Тема 1. ОСНОВИ МОВИ ПРОГРАМУВАННЯ JavaScript

1.1. 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>

```
        <title>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>
        <title>Використання 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 рис. 6.1.

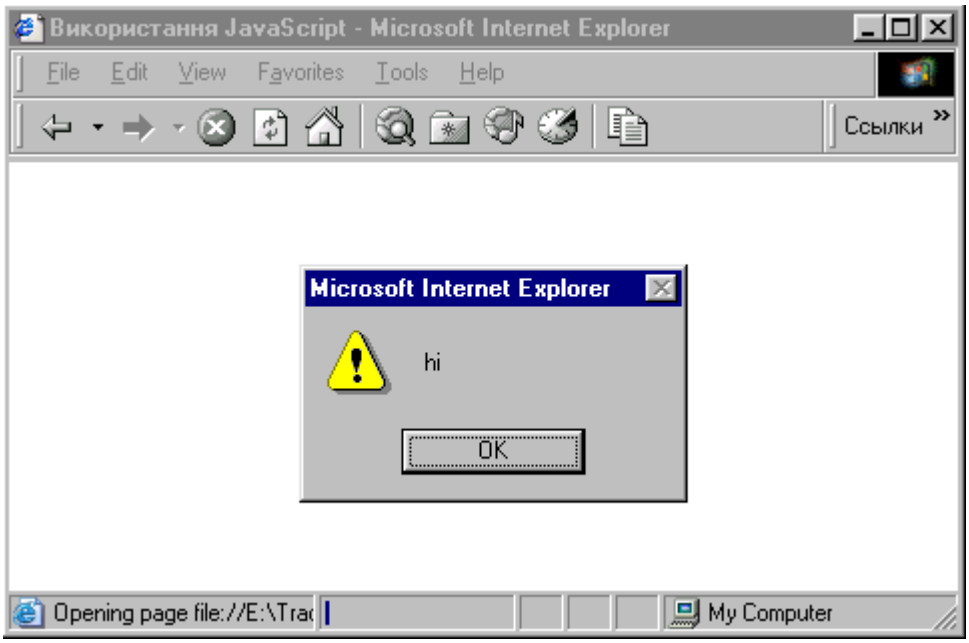

Рис. 6.1 Показ вікна повідомлення засобами JavaScript.

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 :

* текст зеленого кольору *

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

1.2. 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. 6.1. Для присвоєння змінним значень основних типів використовуються ***літерали***.

Таблиця 6.1

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.

1.3. 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 (==, >, >, =<, <=,!=).

Перетворення типів для примітивів

Система перетворення типів в JavaScript дуже проста, але відрізняється від інших мов. Тому вона часто служить «каменем спотикання» для програмістів, що приходять з інших мов.

Всього є три перетворення:

1. Рядкове перетворення.
2. Числове перетворення.
3. Перетворення до логічного значення.

1. Рядкове перетворення

Рядкове перетворення відбувається, коли потрібно представлення чого-небудь у вигляді рядка. Наприклад, його проводить функція `alert`.

```
var a = true;

alert( a ); // "true"
```

Можна також здійснити перетворення явним викликом `String(val)` :

```
alert( String(null) === "null" ); // true
```

Як видно з прикладів вище, перетворення відбувається найбільш очевидним способом, «як є»: `false` стає «`false`», `null` — «`null`», `undefined` — «`undefined`» і тому подібне

Також для явного перетворення застосовується оператор `"+"`, у якого один з аргументів рядок. В цьому випадку він призводить до рядка і інший аргумент, наприклад:

```
alert( true + "test" ); // "truetest"
alert( "123" + undefined ); // "123undefined"
```

4. Чисельне перетворення

Чисельне перетворення відбувається в математичних функціях і виразах, а також при порівнянні цих різних типів (окрім порівнянь `===`, `!==`).

Для перетворення до числа в явному виді можна викликати `Number(val)`, або, що коротше, поставити перед вираженням унарний плюс `"+"`:

```
var a = +"123"; // 123
var a = Number("123"); // 123, той же ефект
```

Значення	Перетвориться ст.
`undefined`	`NaN`
`null`	`0`
`true / false`	`1 / 0`
Рядок	Пробільні символи по краях обрізуються. Далі, якщо залишається порожній рядок, то `0`, інакше з непорожнього рядка «прочитується» число, при помилці результат `NaN`.

Наприклад:

```
// після обрізання пробільних символів залишиться "123"
alert( +"   \n  123   \n  \n" ); // 123
```

Ще приклади:

- Логічні значення:

- ```
  alert( +true ); // 1
  alert( +false ); // 0
  ```

- Порівняння різних типів — означає чисельне перетворення:

```
alert( "\n0 " == 0 ); // true
```

При цьому рядок `""` перетвориться до числа, як вказано вище: початкові і кінцеві пропуски обрізуються, виходить рядок `"0"`, яка дорівнює `0`.

- З логічними значеннями:

```
alert( "\n" == false );
alert( "1" == true );
```

- Тут порівняння `"=="` знову приводить обидві частини до числа. У першому рядку ліворуч і справа виходить `0`, в другій 1 .

Спеціальні значення

Подивимося на поведінку спеціальних значень уважніше.

Інтуїтивно, значення `null/undefined` асоціюються з нулем, але при перетвореннях поводяться інакше.

Спеціальні значення перетворяться до числа так:

Значення	Перетвориться ст.
`undefined`	`NaN`
`null`	`0`

Це перетворення здійснюється при арифметичних операціях і порівняннях `> >= < <=`, але не при перевірці рівності `==`. Алгоритм перевірки рівності для цих значень в специфікації прописаний окремо (пункт 11.9.3). У нім вважається, що null і `undefined` рівні `"=="` між собою, але ці значення не дорівнюють ніякому іншому значенню.

Це веде до забавних наслідків.

Наприклад, `null` не підкоряється законам математики — він «більше або дорівнює нулю»: `null>=0`, але не більше і не рівний:

```
alert( null >= 0 ); // true, оскільки null перетвориться до 0
alert( null > 0 ); // false (не більше), оскільки null перетвориться до 0
alert( null == 0 ); // false (і не рівний!), оскільки == розглядає null
особливо.
```

Значення `undefined` взагалі «незрівняно»:

```
alert( undefined > 0 ); // false, оскільки undefined -> NaN
alert( undefined == 0 ); // false, оскільки це undefined (без
перетворення)
alert( undefined < 0 ); // false, оскільки undefined -> NaN
```
```

Для очевиднішої роботи коду і щоб уникнути помилок краще не давати спеціальним значенням брати участь в порівняннях `> >= < <=`.

Використайте в таких випадках змінні-числа або призводьте до числа явно.

5. Логічне перетворення

Перетворення до `true/false` відбувається в логічному контексті, такому як `if(value)`, і при застосуванні логічних операторів.

Усі значення, які інтуїтивно «порожні», стають `false`. Їх декілька: `0`, порожній рядок, `null`, `undefined` і `NaN`.

Решта, у тому числі і будь-які об'єкти — `true`.

Повна таблиця перетворень :

Значення	Перетвориться ст.
`undefined`, `null`	`false`
Числа	Усе `true`, окрім `0`, `NaN` — `false`.
Рядки	Усе `true`, окрім порожнього рядка `""` — `false`
Об'єкти	Завжди `true`

Для явного перетворення використовується подвійне логічне заперечення `!!value` або виклик `Boolean(value)`.

Зверніть увагу: рядок `"0"` стає `true`
На відміну від багатьох мов програмування (наприклад PHP), `"0"` в JavaScript являється `true`, як і рядок з пропусків:

```
alert( "!!0" ); // true
alert( "!! " ); // будь-які непорожні рядки, навіть з пропусків - true!
```

Логічне перетворення цікаво тим, як воно поєднується з чисельним.

Два значення можуть бути рівні, але одне з них в логічному контексті `true`, інше — `false`.

Наприклад, рівність в наступному прикладі вірно, оскільки відбувається чисельне перетворення:

```
alert( 0 == "\n0\n" ); // true
```

А в логічному контексті ліва частина дасть `false`, права, — `true`:

```
if (""){
  alert( "true, зовсім не як 0"! );
}
```

З точки зору перетворення типів в JavaScript це абсолютно нормально. При порівнянні з допомогою "==" — чисельне перетворення, а в `if` — логічне, тільки і усього.

Масиви в JavaScript

Масив - це тип даних, що зберігає пронумеровані значення. Кожне пронумероване значення називається елементом масиву, а число, з яким зв'язується елемент, називається його індексом. Масиви JavaScript не типізуються, це означає, що елемент масиву може мати будь-який тип, причому різні елементи одного масиву можуть мати різні типи. Окрім цього масиви JavaScript є динамічними, це означає, що оголошувати фіксований розмір не треба і можна додати нові елементи у будь-який час.

Створення масиву

Масив можна створити двома способами, перший: створити масив за допомогою літерала масиву - квадратні дужки, усередині яких розташований список елементів, розділених комами.

```
var empty =[];  //порожній масив
var numbers =[4, 1, 2, 5]; //масив з 5 числовими елементами
var diff =[1.5, false, "текст"];  //масив з 3 елементами різного типу
```

Значення не обов'язково мають бути простими (числа або рядки) - це також можуть бути і будь-які інші вирази, наприклад: літерали об'єктів, інші масиви або функції.

```
var num = 700;
var tab =[function(a){ alert(a)}, { name: 'Петя' }, [1, 2, 3], num + 1];
```

Другий спосіб створення масиву - виклик конструктора Array(). Викликати конструктор Array() можна трьома способами.

Виклик конструктора без аргументів:

```
var b = new Array();
```

В цьому випадку створюється порожній масив, еквівалентний порожньому літералу [].

У конструкторі явно вказуються значення n елементів масиву :

```
var b = new Array(1, 3, 5, 8, "рядок", true);
```

В цьому випадку конструктор отримує список аргументів, які стають елементами нового масиву. Аргументи записуються в масив в тому порядку, в якому вказані.

Виділення місця для подальшого привласнення значень. Це робиться шляхом вказівки при визначенні масиву одного числа в круглих дужках:

```
var b = new Array(5);
```

Цей спосіб визначення масиву припускає виділення масиву певної кількості елементів (кожен з яких має значення undefined) з можливістю подальшого привласнення значень по ходу сценарію. Така форма зазвичай використовується для попереднього розміщення масиву, якщо його довжина відома заздалегідь.

Читання, запис і додавання елементів масиву

Доступ до елементів масиву здійснюється за допомогою оператора []. Елементи масиву в JavaScript нумеруються, починаючи з нуля. Щоб отримати потрібний елемент масиву, потрібно вказати його номер в квадратних дужках.

```
var numbers =[4, 1, 2, 5];
document.write(numbers[0] +; //перший елемент масиву
document.write(numbers[1] +; //другий елемент масиву
document.write(numbers[2] +; //третій елемент масиву
document.write(numbers[3]);        //четвертий елемент масиву
```

Елементи масиву можна змінювати:

```
var numbers =[4, 1, 2, 5];
numbers[0] = 10; //змінили перший елемент масиву - [10, 1, 2, 5]
```

Щоб додати новий елемент масиву, досить присвоїти нове значення:

```
var numbers =[4, 1];
numbers[2] = 7; //стало [4, 1, 7]
```

Примітка: в масивах JavaScript може зберігатися будь-яке число елементів будь-якого типу.

Довжина масиву

Усі масиви, як створені за допомогою конструктора Array(), так і визначені за допомогою літерала масиву, мають спеціальну властивість length, яке повертає загальне число елементів, що зберігаються в масиві. Оскільки масиви можуть мати невизначені елементи (що мають значення undefined), точніше формулювання звучить так: властивість length завжди на одиницю більше, ніж найбільший індекс (номер) елементу масиву. Властивість length автоматично оновлюється, залишаючись коректним при додаванні нових елементів в масив.

```
var v = new Array();  // v.length == 0 (жоден елемент не визначений)
v = new Array(1,2,3); // v.length == 3 (визначені елементи 0-2)
v =[4, 5];           // v.length == 2 (визначені елементи 0 і 1)
document.write(v.length);
```

Щоб отримати останній елемент масиву можна скористатися так само властивістю length :

```
var v =["JavaScript", "Властивість", "Масиви"];
```

```
document.write(v[v.length - 1]);
```

Останній елемент має індекс на 1 менше ніж, довжина масиву, оскільки відлік розпочинається з нуля. Тому, якщо ви не знаєте точну к-ть елементів, але вам потрібно звернутися до останнього елементу масива використовується запис: v.length - 1.

Перебір елементів масиву

Найчастіше властивість length використовується для перебору елементів масиву в циклі:

```
var fruits =["яблуко", "банан", "полуниця", "персик"];
for(var i = 0; i < fruits.length; i++)
  document.write(fruits[i] + "<br>");
```

В даному прикладі передбачається, що елементи масиву розташовані безперервно і розпочинаються з першого елементу (з індексом 0). Якщо це не так, перед зверненням до кожного елементу масиву треба перевіряти, чи визначений він:

```
var fruits =["яблуко", "банан", "полуниця", "персик"];
for(var i = 0; i < fruits.length; i++)
  if (fruits[i] != undefined)
    document.write(fruits[i] + "<br>");
```

Багатовимірні масиви

Нагадаємо, що масиви JavaScript можуть містити в якості елементів інші масиви. Цю особливість можна використати для створення багатовимірних масивів. Для доступу до елементів в масиві масивів досить використати кадратные дужки двічі.

```
var matrix =[
  [3, 3, 3],
  [1, 5, 1],
  [2, 2, 2]
];
document.write(matrix[1][1]); //вибраний центральний елемент
```

Розберемо, що написано в прикладі, matrix - це масив масивів чисел. Будь-який елемент matrix[n] - це масив чисел. Для доступу до певного числа в масиві потрібно написати matrix[n][n], у других квадратних дужках вказується індекс елементу внутрішнього масиву.

```
//аналог попереднього запису - створення за допомогою конструктора
var table = new Array(3);
for(var i = 0; i < table.length; i++)
  table[i] = new Array(3);
```

```
// Ініціалізація масиву
for(var j = 0; j < table.length; j++){
  for(n = 0; n < table[j].length; n++){
    table[j][n] = j * n;
  }
}
document.write(table[1][1]); //вибір центрального елементу
```

Асоціативні масиви

Об'єкти можна використати в якості асоціативних масивів. Трохи теорії - асоціативні масиви (що називаються також хэш-таблицами) дозволяють замість індексів використати рядки. Застосування асоціативних масивів дуже схоже на використання імені властивості звичайного об'єкту, але в даному випадку при роботі у форматі масиву. Оскільки в JavaScript немає методів для роботи з асоціативними масивами вони застосовуються значно рідше, ніж звичайні, хоча точно також можуть бути корисні для зберігання інформації і полегшують запам'ятовування елементів, до яких треба отримати доступ.

```
var s_list = new Object();
 s_list["fat"] = "Товстий";
s_list["small"] = "Маленький";
s_list["name"] = "Гомер";
for (var x in s_list) //виведемо на екран усі елементи
  document.write(s_list[x] + "<br>");
```

1.4. Оператори вибору

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 *значення1*, то виконується група операторів *оператори1*, якщо дорівнює *значення2*, то виконується група операторів *оператори2* 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*.

1.5. 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 = {"a": "Літера a", "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 виключень

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*$_1$ — 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*$_2$, який має містити код опрацювання цього виключення. Якщо виключення неможливо на даному рівні опрацювати, то *оператор*$_2$ 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="Неправильний № місяця"};
document.write(monthName);
</script></body></html>
```

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

Помилку, про якій catch не знає, він не повинен обробляти.

Така техніка називається «*Кидок виключення»* : в catch(e) ми аналізуємо об'єкт помилки, і якщо він нам не підходить, то робимо throw e.

При цьому помилка «випадає» з try.catch назовні. Далі вона може бути спіймана або зовнішнім блоком try.catch (якщо є), або «повалить» скрипт.

1.5. Використання функцій

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() {
        alert("Це виклик функції")
        }
        </script>
</head><body>
<script>
Go();
</script></body></html>
```