## 3.4. Стандартні об’єкти і функції JavaScript

В ядрі JavaScript визначені об’єкти та функції, які можливо використовувати не використовуючи контекст завантаженої сторінки. До основних об’єктів відносяться: Array, Date, Math, String.

***Array*** –масив. Масив це упорядкований набір однотипних даних, до елементів якого можливо звернутись по імені або по індексу. Для створення масиву необхідно використати одну із двох конструкцій:

*ім’я_масиву = new Array([елемент1], [елемент2], [елемент3],...)*

*ім’я_масиву = new Array([довжина масиву])*

Відзначимо, що перший елемент масиву має номер 0.

В першій конструкції в якості параметрів використовуються елементи масиву, в другій конструкції використовується довжина масиву.

Наприклад:

*ar1 = new Array(1, 2, 3)*

*ar2 = new Array(3)*

Для доступу до значень елементів масиву в квадратних дужках біля імені масиву необхідно вказати порядковий номер елементу. Наприклад:

*a = ar1[2]*

*ar1[0] = 7*

В цьому прикладі, змінній *a* присвоюється значення елементу масиву за номером 2, а в елемент масиву за номером 0 записується значення 3.

Особливістю масивів JavaScript є те, що розмір масиву може встановлюватись динамічно. Наприклад, якщо для масиву із попереднього прикладу написати:

*ar1[100] = 7,*

то розмір масиву буде автоматично установлений рівним 101.

Для визначення довжини масиву можна скористатись властивістю *length*. Наприклад:

*a = ar1. length*

Зручність використання масивів забезпечується рядом методів, представлених в табл. 3.2

Таблиця 3.2

**Методи об’єкту Array**

| Метод | Призначення |
|---|---|
| concat | Об’єднує два масиви в один.<br>`var newArray = array.concat(value1, value2, ..., valueN)` |
| join | Об’єднує всі елементи масиву в один рядок.<br>`arrayObj.join([glue]),  де glue – з’єднувач, наприклад ‘+’` |
| pop | Знищує останній елемент із масиву і повертає його значення.<br>`myFish = ["angel", "clown", "mandarin", "surgeon"];`<br>`popped = myFish.pop();` |
| push | Додає один або декілька елементів в кінець масиву і повертає останній добавлений елемент.<br>`var array = ["one", "two" ];`<br>`var pushed = array.push("three", "four");` |
| reverse | Переставляє елементи масиву в зворотному порядку: перший елемент стає останнім, а останній першим.<br>`arr = [1,2,3]; a = arr.reverse();` |
| shift | Знищує перший елемент масиву і повертає його значення.<br>`var arr = ["мій","маленький", "масив"]`<br>`var my = arr.shift()` |
| slice | Створює перетин масиву в вигляді нового масиву<br>Синтаксис: `arrayObj.slice( start[, end] )`<br>`var arr = [ 1, 2, 3, 4, 5 ]`<br>`arr.slice(2) // => [3, 4, 5]`<br>`arr.slice(1, 4) // => [2, 3, 4]` |
| splice | Додає та/або знищує елементи масиву<br>`arrayObj.splice(start,deleteCount,[elem1[,elem2[,...[,elemN]]]])`<br>`arr = [ "a", "b", "c", "d", "e" ]`<br>`removed = arr.splice(1,2)` |
| sort | Сортує елементи масиву<br>`arr = [1,-1, 0]`<br>`a = arr.sort()` |
| unsift | Додає один або більше елементів в початок масиву та повертає нову довжину масиву<br>`var arr = ["a", "b"]`<br>`unshifted = arr.unshift(-2, -1);` |

Синтаксис

### concat

```
var newArray = array.concat(value1, value2, ...,
valueN)
```

**value1, value2, ...**

Масиви або значення для приєднання

Створює новий масив. Копіює в нього поточний об'єкт `array` і значення `value1, value2, ...`

---

## join

### Синтаксис

```
arrayObj.join( [glue] )
```

### Аргументи

**glue**

Строковий аргумент, за допомогою якого будуть сполучені в рядок усі елементи масиву. Якщо аргумент не заданий, елементи будуть сполучені комами.

### Опис, приклади

```
var arr = [ 1, 2 , 3 ]
arr.join('+')  // "1+2+3"
arr.join()  // "1,2,3"
```

---

- ## slice

### Синтаксис

```
arrayObj.slice( start[, end] )
```

### Аргументы

**start**

Індекс елементу в масиві, з якого розпочинатиметься новий масив.

**end**

Необов'язковий параметр. Індекс елементу в масиві, на якому новий масив завершиться. При цьому останнім в новому масиві буде елемент з індексом `end - 1`

- Якщо start негативний, то він трактуватиметься як `arrayObj.length+start` (тобто `start 'ый` елемент з кінця масиву).
- Якщо end негативний, то він трактуватиметься як `arrayObj.length+end` (тобто `end-й` елемент з кінця масиву).
- Якщо другий параметр не вказаний, то екстракція продовжиться до кінця масиву. Якщо `end < start`, то буде створений порожній масив.

### Опис, приклади

Цей метод не змінює початковий масив, а просто повертає його частину.

### Приклад: Елементи з середини

```
var arr = [ 1, 2, 3, 4, 5 ]
arr.slice(2) // => [3, 4, 5]
arr.slice(1, 4) // => [2, 3, 4]
arr.slice(2, 3) // => [3]
```

### Приклад: Відлік з кінця

```
arr = [1, 2, 3, 4, 5]
arr.slice(-2) // => [4, 5]
arr.slice(arr.length - 2) // те ж саме

arr.slice(-3, -1) // [3, 4]
arr.slice(arr.length-3, arr.length-1) // те ж саме
```

- ## splice

## Синтаксис

```
arrayObj.splice( start, deleteCount, [elem1[, elem2[, ...[, elemN]]]] )
```

## Аргументи

**start**
: Індекс в масиві, з якого починати видалення.

**deleteCount**
: К-ть елементів, яку вимагається видалити, починаючи з індексу *start*.

- **IE**: якщо цей параметр не вказаний, то жоден елемент не буде видалений.
- **Firefox**: якщо цей параметр не вказаний, то будуть видалені усі елементи, починаючи з індексу *start*.
- **Opera**: (досліджено у версії 9.61) якщо цей параметр не вказаний, то будуть видалений 1 елемент, що має індекс *start*+1. У цьому ж випадку замість видаленого елементу буде повернений порожній рядок.

**elem1, elem2, ..., elemN**
: Елементи, що додаються, в масив. Додавання розпочинається з позиції *start*.

## Опис, приклади

Мабуть, самий комплексний метод для роботи з масивом.

Він об'єднує в собі дві різних функціональності: видаляє частину масиву і додає нові елементи на місце видалених.

При цьому можна звести до нуля кількість елементів, що видаляються, - тоді це буде просто додавання.

І можна не додавати елементів - тоді це буде просто видалення.

Метод повертає масив з видалених елементів.

### Приклад: Видалення

```
arr = [ "a", "b", "c", "d", "e" ]
removed = arr.splice(1,2)
// removed = [ "b", "c"] (2 елементи з arr[1])
// arr = ["a", "d", "e"] (ті що залишились)
```

### Приклад: Додавання елементів

```
arr = [ "a", "b", "c", "d", "e" ]
// починаючи з індексу 2 видалимо 0 елементів
// і додамо "b+"
 arr.splice(2,0,"b+")
 // arr = ["a", "b", "b+", "c", "d", "e"]
```

### Приклад: З кінця

```
arr = [ "a", "b", "c", "d", "e" ]
//видалимо з індексу 1 починаючи з кінця 1 елемент
arr.splice(-1,1)
```

---

- ## sort

## Синтаксис

```
arrayObj.sort( [sortFunction] )
```

## Аргументи

**sortFunction**

Необов'язковий параметр - функція сортування.

- Якщо вказана функція, то елементи масиву будуть відсортовані згідно зі значеннями, повертаних функцією. Умови на функцію можна записати таким чином:

```
function sortFunction(a, b){
  if(a менше, ніж b за деяким критерієм)
     return - 1 // Або будь-яке число, менше нуля
  if(a більше, ніж b за деяким критерієм)
     return 1  // Або будь-яке число, більше нуля
  // у випадку a = b повернути 0
  return 0
}
```

- Якщо параметр не вказаний, масив буде відсортований в лексикографічному порядку (зростаючий порядок дотримання символів в таблиці ASCII).

## Опис, приклади

Цей метод змінює початковий масив. Масив, що вийшов, також повертається в якості результату.

```
arr = [1,-1, 0]
 a = arr.sort()
```

```
// => arr = [ -1, 0, 1 ]
alert(a === arr) // => true, это тот же сортированный массив
```

---

# unshift

## Синтаксис

```
arrayObj.unshift( [elem1[, elem2[, ...[, elemN]]]] )
```

## Аргументи

**elem1, elem2, ..., elemN**
: Елементи, що додаються в початок масиву. Додані елементи збережуть порядок дотримання.

## Опис, приклади

Цей метод змінює початковий масив. Додає в нього аргументи і повертає довжину, що вийшла.

```
var arr = ["a", "b"]
unshifted = arr.unshift(-2, -1);
alert(arr ); // [ -2, -1, "a", "b"]
alert("New length: " + unshifted); // 4
```

Об’єкт ***Date*** використовується для роботи з датами. Синтаксис оператора створення екземпляра об’єкту дати:

*ім’я_об’єкту_дати = new Date([параметри])*

Якщо параметри відсутні, то значенням об’єкту буде поточна дата. Параметром може бути рядок типу: "місяць день, рік часи:хвилини;секунди".

Наприклад, для створення дати – "5 лютого 2005 року 23:12:07" необхідно:

*day = new Date("February 5, 2005 23:12:07")*

Прочитати або змінити параметри створеного об’єкту Date можливо за допомогою ряду методів.

Найбільш вживані методи показані в табл. 3.3

Таблиця 3.3

**Методи об’єкту Date**

| **Метод** | **Призначення** |
|---|---|
| getDate | Повертає число місяця для вказаної дати<br>`var sputnikLaunch = new Date("October 4, 1957 19:28:34 GMT")`<br>`day = sputnikLaunch.getDate()` |
| getDay | Повертає день тижня для вказаної дати |
| getHours | Повертає годину для вказаної дати |
| getMinutes | Повертає хвилини для вказаної дати |
| getMonth | Повертає місяць для вказаної дати |
| getSeconds | Повертає секунди для поточного часу |
| getYear | Повертає рік для вказаної дати |
| setDate | Встановлює число місяця для вказаної дати<br>setMonth(monthValue[, dayValue])<br>`theBigDay = new Date();`<br>`theBigDay.setMonth(6);` |
| setDay | Встановлює день тижня для вказаної дати |
| setHours | Встановлює годину для вказаної дати |
| setMinutes | Встановлює хвилини для вказаної дати |
| setMonth | Встановлює місяць для вказаної дати |
| setSeconds | Встановлює секунди для вказаної дати |
| setYear | Встановлює рік для вказаної дати |

Об’єкт ***Math*** дозволяє використовувати вбудовані в JavaScript математичні функції та константи. При зверненні до методів та властивостей цього об’єкту створювати його не потрібно, але необхідно явно вказувати його ім’я.

Наприклад для того, щоб записати в змінну a результат розрахунку функції sin від 1 радіану необхідно:

*a = Math.sin(1)*

Для того, щоб записати в змінну a результат виразу 5 в степені 6 необхідно:

*a = Math. pow(5,6)*

Методи об’єкту Math, що використовуються найбільш часто представлені в табл. 3.4.

Таблиця 3.4

**Методи об’єкту Math**

| **Метод** | **Призначення** |
|---|---|
| abs | Повертає абсолютне значення змінної. |
| sin, cos, tan | Повертають значення тригонометричних функцій. Аргументи задаються в радіанах. |
| acos, asin, atan | Повертають значення обернених тригонометричних функцій |
| exp, log | Повертають значення експоненціальної функції та функції натурального логарифму |
| ceil | Повертає найменше ціле число, більше або рівне значенню аргументу |
| floor | Повертає найменше ціле число, менше або рівне значенню аргументу |
| min, max | Повертає найбільше/ найменше значення з двох аргументів |
| pow | Повертає значення функції: $pow(x,y)=x^y$ |
| round | Повертає значення аргументу, округлене до найближчого цілого числа |
| sqrt | Повертає квадратний корінь аргументу |

Об’єкт ***String*** використовується для роботи з рядковими типами даних. Створення об’єкту String  відбувається коли змінній присвоюється рядковий літерал:

*var a = "Не явний спосіб створення рядкового об’єкту"*

Крім того, можливо явно створити рядковий об’єкт, використовуючи оператор new та конструктор String:

*ім’я_об’єкту = new String(Рядок)*

Параметром конструктору може бути будь-який рядок. Наприклад:

*a=new String("Явний спосіб створення рядкового об’єкту")*

Єдиною властивістю об’єкту String є ***length***, що зберігає довжину рядка. Наприклад для запису в змінну h довжини рядка a необхідно:

*h=a.length*

Методи об’єкту String, що використовуються найбільш часто перераховані в табл. 3.5. Наведемо приклад використання методу toLowerCase для переводу рядкової змінної a в верхній регістр:

*a=a.toLowerCase();*

Таблиця 3.5

**Основні методи об'єкту String**

| Метод | Призначення |
|---|---|
| link | Створює гіпертекстове посилання, по якій можливо перейти на інший URL.<br>`var hotText="Netscape"`<br>`var URL="http://home.netscape.com"`<br>`document.write("Click to return to " + hotText.link(URL))` |
| fontsize<br>small<br>big | Виводить рядок, з встановленим розміром шрифта<br>var worldString="Hello, world"<br>`document.write(worldString.small())`<br>`document.write("<P>" + worldString.big())`<br>`document.write("<P>" + worldString.fontsize(7))` |
| blink<br>bold<br>italics<br>strike | Виводить рядок, що відображається блимаючим, напівжирним, курсивом чи перекресленим шрифтом<br>var worldString="Hello, world"<br>document.write(worldString.blink())<br>document.write("<P>" + worldString.bold())<br>document.write("<P>" + worldString.italics())<br>document.write("<P>" + worldString.strike()) |
| substring | Повертає частину рядка об'єкта string<br>`var anyString="Netscape"`<br>`document.write(anyString.substring(0,3))` |
| sub<br>sup | Виводить рядок, що відображається як нижній або верхній індекс<br>`var subText="subscript"`<br>`document.write("This is what a " + superText.sup() + " looks like.")` |
| toLowerCase,<br>toUperCase | Переводить зміст рядка в нижній/верхній регістр<br>`var upperText="ALPHABET"`<br>`document.write(upperText.toLowerCase())` |
| parseFloat<br>parseInt | Аналізує рядковий аргумент та повертає число з плаваючою крапкою або ціле число |

| | |
|---|---|
| | parseFloat("3.14more non-digit characters"); |

На додаток до стандартних об’єктів JavaScript існує декілька функцій, для виклику яких не потрібно створювати об’єктів. Ці функції дістали назву "функцій верхнього рівня". До цих функцій відносяться: ***parseFloat(параметр)*** та ***parseInt(параметр)***. Їх призначенням є аналіз рядкового аргументу та повернення відповідно число з плаваючою крапкою або цілого числа.

- ## parseInt

### Синтаксис

```
var intValue = parseInt(string[, radix])
```

### Аргументи

**string**
строкове представлення числа
**radix**
основа системи числення

### Опис, приклади

Функція `parseInt` перетворить перший аргумент в число по вказаній основі, а якщо це неможливо - повертає `NaN`.

Наприклад, `radix=10` дасть десяткове число, 16 - шістнадцяткове і тому подібне. Для `radix>10` цифр після дев'яти представлені буквами латинського алфавіту.

Якщо в процесі перетворення `parseInt` виявляє цифру, яка не є цифрою в системі числення з основою `radix`, наприклад G в 16-вій системі або A в десятковій, то процес перетворення тут же завершується і повертається значення, отримане з рядка на даний момент.

`parseInt` округлює дробові числа, т.к зупиняється на десятковій точці.

Якщо `radix` не вказаний або дорівнює 0, то javascript припускає наступне:

- Якщо вхідний рядок починається з "0x", то `radix = 16`
- Якщо вхідний рядок починається з "0", то `radix = 8`. Цей пункт залежить від реалізації і в деяких браузерах (Google Chrome) відсутній.
- У будь-якому іншому випадку `radix=10`

## parseFloat

### Синтаксис

```
val FValue = parseFloat(strVal)
```

### Аргументы

**strVal**
строка, представляющая числовое значение

Об'єкт **`Number`** у JavaScript — це об'єкт-обгортка, що дозволяє працювати з числовими значеннями, такими як `37` чи `-9.25`.

Конструктор **`Number`** містить константи та методи для роботи з числами. Значення інших типів можуть бути перетворені на числа за допомогою **функції `Number()`**.

При використанні в якості функції, `Number(value)` перетворює рядок чи інше значення на тип Number. Якщо значення не можна перетворити, повертається `NaN`.

```
Number('123')  // повертає число 123
Number('123') === 123  // true

Number("коник")    // NaN
Number(undefined)  // NaN
```

## 3.5 Використання об’єктів window, document, location.

Об’єкт ***window*** створюється автоматично при запуску браузера. Крім того нове вікно можливо створити і засобами JavaScript. Для цього необхідно використати метод open. Синтаксис методу такий:

*Ім’я_змінної = window.open([Ім’я_ файлу],[Ім’я_вікна],[Параметри])*

*Ім’я_змінної* – це ім’я для звернення до нового вікна в програмі JavaScript.

*Ім’я_ файлу* – це повна або відносна URL-адреса документу, що буде відкриватись в вікні браузеру.

*Ім’я_вікна* – це ім’я, що буде вказане в якості цілі в гіпертекстовому посиланні на це вікно із іншого HTML-документу.

*Параметри* – задають значення параметрів вікна. Основні параметри представлені в табл. 3.6. Якщо можливе значення властивості yes або no, то при стандартних настройках використовується значення yes.

Таблиця 3.6

**Основні параметри вікна**

| Назва | Призначення | Можливі значення |
|---|---|---|
| directories | Наявність/відсутність панелі "Ссылки" | yes/no |
| height | Висота вікна | кількість пікселів |
| location | Наявність/відсутність адресного рядка | yes/no |
| menubar | Наявність/відсутність рядка меню | yes/no |
| resizable | Можливість/не можливість зміни розмірів віна користувачем | yes/no |
| scrollbars | Наявність/відсутність смуг прокрутки віна | yes/no |
| status | Наявність/відсутність рядка стану браузера | yes/no |
| toolbar | Наявність/відсутність панелей інструментів | yes/no |
| width | Ширина вікна | кількість пікселів |

Наприклад, для створення нового вікна браузера в якому буде завантажено файл a.html необхідно:

*<script>*

*myw=window.open("a.html","displayWindow",*

*"width=400,height=300,status=no,toolbar=no,menubar=no")*

*</script>*

При цьому, ширина вікна дорівнює 400 пікселів, висота вікна 300 пікселів, рядок стану вікна, панель інструментів та рядок меню будуть відсутні. Відзначимо, що наведений код необхідно записати в одному рядку.

Для закриття вікна браузера використовується метод close. Наприклад для закриття вікна попереднього прикладу необхідно:

*myw.close()*

Відзначимо, що при звернення до методів та властивостей вікна браузера в якому знаходиться програма JavaScript імені вікна або ключового слова window можна не вказувати. Наприклад, закрити поточне вікна браузера можливо так:

*close()*

---

*window.focus*

*Зазвичай, якщо вікно не мінімізоване, то воно робиться його поточним і виводиться на передній план.*

*Якщо ж вікно мінімізоване, то його позначення в списку мінімізованих вікон починає блимати.*

| Метод | Опис |
|---|---|
| alert() | Викликає вікно сповіщення, яке містить текст повідомлення і клавішу ОК.<br>`window.alert("Вітаю, це сповіщення");` |
| blur() | Робить вікно неактивним.<br>`win.blur()` |
| clearInterval() | Припиняє повторне виконання коду заданого setInterval().<br>`clearInterval(id);` |
| clearTimeout() | Відміняє заплановане методом setTimeout() виконання коду.<br>`clearTimeout(id)` |
| close() | Закриває вікно.<br>`close()` |
| confirm() | Викликає вікно підтвердження що містить текст повідомлення і клавіші ОК і Відміна. Повертає true або false |

| | |
|---|---|
| | `var x = confirm('Натисніть  ОК або Отмена.');` |
| focus() | Робить вікно активним.<br>`window.focus()` |
| moveBy() | Зміщує вікно відносно його поточної позиції.<br>`win = window.open();`<br>`win.moveBy(200,100);` |
| moveTo() | Переміщає вікно на вказану позицію.<br>`win.moveTo(500,400);`<br>`win.focus();` |
| open() | Відкриває нове вікно браузеру.<br>`myw=window.open("a.html","displayWindow",`<br>`"width=400,height=300,status=no,toolbar=no,menubar=no")` |
| scrollBy() | Прокручує вміст вікна на вказану кількість пікселів.<br>`scrollBy(0,100);` |
| print() | Роздруковує вміст поточного вікна.<br>`print();` |
| prompt() | Викликає вікно запиту, спонукаюче відвідувача ввести в нього певні дані.<br>`<html>`<br>`<head>`<br>`<script type='text/javascript'>`<br>`function message()`<br>`{ var x = prompt('Введіть Ваше ім'я:','Ім'я');`<br>`//Виведемо введені користувачем дані на сторінку`<br>`document.getElementById('mes').innerHTML = x; }`<br>`</script>`<br>`</head>`<br>`<body>`<br>`<input type='button' value='Викликати вікно запиту'`<br>`onclick='message()' />`<br>`<br /><br />`<br>`<p style='display:inline'> Ваше ім'я: </p>`<br>`<b><div style='display:inline' id='mes'>Невідомо </div></b>`<br>`</body>`<br>`</html>` |
| scrollTo() | Прокручує вміст вікна до вказаних координат.<br>`scrollTo(0,960);` |
| setInterval() | Викликає функцію або виконує код через певні проміжки часу.<br>`id=setInterval('time()',1000);` |

| | |
|---|---|
| setTimeout() | Викликає функцію або виконує код після вказаної кількості мілісекунд один раз.<br>`id=setTimeout('document.write(txt)',3000);` |

Цікавим методом об’єкту window є метод ***setTimeout***, за допомогою якого можливо запрограмувати виконання деяких команд після закінчення встановленого терміну часу. Синтаксис методу такий: *setTimeout("Код_JavaScript",інтервал_часу)*

В якості першого параметру функції setTimeout, як правило використовують функцію. Відзначимо, що цю функцію необхідно визначити на HTML-сторінці до використання функції setTimeout. Другим параметром функції setTimeout є інтервал часу закінчення якого буде сигналом про початок виконання команд JavaScript. Цей параметр задається в мілісекундах. Наприклад, виконання функції myfunction через 3000 мілісекунд після завантаження HTML-сторінки можливо запрограмувати так:

*setTimeout("myfunction()",3000)*

Досить часто функція setTimeout використовується для створення анімаційних ефектів. Це може бути, наприклад, циклічна зміна кольору тексту, або циклічна заміна одного зображення іншим.

Об’єкт ***document*** містить інформацію про завантажену сторінку. Всі елементи HTML-сторінки є властивостями цього об’єкту.

# Методи об'єкту Document

| Метод | Опис |
|---|---|
| createElement | Створює елемент. |
| getElementById | Дозволяє звернутися до елементу з вказаним id. |
| getElementsByName | Дозволяє звернутися до усіх елементів на сторінці з вказаним ім'ям. |
| getElementsByTagName | Дозволяє звернутися до усіх вказаних тегів на сторінці. |
| write | Виводить переданий текст на сторінку. |
| writeln | Виводить переданий текст на сторінку, відступаючи при цьому новий рядок, після кожного виводу. |

Найбільш використовуваним методом об’єкту document є метод ***write***, за допомогою якого можливо зробити запис в HTML-сторінку. Наприклад, для запису рядка "Привіт JavaScript" в документ в якому знаходиться сценарій необхідно:

*document.write("Привіт JavaScript")*

Ще одним широко вживаним методом цього об’єкту є ***getElementById***, що реалізує доступу до будь-якого об’єкту з визначеним id. Синтаксис методу такий:

***document.getElementById(ім’я_id)***

Наприклад, встановлення значення стилю display елемента з id=myid рівним block, можливо реалізувати так:

*document.getElementById("myid").style.display="block"*

Властивості об’єкту ***location*** дозволяють одержати інформацію про URL-адресу завантаженої HTML-сторінки.

- # window.location

Отримує/встановлює URL вікна і його компоненти

### Опис, приклади

Значенням цієї властивості є об'єкт типу `Location`.

### Об'єкт `Location`

Метод `toString` цього об'єкту повертає URL, а різні властивості дозволяють отримати/встановити окремі компоненти адреси.

Для деяких строкових операцій необхідно явно перетворити `Location` до рядка:

```
window.location.toString().charAt(17)
```

- **Властивості об'єкту Location**

Усі наступні властивості є рядками.

Колонка "Приклад" містить їх значення для URL:

- http://www.google.com: 80/search?q=javascript#test

| Властивість | Опис | Приклад |
|---|---|---|
| hash | частина URL, яка йде після символу '#', включаючи символ '#' | #test |
| host | хост і порт | www.google.com: 80 |
| href | увесь URL | http://www.google.com: 80/search?q=javascript#test |
| hostname | хост (без порту) | www.google.com |
| pathname | рядок шляху (відносно хоста) | /search |
| port | номер порту | 80 |
| protocol | протокол | http: |
| search | частина адреси після символу ?, включаючи символ ? | ?q=javascript |

У Firefox є баг: якщо hash -компонент адреси містить закодовані (див. encodeURIComponent) символи, властивість hash повертає розкодований компонент. Наприклад, замість %20 буде пропуск і тому подібне. Інші браузери поводяться коректно і не розкодують hash.

**Методи об'єкту `Location`**

assign(url)
: завантажити документ по цьому url

reload([forceget])
: перезавантажити документ по поточному URL. Аргумент forceget - булеве значення, якщо воно true, то документ перезавантажується завжди з сервера, якщо false або не вказано, то браузер може узяти сторінку зі свого кеша.

replace(url)
: замінити поточний документ на документ по вказаному url. Різниця, в порівнянні з assign() полягає в тому, що після використання replace() сторінка не записується в історії відвідувань. Зокрема, це означає, що відвідувач не зможе використати для повернення кнопку браузеру "Назад".

toString()
: повертає строкове представлення URL для об'єкту Location

При зміні будь-яких властивостей window.location, окрім hash, документ буде перезавантажений, начебто для модифікованого url був викликаний метод window.location.assign().

## 3.6. Контрольні запитання

1. Правила запису функцій.
2. Що таке об’єкт в мові JavaScript?
3. Як визначити об’єкт користувача в JavaScript?
4. Стандартні об’єкти JavaScript.
5. Як реалізована інтерактивність в JavaScript?
6. Стандартні функції JavaScript.

7. Методи та властивості об’єкту Array.
8. Методи та властивості об’єкту Date.
9. Методи та властивості об’єкту Math.
10. Методи та властивості об’єкту String.
11. Методи та властивості об’єкту Window?
12. Як за допомогою JavaScript відкрити нове вікно браузеру ?
13. Як за допомогою JavaScript запрограмувати виконання деяких команд після закінчення встановленого терміну часу?
14. Методи та властивості об’єкту Document.
15. Методи та властивості об’єкту Location.