

# Web-технології та web-дизайн. JavaScript

2023/2024  
для студентів 2-го курсу

## Самостійна робота до лекції №7

Лекції: ст. викладач каф. Штучного інтелекту  
Гріньова Олена Євгенівна  
olena.hrynova@nure.ua

# Зміст

Функції (продовження)

# Функції (продовження)

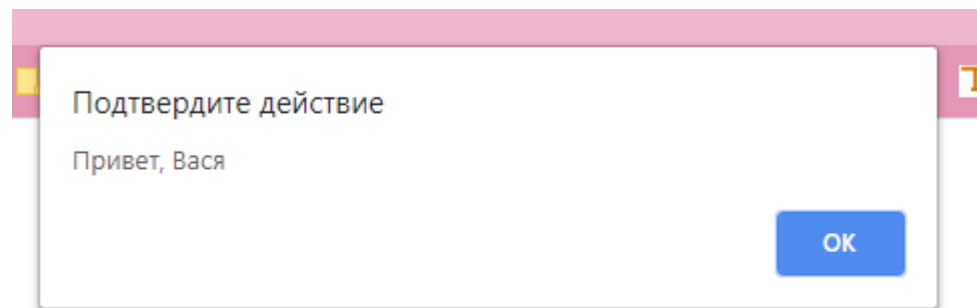
# Звернення до глобальної змінної в тулубі функції

У функції є доступ до зовнішніх змінних

```
let userName = 'Вася';
```

```
function showMessage() {  
  let message = 'Привіт, ' + userName;  
  alert(message);  
}
```

```
showMessage(); // Привіт, Вася
```



Функція має повний доступ до зовнішніх змінних і може змінювати їх значення

```
let userName = 'Вася';
```

```
function showMessage() {  
  userName = "Петя"; // змінюємо значення зовнішньої змінної  
  let message = 'Привіт, ' + userName;  
  alert(message);  
}
```

```
// до визову функції showMessage()  
alert(userName); // Вася
```

```
showMessage(); // Привіт, Петя  
// Наразі userName стало Петя
```

```
//значення зовнішньої змінної було зменено функцією  
alert(userName); // Петя
```

# Створення глобальної змінної при виконанні функції

Створення глобальної змінної у функції

// Опис функції:

```
function show(){
```

// Глобальна змінна:

```
myText="Глобальна змінна"
```

// Відображення значення змінної:

```
document.write(myText+"<br>")
```

```
}
```

// 1. Виклик функції:

```
show()
```

// 2. Відображення значення глобальної змінної:

```
document.write(myText+"<br>")
```

Глобальная переменная  
Глобальная переменная

# Створення глобальної змінної при викликанні функції

Код у тулубі функції виконується тільки при викликанні функції. Якщо функція не була викликана, то її код не виконується.

```
myText="Глобальная переменная"
```

```
// Опис функції:
```

```
function show(){
```

```
// Глобальна змінна:
```

```
myText=" Глобальна змінна1"
```

```
// Відображення значення змінної:
```

```
document.write(myText+"<br>")
```

```
}
```

```
// 1. Відображення значення глобальної змінної:
```

```
document.write(myText+"<br>")
```

```
show()
```

```
document.write(myText+"<br>")
```

---

Глобальная переменная  
Глобальная переменная1  
Глобальная переменная1

# Створення глобальної змінної при виконанні функції

Якщо функція викликається кілька разів, то змінна створюється при першому виклику функції. Якби ми в тілі функції `show ()` в команді привласнення значення змінної `myText` використовували ключове слово `var`, то при виконанні функції створювалася б не глобальна, а локальна змінна.

```
myText=«Глобальна змінная" // Глобальна змінная
```

```
// Опис функції:
```

```
function show(){
```

```
var myText="Локальна змінна"
```

```
// Відображення значення змінної:
```

```
document.write(myText+"<br>")
```

```
}
```

```
// 1. Відображення значення глобальної змінної:
```

```
document.write(myText+"<br>")
```

```
show() // 2. Відображення значення локальної змінної:
```

```
document.write(myText+"<br>") // 3. Відображення глобальної змінної :
```

Глобальная переменная Локальная переменная Глобальная переменная
--



# Локальні та глобальні змінні

Зовнішня змінна використовується тільки тоді, коли всередині функції немає такої локальної. Якщо однойменна змінна оголошується всередині функції, тоді вона перекриває зовнішню.

```
let userName = 'Вася';
```

```
function showMessage() {  
    let userName = "Петя"; // оголошуємо локальну змінну  
    let message = 'Привіт, ' + userName;  
    alert(message);  
}
```

// функція створить і буде використовувати свою власну локальну змінну **userName**

```
showMessage(); // Привіт, Петя
```

```
alert( userName ); // Вася, не змінилася, функція не чіпала зовнішню змінну
```

# Локальні та глобальні змінні

Подтвердите действие

Привет, Петя

OK

Подтвердите действие

Вася

OK

# var vs let

Директива **var** визначає нову змінну в поточній області видимості. Область видимості змінної, оголошеної через **var**, це її поточний контекст виконання, який може обмежуватися функцією або бути глобальним, для змінних, оголошених за межами функції.

ЕСМА-262 <https://tc39.es/ecma262/#sec-variable-statement>

Директива **let** дозволяє оголосити локальну змінну з **областю видимості, обмеженою поточним блоком коду**. На відміну від ключового слова **var**, яке оголошує змінну глобально або локально в усій функції, незалежно від області блоку.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

# Передача функціям змінних

Можна передавати функціям необмежену кількість змінних. Параметри функцій видно в області видимості і не пов'язані з глобальними змінними.

Щоб звертатися до глобальної змінної з функції, а не її копії використовуйте **window.им'я\_змінної**.

Якщо в тілі функції є локальна змінна з ім'ям **myText** і потрібно звернутися до глобальної з таким же ім'ям, то просто ім'я **myText** означатиме локальну змінну, а інструкція **window.myText** є звернення до глобальної змінної

# Передача функціям змінних

У тілі функції використовуються глобальна і локальна змінні з однаковими назвами

```
var myText="Глобальна змінна" // Глобальна змінна
document.write(myText+"<br>")// Відображення значення глобальної змінної
function show(){ // Опис функції
var myText="Локальна змінна"// Оголошення локальної змінної

//Присвоєння значення глобальної змінної :
window.myText="Змінна з натяком на глобальність"
document.write(myText+"<br>")//Відображення значення локальної змінної
document.write(window.myText+"<br>")// Відображення значення глобальної зм.
}
show() // Виклик функції

// Відображення значення глобальної змінної:
document.write(myText+"<br>")
```

Глобальная переменная
Локальная переменная
Переменная с намеком на глобальность
Переменная с намеком на глобальность

# Параметри

**Параметри функції** - це імена, перераховані у визначенні функції. Аргументи функції - це реальні значення, передані (і одержувані) функцією.

Можна передати всередину функції будь-яку інформацію, використовуючи параметри (так звані аргументи функції).

**Від локальних змінних аргументи відрізняються двома аспектами:**  
По-перше, аргументи просто вказуються (в той час як локальні змінні оголошуються в тілі функції з ключовим словом **var**)

По-друге, для аргументів функції значення зазвичай не присвоюються, в той час як для використання локальної змінної їй необхідно привласнити значення.

```
function showMessage(from, text) { // аргументи: from, text
    alert(from + ': ' + text);
}
showMessage('Аня', 'Привіт!'); // Аня: Привіт!
showMessage('Аня', «Як справи?»); // Аня: Як справи?
```

# Правила параметрів

**У визначеннях функцій JavaScript не вказуються типи даних для параметрів.**

**Функції JavaScript не виконують перевірку типу переданих аргументів.**

**Функції JavaScript не перевіряють кількість отриманих аргументів.**

# Параметри

// 1. Глобальна змінна:

```
var x="Альфа"
```

```
document.write("Глобальна змінна: "+x+"<br>")
```

// Виклик функції:

```
show("Браво")
```

// Опис функції з аргументом :

```
function show(x){
```

```
document.write("<h4>Виконування функції</h4>")
```

// Звернення до аргументу функції :

```
document.write("Аргумент: "+x+"<br>")
```

// Звернення до глобальної змінної:

```
document.write("Глобальна змінна: "+window.x+"<br>")
```

```
}
```

Глобальная переменная: Альфа

**Выполнение функции**

Аргумент: Браво

Глобальная переменная: Альфа



# Кількість аргументів функції

При виконанні функції кількість переданих їй аргументів може:

- співпадати з кількістю аргументів, зазначених при описі функції;
- перевищувати кількість аргументів, зазначених при описі функції;
- бути меншою за кількість аргументів, зазначених при описі функції.

```
function showMessage(from, text = "текст не додано ") {  
    alert( from + ": " + text );  
}  
showMessage("Аня"); // Аня: текст не додано
```

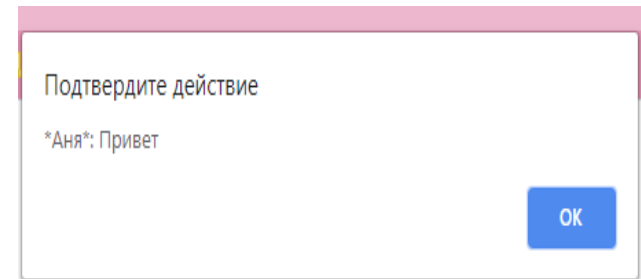
Тепер, якщо параметр **text** не вказано, його значенням буде **"текст не додано"**

# Параметри

Функція змінює значення **from**, але ця зміна не видно зовні. Функція завжди отримує тільки **копію значення**.

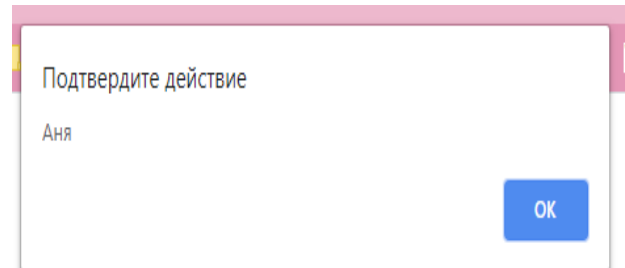
```
function showMessage(from, text) {  
  from = '*' + from + '*'; // трохи прикрасимо "from"  
  alert( from + ': ' + text );  
}
```

```
let from = "Аня";  
showMessage(from, "Привіт");  
// *Аня*: Привіт
```



// значення "from" залишилося колишнім, функція змінила значення локальної змінної

```
alert( from ); // Аня
```



# Параметри за замовчанням

Якщо параметр не вказано, то його значенням стає **undefined**.

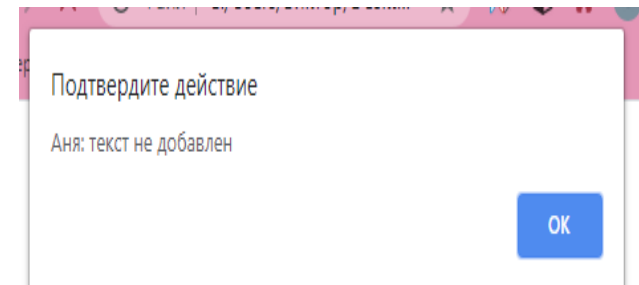
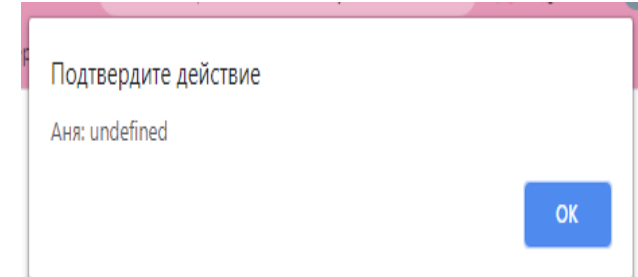
Наприклад, функція **showMessage(from, text)** може бути викликана з одним аргументом:  
**showMessage("Аня");**

Це не призведе до помилки. Такий виклик виведе "Аня: undefined". У виклику не вказано параметр **text**, тому передбачається, що **text === undefined**.

Щоб задати параметру **text** значення за **замовчанням**, потрібно його вказати після=

```
function showMessage(from, text = "текст не  
додано") {  
    alert( from + ": " + text );  
}  
showMessage("Аня"); // Аня: текст не додано
```

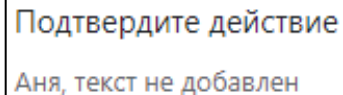
Наразі, якщо параметр **text** не вказано, його значенням буде "**текст не додано**"



# Параметри за замовчанням

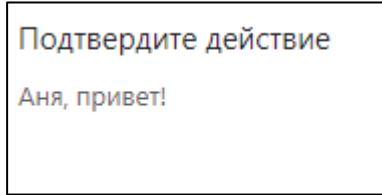
Якщо параметр не вказано, то його значенням стає **undefined**.  
Наприклад, функція `showMessage(from, text)` може бути викликана з одним аргументом:

```
function showMessage(from , text) {  
  if (text === undefined) {  
    text = 'текст не додано';  
  }  
  alert( from + ", " + text);  
}  
showMessage("Аня");
```



Подтвердите действие  
Аня, текст не добавлен

```
showMessage("Аня", "привіт!" );
```



Подтвердите действие  
Аня, привіт!

# Повернення значення, return

Функція може повернути результат, який буде переданий в який викликав її код. За допомогою команди **return** можна повертати з функцій значення.

```
function sum(a, b) {  
  return a + b;  
}
```

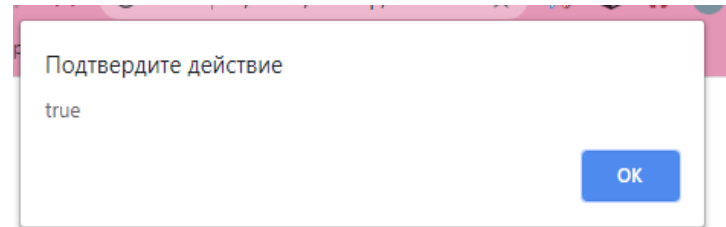
```
let result = sum(1, 2);  
alert( result ); // 3
```

Викликів **return** може бути кілька

# Повернення значення, return

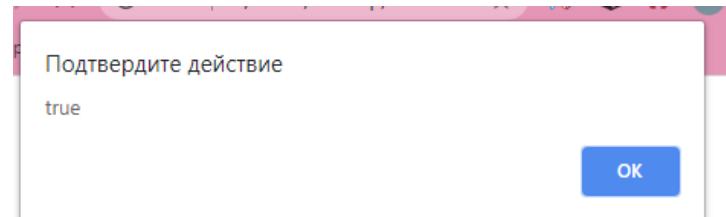
Якщо функція не повертає значення, це рівнозначно тому, як якщо б вона повертала **undefined**

```
function doNothing() { /* порожньо */ }  
alert( doNothing() === undefined ); // true
```



Порожній **return** аналогічний **return undefined**

```
function doNothing() {  
    return;  
}  
  
alert( doNothing() === undefined ); // true
```



# Повернення значення, return

**Важливо!** Якщо повертаючий вираз займає кілька рядків, потрібно почати його на тому ж рядку, що і **return**. Або, хоча б, поставити там відкриваючу дужку.

```
return (  
    some + long + expression  
    + or +  
    whatever * f(a) + f(b)  
)
```

# Передача функції аргументом

Аргументом функції може передаватися функція. Це дуже потужна властивість, яка дозволяє з успіхом, легко і ефективно вирішувати широкий клас задач.

З технічної точки зору передати аргументом функції іншу функцію досить просто: відповідний аргумент в тілі функції обробляється як **ім'я** функції. Відповідно, якщо при виконанні функції аргументом їй передається інша функція, то такий аргумент - це ім'я функції (яка передається аргументом).

Функція оголошена всередині іншої функції ще має доступ і до всіх змінних її батьківської функції та іншим змінним, до яких ця батьківська функція має доступ.



# Рекурсія

Під **рекурсією** вважають ситуацію, коли в програмному коді опису функції викликається ця ж функція (зазвичай з іншим аргументом або аргументами).

Функція рекурсивного обчислення факторіала:

```
function factorial(n) {  
  if ((n === 0) || (n === 1))  
    return 1;  
  else  
    return (n * factorial(n - 1));  
}
```

# Замикання

В JavaScript дозволяється в тілі функції описувати інші функції. Функції, описані в тілі функцій, називаються **внутрішніми**. Функцію, що містить опис внутрішньої функції, будемо називати **зовнішньою**.

Головна особливість внутрішньої функції пов'язана з тим, що вона має доступ до локальних змінних зовнішньої функції, ця особливість внутрішніх функцій називається **замиканням**. При цьому зовнішня функція **не має доступу** до локальних змінних внутрішньої функції.

# Анонімні функції

Функції, які не містять імені при оголошенні, називаються **анонімними**.

```
function(аргументи){  
// код функції  
}
```

Такий об'єкт можна привласнити як значення змінної, а також його можна викликати.

```
var переменная=function(аргументы){  
// код функции  
}
```

# Анонімні функції

// Змінній значенням присвоюється анонімна функція :

```
var f=function(msg){  
document.write(msg+"<br>")  
}
```

// Виклик анонімної функції через змінну:  
f("Анонимная функция")

Анонимная функция

// Виклик іншої анонімної функції без

// присвоювання її значення змінній:

```
(function(msg){  
document.write("<b>" + msg + "</b><br>")  
})(«Ще одна функція»)
```

Еще одна функция

**Зовнішні круглі дужки**, в які полягає вся конструкція, потрібні для того, щоб «об'єднати» код створення іншої функції і дужки з аргументом в одне ціле.

# Анонімні функції

При запуску сценарію на виконання попередньо створюються **всі описані в ньому функції**, причому незалежно від того, в якому конкретно місці сценарію функції описані.

А якщо функція **анонімна**, то створюється вона в тому місці, де в сценарії розміщена команда створення функції. Тому **не можна розміщувати команду виклику анонімної функції через змінну f до того, як функція присвоєна значенням змінної**.

```
f("Анонімна функція")  
// Змінної значенням присвоюється анонімна функція :  
var f=function(msg){  
document.write(msg+"<br>")  
}
```

# return як функція

Функція результатом може повертати іншу функцію.

Поліном другого ступеню  $P(x) = a + b*x + c*x*x$

```
// Функція з результатом-функцією :  
function makePolynom(a,b,c){  
  // Результат функції:  
  return function(x){  
    // Результат анонімної функції:  
    return a+b*x+c*x*x  
  }  
} // Закінчення опису функції  
// Змінні:  
var P  
P=makePolynom(1,5,2)  
var z=2  
// Обчислення значень поліномів:  
document.write("P("+z+") = "+ P(z) + "<br>")
```

P(2) = 19

# Вбудовані функції

Крім визначених користувачем функцій в JavaScript існують ще й **вбудовані** функції.

Наприклад :

**decodeURI()** декодує URI

**encodeURI()** кодує URI

**escape()** кодує рядок

**eval()** аналізує рядок і виконує її як JavaScript код

**isFinite()** визначає чи є число допустимим

**isNaN()** визначає чи є об'єкт не числом

# Вибір імені функції

Як правило, використовуються дієслівні префікси, що позначають загальний характер дії, після яких слід уточнення. Функція - це дія, тому її ім'я зазвичай є **дієсловом**.

Функція повинна робити тільки те, що явно мається на увазі її назвою. Ім'я функції має зрозуміло і чітко відображати, що вона робить, і що повертає.

І це повинно бути однією дією. **Две незалежних дії зазвичай мають на увазі дві функції**, навіть якщо передбачається, що вони будуть викликатися разом (в цьому випадку ми можемо створити третю функцію, яка буде їх викликати).

```
showMessage(..)  // показує повідомлення  
getAge(..)       // повертає вік (в будь-якому значенні)  
calcSum(..)      // обчислює суму і повертає результат  
createForm(..)   // створює форму (і зазвичай повертає її)  
checkPermission(..) // перевіряє доступ, повертаючи true/false
```



# Вибір імені функції

Функції повинні бути короткими і робити тільки щось одне. Якщо це щось велике, має сенс **розбити функцію на кілька менших**. Приклад функції виводить просте число до n

1 варіант

```
function showPrimes(n) {  
  nextPrime: for (let i = 2; i < n; i++) {  
  
    for (let j = 2; j < i; j++) {  
      if (i % j == 0) continue nextPrime;  
    }  
  
    alert( i ); // просте  
  }  
}
```

2 варіант

```
function showPrimes(n) {  
  for (let i = 2; i < n; i++) {  
    if (!isPrime(i)) continue;  
    alert(i); // просте  
  }  
}  
function isPrime(n) {  
  for (let i = 2; i < n; i++) {  
    if ( n % i == 0) return false;  
  }  
  return true;  
}
```

# Функція як об'єкт

**Функція є об'єктом.** Як у будь-якого об'єкта, у функції є **властивості**.

Крім способу створення функцій (опис функції з ключовим словом **function** і присвоювання змінної значенням анонімної функції), існує ще один спосіб, який передбачає використання **конструктора Function**.

**var ім'я\_функції=new Function(аргументи, код функції)**

Початкові аргументи конструктора **Function** - текстові назви аргументів функції, а останній аргумент - текстовий рядок з програмним кодом, виконуваних при виконанні функції.

```
var f=new Function("x","y","return x+y")  
//обчислює суму двох своїх аргументів  
//після цього змінну f можемо викликати як функцію  
f(3,4)
```

# Функція як об'єкт

**arguments.length** повертає властивість **числа аргументів**

```
<body>
<p>The arguments.length property returns the number of arguments received
by the function:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return arguments.length;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
```

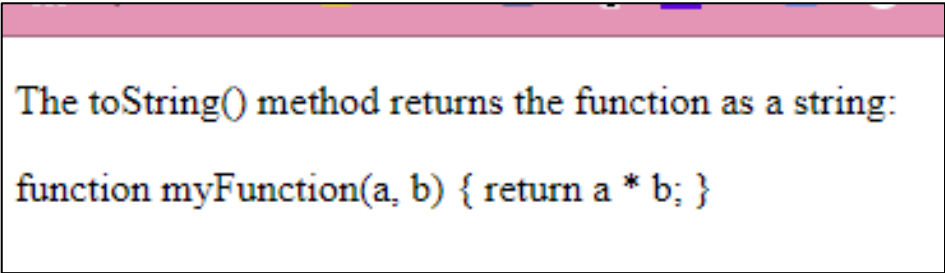
The arguments.length property returns the number of arguments received  
by the function:

2

# Функція як об'єкт

Метод **toString()** повертає функцію у вигляді строки

```
<body>
<p>The toString() method returns the function as a string:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction.toString();
</script>
</body>
```



```
The toString() method returns the function as a string:
function myFunction(a, b) { return a * b; }
```

# Функція як об'єкт

Функція, певна як властивість об'єкта, називається **методом об'єкта**.

Функція, призначена для створення нових об'єктів, називається **конструктором об'єктів**.

# Функції, підсумок

Функція - блок коду, який можна викликати по імені. У функції можуть бути аргументи, і функції можуть повертати результат. В описі функції використовується ключове слово **function**, після якого вказується ім'я функції, список аргументів і тіло функції з командами (полягають у фігурні дужки).

Для повернення значення у функції використовують ключове слово **return**.

Функція може бути описана в будь-якому місці сценарію. Команди використання функції можуть в сценарії перебувати до опису функції. При запуску сценарію перед виконанням команд створюються всі функції, описані в сценарії.

Змінні, оголошені в тілі функції з ключовим словом **var**, є локальними і доступні тільки в тілі функції. Аргументи функції мають силу локальних змінних. Якщо локальна і глобальна змінні мають однакові назви, то в тілі функції пріоритет залишається за локальною змінною.

Для отримання доступу до зовнішніх (глобальних) змінних в такому випадку використовують посилання **window** на об'єкт вікна.

# Функції, підсумок

Аргументи функцій передаються за значенням: насправді створюються технічні копії аргументів, які і використовуються при обчисленнях. Аргументом функції допускається передавати, крім іншого, функцію. При виконанні функції кількість аргументів може не збігатися з тією кількістю аргументів, яке було вказано при описі функції.

Передані значення копіюються в параметри функції і стають локальними змінними.

Функції мають доступ до зовнішніх змінних. Але це працює тільки зсередини назовні. Код поза функції не має доступу до її локальних змінних.

Функція може повертати значення. Якщо цього не відбувається, тоді результат дорівнює **undefined**.

Для того, щоб зробити код більш чистим і зрозумілим, рекомендується використовувати локальні змінні і параметри функцій, не користуючись зовнішніми змінними.

Функція, яка отримує параметри, працює з ними і потім повертає результат, набагато зрозуміліше функції, що викликається без параметрів, але змінює зовнішні змінні, що загрожує побічними ефектами.

# Література

Он-line посібники:

<http://www.w3schools.com/js/>

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





Дякую за увагу!