### Рядки

Посилання:

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String>

<https://www.w3schools.com/js/js_strings.asp>

<https://metanit.com/web/javascript/6.1.php>

**Літерали рядків**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Запис літералів за допомогою одинарних чи подвійних лапок | ‘ рядок\_тексту ’  “ рядок\_тексту ”  var змінна = “рядок\_тексту” | Літерал у одинарних лапках: 'Це рядок'  Літерал у подвійних лапках: ”це теж рядок”  **Відкриваючі і закриваючі лапки повинні бути ідентичні:**'помилка”, ”теж помилка'  Змінна рядкового типу:  var str='Приклад'; |
| Запис великих літералів у декількох рядках:   * записати в кінці символ «\»; * використовувати шаблони текстів (записувати рядок у символах “ ` ”); при цьому зберігаються також символи переходу на новий рядок. | “ перша\_частина\_тексту \  друга\_частина\_тексту ”  ' перша\_частина\_тексту  друга\_частина\_тексту ' | var s = "Hello \  world";  alert(s);    ---------------------------------------------  var s = `Hello  world`;  alert(s); |
| Ескейп послідовності | прості ескейп-послідовності – службові символи починаються з символу «\» | \n - перехід на новий рядок,  \t - горизонтальна табуляція,  \r – повернення каретки |
| Символи (незалежно від таблиці кодування, яка була використана при написанні скрипта, рядки завжди кодуються з використанням Unicode ). Символи можуть бути вставлені у текст за допомогою їх кодів у таблиці кодування Unicode у форматі \uNNNN | ескейп-послідовності Unicode – символи «\u», за якими вказують код символу з чотирьох цифр у шістнадцятковій системі числення | Наприклад, \ u00A9 - символу копірайту © |
| Екранування символів  Якщо необхідно вставити у рядок деякий символ, який має спеціальне призначення, то необхідно його «екранувати» за допомогою символу “\” | \ символ\_спеціального\_призначення | \’ – апостороф  \” – подвійні лапки  var s1='здоров\'я',  s2=”готель \”Ужгород\” ” ,  s3=”здоров'я”; |
| Рядки як об’єкти  (бажано не використовувати, оскільки призводить до сповільнення роботи програми) | var змінна =new String( рядок\_тексту ) | var x= "John"; var y= new String("John");  // typeof x will return string  // typeof y will return object |
| Створення рядка, як багаторазової копії заданого фрагмента | рядок . **repeat**( кількість\_повторень ) | let hello = "hello ";  console.log(hello.**repeat**(3));  //hello hello hello  -----------------------------------  alert("hello".**repeat**(3)); |
| Шаблони рядків  (шаблони дозволяють вставити значення у рядок); шаблони записуються у похилих лапках  ` текст `  значення вставляють у фігурних дужках, перед якими записують символ “$”  ` . . . ${ значення} . . . ` | **`**текст **${** **значення\_для\_вставки** **}** текст ` | let name = "Tom";  let hello = `Hello **${name}**`;  console.log(hello);     // Hello Tom    let age = 23;  let info = `**${name}** is **${age}** years old`;  console.log(info);   // Tom is 23 years old |
|  |  |  |

**Зміна регістра**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Приведення усіх символів до нижнього регістру | рядок . toLowerCase() | let hello = "Привет Том";  console.log(hello.**toLowerCase**());  // привет том |
| Приведення усіх символів до верхнього регістру | рядок . toUpperCase() | let hello = "Привет Том";  console.log(hello.**toUpperCase**());  // ПРИВЕТ ТОМ |

**Звертання до символів**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| зчитування символу – charAt | рядок . **charAt**( індекс ) | let hello = "Привет Том";  console.log(hello.**charAt**(2));       // и |
| зчитування символу з використанням індексатора (використати квадратні дужки) | рядок **[** індекс **]** | let hello = "Привет Том";  console.log(hello **[ 2 ]** ));   // 1080 |
| зчитування коду символу | рядок . **charCodeAt** ( індекс ) | let hello = «Привет Том»;  console.log(hello.**charCodeAt**(2));   // 1080 |
| одержання символу за його кодом | String . **fromCharCode**( код\_символу ) | alert( String.**fromCharCode**(1072) ); // 'а' |

**Об’єднання**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| З використанням методу concat  (створюється новий рядок, який є об’єднанням даних) | рядок1 . **concat** ( рядок2 ) | let hello = "Привет ";  let world = "мир";  hello = hello.**concat**(world);  console.log(hello); // Привет мир |
| З використанням оператора «+»  (створюється новий рядок, який є об’єднанням даних) | рядок1 + рядок2 | let hello = "Привет ";  let world = "мир";  hello = hello **+** world ;  console.log(hello); // Привет мир |

**Вибір підрядка**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Копіювання з вказаної позиції вказаної кількості | рядок . **substr**(start [, length]) | let hello = "привет мир. пока мир";  let bye = hello.substr(12, 4);  console.log(bye); // пока |
| копіювання з початкової до кінцевої позиції | рядок . **substring**(start [, end]) | let hello = "привет мир. пока мир";  let world = hello.substring(7, 10); // с 7-го по 10-й индекс  console.log(world); // мир |
| копіювання з початкової до кінцевої позиції | рядок .**slice**(start [, end]) |  |

**Пошук у рядку**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| визначення входження чи невходження деякого підрядка у рядку (повертає false, якщо підрядок не входить у рядок) | рядок . **includes** ( підрядок ) | let hello = "привет мир. пока мир";    console.log(hello.**includes**("мир")); // true  console.log(hello.**includes**("миг")); // false |
| пошук першого входження деякого підрядка у рядку (повертає -1, якщо підрядок не входить у рядок) | рядок . **indexOf**( підрядок ) | let hello = "привет мир. пока мир";  let key = "мир";  let lastPos = hello.**indexOf**(key);  console.log("Первое вхождение: ", firstPos);    // 7 |
| знаходження останнього входження деякого фрагмента (пошук з кінця) | рядок . **lastIndexOf**( підрядок ) | let hello = "привет мир. пока мир";  let key = "мир";  let lastPos = hello.**lastIndexOf**(key);  console.log("Последнее вхождение: ", lastPos);  // 17 |
| чи починається з вказаного фрагменту | startsWith() | let hello = "let me speak from my heart";  console.log(hello.startsWith("let"));       // true  console.log(hello.startsWith("Let"));       // false  console.log(hello.startsWith("lets"));      // false |
| чи закінчується вказаним фрагментом | **endsWith()** | let hello = "let me speak from my heart";  console.log(hello.endsWith("heart"));       // true  console.log(hello.startsWith("bart"));      // false |

**Порівняння**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Порівння з врахуванням національних особливостей | рядок . localeCompare( інший\_рядок )  a[a.length-1]. localeCompare(b[b.length-1]) | var str = "Ёлки";  alert( str.localeCompare("Яблони") ); // -1 |
|  |  |  |

**Заміна фрагментів у рядку**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Заміна старого фрагменту тексту на новий | str.replace(regexp|substr, newSubStr|function[, flags])  рядок.replace(**старий\_фрагмент**, новий\_фрагмент) | let hello = "Добрый день";  hello = hello.replace("день", "вечер");  console.log(hello); // Добрый вечер |
|  |  |  |
|  |  |  |

**Інші методи**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Вилучення пробілів з початку і кінця рядка | trim(): | et hello = "   Привет Том  ";  let beforeLength = hello.length;  hello = hello.trim();  let afterLength = hello.length;  console.log("Длина строки до: ", beforeLength);     // 15  console.log("Длина строки после: ", afterLength);   // 10 |
|  |  |  |
|  |  |  |
| Розділення рядка  (формується масив з елементів між символами-розділювачами) | рядок . **split**( символ\_розділювач ) | var message = "Сегодня была прекрасная погода";  var stringArray = message.**split**(" ");  for(var str in stringArray)      console.log(stringArray[str]); |
| формування рядка шляхом об’єднання елементів масиву та використання символу-розділювача | масив. **join**( символ\_розділювач ) | var s = [1, 2, 3].join('-');  document.write(s); //1-2-3 |

**Регулярні вирази**

Регулярні вирази є об’єктами типу **RegExp** . Вони дозволяють описувати шаблони фрагментів тексту.

**Задання регулярного виразу**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| літерал | / шаблон / модифікатори ; | var myExp = /hello/; |
| створення як об’єкта | new RegExp(“шаблон”, “модифікатори”) | var myExp = new RegExp("hello");  var r2 = new RegExp("[^xyz]", "g") |

Модифікатори

|  |  |  |
| --- | --- | --- |
| модиікатор | Загальна форма | Приклад |
| **і** – ігнорування регістру | / регулярний\_вираз / **і** | var str = "Visit W3Schools";  var patt1 = /w3schools/i;  var result = str.match(patt1); |
| **g –** знаходження усіх фрагментів, які відповідають шаблону (пошук не зупиниться після знаходження першого фрагменту) | / регулярний\_вираз / **g** | var str = "Is this all there is?";  var patt1 = /is/g;  var result = str.match(patt1);  document.write(result); |
| **m –** пошук у багаторядковому тексті | / регулярний\_вираз / **m** | var str = "\nIs th\nis it?";  var patt1 = /^is/m;  var result = str.match(patt1); |

Допустимими також є комбінації модифікаторів

Приклад

var menu = "Завтрак: каша, чай. Обед: суп, чай. Ужин: салат, чай.";

var exp = /чай/**gi**; //🡨 ігнорування регістрів і глобальний пошук

menu = menu.replace(exp, "кофе");

document.write(menu); // Завтрак: каша, кофе. Обед: суп, кофе. Ужин: салат, кофе.

Групи символів

|  |  |  |
| --- | --- | --- |
| задаються за допомогою квадратних дужок |  |  |
| Один з вказаних символів  (у квадратних дужках перераховуємо допустимі символи) | [ список\_символів ] | var initialText = "обороноспособность";  var exp = **/[абв]/; //🡨 повинен містити одну із букв**  var result = exp.test(initialText);  document.write(result + "<br/>"); // true    initialText = "город";  result = exp.test(initialText);  document.write(result); // false |
| Хоча би один не з вказаних символів  (перераховуємо заборонені символи і спочатку ставимо символ « ^ ») | [ ^ список\_символів ] | var str = "Do you know if this is all there is?";  var patt1 = /[^is]/gi;  var result = str.match(patt1);  //D,o, ,y,o,u, ,k,n,o,w, ,f, ,t,h, , ,a,l,l, ,t,h,e,r,e, ,? |
| Символи з діапазону | [ start - end] | var initialText = "обороноспособность";  var exp= **/[а-я]/; //🡨хоча би один символ з діапазону**  var result = exp.test(initialText);  document.write(result + "<br/>"); // true  //-----------------------  initialText = "3di0789";  result = exp.test(initialText);  document.write(result); // false  //-----------------------  var initialText = '<img src= "picture.png" />';  var exp = /**[a-z]**+\.(png|jpg)/i;  var result = initialText.match(exp);  result.forEach(function(value, index, array){      document.write(value + "<br/>");  }) |
| Не лише символи з діапазону | [^ start - end] | var initialText = "обороноспособность";  var exp = /**[^а-я]**/; **//🡨не лише символи з діапазону**  var result = exp.test(initialText);  document.write(result + "<br/>"); // false    initialText = "3di0789";  exp = **/[^0-9]/**; **//🡨не лише цифри**  result = exp.test(initialText);  document.write(result); // true |
| альтернативні варіанти  (варіанти перераховуються у круглих дужках, розідені символами « | » | (варіант1 | варіант2| …) | var str = "re, green, red, green, gren, gr, blue, yellow";  var patt1 = /(**red|green)**/g;  var result = str.match(patt1);  //green,red,green  //-----------------------  var str = "01234567890123456789";  var patt1 = /(0|5|7)/g;  var result = str.match(patt1);  //0,5,7,0,5,7 |

Метасимволи

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| . – довільний символ за винятком символу кінця рядка або нового рядка |  | var str = "That's hot!";  var patt1 = **/h.t/g**;  var result = str.match(patt1);  //hat,hot |
| \d – довільна цифра (0–9) |  |  |
| \D -- не цифра |  |  |
| \w – буква (латинського алфавіту), цифра, символ нижнього підкреслювання (A–Z, a–z, 0–9) |  |  |
| \W – не буква, не цифра, не символ нижнього підереслювання |  |  |
| \s – пробільний символ |  |  |
| \S – не пробільний символ |  |  |
| \n – символ нового рядка |  |  |
| \t – символ табуляції |  |  |

Квантифікатори

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| **+** -- принаймні одне входження (попереднього символу) | символ + | var str = "Hellooo World! Hello W3Schools!";  var patt1 = **/o+/g**;  var result = str.match(patt1);  //ooo,o,o,oo |
| \* -- довільна кількість або відсутність (попереднього символу) | символ \* | var str = "Hellooo World! Hello W3Schools!";  var patt1 = **/lo\*/g;**  var result = str.match(patt1);  //l,looo,l,l,lo,l |
| ? – входить один раз або відсутність | символ ? | var str = "1, 100 or 1000?";  var patt1 = **/10?/g**;  ar result = str.match(patt1);  //1,10,10 |
| **рівно n символів** | **{n}** | var str = "100, 1000 or 10000?";  var patt1 = /\d{4}/g;  var result = str.match(patt1);  //1000,1000 |
| **не менше n символів** | **{n,}** | var str = "100, 1000 or 10000?"; var patt1 = /\d{3,}/g;  //100,1000,10000 |
| **від n до m** | **{n, m}** | var str = "100, 1000 or 10000?"; var patt1 = /\d{3,4}/g;  //100,1000,1000 |

/\d{1,2}\/\d{1,2}\/\d{4}/g

12 / 11 / 2001

Позиція (початок, кінець)

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| початок рядка « ^ » |  | var str = "Is this his";  var patt1 = /^Is/g;  var result = str.match(patt1);  //Is |
| кінець рядка « $ » |  | var str = "Is this his";  var patt1 = /is$/g;  //is  var str = "Is this his";  var patt1 = /\w\*is$/g;  //his |
| початок або кінець слова « \b » |  | var str = "Visit W3Schools";  var patt1 = /\bW3/g;  var result = str.match(patt1);  // W3 |
| не початок або кінець слова «\B» |  | var str = "Visit W3Schools";  var patt1 = /\BSchool/g;  var result = str.match(patt1);  // School |
| групування, запам’ятовування знайдених фрагментів (якщо вказуємо у круглих дужках)  (для доступу використовується: $номер) | **(pattern)** | var re = /(\w+)\s(\w+)/i;  var str = "Mikhail Bulgakov";  document.write(str.replace(re, "$2, $1"))  // Bulgakov, Mikhail |
| \ номер\_збереженої\_групи |  | var str = "He said: \"She's the one!\".";  var reg = /(['"])(.\*?)**\1**/g;  //Закривача лапка як і відкриваюча  alert( str.match(reg) ); // "She's the one |
| не запам’ятовуються, використовуються для групування частин | **(?:pattern)** | /ко(?:т|шка)/ - это краткая запись выражения /кот|кошка/. |
| якщо після слідує вказаний фрагмент | (?= фрагмент ) | var str = "Is this all there is";  var patt1 = /...is**(?= all)**/;  var result = str.match(patt1);  //this |
| якщо після не слідує вказаний фрагмент | (?! фрагмент ) | var str = "Is this all there is";  var patt1 = /is**(?! all)/**gi;  var result = str.match(patt1);  //Is,is |
|  |  |  |

**Деякі методи**

|  |  |  |
| --- | --- | --- |
|  | Загальна форма | Приклад |
| Перевірка рядка тексту на відповідність регулярному виразу (повертає true/false) | регулярний\_вираз .**test**(рядок\_тексту) | var initialText = "hello world!";  var exp = /hello/;  var result = exp.**test**(initialText);  document.write(result + "<br/>"); // true    initialText = "beautifull wheather";  result = exp.**test**(initialText);  document.write(result);  // false - в строке initialText нет "hello" |
| Повертає частину рядка, яка відповідає регулярному виразу (null, якщо немає) | регулярний\_вираз .**exec** (рядок\_тексту) | var initialText = "hello world!";  var exp = /hello/;  var result = exp.**exec**(initialText);  document.write(result + "<br/>"); // hello    initialText = "beautifull wheather";  result = exp.**exec**(initialText);  document.write(result); // null  //------------------------------------ |
| Знаходження позиції першого входження фрагменту, що відповідає шаблону (регулярному виразу) у рядку (-1 якщо немає) | рядок .**search**( регуляний\_вираз ) | var str = "Visit W3Schools";  var n = str.**search**(/w3schools/i); //6 |
| Пошук усіх фрагментів, що відповідають заданому фрагменту (null, якщо немає) | рядок . **match** ( регуляний\_вираз ) | var initialText = "Он пришел домой и сделал домашнюю работу";  var exp = /дом[а-я]\*/gi;  var result = initialText.**match**(exp);  result.forEach(function(value, index, array){        document.write(value + "<br/>");  })  домой  домашнюю |
| Розділення рядка за шаблоном фрагмента-розділювача (не обов’язково одного символу) | рядок . **split** ( регуляний\_вираз ) | var initialText = "Сегодня была прекрасная погода";  var exp = /\s/;  var result = initialText.**split**(exp);  result.forEach(function(value, index, array){        document.write(value + "<br/>");  }) |
| Заміна значень заданих шаблоном на новий фрагмент тексту | рядок.replace(**регулярний\_вираз**, новий\_фрагмент) | var **re = /яблоки/gi;**  var str = 'Яблоки круглые и яблоки сочные.';  var newstr = str.**replace**(**re**, 'апельсины');  console.log(newstr); // апельсины круглые и апельсины сочные.  ------ |
| Заміна рядка на основі функції формування нового рядка | рядок.replace(**регулярний\_вираз**, функція) | function replacer(match, p1, p2, p3, offset, string) {  // p1 - не цифры, p2 - цифры, p3 - не буквы и не цифры  return [p1, p2, p3].join(' - ');  }  var newString = 'abc12345#$\*%'.replace(/([^\d]\*)(\d\*)([^\w]\*)/, replacer);  //abc - 12345 - #$\*% |

Жадний і «лінивий» пошук ( квантифікатор ?)

|  |  |  |
| --- | --- | --- |
| Жадний пошук  (вибір максимально можливої кількості (перехід до наступного символу максимальну допустиму кількість раз) символів, які підходять за шаблоном) |  | var reg = /".+"/g;  var str = 'a "witch" and her "broom" is one';  alert( str.match(reg) ); // "witch" and her "broom" |
| «лінивий» пошук  (перехід до наступного символу згідно з шаблоном мінімальну можливу кількість )  (для цього ставимо знак «?» після квантифікатора: \*? , +? , ??) |  | var reg = /".+?"/g;  var str = 'a "witch" and her "broom" is one';  alert( str.match(reg) ); // witch, broom |

У JavaScript будь-які текстові дані є рядками. Не існує окремого типу «символ», який є в ряді інших мов. Рядкові величини обмежуються одинарними, або двійними лапками. Вони є рівноправними, але відкриваюча і закриваюча повинні співпадати.

'Це рядок' ”це теж рядок” 'помилка” ”теж помилка'

var str='Приклад';

Рядки можуть містити спеціальні символи:

|  |  |
| --- | --- |
| \b | Backspace |
| \n | Перехід на новий рядок |
| \t | Tab |
| \uNNNN | Символ в кодуванні Unicode з шістнадцятковій кодом NNNN.  Наприклад, \ u00A9 - символу копірайту © |

Якщо рядок в одинарних лапках, то внутрішні одинарні лапки всередині повинні бути екрановані: перед ними додається символ \. У подвійних лапках - екрануються внутрішні подвійні. Одинарні лапки всередині подвійних і навпаки екранувати не обов’язково. Екранування служить виключно для правильного сприйняття рядка JavaScript. У пам'яті рядок буде містити сам символ без '\'. Сам символ зворотного слеша "\" є службовим, тому завжди екранується, тобто пишеться як \\.

var s1='здоров\'я', s2=”готель \”Ужгород\”” , s3=”здоров'я”;

Єдиою операцію, яку можна застосовувати до рядків є додавання. Вона об'єднує операнди в один:

var a = 'Мій' + 'рядок'; //'Мійрядок'

alert(str+' '+a); // 'Приклад Мійрядок'

Якщо хоча б один аргумент є рядком, то другий буде також перетворений до рядка.  
Причому не важливо, праворуч або ліворуч знаходиться операнд-рядок, в будь-якому випадку нерядковий аргумент буде перетворений:

1+'2'==='1'+2;// '12' 'Це '+(0<1)=='Це true';

Будь-який інший арифметичний оператор, навпаки, завжди перетворює операнди на число. Якщо неможливо перетворити рядковий операнд, то отримаємо NaN :

5-'2'==3; '10'/2==5; 'Не число'\*2=NaN;

Рядки – є об’єктами, і відповідно мають деякі властивості і методи.

Властивість length містить довжину рядка.

alert (str.length); //7

Щоб отримати символ, використовується метод charAt (позиція). Перший символ має позицію 0. В JavaScript немає окремого типу «символ», тому charAt повертає рядок, що складається з одного символу. У сучасних браузерах (крім IE <8) для доступу до символу можна також використовувати квадратні дужки:

alert (str.charAt (0)); // "П"

str[3]===str.charAt (3); // "к"

Можна прочитати символ, але не можна замінити його. Оператори виду str[3]='x' буде проігноровано.

Методи toLowerCase() і toUpperCase() міняють регістр символів рядка на нижній і верхній, відповідно.

str.toUpperCase()=='ПРИКЛАД';

str[0].toLowerCase()=="п";

Для пошуку підрядка є метод indexOf (підрядок [, початкова позиція]). Він повертає позицію, починаючи з якого входить підрядок або -1, якщо нічого не знайдено. Необов'язковий другий аргумент дозволяє шукати, починаючи з вказаної позиції. Також існує аналогічний метод lastIndexOf , який шукає не з початку, а з кінця рядка.

var str='підрядок цей рядок';

alert( str.indexOf( 'під')); // 0 – починається одразу з першого символу

alert( str.indexOf( 'рядок')); // 3 – це перше входження

alert( str.indexOf( 'рядок',5)); // 13 – шукає починаючи з 5 символу

alert( str.indexOf( 'нема')); // -1 – такого нема

alert( str.indexOf( 'Рядок')); // -1 – такого теж нема 'Рядок'!= 'рядок'

В JavaScript існують цілих три Методу для взяття підрядка, з невеликими відмінностями між ними.

Метод substring (початкова позиція [, кінцева позиція]) повертає підрядок від початкової позиції до кінцевої, не включаючи її. Якщо другий аргумент відсутній, то підрядок береться до кінця.

substr (початкова позиція [, кількість символів]). Перший аргумент має такий же зміст, як і в substring, а другий містить не кінцеву позицію, а кількість символів, що треба скопіювати до підрядка.

slice (початкова позиція [, кінцева позиція]) теж повертає частину рядка від початкової позиції до кінцевої, але не включаючи її. Сенс параметрів - такий же як в substring.

Різниця між substring і slice - в тому, як вони працюють з від’ємними і перевищуючими довжину рядка параметрами:

В substring від’ємні аргументи інтерпретуються як рівні нулю. Занадто великі значення усікаються до довжини рядка. Крім того, якщо початкове значення більше за кінцеве, то аргументи міняються місцями, тобто повертається ділянку рядка від кінцевого до початкового значення.

В slice від'ємні значення відраховуються від кінця рядка.

var s='Приклад';

s.slice(2,4)== 'ик'; alert(s.slice(-2,10))=='ад';

s.substring(2,4)== 'ик'; alert(s.substring(-2,10))== 'Приклад';

s.substr(2,4) =='икла';

Взагалі, в світі багато різних кодувань, і сторінка може бути в будь-який з них. Але всередині JavaScript-інтерпретатора всі рядки перебувають в кодуванні юнікод.

Є метод для отримання символу за його цифровому коду: String.fromCharCode (код)  
    Повертає символ за кодом code:

alert (String.fromCharCode (1072)); / / 'а'

І метод для отримання цифрового коду: charCodeAt (позиція). Повертає код символу на відповідній позиції. Відлік позиції починається з нуля.

Рядки порівнюються в лексикографічному порядку. Порівняння рядків s1 і s2 відбувається за наступним алгоритмом:

Порівнюються перші символи: s1.charAt (0) і s2.charAt (0). Якщо вони однакові, то порівнюються другий символ, потім треті і так далі ... Якщо символи не рівні то більшим буде той рядок, відповідний символ якого більший. Якщо в одному рядку закінчилися символи, то він менший. Якщо в обох закінчилися - вони рівні.

"Яма">"Aма" / / true, Я>А

"Вася"> "Ваня" / / true, с> н

"Aaа"> "Аa" / / true, в 1-му рядку більше символів

Буває, що числа приходять в скрипт у вигляді рядків, наприклад як результат prompt. У цьому випадку результат їх порівняння буде невірним. Числа у вигляді рядків порівнюються як рядки.

alert ("2"> "14"); / / true, так як це рядки, і для перших символів вірно "2"> "1"

Щоб отримувати вірний результат, хоча б один з аргументів має не бути рядком. Тоді й інший буде перетворений до числа:

alert (2> "14"); / / false

Cимволи порівнюються не за алфавітом, а за кодом. У кого код більше - той і більше. У Юнікоді є багато різних символів. Кирилиці відповідає лише невелика частина з них.

Виведемо відрізок символів Юнікоду з кодами від 1035 до 1115:

var str='';

for (var i=1035; i<=1115; i++) {

str += String.fromCharCode (i);

}

alert (str);

результат: ЌЍЎЏАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюяѐёђѓєѕіїјљњћ

Слід мати на увазі, що літери 'і' та 'ї' знаходяться поза основним алфавітом, і як не парадоксально, але 'і'>'я'.