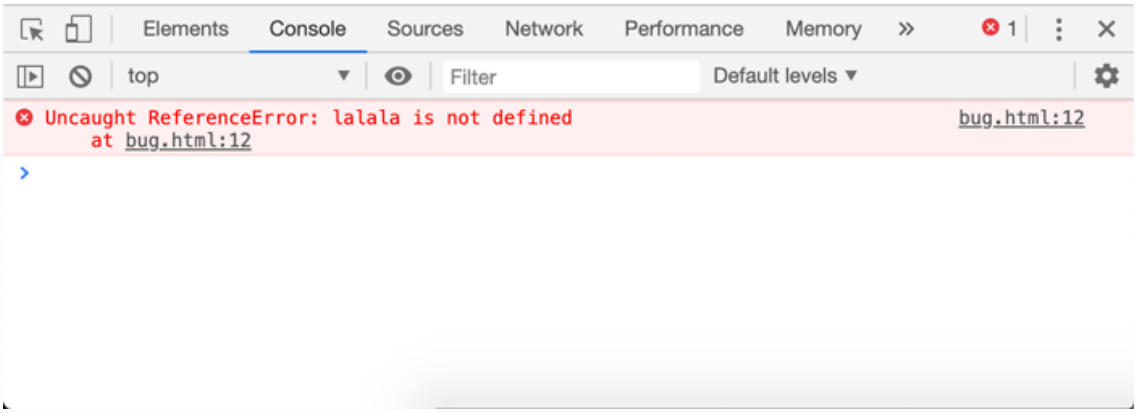
Для прикладів ми будемо використовувати браузер Google Chrome . Інструменти розробника в ньому показуються лише англійською мовою, незалежно від налаштувань браузера.

Відкрийте сторінку bug.html. На ній є помилка в коді JavaScript. Вона прихована для звичайних користувачів, тому потрібно відкрити інструменти розробника, щоб її побачити.

Натисніть клавішу `F12` або, якщо у вас Mac, комбінацію клавіш `Cmd+Opt+J`.

Інструменти розробника типово відкриваються на вкладці “Console” (консоль).

Ось так відображається помилка в консолі:



Точний вигляд інструментів розробника може відрізнятися в залежності від вашої версії Chrome. Вони міняються час від часу, але в основному це вікно повинно бути схожим.

- Тут ми можемо побачити червоне повідомлення про помилку. У нашому випадку, скрипт має невизначену команду “lalala”.
- З правого боку є посилання на джерело `bug.html:12` з номером рядка, де ця помилка виникла. При натисканні на це посилання, вас перенаправить на вкладку “Sources” (файли з кодом сторінки), де відкриється файл і перейде на рядок, в якому трапилася помилка.

Нижче повідомлення про помилку є синій символ `>`. Цей символ позначає “командний рядок”, де ми можемо вводити команди JavaScript. Натисніть `Enter`, щоб їх виконати.

Тепер ми бачимо помилки, цього достатньо, щоб почати. Ми пізніше повернемося до інструментів розробника, щоб розглянути налагодження коду у розділі Налагодження в браузері.

Введення декількох рядків

Зазвичай, коли ми вводимо один рядок коду в консоль і натискаємо `Enter`, він виконується.

Щоб ввести декілька рядків коду, натисніть `Shift+Enter`. Таким чином можна вводити і виконувати довгі фрагменти JavaScript коду.

Firefox, Edge та інші

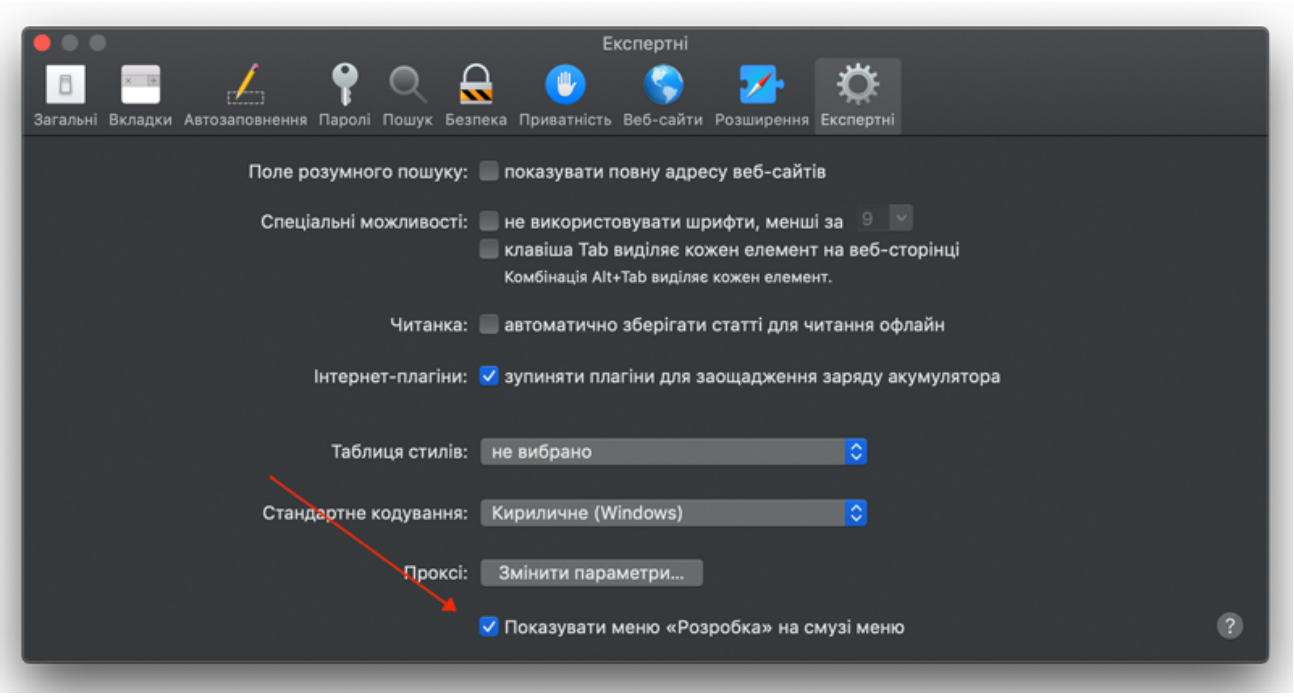
Більшість браузерів використовують клавішу `F12`, щоб відкрити консоль розробника.

Їх вигляд зазвичай схожий. Якщо ви навчитеся використовувати однин з них (можете почати з Chrome), ви зможете легко переключитися на інший браузер.

Safari

Safari (стандартний браузер у macOS, не підтримується Windows/Linux) має свої нюанси. Спочатку нам потрібно увімкнути меню “Розробка”.

Відкрийте Параметри і перейдіть на панель “Експертні”. Знизу буде галочка, яку необхідно вибрати:



Тепер комбінація клавіш `Cmd+Opt+C` може переключати консоль. Також зауважте, що з'явився новий пункт "Розробка" у верхньому меню. Це меню має багато команд та опцій.

Підсумки

- Інструменти розробника дозволяють нам переглядати помилки, виконувати команди, досліджувати змінні та багато іншого.
- Їх можна відкрити клавішою `F12` для більшості браузерів в Windows. В Chrome для Mac потрібно натиснути комбінацію клавіш `Cmd+Opt+J`, в Safari: `Cmd+Opt+C` (але спочатку інструменти потрібно увімкнути).

Тепер у нас є готове середовище. В наступній темі ми приступимо до самогоJavaScript.

Основи JavaScript

У цьому розділі розглядаються основні поняття в JavaScript.

Привіт, світ!

Ця частина посібника розповідає про основи JavaScript, про саму мову.

Але нам необхідне робоче середовище для запуску наших скриптів і, оскільки ця книга є онлайн, браузер є гарним вибором для цього. Ми зведемо до мінімуму кількість специфічних команд для браузера (наприклад `alert`), щоб ви не витрачали час на них, якщо плануєте зосередитися на іншому середовищі (як Node.js). Ми зосередимось наJavaScript в браузері у наступній темі.

Насамперед, давайте подивимось, як додавати скрипти до сторінки. Для серверних середовищ (як Node.js) ви можете виконати скрипти за допомогою команди `"node my.js"`.

Тег “script”

Програми JavaScript можна вставити у будь-яку частину HTML документа, використовуючи тег `<script>`.

Наприклад:

```
<!DOCTYPE HTML>
<html>

<body>

  <p>Текст перед скриптом...</p>

  <script>
    alert( 'Привіт, світ!' );
  </script>

  <p>...Після скрипта.</p>

</body>

</html>
```

Тег `<script>` містить код JavaScript, який автоматично виконується, коли браузер обробляє тег.

Сучасна розмітка

Тег `<script>` має декілька атрибутів, які рідко використовуються сьогодні, але їх ще можна знайти в старому коді.

Атрибут `type`: `<script type=…>`

Старий стандарт HTML, HTML4, вимагав, щоб у `<script>` був атрибут `type`. Зазвичай це був `type="text/javascript"`. Зараз в ньому немає необхідності — сучасний стандарт HTML повністю змінив зміст цього атрибута. Тепер його можна використовувати для модулів JavaScript. Але це просунута тема, ми поговоримо про модулі в іншій частині посібника.

Атрибут `language`: `<script language=…>`

Цей атрибут вказував на мову скрипта. Цей атрибут більше немає сенсу, оскільки JavaScript є усталеною мовою. Його більше не потрібно використовувати.

Коментарі до та після скриптів.

У дійсно старих посібниках і книгах ви можете знайти коментарі всередині тега `<script>`, наприклад:

```
<script type="text/javascript"><!--
    ...
//--></script>
```

Такий спосіб не використовується у сучасному JavaScript. Ці коментарі ховали JavaScript для старих браузерів, які не знали, як обробляти тег `<script>`. Ті браузери, які були випущені за останні 15 років, не мають цієї проблеми. Такий вид коментарів означає, що це дійсно старий код.

Зовнішні скрипти

Якщо ми маємо багато коду на JavaScript, ми можемо розділити його на окремі файли.

Файл скрипта можна додати до HTML за допомогою атрибута `src`:

```
<script src="/path/to/script.js"></script>
```

Тут `/path/to/script.js` — абсолютний шлях до файлу скрипта з кореня сайту. Також можна вказати відносний шлях з поточної сторінки. Наприклад, `src="script.js"`, так само як `src="./script.js"`, означатиме, що файл `"script.js"` у поточній директорії.

Ми також можемо вказати повну URL-адресу. Наприклад:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js"></script>
```

Щоб додати кілька скриптів, використовуйте кілька тегів:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
…
```

Будь ласка, зверніть увагу:

Як правило, тільки найпростіші скрипти поміщаються безпосередньо в HTML. Більш складні знаходяться в окремих файлах.

Перевага окремого файлу полягає в тому, що браузер завантажує його та зберігає у своєму кеші .

Інші сторінки, які посилаються на один і той самий скрипт, замість того, щоб завантажувати повторно, будуть брати його з кешу, тому файл буде завантажено лише один раз.

Це зменшує трафік і робить сторінки швидшими.

Якщо вказаний `src`, вміст скрипта ігнорується.

Один тег `<script>` не може мати атрибут `src` і код всередині.

Це не спрацює:

```
<script src="file.js">
  alert(1); // вміст ігнорується, оскільки задано src
</script>
```

Ми повинні вибрати або зовнішній `<script src="…">`, або звичайний `<script>` з кодом.

Наведений вище приклад можна розділити на два скрипти:

```
<script src="file.js"></script>
<script>
  alert(1);
</script>
```

Підсумки

- Ми можемо використовувати тег `<script>` для додавання коду JavaScript на сторінку.
- Атрибути `type` і `language` не потрібні.
- Скрипти у зовнішньому файлі можна вставити за допомогою `<script src="path/to/script.js"></script>`.

Існує набагато більше інформації про браузерні скрипти та їхню взаємодію з веб-сторінкою. Але майте на увазі, що ця частина посібника присвячена мові JavaScript, тому ми не повинні відволікатись на деталі реалізації в браузері. Ми будемо використовувати браузер як спосіб запуску JavaScript, що є дуже зручним для читання в Інтернеті, але це лише один із багатьох можливих варіантів.

Завдання

Challenge #1 Показати сповіщення

Створіть сторінку, яка показуватиме повідомлення “Я – JavaScript!”.

Зробіть це в plunker або на жорсткому дискові; немає значення, головне переконайтеся,що це працює.

Challenge #2 **Показати повідомлення із зовнішнього скрипта**

Використайте розв'язок з попереднього завдання Показати сповіщення. Змініть його так, щоб вміст скрипта знаходився в зовнішньому файлі `alert.js`, який буде знаходитися в тій самій папці.

Відкрийте сторінку і переконайтеся, що повідомлення показується.

Структура коду

Спочатку розглянемо "будівельні блоки" коду.

Інструкції

Інструкції – це синтаксичні конструкції та команди, які виконують дії.

Ми вже бачили інструкцію `alert('Привіт, світ!')`, яка показує повідомлення "Привіт, світ!".

Можна писати стільки інструкцій, скільки завгодно. Інструкції можна розділяти крапкою з комою.

Наприклад, тут ми розділити "Привіт, світ" на два виклики alert:

```
alert('Привіт'); alert('Світ');
```

Зазвичай кожну інструкцію пишуть з нового рядка, щоби код було легше читати:

```
alert('Привіт');
alert('Світ');
```

Крапка з комою

Здебільшого крапку з комою можна пропустити, якщо є перенесення на новий рядок.

Такий код буде працювати:

```
alert('Привіт')
alert('Світ')
```

У цьому разі JavaScript інтерпретує перенесення рядка як "неявну" крапку з комою. Це називається автоматичне вставлення крапки з комою .

Переважно новий рядок означає крапку з комою. Але “переважно” не означає “завжди”!

У деяких випадках новий рядок не означає крапку з комою. Наприклад:

```
alert(3 +
1
+ 2);
```

Код виведе `6`, тому що JavaScript не вставить тут крапку з комою. Інтуїтивно зрозуміло, що, якщо рядок закінчується плюсом `"+"`, то це “незакінчений вираз”, тому крапка з комою не потрібна. І в цьому випадку все працює як задумано.

Але є ситуації, коли JavaScript “забуває” вставити крапку з комою там, де це дійсно потрібно.

Помилки, які виникають у таких ситуаціях, досить важко виявити й виправити.

Приклад такої помилки

Якщо хочете побачити конкретний приклад такої помилки, зверніть увагу на цей код:

```
alert("Привіт");

[1, 2].forEach(alert);
```

Поки що не задумуйтеся, що означають квадратні дужки `[]` і `forEach`. Ми вивчимо їх пізніше. Зараз просто запам'ятайте результат виконання коду: спочатку виведеться `Привіт`, далі `1`, а потім `2`.

А тепер видалимо крапку з комою після першого `alert`:

```
alert("Привіт")

[1, 2].forEach(alert);
```

Різниця з кодом вище лише в одному символі: крапка з комою, яку ми видалити в кінці першого рядка.

Якщо ми запустимо цей код, виведеться лише `Привіт` (а потім виникне помилка, яку можна побачити в консолі). Числа більше не виводяться.

Це тому що JavaScript не вставляє крапку з комою перед квадратними дужками `[...]`. Оскільки крапка з комою автоматично не вставиться, код у цьому прикладі виконається як одна інструкція.

Ось як рушій побачить її:

```
alert("Привіт")[1, 2].forEach(alert);
```

Виглядає дивно, чи не так? У цьому випадку таке об'єднання неправильне. Щоби код правильно працював, нам потрібно поставити крапку з комою після `alert`.

Це може статися в інших випадках.

Ми рекомендуємо ставити крапку з комою між інструкціями, навіть якщо вони розділені новими рядками. Це правило широко використовується в спільноті розробників. Варто повторити ще раз – здебільшого *можна* пропускати крапки з комою. Але безпечніше – особливо для новачка – використовувати їх.

Коментарі

З часом програми стають все складнішими та складнішими. Виникає потреба додавати *коментарі*, які будуть описувати що робить код і чому.

Коментарі можна писати в будь-якому місці скрипту. Вони не впливають на його виконання, тому що рушій просто ігнорує коментарі.

Однорядкові коментарі починаються з подвійної косої риски `//`.

Частина рядка після `//` вважається коментарем. Такий коментар може займати весь рядок, або міститися після інструкції.

Ось так:

```
// Цей коментар займає весь рядок
alert('Привіт');

alert('Світ'); // Цей коментар міститься після інструкції
```

Багаторядкові коментарі починаються з косої риски з зірочкою `/*` і закінчується зірочкою з косою рискою `*/`.

Ось так:

```
/* Приклад із двома повідомленнями.
Це багаторядковий коментар.
*/
alert('Привіт');
alert('Світ');
```

Вміст коментаря ігнорується, тому, якщо ми напишемо всередині `/* … */` код, він не виконається.

Деколи виникає необхідність тимчасово вимкнути частину коду:

```
/* Закоментований код
alert('Привіт');
*/
alert('Світ');
```

Використовуйте комбінації клавіш!

У більшості редакторів рядок коду можна закоментувати, натиснувши комбінацію клавіш `Ctrl+/`, а щоби закоментувати декілька рядків – виділіть потрібні рядки та натисніть комбінацію клавіш `Ctrl+Shift+/`. У macOS потрібно натискати клавішу `Cmd` замість `Ctrl` і клавішу `Option` замість `Shift`.

Вкладені коментарі не підтримуються!

Не може бути `/*...*/` всередині `/*...*/`.

Такий код “помре” з помилкою:

```
/*
  /* вкладений коментар ?!? */
*/
alert( 'Світ' );
```

Будь ласка, не вагайтесь коментувати ваш код.

Коментарі збільшують розмір коду, але це не проблема. Є багато інструментів, які мініфікують код перед публікацією на робочий сервер. Вони видалять коментарі, тому їх не буде в робочих скриптах. Отже, коментарі жодним чином не впливають на робочий код.

Сучасний режим, "use strict"

Впродовж тривалого часу JavaScript розвивався без проблем із сумісністю. До мови додавалися нові функції, а стара функціональність залишалася незмінною.

Перевагою цього було те, що існуючий код не ламався. Проте, будь-яка помилка або неідеальне рішення назавжди ставали частиною JavaScript, тому що цей код не змінювався.

Так було до 2009 року, коли з'явився стандарт ECMAScript 5 (ES5). Він додав нові функції до мови і змінив деякі існуючі. Щоб старий код лишався робочим, більшість таких модифікацій усталено було вимкнено. Щоб увімкнути цей функціонал, потрібно прописати спеціальну директиву: `"use strict"`.

"use strict"

Директива виглядає як рядок: `"use strict"` чи `'use strict'` і дослівно перекладається як "використовувати суворий (режим)". Якщо вона прописана на початку скрипта, він буде виконуватися у "сучасному" режимі.

Наприклад:

```
"use strict";

// цей код працюватиме у сучасному режимі
...
```

Незабаром ми будемо вивчати функції (такий собі спосіб групування команд). Забігаючи наперед, маймо на увазі, що `"use strict"` можна писати на початку функції. Таким чином, суворий режим буде використовуватися лише в межах цієї функції. Проте зазвичай люди використовують цей режим для всього скрипта.

Переконайтеся, що “use strict” написано зверху

Будь ласка, завжди переконуйтеся в тому, що директива `"use strict"` написана зверху ваших скриптів, інакше суворий режим не увімкнеться.

Тут суворий режим не спрацює:

```
alert("деякий код");
// "use strict" нижче alert(), і тому ігнорується -- він повинен бути зверху

"use strict";

// суворий режим не активовано
```

Лише коментарі можуть бути вище `"use strict"`.

Неможливо скасувати `use strict`

Немає директиви на зразок `"no use strict"`, яка могла б вернути старий режим.

Як тільки ми увійшли в суворий режим, назад дороги немає.

Консоль браузера

Коли ви використовуєте консоль розробника для виконання коду, майте на увазі, що консоль усталено не використовує суворий режим.

В тих випадках, коли `use strict` впливає на роботу коду, ви отримаєте невірні результати в консолі.

Як тоді увімкнути `use strict` в консолі?

По-перше, можна використовувати `Shift+Enter`, щоб ввести декілька рядків, і на початку написати `use strict`, ось так:

```
'use strict'; <Shift+Enter для нового рядка>
//  ...ваш код
<натисніть Enter, щоб виконати>
```

Це працюватиме в більшості браузерів, зокрема в Firefox і Chrome.

Якщо не спрацює, наприклад, в старих браузерах, тоді найнадійнішим варіантом буде використати `use strict` всередині функції-обгортки (хоч це, звичайно, виглядатиме потворно). Ось так:

```
(function() {
  'use strict';

  // ...тут буде ваш код...
})()
```

Чи повинні ми використовувати суворий режим?

Питання може здатися очевидним, але це не так.

Одні можуть порекомендувати ставити `"use strict"` на початку скриптів… Але знаєте, що круто?

Сучасний JavaScript підтримує “класи” і “модулі” – просунуті структури мови (ми їх, звичайно, будемо вивчати), які автоматично вмикають `use strict`. Тому, якщо ми використовуємо ці структури, нам не потрібно прописувати директиву `"use strict"`.

Отож зараз бажано ставити `"use strict";` на початку скриптів. Але пізніше, коли наш код “дорослє” до класів і модулів, ми зможемо пропускати цю директиву.

Зараз ми знаємо про `use strict` в загальному.

У наступних темах, в процесі вивчення особливостей мови, ми помітимо відмінності міжсуворим і усталеним режимами. На щастя, їх не багато, і вони справді роблять наше життякращим.

Змінні

Найчастіше застосункам на JavaScript потрібно працювати з інформацією. Ось два приклади:

1. Онлайн-магазин – інформацією можуть бути товари, які продаються, і вміст кошика.
2. Застосунок для чату – інформація може включати користувачів, повідомлення та багато іншого.

Змінні використовуються для зберігання цієї інформації.

Змінна

Змінна це “іменована частинка сховища”, в якій зберігаються дані. Ми можемо використовувати змінні, щоб зберігати товари, відвідувачів та інші дані.

Щоб створити змінну, використовуйте ключове слово `let`.

Цей рядок нижче створить (інакше кажучи, *оголосить*) змінну з ім’ям “message”:

```
let message;
```

Тепер ми можемо покласти деякі дані в цю змінну, використовуючи оператор присвоєння `=`:

```
let message;

message = 'Привіт'; // збереження рядка 'Привіт' у змінній `message`
```

Тепер рядок збережено в частину пам’яті, яка зв’язана з цією змінною. Ми можемо отримати доступ до даних, використовуючи ім’я змінної:

```
let message;
message = 'Привіт!';

alert(message); // показує вміст змінної
```

Щоб писати менше коду, ми можемо суміщати оголошення змінної та її присвоєння в одному рядку:

```
let message = 'Привіт!'; // оголошення змінної і присвоєння значення

alert(message); // Привіт!
```

Ми також можемо оголосити декілька змінних в одному рядку:

```
let user = 'Іван', age = 25, message = 'Привіт';
```

Таке оголошення може виглядати коротшим, проте ми не рекомендуємо так писати. Заради кращої читабельності, будь ласка, оголошуйте змінні з нового рядка.

Багаторядковий спосіб трохи довший, проте його легше читати:

```
let user = 'Іван';
let age = 25;
let message = 'Привіт';
```

Деякі люди також оголошують змінні в такому багаторядковому стилі:

```
let user = 'Іван',
  age = 25,
  message = 'Привіт';
```

…або навіть у стилі “кома спочатку”:

```
let user = 'Іван'
  , age = 25
  , message = 'Привіт';
```

Технічно, всі ці способи роблять одне й те ж. Тому це питання особистого смаку та естетики.

`var` замість `let`

У старих скриптах ви можете знайти інше ключове слово: `var` замість `let`:

```
var message = 'Привіт';
```

Ключове слово `var` *майже* таке, як `let`. Воно теж оголошує змінну, але дещо іншим, “застарілим” способом.

Є деякі відмінності між `let` і `var`, але вони поки що не мають для нас значення. Ми дізнаємося більше про ці відмінності в розділі Застаріле ключове слово "var".

Аналогія з життя

Ми легко зрозуміємо концепцію “змінної”, якщо уявимо її у вигляді “коробки” для даних з унікальною назвою на наклейці.

Наприклад, змінну `message` можна уявити як коробку з підписом `"Повідомлення"` зі значенням `"Привіт!"` всередині:

Ми можемо покласти будь-яке значення в цю коробку.

Ми також можемо змінювати його стільки разів, скільки захочемо:

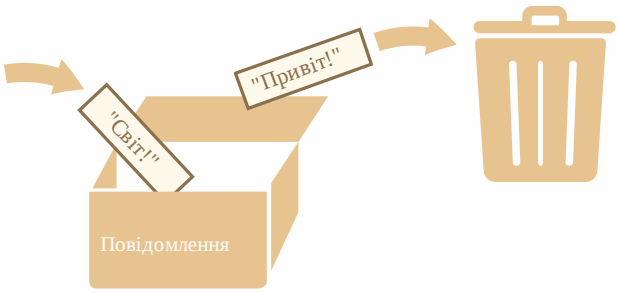
```
let message;

message = 'Привіт!';

message = 'Світ!'; // значення змінено

alert(message);
```

Коли значення змінюється, старі дані видаляються зі змінної:



Ми також можемо оголосити дві змінні і скопіювати дані з однієї в іншу.

```
let hello = 'Привіт світ!';

let message;

// копіюємо 'Привіт світ' з hello в message
message = hello;

// тепер дві змінні мають однакові дані
alert(hello); // Привіт світ!
alert(message); // Привіт світ!
```

Оголошення змінної вдруге призведе до помилки

Оголошувати змінну можна лише один раз.

Повторне оголошення тієї ж змінної спричинить помилку:

```
let message = "Це";

// повторне 'let' призведе до синтаксичної помилки
let message = "Той"; // SyntaxError: 'message' has already been declared
```

Тому ми маємо оголосити змінну лише раз, а потім просто посилатися на неї, без `let`.

Функційне програмування

Цікаво зазначити, що функційні мови програмування, такі як Scala або Erlang, забороняють змінювати значення змінних.

У таких мовах збережені в “коробку” значення залишаються там назавжди. Якщо нам потрібно зберегти щось інше, мова змусить нас створити нову коробку (оголосити нову змінну). Ми не можемо використати стару змінну.

Хоча на перший погляд це може здатися дивним, проте ці мови цілком підходять для серйозної розробки. Ба більше, є такі галузі, як от паралельні обчислення, де це обмеження дає певні переваги. Вивчення такої мови (навіть якщо ви не плануєте користуватися нею найближчим часом) рекомендується для розширення кругозору.

Іменування змінних

В JavaScript є два обмеження, які стосуються імен змінних:

1. Ім’я має містити лише букви, цифри або символи `$` і `_`.
2. Перший символ не має бути числом.

Ось приклади допустимих імен:

```
let userName;
let test123;
```

Для написання імені, яке містить декілька слів, зазвичай використовують “верблюжий регістр” (camelCase). Тобто слова йдуть одне за одним, і кожне слово пишуть із великої букви й без пробілів: `myVeryLongName`. Зауважте, що перше слово пишеться з маленької букви.

Що цікаво – знак долара `'$'` і знак підкреслення `'_'` також можна використовувати в іменах. Це звичайні символи, такі ж, як і букви, без будь-якого особливого значення.

Ці імена також допустимі:

```
let $ = 1; // оголошено змінну з ім'ям "$"
let _ = 2; // а тепер змінна з ім'ям "_"

alert($ + _); // 3
```

Приклади недопустимих імен змінних:

```
let 1a; // не може починатися з цифри

let my-name; // дефіс '-' недопустимий в імені
```

Регістр має значення

Змінні з іменами `apple` і `AppLE` – це дві різні змінні.

Не-латинські букви дозволені, але не рекомендуються

Можна використовувати будь-яку мову, включно з кирилицею або навіть ієрогліфами, наприклад:

```
let назва = '...';
let 我 = '...';
```

Технічно тут немає помилки. Такі імена дозволені, проте є міжнародна традиція використовувати англійську мову в іменах змінних (наприклад, `yaLyublyuUkrainu` => `iLoveUkraine`). Навіть якщо ми пишемо маленький скрипт, у нього може бути тривале життя попереду. Людям з інших країн, можливо, доведеться прочитати його не один раз.

Зарезервовані слова

Є список зарезервованих слів , які не можна використовувати як імена змінних, тому що ці слова використовує сама мова.

Наприклад: `let`, `class`, `return` і `function` зарезервовані.

Такий код видаватиме синтаксичну помилку:

```
let let = 5; // не можна назвати змінну "let", помилка!
let return = 5; // також не можна називати змінну "return", помилка!
```

Створення змінної без використання `use strict`

Зазвичай нам потрібно оголосити змінну перед її використанням. Але в старі часи було технічно можливим створити змінну простим присвоєнням значення, без використання `let`. Це все ще працює, якщо не вмикати `суворий режим` у наших скриптах для підтримання сумісності зі старими сценаріями.

```
// "use strict" в цьому прикладі не використовується

num = 5; // якщо змінна "num" не існувала, її буде створено

alert(num); // 5
```

Це погана практика, яка призведе до помилки в суворому режимі:

```
"use strict";

num = 5; // помилка: num не оголошено
```

Константи

Щоб оголосити константу (незмінювану) змінну, використовуйте ключове слово `const` замість `let`:

```
const myBirthday = '18.04.1982';
```

Змінні, оголошені за допомогою `const`, називаються “константами”. Їхні значення не можна переприсвоювати. Спроба це зробити призведе до помилки:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // помилка, не можна перевизначати константу!
```

Коли програміст впевнений, що змінна ніколи не буде змінюватися, він може оголосити її через `const`, що гарантує постійність і буде зрозумілим для кожного.

Константи в верхньому регістрі

Широко поширена практика використання констант як псевдонімів для значень, які важко запам'ятати і які відомі до початку виконання скрипта.

Такі константи пишуться в верхньому регістрі з використанням підкреслень.

Наприклад, давайте створимо константи, в які запишемо значення кольорів у так званому "вебформаті" (в шістнадцятковій системі):

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...коли потрібно вибрати колір
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Переваги:

- `COLOR_ORANGE` набагато легше запам'ятати, ніж `"#FF7F00"`.
- Набагато легше допустити помилку в `"#FF7F00"`, ніж під час введення `COLOR_ORANGE`.
- Під час читання коду `COLOR_ORANGE` набагато зрозуміліше, ніж `#FF7F00`.

Коли ми маємо використовувати для констант великі букви, а коли звичайні? Давайте це з'ясуємо.

Назва "константа" лише означає, що змінна ніколи не зміниться. Але є константи, які відомі нам до виконання скрипта (наприклад, шістнадцяткове значення для червоного кольору), а є константи, які *вираховуються* в процесі виконання скрипта, але не змінюються після їхнього початкового присвоєння.

Наприклад:

```
const pageLoadTime = /* час, потрачений на завантаження вебсторінки */;
```

Значення `pageLoadTime` невідоме до завантаження сторінки, тому її ім'я записано звичайними, а не великими буквами. Але це все ще константа, тому що вона не змінює значення після присвоєння.

Інакше кажучи, константи з великими буквами використовуються як псевдоніми для "жорстко закодованих" значень.

Придумуйте правильні імена

У розмові про змінні необхідно згадати ще одну дуже важливу річ – правильні імена змінних.

Такі імена повинні мати чіткий і зрозумілий сенс, який описує дані, що в них зберігаються.

Іменування змінних – одна з найважливіших і найскладніших навичок у програмуванні. Швидкий перегляд змінних може показати, чи код був написаний новачком чи досвідченим розробником.

У реальному проєкті більшість часу тратиться на змінення і розширення наявної кодової бази, а не на написання чогось цілком нового. Коли ми повертаємося до якогось коду після виконання чогось іншого впродовж тривалого часу, набагато легше знайти інформацію, яку добре позначено. Або, інакше кажучи, коли змінні мають хороші імена.

Будь ласка, приділіть час на обдумування правильного імені для змінної перед її оголошенням. Робіть так, і будете винагороджені.

Декілька хороших правил:

- Використовуйте імена, які легко прочитати, наприклад, `userName` або `shoppingCart`.
- Уникайте використання абревіатур або коротких імен, таких як `a`, `b`, `c`, окрім тих випадків, коли ви точно знаєте, що так потрібно.
- Робіть імена максимально описовими і лаконічними. Наприклад, такі імена погані: `data` і `value`. Такі імена нічого не говорять. Їх можна використовувати лише тоді, коли з контексту очевидно, на які дані або значення посилається змінна.
- Погоджуйте з вашою командою, які терміни будуть використовуватися. Якщо відвідувач сайту називається “user”, тоді ми маємо давати відповідні імена іншим пов’язаним змінним: `currentUser` або `newUser`, замість `currentVisitor` або `newManInTown`.

Звучить легко? Це дійсно так, проте на практиці створення зрозумілих і коротких імен – рідкість. Дійте.

Перевикористовувати чи створювати?

І остання примітка. Є ліниві програмісти, які замість оголошення нових змінних повторно використовують наявні.

У результаті їхні змінні схожі на коробки, в які люди кидають різні речі, не змінюючи на них наклейки. Що зараз міститься всередині коробки? Хто знає? Нам необхідно підійти поближче і перевірити.

Такі програмісти економлять трішки часу на оголошенні змінних, але втрачають вдесятеро більше під час відлагодження.

Додаткова змінна – це добро, а не зло.

Сучасні JavaScript-мініфікатори і браузери оптимізують код досить добре, тому додаткові змінні не погіршують продуктивність. Використання різних змінних для різних значень може навіть допомогти рушію оптимізувати ваш код.

Підсумки

Ми можемо оголосити змінні для збереження даних за допомогою ключових слів `var`, `let` чи `const`.

- `let` – це сучасний спосіб оголошення.
- `var` – це застарілий спосіб оголошення змінної. Зазвичай ми не використовуємо його взагалі, але ми розглянемо тонкі відмінності від `let` в розділі Застаріле ключове слово

"var", на випадок, якщо це все-таки знадобиться.

- `const` – це як `let`, але значення змінної не може змінюватися.

Змінні мають називатися так, щоб ми могли легко зрозуміти, що в середині них.

Завдання

Challenge #3 Робота зі змінними

1. Оголосіть дві змінні: `admin` та `name`.
2. Присвойте значення `"Іван"` змінній `name`.
3. Скопіюйте значення зі змінної `name` в `admin`.
4. Виведіть значення змінної `admin`, використовуючи функцію `alert` (яка повинна показати “Іван”).

Challenge #4 Придумайте правильні імена

1. Створіть змінну із назвою нашої планети. Як би ви назвали таку змінну?
2. Створіть змінну для зберігання імені поточного відвідувача сайту. Як би ви назвали таку змінну?

Challenge #5 Використовувати великі чи маленькі букви для імен констант?

Переглянемо наступний код:

```
const birthday = '18.04.1982';

const age = someCode(birthday);
```

В нас є константа `birthday`, а також `age`, яка вираховується за допомогою функції, використовуючи значення із `birthday` (в даному випадку деталі не мають значення, тому код функції не розглядається).

Чи можна використовувати великі букви для імені `birthday`? А для `age`? Чи для обох змінних?

```
const BIRTHDAY = '18.04.1982'; // використовувати великі букви?

const AGE = someCode(BIRTHDAY); // а тут?
```