Лабораторна робота №1

Знайомство з JavaScript

Мета роботи: вивчити основні типи даних та оператори мови JavaScript .

Теоретична частина

Для написання сценаріїв на мові JavaScript достатньо будь-якого текстового редактора, але краще використовувати спеціалізовані, такі як Notepad ++, Visual Studio, DreamWeaver.

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

```
<!DOCTYPE HTML>
<html>
     <body>
          <p>Текст перед скриптом...</p>
               <script>
                    alert('Привіт, світ!');
               </script>
          <p>...Після скрипта.</p>
     </body>
</html>
```

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 старих посібниках можна 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 коду на JavaScript є багато, можна 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.

Також можна 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, використовують кілька тегів:

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

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 вказано атрибут `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>
```

Потрібно 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>
```

- Для додавання коду JavaScript на сторінку використовують тег `<script>`.

- Атрибути `type` і `language` не потрібні.
- Скрипти у зовнішньому файлі можна вставити за допомогою `<script src="path/to/script.js"></script>`.

Синтаксичні конструкції та команди

Сценарій JavaScript виконується з використанням інструкцій, які представляють синтаксичні конструкції та команди, що виконують дії. Зазвичай, кожну інструкцію пишуть на новому рядку, щоб код було легше читати, і закінчують крапкою з комою.

```
alert ('Hello');
alert ('World');
```

Для пояснення програмного коду використовуються коментарі. У JavaScript коментарі можуть знаходитися в будь-якому місці скрипта і можуть бути однорядковими та багаторядковими.

```
// Цей коментар займає весь рядок
/*
Це – багаторядковий коментар.
*/
```

Зазвичай, веб-програма обробляє певну інформацію. Для зберігання інформації в JavaScript використовуються змінні та константи.

Для створення змінної JavaScript використовують ключове слово `let`:

```
let message; //оголошення змінної
message = 'Hello'; //привласнення значення
alert(message); //показує вміст змінної
```

Щоб писати менше коду, можна суміщати оголошення змінної та її привласнення в одному рядку:

```
let message = 'Привіт!'; // оголошення змінної і привласнення значення
alert(message); // показує вміст змінної
```

У старих скриптах можна знайти інше ключове слово: `var` замість `let` . Ключове слово `var` – майже те саме, що й `let`. Воно оголошує змінну, але іншим, застарілим способом.

Значення, що заносяться до змінної з використанням оператора присвоєння (=), можуть бути одного з 8 основних типів:

- `number` для будь-яких чисел: цілих чисел з плаваючою точкою, цілі числові значення обмежені діапазоном $\pm 2^{53}$.
- `bigint` для цілих чисел довільної довжини.
- `string` для рядків. Рядок може містити один або більше символів, немає окремого символьного типу.
- `boolean` для true / false.
- `null` для невідомих значень – окремий тип, що має одне значення null.
- `undefined` для не присвоєних значень – окремий тип, що має одне значення undefined.

- `object` для складніших структур даних.
- `symbol` для унікальних ідентифікаторів.

Є багато способів оголосити декілька змінних. Наприклад, в одному рядку:

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

Таке оголошення може виглядати коротшим, проте так писати не рекомендується. Заради кращої читабельності, варто оголошуйте змінні з нового рядка.

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

Багаторядковий спосіб трохи довший, проте його легше читати:

Деякі розробники 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.

Іменування змінних

В JavaScript є два обмеження, які стосуються імен змінних:

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

Ось приклади допустимих імен:

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

Для написання імені, яке містить кілька 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.

Приклади допустимих імен змінних:

```
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
```

При назві змінних потрібно враховувати, що JavaScript реагує на різні регістри символів, до прикладу нижче наведено дві різні змінні:

```
var myIncome;
var MyIncome;
```

Нелатинські 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. Програмістам 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!
```

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 варто використовувати для констант великі букви, а коли звичайні? Назва "константа" лише означає, що змінна ніколи не зміниться. Але є константи, які відомі 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.

Константи з великими буквами використовуються як псевдоніми для "жорстко закодованих" значень.

Правильні імена

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. В такому коді набагато легше знайти інформацію, яку добре позначено.

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`.

Перетворення даних

Оператор `typeof` показує, який тип даних збережений у змінній:

- Має дві форми: `typeof x` або `typeof (x)`.
- Повертає рядок із назвою типу. Наприклад, "`string`".
- Для `null` повертається "`object`" - це помилка у мові, насправді це не об'єкт.

Часто значення, що заносяться до змінної, є результатом обчислення виразу. Вираз є набором операндів та операторів, що виконують дії над операндами. JavaScript є нащадком мови С. Тому, набір операторів і керуючих конструкцій у мові такий самий як і в С.

Мова JavaScript є інтерпретованою і слабко типізованою, на відміну компільованих. Тип змінної визначається за введеним значенням або результатом обчислення виразу, тому в процесі виконання сценарію тип змінної може змінюватися. Потрібно уважно стежити за перетворенням типів.

Для перетворення рядків у числа JavaScript передбачено вбудовані функції `parseInt()` і `parseFloat()`. Функція `parseInt(рядок,основа)` перетворює вказаний у параметрі рядок у ціле число в системі числення за вказаною основою (8, 10 або 16). Якщо основу не зазначено, то передбачається десяткова система числення. Функція `parseFloat(рядок)` перетворює зазначений рядок на число з плаваючою роздільною (десятковою, основа) точкою:

```
parseInt ("3.14") // результат = 3
parseInt ("Вася") // результат = NaN , тобто не є числом
parseInt ("0xFF", 16) // результат = 255
parseFloat ("3.14") // результат = 3.14
parseFloat ("-7.875") // результат = -7.875
```

В JavaScript передбачено досить мало засобів для введення та виведення даних, оскільки JavaScript створювався насамперед як мова сценаріїв для веб-сторінок. Можна скористатися трьома стандартними методами глобального об'єкта `window` для введення та виведення даних: `alert(), prompt(), confirm()`.

Метод `alert()` дозволяє виводити діалогове вікно із заданим повідомленням та кнопкою ОК:

```
alert (“повідомлення”);
```

Повідомлення представляє дані будь-якого типу: послідовність символів, укладені в лапки, число (у лапках або без них), змінну або вираз.

Метод `confirm()` дозволяє вивести діалогове вікно з повідомленням та двома кнопками – ОК та Скасувати (Cancel). На відміну від методу `alert`, цей метод повертає логічну величину, значення якої залежить від того, на якій із двох кнопок клацнув користувач. Якщо він клацнув на кнопці ОК то повертається значення `true(істина)`, якщо він клацнув на кнопці Скасувати, то повертається значення `false(хиба)`. Синтаксис застосування методу `confirm` має такий вигляд:

```
let answer = confirm (“повідомлення”);
```

Метод `prompt()` дозволяє вивести на екран діалогове вікно з повідомленням, а також текстове поле, в яке користувач може ввести дані. Крім того, у цьому вікні передбачені дві кнопки: ОК та Cancel). Цей метод приймає два параметри: повідомлення та значення, яке має з'явитися у текстовому полі введення даних за замовченням. Якщо користувач клацне на кнопці ОК, метод поверне вміст поля введення даних, якщо він клацне на кнопці Скасувати, то повертається логічне значення `false(хиба)`.

```
let input = prompt ("повідомлення" , " початкове _значення ");
```

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"`.

Директива дослівно перекладається як “використовувати суворий (режим)”. Якщо вона прописана на початку скрипта, він буде виконуватися у “сучасному” режимі.

Директиву `"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, тоді `“use strict”` пишуть зверху, інакше суворий режим не увімкнеться.

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

Вище `"use strict"` можуть бути лише коментарі. 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.

Метод document.write

Метод document.write – один із найдавніших методів додавання тексту до документа. Він має суттєві обмеження, тому використовується рідко, хоча за своєю суттю є унікальним і корисним.

Метод document.write(str) працює лише доки HTML-сторінка перебуває у процесі завантаження. Він дописує текст у поточне місце HTML ще до того, як браузер збудує з нього DOM. HTML-документ нижче буде містити цифри `1 2 3`.

```
<body>
      1
   <script>
       document.write(2);
   </script>
      3
</body>
```

Немає жодних обмежень на вміст document.write. Рядок просто пишеться в HTML-документі без перевірки структури тегів, ніби вона завжди там була. Наприклад:

```
<body>
      <script>
            document.write('<style> td {color: #F40; border:1px solid #000;
background:#ccc}</style>');
      </script>
      <table>
            <tr>
                  <script>
                        document.write('<td>')
                  </script>
                  Текст всередині комірки.
                  <script>
                        document.write('</td>')
                  </script>
            </tr>
      </table>
</body>
```

Переваги перед innerHTML

Метод document.write – один із найдавніших методів, попередник сучасного стандартного методу innerHTML, потреба в ньому виникає рідко, але деякі переваги все ж таки є.

- Метод document.write працює швидше, фактично, це найшвидший спосіб додати на сторінку текст, згенерований скриптом.
- Це природно, адже він не модифікує існуючий DOM, а пише текст сторінки до його генерації.
- Метод document.write вставляє будь-який текст на сторінку «як є», тоді як innerHTML може вписати лише валідний HTML (при спробі підсунути невалідний – браузер скоригує його). Ці переваги є скоріше засобом оптимізації, який потрібно використовувати саме там, де така оптимізація потрібна чи доречна.

Практична частина:

Для виконання практичної частини уважно ознайомтеся з наведеними прикладами. Як індивідуальне завдання дані приклади мають бути модифіковані і збагачені іншими конструкціями.

У всіх скриптах, у заголовку вікна браузера зазначити групу і прізвище студента.

Завдання 1. Робота з діалоговими вікнами.

У редакторі створіть файл lab1_1.html і помістіть до нього наступний код. Відкрийте цей файл у браузері та перегляньте результат.

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset="UTF-8">
            <title>Робота з діалоговими вікнами</title>
      </head>
      <body>
            <script>
                  "use strict";
                  alert("Моє перше діалогове вікно");
                  const   answer=confirm("Хочете   продовжити   виконання
сценарію на JavaScript");
                  if (answer) {
                        const name=prompt ("Введіть ваше ім'я", "");
                        alert(name + ", у вас вже починає виходити!");
                  }
                  else alert ("Шкода, можна було б ще попрацювати!");
            </script>
      </body>
</html>
```

Завдання 2. Обчислення арифметичних виразів.

Створіть файл lab1_2.html і помістіть до нього наступний код. Перегляньте у браузері результати роботи скрипта.

```
<!DOCTYPE html >
<html>
      <head>
            <meta charset="UTF-8">
            <title>Обчислення арифметичних виразів</title>
      </head>
      <body>
            <script>
                  "use strict";
                  const x=parseInt(prompt("Введіть значення x", ""));
                  let a=x*x-7*x+10;
                  let b=x*x-8*x+12;
                  let c = a/b;
                  alert("c="+c);
            </script>
      </body>
</html>
```

Завдання 3. Обчислення площі і периметра правильного n-кутника, описаного біля кола радіуса R.

Створіть файл lab1_3.html і помістіть до нього наступний код. Перегляньте у браузері результати роботи скрипта.

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset="UTF-8">
            <title>Лінійні алгоритми</title>
      </head>
      <body>
            <script>
                  "use strict";
                  let n=parseInt(prompt("Введіть кількість кутів n", ""));
                  let r=parseInt(prompt("Введіть радіус r", ""));
                  let a=2*r*Math.tan(Math.PI/n);
                  let p=a*n;
                  let s=(1/2)*n*a*r;
                  alert("Площа =" + s.toFixed(2));
                  alert("Периметр =" + p.toFixed (2));
            </script>
      </body>
</html>
```

Для обчислення арифметичного виразу використовуються методи об'єкта Math:

`abs(x)` Повертає абсолютне значення (модуль) числа x.

`acos(x)` Повертає арккосинус числа x у радіанах.

`asin(x)` Повертає арксинус числа x у радіанах.

`atan(x)` Повертає арктангенс числа x як чисельне значення між -PI/2 та PI/2.

`ceil(x)` Округлює значення x до першого більшого цілого числа.

`cos(x)` Повертає косинус числа x (число x визначається в радіанах).

`exp(x)` Повертає значення E у ступені x.

`floor(x)` Округлює значення x до першого меншого цілого числа.

`log(x)` Повертає натуральний логарифм (з основою E) x.

`max(x1,x2,...xn)` Повертає більше із чисел x1,x2,...xn.

`min(x1,x2,...xn)` Повертає найменше з чисел x1,x2,...xn.

`pow(x,y)` Зводить x до ступеня y і повертає результат.

`random()` Повертає випадкове число між 0 та 1 (наприклад 0.6230522912910803).

`round(x)` Округлює значення x до найближчого цілого числа.

`sin(x)` Повертає синус числа x (число x визначається в радіанах).

`sqrt(x)` Повертає квадратний корінь x.

`tan(x)` Повертає тангенс кута.

Завдання 4. Формування динамічних сторінок за допомогою методу write об'єкта document.

Створіть файл lab1_4.html і помістіть до нього наступний код. Перегляньте у браузері результати роботи скрипта.

```
<!DOCTYPE html>
<html>
      <head>
            <meta charset = "UTF-8">
            <title>Динамічно сформована сторінка</title>
      </head>
      <body>
            <script>
                  "use strict";
                  document.write ("Початок сформованої сторінки");
                  document.write ("<h1>Заголовок першого рівня</h1>");
                  document.write     ('<p     style="text-align:center;font-
size:18px;color:red;">Зміна розміру та кольору шрифту');
                  document.write ("<p>Кінець формування сторінки, що містить
сценарій");
            </script>
      </body>
</html>
```

Література

1. Основи JavaScript - https://uk.javascript.info/first-steps

Контрольні питання

1. Які види діалогових вікон ви знаєте?
2. Як ввести дані користувача?
3. Як перетворити рядок на число?
4. Які типи змінних використовуються JavaScript?
5. Як вивести повідомлення у діалогове вікно?
6. Навіщо використовується ключове слово var?
7. Навіщо використовується ключове слово let?
8. Навіщо використовується ключове слово const?
9. Навіщо використовується метод write об'єкта document?
10. Навіщо використовується вбудований об'єкт Math?
11. Як записати складний арифметичний вираз?