==================== ТЕОРІЯ =======================

## Основи синтаксису

Скрипт - це послідовність команд (або файл-сценарій), які автоматизовують виконання деяких задач.

***Підключення (додавання) скриптів***

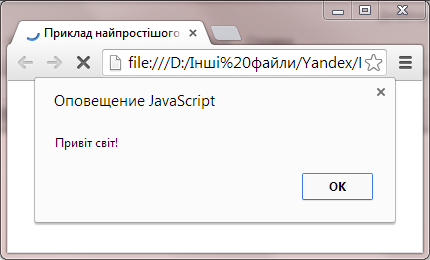
Скрипт можна додати в будь-яке місце HTML документу, за допомогою елемента script. Стандарти HTML4 вимагають обов'язкового вказування атрибуту type= "text/javascript" (у HTML5 таких вимог вже немає). Є кілька способів для додавання скипту на веб сторінку.

***1) безпосереднє включення в текст сторінки.***

Для цього оператори записують безпосередньо між тегами

|  |  |
| --- | --- |
| ***Загальна форма*** | ***Приклад*** |
| <scrip type="text/javascript">скрипт</script> | <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>Приклад найпростішого скрипта</title>  **<script >**  **alert("Привіт світ!")**  **</script>**  </head>  <body>  Скрипт виконано.  </body>  </html> |

Для його запуску необхідно створити html сторінку, що містить наступний текст і відкрити за допомогою браузера.



***2) підключення окремих файлів з скриптами (мають розлишення \*.js)***

Для цього необхідно створити текстовий файл, що містить скрипт і зберегти його з розширенням js. А на сторінку він підключається за допомогою елемента script з атрибутом src.

|  |  |
| --- | --- |
| Загальна форма | <script type="text/javascript" src="шлях/ім'я.js"> </script> |
| Приклад | ----------- Файл Hello.js---------  alert("Привіт світ!");  ----------- Сторінка HTML ---------  <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>Приклад найпростішого скрипта</title>  **<script src="Hello world.js"> </script>**  </head>  <body>  Скрипт виконано.  </body>  </html> |

Надалі будемо вважати, що html сторінка вже створена і ми редагуємо тільки віповідний файл зі скриптом.

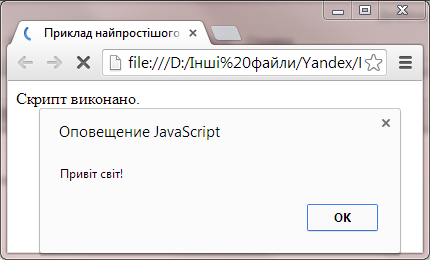
Варто зазначити, що інтерпретатор будує сторінку послідовно, відповідно знайшовши на ній скрипт - відкладає подальшу побудову до закінчення виконання скрипта. Якщо скрипт розмістити в заголовочній частині документу, то він виконається до створення тіла документа, відповідно спочатку з'явиться повідомлення на фоні порожнього документу і тільки після його закриття з'явиться надпис "Скрипт виконано". Якщо ж розмістити елемент скрипта в тілі документа після відповідного надпису, то вікно з'явиться на фоні документа, що вже містить надпис.

<body>

Скрипт виконано.

<script type="text/javascript" src="…../Hello world.js"></script>

</body>



Якщо документ містить кілька скриптів, то вони виконуються послідовно, в порядку розміщення на сторінці.

Оператори розділяються за допомогою символу ";". Перехід на новий рядок не розділяє оператори, але при виконанні скрипта інтерпретатор намагається виправити помилки і може сам додати розділення, але це не завжди приводить до правильного результату.

***3)порядок виконання скриптів***

Загалом, виконання скрипта здійснюється одразу після завантаження файлу, що його містить. При цьому поки скрипт не буде виконано завантаження сторінки буде призупинено. У зв’язку з цим часто скрипти розміщують в кінці документу. Але іноді такий підхід є неприйнятним, оскільки інформація, яка знаходиться на сторінці може залежати від результатів виконання скрипта. Тому з метою деблокування процесу завантаження сторінки можна використовувати при підключенні зовнішніх скриптів атрибуту async

<script **async** src="шлях до скрипта">

Таким чином дається команда на завантаження скрипта (він одразу буде виконаний після завнтаження) і продовжується обробка веб-сторінки.

Але якщо у програмі підлючається декілька файлів скриптів, то виконання кожного наступного здійснюється після закінчення виконання попереднього. Тому, наприклад, при підлюченні двох сриптів у порядку

<script src="1.js" ></script>

<script src="2.js" ></script>

Виконаним буде спочатку скрипт 1 і лише після закінчення його буде завантажуватися скрипт 2. Якщо використати атрибут async,

<script src="1.js" **async**></script>

<script src="2.js" **async**></script>

то виконаним буде той скрипт, який завантажиться раніше.

Іноді потрібно дотриматися порядку виконання скриптів і при цьому не блокувати процес обробки сторінки браузером. В цьому випадку необхідно використати атрибут defer.

<script src="1.js" defer></script>

<script src="2.js" defer></script>

В цьому блокування сторінки не буде і першим обов’язково буде виконано перший скрипт.

Слід також зазначити, що скрипти, які підключені з використанням атрибуту defer виконуються після того, як весь HTML документ буде опрацьовано браузером.

Приклад.

<script src="defer2.js" defer></script>

<script src="async1.js" async></script>

Скрипт async1.js буде буде виконано як тільки він буде завантажений. Скрипт defer2.js буде виконано після опрацювання документа браузеном.

**Типи даних**

У JS підтримуються :

* прості типи даних (відповідають скалярним значенням: числам, логічним значенням та ін.);
* складні типи даних посилання (об’єкти, функції).

|  |  |
| --- | --- |
| number | як цілі так і дробові, мають тип Number |
| string | текстові дані (як окремі символи, так і рядки) |
| boolean | логічні значення начення ( true (істина) і false (неправда)) |
| undefined | спеціальне значення, яке використовується як значення змінної, якій не присвоєно значення |
| object | тип даних посилання, використовується при описі об’єктів  (null - спеціальне значення, яке використовується для позначення «порожньої адреси») |

Для визначення типу даних може бути використана функцію **typeof**, яка повертає назву типу даних (рядок тексту)

typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof function myFunc(){}   //"function"

typeof {} // "object"

typeof null // "object"

typeof  {name:'John', age:34} // "object"

typeof  [1,2,3,4]             // "object"

Інші підходи до визначення типів даних будуть розглянуті пізніше.

**Коментарі**

*Коментар –* невиконувана частина тексту програми, що ігнорується компілятором і служить для вставки деяких поміток у програмі тільки для програміста. Коментарі бувають однорядковими та багаторядковими. *Однорядковий коментар* починається з символів «//» і закінчується у кінці рядка. Тобто всі символи до кінця рядка вважаються коментарем.

Приклад.

// Це однорядковий коментар \_

*Багаторядковий коментар* починається з символів «**/\***» і закінчується символами «**\*/**».

Приrлад.

**/\*** Це

багаторядковий

коментар **\*/**

*Літерали*

*Літерал* – це явно вказане значення декого типу. Розрізняють такі типи літералів:

|  |  |  |
| --- | --- | --- |
| *тип літерала* | *опис* | *приклади* |
| масив |  | *var a=[]*  *var b=[23,12,89]* |
| цілого типу | у десятковій системі числення – звичне нам ціле число | 125, -89, 108; |
| дійсного типу | у форматі з фіксованою крапкою – у записі числа є крапка, що розділяє цілу і дробову частити | 25.69, 2.0, 145.058 |
| у форматі з плаваючою крапкою – у записі використано символ «е» або «Е», що розділяє мантису від порядку | 3.5е5, 3е12, 678е2 |
| символьний | заданий у явному вигляді рядок у одинарних або подвійних лапках | var a=”Hello”  var b = “Peace”  var c=”” |
| прості ескейп-послідовності – службові символи починаються з символу «\» | \n -перехід на новий рядок,  \t -горизонтальна табуляція,  \’ - апостороф, і т. д. |
|  | ` … текст … ${ значення } … текст ` | // Інтерполяція рядків  var name = "Бобби", time ="сегодня";  `Привет ${name}, как ты ${time}?`  “Привет” +name+”, как ты”+time |
| логічний | може приймати два значення | true,  false |
| літерал об’єкта |  | var a={}  var b={name=”Ivan”} |
| порожньої адреси (нульовий літерал) | використовується у випадку, коли покажчик не містить жодної адреси | null |

| **Специальные символы в JavaScript** | |
| --- | --- |
| **Символ** | **Значение** |
| \b | Возврат (Backspace) |
| \f | Перевод или прогон страницы (Form feed) |
| \n | Перевод строки (New line) |
| \r | Возврат каретки (Carriage return) |
| \t | Табуляция (Tab) |
| \v | Вертикальная табуляция (Vertical tab) |
| \' | Апостроф или одинарная кавычка |
| \" | Двойная кавычка |
| \\ | Обратная косая черта (Backslash) |
| \*XXX* | Символ в кодировке Latin-1, представленный тремя восьмеричными числами *XXX* от 0 до 377. Например, \251 (символ ©). |
| \x*XX* | Символ в кодировке Latin-1, представленный двумя шестнадцатеричными числами *XX* от 00 до FF. Например, \xA9 (символ ©). |
| \u*XXXX* | Символ в Unicode, представленный четырьмя шестнадцатеричными числами *XXXX*. Например, \u00A9 (символ ©). |
| \u*{XXXXX}* | Символ в UTF-32BE. Например, \u{2F804} обозначает то же, что обычная запись \uD87E\uDC04. |

## Опис змінних

Змінна – іменована область пам’яті, призначена для збереження певних даних, які можуть змінюватися. Для використання змінних їх слід описати (оголосити).

Опис може бути здійснено декількома способами:

* як глобальна змінна (як властивість window);
* за допомогою var;
* за допомогою let.

***Опис змінних, як властивостей об’єкта window***

У попередніх версіях JS дозволялось створювати змінні лише задаючи їх ім’я і значення.

|  |  |
| --- | --- |
| Загальна форма | Приклад |
| змінна = значення; | a = 90; // рівносильно window.a=90 |

Це по суті призводило до створення відповідної властивості об’єкта **window** (глобальної змінної) і могло призвести до конфлікту імен у різних скриптах. Тому на даний момент такий підхід не використовуються, а при використанні сучасних стандартів (використанні режиму "use strict") -- призводить до помилки.

"use strict";

a = 90; // error: a is not defined

*Опис змінних за допомогою var*

|  |  |
| --- | --- |
| Загальна форма | Приклад |
| var змінна; | var a;  var b,c; |

При оголошенні також можна задавати початкове значення змінної.

|  |  |
| --- | --- |
| Загальна форма | Приклад |
| var змінна = початкове значення; | var a=7;  var b= “Hello”;  a=”Hi” |

Ім'я кожної змінної повинно бути унікальним правильним ідентифікатором. Правильний ідентифікатор (назва змінної) на мові JS повинен містити тільки букви латинського алфавіту, цифри та символи "$" і "\_". При цьому ідентифікатор не може починатись з цифри і співпадати із зарезервованими словами.

Правильні ідентифікатори: MyID, identifier, x, $, abc123, my\_id, \_true.

Неправильні:

my id //містить пробіл

Ok! // містить недозволений знак оклику !

12th //починається з цифри

100 //починається з цифри

for //зарезервоване слово.

Стосовно змінних є декілька важливих зауважень:

* мова JS є регістро-чутливою (розрізняє ВЕЛИКІ та малі букви). Тому myid та MyID - це різні ідентифікатори;
* змінні оголошуються глобально ( вони доступні в будь-якому скрипті, що є на сторінці);
* змінні в JS не є типізованими ( не можна строго вказати тип змінної, вона може змінювати свій тип, залежно від значення).

***Приклад.***

var x; // x – змінна (значення undefined);

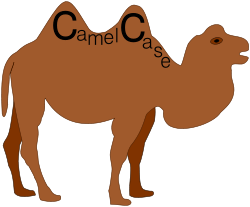
var y = 2;, у - змінна числового типу, рівна 2

x = y / 8; // х - числового типу, рівна 0,25

x = "String"; // змінює свій тип на рядковий.

***Вибір імені змінної***

**CamelCase** — стиль (нотація) формування ідентифікаторів при якому декілька слів пишуться разом (без пробілів) і при цьому кожне слово починається з великої літери



Розрізняють два види CamelCase-нотації у з

алежності від того, велика чи маленька перша літера:

* **UpperCamelCase** (*PascalCase*) (перша літера велика);
* **lowerCamelCase** (перша літера маленька).

Приклади імен: BackColor, backColor, CamelCase, itemCount, totalSum.

***Локальні змінні***

Іноді необхідно описати змінну, яка буде видима (доступна) тільки в межах блоку (в межах циклу, умовного оператора чи блоку, описаного з використанням фігурних дужок)

{

… блок …

}

Для цього необхідно описати ці змінні з використанням let

|  |  |
| --- | --- |
| Загальна форма | Приклад |
| {  let змінна;  } | {  let a;  a=23; 🡨 **Доступ можливий**  let b,c;  ...  a=a+5; 🡨 **Доступ можливий**  }  a=9; 🡨 **ПОМИЛКА! Змінна недоступна** |

## Опис констант

Константи описуються за допомогою оператора const.

|  |  |
| --- | --- |
| Загальна форма | Приклад |
| const КОНСТАНТА = значення; | const RATE=15; |

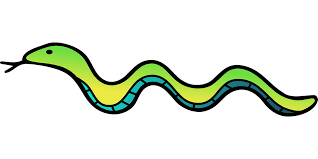
Константи, на відміну від змінних, не можуть змінювати свого значення. Традиційно імена констант записують великими літерами.

***Приклад.*** const PI = 3.14, ZERO = 0, NAME = "Name";

**Snake case** – стиль (нотація) формування ідентифікаторів, коли окремі слова у назві відділяються символом нижнього підкреслювання (“\_”). При цьому слова можуть писатися як маленькими, так і великими літерами. У багатьох мовах програмування ідентифікатори констант прийнято формувати з використанням цього стилю.

Приклад: **SCREAMING\_SNAKE\_CASE, MAX\_LENGTH, LOWER\_BOUND**

**const MAX\_LENGTH=100;**



## Введення та виведення даних

***Виведення даних***

Виведення даних може бути здійснено багатьма способами:

* виведення з використанням діалогового вікна **window.alert();**
* виведення в консоль браузера **console.log()**;
* виведення на сторінку HTML з використанням **document.write()**;
* виведення в HTML елемент, з використанням властивостей **innerHTML, innerText;**
* за допомогою форм.

Розглянемо деякі з них.

***Виведення даних за допомогою alert***

За допомогою виклику alert (або ж window.alert) відображаємо діалогове вікно, у якому і відображається вказана інформація.

|  |  |  |
| --- | --- | --- |
| Загальна форма | Приклад | На екрані |
| alert (текст повідомлення); | alert (“Hello”); |  |

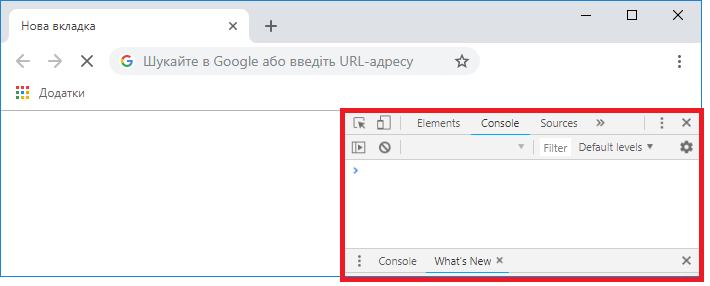
За допомогою цього діалогового вінка можна виводити і не тільки текстові дані, а й дані інших типів. При цьому параметр (якщо він не текстового типу) приводитися до текстового типу.

Приклад. Виведення значення змінної

|  |  |
| --- | --- |
| Загальна форма | alert (“ім’я змінної =”+змінна); |
| Приклад (вивести суму) | var a=9;  var b=7;  var sum=a+b;  **alert(“Sum=”+sum);** |
| На екрані |  |

***Виведення даних за допомогою console.log***

При використанні даного підходу інформація відображається у вікні консолі браузера (у Chrome для відображення вінка консолі необхідно скористатися F12).



|  |  |
| --- | --- |
| Загальна форма | ***console.log*** (“текстове повідомлення”); |
| Приклад (вивести суму) | var a = 9;  var b = 7;  var sum = a + b;  **console.log("Sum ="+sum);** |
| На екрані |  |

***Виведення даних за допомогою document.write***

При використанні даного підходу інформація відображається безпосередньо на HTML сторінці. Зауважимо, що при цьому можна виводити також розмітку HTML. Наприклад, для переходу на наступний рядок необхідно вивести текст тегу “<br>”.

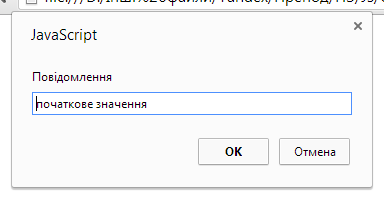
|  |  |
| --- | --- |
| Загальна форма | ***document.write*** (“текстове повідомлення”); |
| Приклад (вивести суму) | **document.write**("Приклад використання <br>");  var a = 9;  var b = 7;  var sum = a + b;  **document.write("Sum ="+sum);** |
| На екрані |  |

**Введення значення змінних**

Для введення даних будемо користуватись функцією prompt;

prompt (повідомлення, початкове значення);

В результаті виводиться модальне вікно, яке містить інформаційне повідомлення, поле для вводу з початковим значенням та кнопки Ok та Відміна. Зазвичай як другий параметр використовується порожній рядок. Якщо його не вказати то в деяких браузерах значення буде undefined.



Функція повертає значення, що було введено користувачем у відповідне поле. Значення повертається як текстова величина. Якщо необхідно ввести числове значення то необхідно скористатись функціями переведення parseInt, parseFloat чи оператором збереження знаку + . Детальніше вони будуть розглянуті в розділі про числовий тип. В разі натискання кнопки відміни функція повертає null.

*Введення цілого числа*

|  |  |
| --- | --- |
| *Загальна форма* | *Приклад* |
| var змінна =parseInt(prompt(“Змінна=”, “0”)) | var a= parseInt(prompt(“a=”, “0”)) |

*Введення дійсного числа*

|  |  |
| --- | --- |
| *Загальна форма* | *Приклад* |
| var змінна =parseFloat(prompt(“Змінна=”, “0”)) | var a= parseFloat(prompt(“a=”, “0”)) |

*Приклад.*  Знайти суму двох чисел

//1. Описуємо і вводимо значення змінних

var number1 = parseInt(prompt ('Введіть ціле число','0'));

var number2 = parseInt(prompt ('Введіть ціле число','0'));

//2. Знаходимо результат

var sum = number1 + number2;

//3. Виводимо результат

alert(“Sum=”+sum);

Зауважимо при цьому, що браузер може заборонити виведення модальних вікон.

## Типи даних

### Числа

Всі числа в JavaScript, як цілі так і дробові, мають тип Number. Для запису числа використовуються арабські цифри. Для відділення дробової частини застосовується крапка.

В JavaScript можна записувати числа не тільки в десятковій, але і в шістнадцятковій (запис починається з 0x), а також вісімковій (починається з 0) системах числення: 0xFF це 255 в шістнадцятковій системі, а 010 - 8 в вісімковій системі. Також доступний запис «з плаваючою крапкою», який виглядає як мантиса e степінь. Наприклад: 1e3 - це 1\*103, тобто 1000, 3e-5 =3\*10-5=0.00003.

В пам’яті числа зберігаються в форматі IEEE-754, також відомому як «double precision». В ньому на число виділяється рівно 8 байт (64 біти). Деякі дробові числа в десятковому форматі мають скінченний запис, а в двійковій системі числення це нескінченна дріб. Так нескінченної дробом є 0.110=0,(00011)2. Двійкове значення нескінченних дробів зберігається лише до певного знака, тому виникає неточність. Переконатись в цьому можна на прикладі .

alert(0.1 + 0.2); // 0.30000000000000004, а не 0.3 як очікувалось

Втрата точності відбувається і з дуже великими числами:

alert(9999999999999999999999); //1e+22

Кожне число має метод toFixed([кількість знаків]), який округлює число до заданої точності і повертає результат у вигляді рядка. Кількість цифр після десяткового знак повинна знаходитися в діапазоні від 0 до 20 включно. Якщо не вказано, то вважається рівним 0. Число округлюється при необхідності, і дробова частина доповнюється нулями до потрібної довжини. Якщо число більше 1e +21, то метод повертає рядкове представлення в експоненціальному записі.

var n = 12.345;

alert (n.toFixed (1)); / / "12.3"

Викликати його можна не тільки у змінних числового типу, а й у числових літералів.

alert ((12.345).toFixed (1)); / / "12.3"

**Арифметичні вирази**

Арифметичний вираз – це вираз, в результаті обчислення якого одержуємо число. Над числами можна виконувати звичайні арифметичні та деякі спеціальні операції.

**Бінарні операції** (необхідно мати два операнди)

|  |  |
| --- | --- |
| x+y | Сума |
| x-y | Різниця |
| x\*y | Добуток |
| x/y | Частка |
| x%y | Остача від ділення |

**Унарні оператори (**необхідним є один операнд**)**

|  |  |
| --- | --- |
| -x | Зміна знаку на протилежний |
| +x | Збереження знаку. |
| x++,++x | Інкримент: збільшення на 1 |
| x--,--xx | Декримент: зменшення на 1 |

Інкремент та декремент можна застосувати тільки до змінної. Код 5++ дасть помилку. Різниця між префіксною та постфіксною формами виявляються там, де потрібно не тільки збільшити / зменшити змінну, але й отримати результат. При використанні постфіксної форми спочатку відбуваються обрахунки, а потім змінюється значення змінної. З префіксною формою навпаки: спочатку змінюється значення, а потім відбуваються обрахунки. Нехай х=1, тоді після виконання у=3\*(х++) буде у=3, х=2. Після у=3\*(++х) отримаємо у=6, х=2.

В більшості мов програмування ділення на нуль викликає помилку. В JavaScript така операція є допустимою, в результаті її виконання отримуємо нескінченність. Infinity - особливе числове значення, яке поводить себе як математична нескінченність ∞. Infinity більше ніж будь-яке число. В результаті додавання/віднімання/множення/ділення нескінченності на число отримуємо нескінченність. Відношення числа до нескінченності дає 0. Також існує мінус нескінченність ‑Infinity.

Якщо математична операція не може бути здійснена, то повертається спеціальне значення NaN (Not-A-Number, не число). Так воно отримується в результаті 0/0, ∞/∞, ∞-∞, sin(∞),  та ін. Значення NaN - єдине, у своєму роді, яке не дорівнює нічому, включаючи себе. В результаті будь-якої операція з NaN отримується NaN. Значення NaN можна перевірити спеціальною функцією isNaN(n), яка повертає true якщо аргумент - NaN і false для будь-якого іншого значення. Функція isFinite (n) повертає true тільки тоді, коли n звичайне число, а false якщо n одне зі спеціальних значень: NaN, Infinity чи -Infinity. Якщо аргумент не число, то він буде автоматично перетворений на число.

При строгому перетворенні, якщо рядок не є точно числом, то отримаємо NaN. Єдиний виняток - пробіли на початку та в кінці рядка, які ігноруються. А також порожній рядок перетворюється на 0.

Якщо один з операндів арифметичної операцій, крім додавання, є числом тоді інший операнд буде також перетворений на число. Унарні арифметичні операції також перетворюють аргумент в число, тому найпростішим способом перетворення на число – унарна операція збереження знаку.

У web-програмуванні багато значення не є точно числами, але мають числовий зміст. Наприклад метрики CSS: 10pt, -1cm чи 12%. Для їх зручного перетворення є функції м’якого перетворення. Вони перетворюють рядок символ за символом, поки це можливо. Звичайно, існують ситуації, коли вони повертають NaN. Це відбувається при помилці в першому ж символі. parseInt перетворює рядок в ціле число, а parseFloat - дробове. parseInt може мати і другий аргумент, що вказує систему числення, з якої відбувається перевід. Це робити не обов’язково, якщо запис починається з 0х – явної вказівки на шістнадцядкову систему числення.

parseInt ('12px ') // 12

parseFloat ('12 .3.4') // 12.3

parseInt ('a123') // NaN

parseInt('0хFF') // 255

В JavaScript нема вбудованих математичних констант та функцій, але вони реалізовані як властивості та методи простору імен Math.

Основні методи:

|  |  |
| --- | --- |
| Math.floor (x) | Округлення вниз |
| Math.ceil (x) | Округлення вгору |
| Math.round (x) | Округлення до найближчого цілого |
| Math.acos (x) | Арккосинус x (в радіанах) |
| Math.asin (x) | Арксинус x (в радіанах) |
| Math.atan (x) | Арктангенс x (в радіанах) |
| Math.atan2 (y, x) | Арктангенс кута до точки (y, x). Виклик еквівалентний Math.atan (y / x). |
| Math.sin (x) | Синус x (в радіанах) |
| Math.cos (x) | Косинус x (в радіанах) |
| Math.tan (x) | Тангенс x (в радіанах) |
| Math.sqrt (x) | Квадратний корінь з x. |
| Math.log (x) | Натуральний логарифм x. |
| Math.pow (x, у) | Піднесення до степеня, повертає xу, наприклад Math.pow (2,3) = 8. |
| Math.abs (x) | Абсолютне значення числа |
| Math.exp (x) | Експонента ex. |
| Math.max (a, b, c ...) | Найбільший зі списку аргументів |
| Math.min (a, b, c ...) | Найменший зі списку аргументів |
| Math.random (x) | Псевдо-випадкове число в інтервалі [0,1) - тобто між 0 (включно) та 1 (не включаючи). Генератор випадкових чисел ініціалізуется поточним часом. |
| Math.PI | Число π |
| Math.Е | Константа Ейлера, е |

*Приклад.*  Скласти скрипт для генерування випадкового числа з проміжку [a,b];

var a=+prompt('a=',''), b=+prompt('b=','');

var rnd=a+Math.floor((b-a+1)\*Math.random());

alert(rnd);

==================== ПРИКЛАДИ =====================

Задача 1. Знайти суму, добуток та частку двох дійсних чисел. Результат вивести у формі таблиці

Задача 2. Дано рік народження. Знайти кількість років.

Задача 3. Дано вартість одиниці товару і кількість. Знайти загальну вартість.

Задача 4. Дано довжину у сантиметрах. Визначати скільки це метрів і кілометрів.

Задача 5. Дано кількість секунд. Визначити скільки це годин і хвилин.

Задача 6. З клавіатури вводяться вартість товару та кількість одиниць 3 товарів. Обчислити вартість кожного товару окремо та загальну вартість. Вивести чек (як у супермаркеті) використовуючи елементи розмітки.