Тема 3. Основи мови Java Script

Загальний огляд мови JavaScript.

JavaScript - це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML. Ця мова програмування була створена фірмами Netscape та Sun Microsystems на базі мови програмування Sun's Java. На сьогодні є декілька версій JavaScript. Однією із найбільш поширених є версія JavaScript 1.3. За допомогою JavaScript на HTML-сторінці можливо зробити те, що не можливо зробити за допомогою стандартних тегів HTML.

Код програми JavaScript розміщується або в середині HTML-сторінки, або в текстовому файлі, що пов'язаний за допомогою спеціальних команд з HTML-сторінкою. Цей код, як правило, розміщується в середині тегу HTML та завантажується в браузер разом з кодом HTML-сторінки. Програма JavaScript не може існувати самостійно, тобто без HTML-сторінки.

Виконання програми JavaScript відбувається при перегляді HTML- сторінки в браузері, звісно, тільки в тому випадку, коли браузер містить інтерпретатор JavaScript. Практично всі сучасні популярні браузери оснащені таким інтерпретатором. Відзначимо, що крім JavaScript на HTML-сторінках можливо використовувати інші мови програмування. Наприклад, VBScript або JScript, яка є варіантом JavaScript від фірми Microsoft. Але виконання програм VBScript та JScript гарантовано коректне тільки при перегляді HTML-сторінки за допомогою браузеру Microsoft Internet Explorer. Тому в більшості випадків використання JavaScript доцільніше, хоча функціональність програм VBScript та JScript дещо краща.

Досить часто програму JavaScript називають *скриптом* або *сценарієм.* Скрипти виконуються в результаті того, що відбулась деяка подія, пов'язана з HTML-сторінкою. В багатьох випадках виконання вказаних подій ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

За допомогою парного тегу SCRIPT;

Як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований в HTML-сторінку з використанням тегу SCRIPT, має наступний формат:

```
<SCRIPT>
    // Код програми
</SCRIPT>
```

Все, що розміщується між тегами <SCRIPT> та </SCRIPT>, інтерпретується як код програми на мові JavaScript. Обсяг вказаного коду не обмежений. Інколи скрипти розміщують в середині HTML-коментарію. Це роблять для того, щоб код JavaScript не розглядався старими браузерами, які не мають інтерпретатора

JavaScript. В цьому випадку сценарій має формат:

```
<SCRIPT>
     <!--
   // Код програми
     -->
</SCRIPT>
```

Тег SCRIPT має декілька необов'язкових параметрів. Найчастіше використовуються параметри *language* та *src.* Параметр *language* дозволяє визначити мову та версію мови сценарію. Параметр src дозволяє задати файл з кодом сценарію. Для пояснення використання параметрів тегу SCRIPT розглянемо задачу.

Задача. Необхідно для HTML-сторінки hi.htm створити сценарій на мові JavaScript 1.3 для показу на екрані вікна повідомлення з текстом "Привіт!".

Відзначимо, що для показу на екрані вікна повідомлення можливо використати функцію alert.

Для ілюстрації можливостей пов'язування скриптів з HTML-кодом вирішення задачі реалізуємо двома варіантами.

Варіант 1. Визначення сценарію безпосередньо на HTML -сторінці hi.htm

```
<html><head>
     <PiP1e>Використання JavaScript</title>
</head>
<body>
<script language="JavaScript1.3">
   alert('hi');
</script>
</body></html>
```

Варіант 2. Визначення сценарію в файлі a.js, пов'язаному з HTML-сторінкою hi.htm за допомогою параметру src тегу SCRIPT. Код HTML- сторінки hi.htm:

```
<html><head>
     <Ш1є>Використзння JavaScript</title>
     <script language="JavaScript1.3" src="a.js"> </script>
</head><body> </body></html>
```

Програмний код, записаний в файлі a.js:

alert('hi');

Результат виконання обох варіантів вирішення задачі однаковий і показаний на рисунку 3.1.

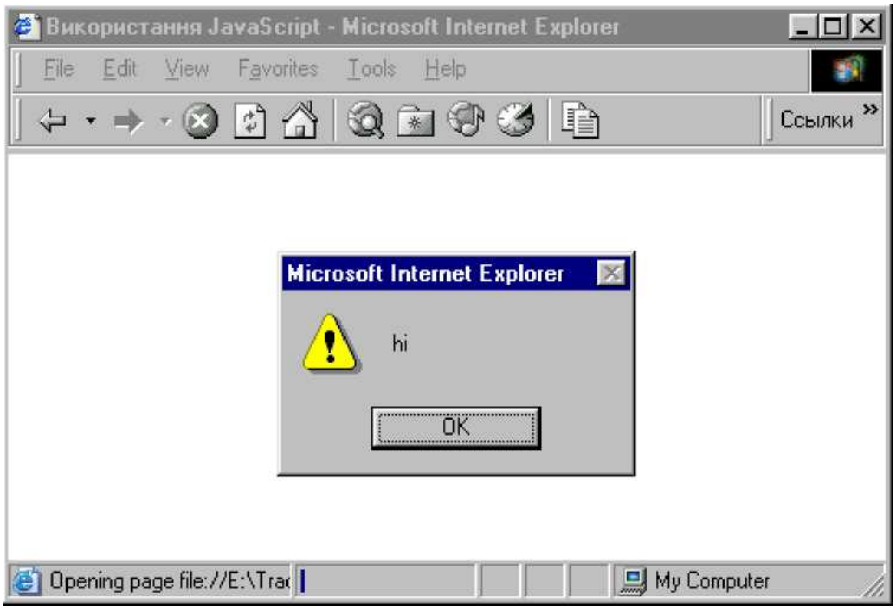

Рисунок 3.1 Показ вікна повідомлення засобами JavaScript

Змінні та вирази JavaScript можливо використовувати в якості значень параметрів тегів HTML. В цьому випадку елементи JavaScript розміщуються між амперсандом (&) та крапкою з комою (;), але повинні бути обмежені фігурними дужками {} і використовуватись тільки в якості значень параметрів тегів.

Наприклад, нехай визначена змінна *c* і їй присвоєно значення *green.* Наступний тег буде виводити текст зеленого кольору:

```
<font color="&{c};"> текст зеленого кольору </font>
```

Відзначимо, що мінімальним комплектом програмного забезпеченням для розробки та тестування програм JavaScript є текстовий редактор та браузер з підтримкою JavaScript.

Синтаксис. Визначення та ініціалізація змінних

Сценарій JavaScript являє собою набір операторів, що послідовно інтерпретуються браузером. Оператори можливо розміщувати як в одному, так і в окремих рядках. Якщо оператори розміщені в одному рядку, то між ними необхідно поставити ;. В протилежному випадку ; не обов'язкова. Будь-який оператор можливо розмістити в декількох рядках без символу продовження.

Будь-яка послідовність символів, розміщених в одному рядку, якій передує //, розглядається як *коментар.* Для визначення *багаторядкових коментарів* використовується конструкція:

```
/*
Багаторядковий коментар
*/
```

В мові JavaScript рядкові та приписні букви вважаються різними символами. JavaScript використовує змінні для зберігання даних визначеного типу. При цьому JavaScript є мовою з вільним використанням типів. Тобто не

обов’язково задавати тип змінної, який залежить від типу даних, що в ній зберігаються. При зміні типу даних автоматично змінюється і тип змінної.

JavaScript підтримує чотири простих типи даних. Ілюстрацією цього є табл. 3.1. Для присвоєння змінним значень основних типів використовуються *літерали.*

Таблиця 3.1 Типи даних JavaScript

Тип даних	Пояснення	Приклад літерала
Цілий	Послідовність цілих чисел	789 +456 -123
З плаваючою крапкою	Числа з крапкою, яка відділяє цілу частину від дробової, або числа в науковій нотації	7.25 0.525e01 71.2E-4
Рядковий	Послідовність алфавітно-цифрових символів, взятих в одинарні (’), або подвійні (") лапки.	"Привіт" "234" "Hello World!!!"
Булевий або логічний	Використовуються для оброблення ситуацій так/ні в операторах порівняння	true false

Ім ’я змінної повинно містити тільки букви латинського алфавіту, символ підкреслення _, арабські цифри та починатись з букви або символу підкреслення _. Довжина імені повинна бути менша від 255 символів. Заборонено використовувати імена, що збігаються з ключовими словами JavaScript. Приклади імен: _hello, go, go123.

Визначити змінну можливо:

Оператором *var,* наприклад, var a;

При *ініціалізації,* за допомогою оператору присвоєння (=), наприклад, b=126.

Визначення та ініціалізацію змінних можливо реалізувати в будь-якому місці програми.

Перетворення типів

Тип змінної залежить від того, який тип інформації в ній зберігається. JavaScript - слаботипізована мова. Це означає, що в декларації змінної не вказується його тип і надалі можна надавати їй значення будь-яких типів. Виконуюча система JavaScript сама виконує автоматичне перетворення типів даних у міру необхідності. Для явного перетворення типів використовуються методи Boolean, Number, Object і String.

Тип змінної привласнюється змінній автоматично протягом виконання скрипта. Так, наприклад, ви можете визначити змінну в такий спосіб: `var answer=42`

А пізніше, ви можете присвоїти тій же змінній значення, що наприклад

випливає:

```
answer="Thanks for all the fish..."
```

Або розглянемо наступний вираз:

```
//приклад
var onestring="1"
var oneint=1
var oneconcatenate=onestring+oneint
// У результаті виходить "11"
var oneaddition=oneint+onestring
// У результаті виходить 2
```

У першій операції додавання перший операнд є рядком. Javascript припускає, що проводиться операція із двома рядками. Коли Javascript виявляє в якості другого операнда ціле число, він у відповідності зі своїми виставами перетворить змінну в рядок.

Оскільки Javascript вільно типізований мова, то це не викличе помилки.

Тому що Javascript не підтримує ніяких методів і властивостей для визначення типу поточного значення змінної, дуже важливо уважно відстежувати типи змінних щоб уникнути несподіваних результатів.

Взагалі, у виразах рядкові значення, що включають числові Javascript перетворює числові значення в строкові. Наприклад, розглянемо наступні твердження:

```
x="The answer is " + 42
y=42 + " is the answer."
```

Перше твердження буде рядок "The answer is - 42 ". Друге твердження повертає рядок " 42 - The answer is".

Вирази та оператори

Вираз - це комбінація змінних, літералів та операторів, в результаті обчислення яких можливо отримати тільки одне значення, яке може бути числовим, рядковим або булевим.

Для реалізації обчислень в JavaScript використовуються арифметичні, рядкові, логічні вирази та декілька типів операторів.

Арифметичні вирази - обчислюють число, наприклад, a=7+5;

Рядкові вирази - обчислюють рядок символів, наприклад, "Джон" або "234";

Логічні вирази - обчислюють true (істина) або false (хибна).

Оператор присвоювання (=) - присвоює, значення лівому операнду, базуючись на значенні правого операнда. Наприклад, для присвоєння змінній *a* значення числа 5 необхідно записати:

a=5

До стандартних *арифметичних операторів* відносяться: оператори

додавання (+), віднімання (-), множення (*), ділення *(/),* остача від ділення чисел *(%),* збільшення числової змінної на 1 (++), зменшення числової змінної на 1 (--).

Відзначимо, що оператор додавання можна використовувати не тільки для чисел, але й для додавання (контрактації / конкатенації) текстових рядків.

Для створення логічних виразів використовуються *логічні оператори* та *оператори порівняння.*

До логічних операторів відносяться - логічне *І (&&),* логічне *АБО* (||), логічне *НІ (!).*

Оператори порівняння не відрізняються від таких операторів в інших мовах програмування. До операторів порівняння відносяться *(==, >, >, =<, <=,!=).*

Умовні та циклічні оператори

Оператори вибору відносяться до *операторів управління,* призначенням яких є зміна напрямку виконання програми. Крім операторів вибору до операторів управління відносяться: оператори циклу та оператори маніпулювання об'єктами.

Оператори вибору призначені для виконання деяких блоків операторів в залежності від істинності деякого логічного виразу. До операторів вибору відносяться: оператор умови *if...else* та перемикач *switch.*

Синтаксис оператора умови такий:

```
if (умова) {
    група операторів 1

}
[else] {
    група операторів 2

}
```

Перша група операторів виконується при умові істинності виразу умова. Необов'язковий блок *else* визначає другу групу операторів, яка буде виконуватись в випадку хибності умови, заданої в блоці if. В середині групи операторів можуть бути використані будь-які інші оператори, в тому числі і інші оператори умови. Це дозволяє створювати групу вкладених операторів умови if та реалізовувати складні алгоритми перевірки. Однак, якщо кількість вкладених операторів if більша ніж три, то програма стає складною для розуміння. В такому випадку доцільно використовувати оператор *switch.* В цьому операторі обчислюється деякий вираз та порівнюється з значенням, заданим в блоках *case.* Синтаксис оператора switch такий: `switch (вираз) { case значення1: [оператори1] break;`

```
case значення2:
[оператори2] break;

default:
```

```
[оператори]
}
```

Якщо значення виразу в блоці *switch* дорівнює *значення!,* то виконується група операторів *оператори!,* якщо дорівнює *значення!,* то виконується група операторів *оператори!* і так далі. Якщо значення виразу не дорівнює ні одному із значень, що задані в блоках *case,* то обчислюється група операторів блоку *default,* якщо це блок заданий, інакше - виконується вихід із оператору *switch.* Необов'язковий оператор *break,* який можливо задавати в кожному із блоків *case,* виконує безумовний вихід із оператору *switch*. Якщо він не заданий, то продовжується виконання операторів в наступних блоках *case* до першого оператору *break* або до кінця оператору *switch*.

Оператори циклу

Цикл - це деяка група команд, що повторюється доки вказана умова не буде виконана. JavaScript 1.3 підтримує дві форми циклу: *for* та *while.* Крім того оператори *break* та *continue* використовуються разом з циклами.

Цикл for повторює групу команд до тих пір, доки вказана умова хибна. Синтаксис оператору for такий:

```
for([initial-expression];[condition];[increment-expression])
{
   statements
}
```

Виконання циклу *for* проходить в такій послідовності:

1. Вираз *initial-expression* служить для ініціалізації змінної лічильника. Цей вираз розраховується один раз на початку виконання циклу

2. Вираз *condition* розраховується на кожній ітерації циклу. Якщо значення виразу *condition* дорівнює true, виконується група операторів *statements* в тілі циклу. Якщо значення виразу *condition* дорівнює false, то цикл for закінчується. Якщо вираз *condition* пропущено, то він вважається рівним true. В цьому випадку цикл продовжується до оператора break.

3. Вираз *increment-expression* використовується для зміни значення змінної лічильника.

4. Розраховується група операторів *statements* та реалізується перехід на наступну ітерацію циклу, тобто на крок 2.

Приклад. Цикл для розрахунку суми цілих чисел від 1 до 100.

```
s = 0
for (i=1;i<101;i++) {
    s=s+1;
}
```

Оператор *while* повторює цикл, доки вказана умова істина. Оператор while виглядає таким чином:

```
while (condition) { statements
}
```

Цикл *while* виконується таким чином. Спочатку перевіряється умова *condition.* Якщо умова істинна, то виконується група операторів *statements* в середині циклу. Перевірка істинності виконується на кожному кроці циклу. Якщо умова хибна, то цикл закінчує своє виконання.

Інколи необхідно закінчити цикл не по умові, що задана в його заголовку, а в результаті виконання деякої умови в тілі циклу. Для цього використовуються оператори *break* та *continue.* Оператор *break* завершує цикл while або for та передає керування програмою першому оператору після циклу. Оператор continue передає управління оператору перевірки істинності умови в циклі while та оператору оновлення значення лічильника в циклі for і продовжує виконання циклу.

Оператор `for...in` виконує задані дії для кожної властивості об'єкта чи для кожного елемента масиву і має такий вигляд:

```
for (змінна in вираз) оператор
```

Оператор `for...in` діє таким чином:

1. *Змінній* надає назву чергової властивості об'єкту чи чергового елемента масиву (залежно від природи *виразу).*
2. Виконують *оператор.*
3. Переходять до етапу 1.

При ітерації властивостей об'єкту неможливо передбачити, в якому порядку їх буде проглянуто. Але їх буде проглянути усі без виключення. Подамо приклад створення об'єкту `ob` з наступним послідовним виведенням усіх його властивостей на екран користувача.

```
<html><script>
var ob = {"а": "Літера а", "б" : 2012};
for (var key in ob) document.write(key+": "+ob[key]
    +"<BR>");</script></html>
```

На екрані побачимо такий текст:

а: Літера а

б: 2012

Оператор `with` задає назву об'єкту за замовчуванням і має такий вигляд:

`with` (*вираз*) *оператор*

Цей оператор діє таким чином. Для кожної назви в *операторі* перевіряють, чи є вона назвою властивості об'єкту, заданого згідно із замовчуванням. Якщо відповідь ствердна, то цю назву вважають назвою відповідної властивості, інакше — назвою змінної. Це допомагає скоротити код. Наприклад, для доступу до математичних функцій маємо кожного разу вказувати назву об'єкту `Math`, як у такому коді:

```
x = Math.cos(Math.PI/2) + Math.sin(Math.LN10) +
Math.tan(2 * Math.E);
```

Того самого можна досягнути таким чином:

```
with (Math) { x=cos(PI/2) +sin(LN10)+tan(2 * E);};
```

Оператор `with` можна застосовувати лише до наявних методів.

Обробка виключних ситуацій

При виконанні сцернарію можливе винекнення помилок, які називають *виключеннями:* звертання до відсутнього об'єкта чи неможливість перетворення величини до заданого типу тощо.

Оператор `try...catch` використовують для опрацювання такого виключення. Він має такий вигляд:

`try { `*оператор*$_1$` } catch ( `*виключення*` ) { `*оператор*$_2$` }`

Тут *виключення* — довільна назва змінної, *оператор*^ — містить код, що може спричинити виключення. Якщо виключення не відбулося, то після виконання *оператору 1* здійснюють перехід до наступного за `try...catch` оператору. Інакше інформацію про виключення зберігають як величину локальної змінної *виключення,* а керування передають *оператор^,* який має містити код опрацювання цього виключення. Якщо виключення неможливо на даному рівні опрацювати, то *оператор*^ має містити оператор `throw` для переходу до опрацювання виключення на вищому рівні (див. далі).

Оператор `throw` породжує виключення, яке вже можна опрацювати оператором `try...catch`. Він має такий вигляд:

```
throw expression
```

Тут *expression* — будь-який вираз. Результат обчислення expression буде кинутий як виняток.

Подамо приклад породження й опрацювання виключення.

```
<html><body><script> var e;
function getMonthName(m)
  {    var    months=new    Array     ("січень","лютий",
  "березень","квітень","травень","червень","липень",
  "серпень", вересень","жовтень","листопад","грудень"); m =
  m -     1;
  if (months[m] != null) return months[m] else throw
  "trow";
}

  try       { monthName = getMonthName(7)}
  catch (e) { monthName="HenpaBM.nbHMh № місяця"};
  document.write(monthName);
```

```
</script></body></html>
```

При заміні `(7)` на `(77)` замість слова «липень на екрані побачимо текст: «Неправильний № місяця».

Використання функцій в JavaScript

Функція - JavaScript це іменована група команд, які вирішують певну задачу та можуть повернути деяке значення. Функція визначається за допомогою оператору *function* , що має такий синтаксис:

```
function Ім’я функції ([параметри])
{
[оператори]
return [значення що повертається]
}
```

Параметри, що передаються функції, розділяються комами. Необов’язковий оператор *return* в тілі функції (блок операторів, що обмежений фігурними дужками), визначає значення, що повертається функцією. Визначення функції тільки задає її ім’я и визначає, що буде робити функція при її визові. Безпосереднє виконання функції реалізується, коли в сценарії відбувається її виклик та передаються необхідні параметри. Відзначимо, що визначення функції необхідно реалізувати на HTML-сторінці до її виклику. Наприклад, для показу на екрані вікна повідомлення з текстом "Це виклик функції" визначимо функцію Go та реалізуємо її виклик:

```
<html><head><title>Використання JavaScript</title>
    <script>
    function Go() {
    alertG’^ виклик функції")
    }
    </script>
</head><body>
<script>
Go();
</script></body></html>
```