**Лекція 4. Функції у JS**

1. **Оголошення функцій**
2. **Функціональні вирази**

Найчастіше нам треба повторювати одне і ту ж дію в багатьох частинах програми.

Наприклад, красиво вивести повідомлення необхідне при вітанні відвідувача, при виході відвідувача з сайту, ще де-небудь.

Щоб не повторювати один і той же код в багатьох місцях, придумані функції. Функції є основними «будівельними блоками» програми.

Приклади вбудованих функцій ви вже бачили - це *alert (message), prompt (message, default)* і *confirm (question)*. Але можна створювати і свої.

*Оголошення*

Приклад оголошення функції:

function showMessage() {

alert( 'Привіт всім присутнім!' );

}

Спочатку йде ключове слово *function*, після нього ім'я функції, потім список параметрів в дужках (в прикладі вище він порожній) і тіло функції - код, який виконується при її виклику.

Оголошена функція доступна по імені, наприклад:

function showMessage() {

alert( 'Привіт всім присутнім!' );

}

showMessage();

showMessage();

Цей код виведе повідомлення два рази. Вже тут видно головну мету створення функцій: позбавлення від дублювання коду.

Якщо знадобиться поміняти повідомлення або спосіб його виведення - досить змінити його в одному місці: в функції, яка його виводить.

***Локальні змінні***

Функція може містити локальні змінні, оголошені через *var*. Такі змінні видно тільки всередині функції:

function showMessage() {

var message = 'Привіт, я - Вася!'; // локальна змінна

alert( message );

}

showMessage(); // 'Привіт, я - Вася!'

alert( message ); // <-- буде помилка, тому що змінну видно тільки всередині

Блоки *if / else, switch, for, while, do..while* не впливають на область видимості змінних. При оголошенні змінної в таких блоках, вона все одно буде видима у всій функції.

Наприклад:

function count() {

// змінні i, j НЕ будуть знищені після закінчення циклу

for (var i = 0; i < 3; i++) {

var j = i \* 2;

}

alert( i ); // i=3, останнє значення i, при ньому цикл перестав працювати

alert( j ); // j=4, останнє значення j, яке обчислив цикл

}

Неважливо, де саме в функції і скільки разів оголошується змінна. Будь-яке оголошення спрацьовує один раз і поширюється на всю функцію. Оголошення змінних в прикладі вище можна пересунути вгору, це ні на що не вплине:

function count() {

var i, j; // пересунули оголошення var в початок

for (i = 0; i < 3; i++) {

j = i \* 2;

}

alert( i ); // i=3

alert( j ); // j=4

}

***Зовнішні змінні***

Функція може звернутися до зовнішньої змінної, наприклад:

var userName = 'Вася';

function showMessage() {

var message = 'Привіт, я ' + userName;

alert(message);

}

showMessage(); // Привіт, я Вася

Доступ можливий не тільки на читання, але і на запис. При цьому, так як змінна зовнішня, то зміни будуть помітні і зовні функції:

var userName = 'Вася';

function showMessage() {

userName = 'Петро'; // (1) привласнення в зовнішню змінну

var message = 'Привіт, я ' + userName;

alert( message );

}

showMessage();

alert( userName ); // Петро, значення зовнішньої змінної змінено функцією

Звичайно, якби всередині функції, в рядку (1), була б оголошена своя локальна змінна *var* *userName*, то всі звернення використовували б її, і зовнішня змінна залишилася б незмінною. Змінні, оголошені на рівні всього скрипта, називають «глобальними змінними». В наведеному вище прикладі змінна *userName* - глобальна.

Робіть глобальними тільки ті змінні, які дійсно мають загальне значення для вашого проекту, а потрібні для вирішення конкретного завдання - нехай будуть локальними у відповідній функції.

*Увага: неявне оголошення глобальних змінних!*

***Параметри***

При виклику функції їй можна передати дані, які та використовує на свій розсуд.

Наприклад, цей код виводить два повідомлення:

function showMessage(from, text) { // параметри from, text

from = "\*\* " + from + " \*\*"; // тут може бути важкий код оформлення

alert(from + ': ' + text);

}

showMessage('Маша', 'Привіт!');

showMessage('Маша', 'Як справи?');

*Параметри копіюються в локальні змінні функції.*

Наприклад, в коді нижче є зовнішня змінна *from*, значення якої при запуску функції копіюється в параметр функції з тим же ім'ям. Далі функція працює вже з параметром:

function showMessage(from, text) {

from = '\*\*' + from + '\*\*'; // міняємо локальну змінну from

alert( from + ': ' + text );

}

var from = "Маша";

showMessage(from, "Привіт");

alert( from ); // старе значення from без змін, в функції була змінена копія

*Аргументи за замовчуванням*

Функцію можна викликати з будь-якою кількістю аргументів. Якщо параметр не переданий при виклику - він вважається рівним undefined. Наприклад, функцію показу повідомлення *showMessage (from, text)* можна викликати з одним аргументом:

showMessage("Маша");

При цьому можна перевірити, і якщо параметр не переданий - привласнити йому значення «за замовчуванням»:

function showMessage(from, text) {

if (text === undefined) {

text = 'текст не передано';

}

alert( from + ": " + text );

}

showMessage("Маша", "Привіт!"); // Маша: Привіт!

showMessage("Маша"); // Маша: текст не передано

При оголошенні функції необов'язкові аргументи, як правило, розташовують в кінці списку.

Для вказівки значення «за замовчуванням», тобто, такого, яке використовується, якщо аргумент не вказано, використовується два способи:

Можна перевірити, чи рівний аргумент *undefined*, і якщо так - то записати в нього значення за замовчуванням. Цей спосіб продемонстрований в прикладі вище.

Використовувати оператор ||:

 function showMessage(from, text) {

text = text || 'текст не передано';

...

}

Другий спосіб вважає, що аргумент відсутній, якщо переданий порожній рядок, 0, або взагалі будь-яке значення, яке в логічному контексті є *false*.

Якщо аргументів передано більше, ніж треба, наприклад *showMessage ("Маша", "привіт", 1, 2, 3)*, то помилки не буде. Але, щоб отримати такі «зайві» аргументи, потрібно буде прочитати їх зі спеціального об'єкта *arguments*. ***Повернення значення***

Функція може повернути результат, який буде переданий у код, що її викликає.

Наприклад, створимо функцію *calcD*, яка буде повертати дискримінант квадратного рівняння за формулою b2 - 4ac:

        function calcD(a, b, c) {

return b\*b - 4\*a\*c;

}

var test = calcD(-4, 2, 1);

alert(test); // 20

Для повернення значення використовується директива *return*. Вона може знаходитися в будь-якому місці функції. Як тільки до неї доходить управління - функція завершується і значення передається назад.

Викликів *return* може бути і кілька, наприклад:

function checkAge(age) {

if (age > 18) {

return true;

} else {

return confirm('Батьки дозволили?');

}

}

var age = prompt('Ваш вік?');

if (checkAge(age)) {

alert( 'Доступ дозволено' );

} else {

alert( 'У доступі відмовлено );

}

Директива *return* може також використовуватися без значення, щоб припинити виконання і вийти з функції.

Наприклад:

function showMovie(age) {

if (!checkAge(age)) {

return;

}

alert( "Фільм не для всіх" ); // (\*)

// ...

}

У коді вище, якщо спрацював *if,* то рядок (\*) і весь код під нею ніколи не виконається, так як *return* завершує виконання функції.

***Значення функції без return і з порожнім return***

У разі, коли функція не повернула значення або *return* був без аргументів, вважається що вона повернула *undefined*:

function doNothing() { /\* пусто \*/ }

alert( doNothing() ); // undefined

Зверніть увагу, ніякої помилки немає. Просто повертається *undefined*.

Ще приклад, на цей раз з *return* без аргументу:

function doNothing() {

return;

}

alert( doNothing() === undefined ); // true

***Вибір імені функції***

Ім'я функції слідує тим же правилам, що і ім'я змінної. Основна відмінність - воно повинно бути дієсловом, тому що функція - це дія. Як правило, використовуються дієслівні префікси, що позначають загальний характер дії, після яких слідує уточнення.

Функції, які починаються з "*show*" - щось показують:

showMessage(..) // префікс show, "показати" повідомлення

Функції, що починаються з "*get*" - отримують, тощо:

getAge(..) // get, "отримує" вік

calcD(..) // calc, "обчислює" дискримінант

createForm(..) // create, "отримує" форму

checkPermission(..) // check, "перевіряє" дозвіл, повертає true/false

Це дуже зручно, оскільки глянувши на функцію - ми вже приблизно уявляємо, що вона робить, навіть якщо функцію написала зовсім інша людина, а в окремих випадках - і якого виду значення вона повертає.

*Одна функція - одна дія*

Функція повинна робити тільки те, що явно мається на увазі її назвою. І це повинно бути одна дія. Якщо вона складна і має на увазі піддії – можливо тоді є сенс виділити їх в окремі функції? Найчастіше це має сенс, щоб краще структурувати код. Але найголовніше - в функції не повинно бути нічого, крім самої дії і піддій, нерозривно пов'язаних з нею. Наприклад, функція перевірки даних (скажімо, "*validate*") не повинна показувати повідомлення про помилку. Її дія - перевірити.

*Надкороткі імена функцій*

Імена функцій, які використовуються дуже часто, іноді роблять над короткими. Наприклад, у фреймворку *jQuery* є функція *$*, у фреймворку *Prototype* - Функція *$$*, а в бібліотеці *LoDash* дуже активно використовується функція з назвою з одного символу підкреслення \_.

***Висновки***

Оголошення функції має вигляд:

function ім’я (параметри, через, кому) {

код функції

}

* Передані значення копіюються в параметри функції і стають локальними змінними.
* Параметри функції копіюються в її локальні змінні.
* Можна оголосити нові локальні змінні за допомогою *var*.
* Значення повертається оператором *return* ....
* Виклик *return* тут же припиняє функцію.
* Якщо *return* викликаний без значення, або функція завершилася без *return*, то її результат дорівнює *undefined*.

При зверненні до неоголошеної змінної функція буде шукати зовнішню змінну з таким ім'ям, але краще, якщо функція використовує тільки локальні змінні:

* Це робить очевидним загальний потік виконання - що передається в функцію і який отримуємо результат.
* Це запобігає можливим конфліктам доступу, коли дві функції, можливо написані в різний час або різними людьми, несподівано один для одного змінюють одну і ту ж зовнішню змінну.

**Іменування функцій:**

Ім'я функції має зрозуміло і чітко відображати, що вона робить. Побачивши її виклик в коді, ви повинні тут же розуміти, що вона робить.

Функція - це дія, тому для надання імен функціям, як правило, використовуються дієслова.

Функції є основними будівельними блоками скриптів. Ми будемо неодноразово повертатися до них і вивчати все більш і більш глибоко.

1. **Функціональні вирази**

В JavaScript функція є значенням, таким же як рядок або число. Як і будь-яке значення, оголошену функцію можна вивести, ось так:

function sayHi() {

alert( "Привіт" );

}

alert( sayHi ); // виведе код функції

Звернемо увагу на те, що в останньому рядку після *sayHi* немає дужок. Тобто, функція не викликається, а просто виводиться на екран.

Функцію можна скопіювати в іншу змінну:

 function sayHi() { // (1)

alert( "Привіт" );

}

var func = sayHi; // (2)

func(); // Привіт // (3)

sayHi = null;

sayHi(); // помилка (4)

Оголошення (1) як би говорить інтерпретатору "створи функцію і помісти її в змінну *sayHi.*

У рядку (2) ми копіюємо функцію в нову змінну *func*. Ще раз зверніть увагу: після *sayHi* немає дужок. Якби вони були, то виклик *var func = sayHi ()* записав би в *func* результат роботи *sayHi ()* (до речі, чому він дорівнює? Правильно, *undefined*, адже всередині *sayHi* немає *return*).

На момент (3) функцію можна викликати і як *sayHi* () і як *func* (). Проте, в будь-який момент значення змінної можна змінити. При цьому, якщо воно не функція, то виклик (4) видасть помилку. Звичайні значення, такі як числа або рядки, є дані. А функцію можна сприймати як дію. Цю дію можна запустити через дужки (), а можна і скопіювати в іншу змінну, як було продемонстровано вище.

***Оголошення Function Expression***

Існує альтернативний синтаксис для оголошення функції, який ще більш наочно показує, що функція - це всього лише різновид значення змінної.

Він називається *«Function Expression»* (функціональний вираз) і виглядає так:

 var f = function(параметри) {

// тіло функції

};

Наприклад:

 var sayHi = function(person) {

alert( "Привіт, " + person );

};

*Порівняння з Function Declaration*

«Класичне» оголошення функції, про яке ми говорили до цього, виду *function ім'я (параметри) {...},* називається в специфікації мови *«Function Declaration».*

*Function Declaration* - функція, оголошена в основному потоці коду. *Function Expression* - оголошення функції в контексті будь-якого виразу, наприклад присвоювання.

Незважаючи на трохи різний вигляд, по суті два ці записи роблять одне і те ж:

// Function Declaration

function sum(a, b) {

return a + b;

}

// Function Expression

var sum = function(a, b) {

return a + b;

}

Обидва цих оголошення говорять інтерпретатора: "Оголоси змінну *sum*, створи функцію з зазначеними параметрами та кодом і збережи її в *sum*".

Основна відмінність між ними: функції, оголошені як *Function Declaration*, створюються інтерпретатором до виконання коду.

Тому їх можна викликати до оголошення, наприклад:

sayHi("Вася"); // Привіт, Вася

function sayHi(name) {

alert( "Привіт, " + name );

}

А якби це було оголошення *Function Expression*, то такий виклик б не спрацював:

sayHi("Вася"); // помилка!

var sayHi = function(name) {

alert( "Привіт, " + name );

}

Це через те, що JavaScript перед запуском коду шукає в ньому *Function Declaration* (їх легко знайти: вони не є частиною виразів і починаються зі слова *function*) і обробляє їх.

А *Function Expression* створюються в процесі виконання виразу, в якому створені, в даному випадку - функція буде створена при операції присвоювання *sayHi = function ...*

Як правило, можливість *Function Declaration* викликати функцію до оголошення - це зручно, так як дає більше свободи в тому, як організувати свій код. Можна розташувати функції внизу, а їх виклик - зверху або навпаки.

***Умовне оголошення функції***

У деяких випадках «додаткову зручність» *Function Declaration* може послужити погану службу. Наприклад, спробуємо, в залежності від умови, оголосити функцію *sayHi* по-різному:

 var age = +prompt("Скільки вам років?", 20);

if (age >= 18) {

function sayHi() {

alert( 'Прошу вас!' );

}

} else {

function sayHi() {

alert( ‘До 18 неможна’ );

}

}

sayHi();

*Function Declaration* при *use strict* видно тільки всередині блоку, в якому оголошено. Так як код в підручнику виконується в режимі *use strict*, то буде помилка.

А що, якщо використовувати *Function Expression*?

 var age = prompt('Скільки вам років?');

var sayHi;

if (age >= 18) {

sayHi = function() {

alert( 'Прошу Вас!' );

}

} else {

sayHi = function() {

alert( 'До 18 неможна' );

}

}

sayHi();

Або навіть так:

var age = prompt('Скільки вам років?');

var sayHi = (age >= 18) ?

function() { alert('Прошу Вас!'); } :

function() { alert('До 18 неможна); };

sayHi();

 Обидва ці варіанти працюють правильно, оскільки, в залежності від умови, створюється саме та функція, яка потрібна.

***Анонімні функції***

Погляньмо ще на один приклад - функцію *ask (question, yes, no)* з трьома параметрами:

*Question* - рядок-питання; *Yes* – функція; *No* – функція.

Вона виводить питання на підтвердження *question* і, в залежності від згоди користувача, викликає функцію *yes ()* або *no ()*:

function ask(question, yes, no) {

if (confirm(question)) yes()

else no();

}

function showOk() {

alert( "Ви погодились." );

}

function showCancel() {

alert( "Ви скасували виконання." );

}

// использование

ask("Ви згодні?", showOk, showCancel);

Якийсь дуже простий код, чи не так? Навіщо, взагалі, може знадобитися така *ask*?

Виявляється, при роботі зі сторінкою такі функції як раз дуже затребувані, тільки ось запитують вони не простим *confirm*, а виводять більш гарне вікно з питанням і можуть інтелектуально обробити введення користувача. Але це все потім, коли перейдемо до роботи з інтерфейсом.

Тут же звернемо увагу на те, що те ж саме можна написати більш коротко:

function ask(question, yes, no) {

if (confirm(question)) yes()

else no();

}

ask(

"Ви згодні?",

function() { alert("Ви погодились."); },

function() { alert("Ви скасували виконання."); }

);

Тут функції оголошені прямо всередині виклику *ask (...),* навіть без присвоєння їм імені. Функціональний вираз, який не записується в змінну, називають анонімної функцією. Дійсно, навіщо нам записувати функцію в змінну, якщо ми не збираємося викликати її ще раз? Можна просто оголосити безпосередньо там, де функція потрібна. Такого роду код виникає природно, він відповідає «духу» JavaScript.

***new Function***

Існує ще один спосіб створення функції, який використовується дуже рідко, але згадаємо і його для повноти картини.

Він дозволяє створювати функцію повністю «на льоту» з рядка, ось так:

var sum = new Function('a,b', ' return a+b; ');

var result = sum(1, 2);

alert( result ); // 3

Тобто, функція створюється викликом *new Function* *(params, code):*

*params*

Параметри функції через кому у вигляді рядка.

*code*

Код функції у вигляді рядка.

Таким чином можна конструювати функцію, код якої невідомий на момент написання програми, але рядок з ним генерується або підвантажується динамічно під час її виконання.

Приклад використання - динамічна компіляція шаблонів на JavaScript, ми зустрінемося з нею пізніше, при роботі з інтерфейсами.

***Висновки***

Функції в JavaScript є значеннями. Їх можна привласнювати, передавати, створювати в будь-якому місці коду.

Якщо функція оголошена в основному потоці коду, то це *Function Declaration*. Якщо функція створена як частина виразу, то це *Function Expression*.

Між цими двома основними способами створення функцій є такі відмінності:

|  |  |  |
| --- | --- | --- |
|  | **Function Declaration** | **Function Expression** |
| Час створення | До виконання першого рядка коду | Коли управління досягає рядка з функцією. |
| Можна викликати до оголошення | Так (тому що створюється заздалегідь) | Немає |
| Умовне оголошення в if | Не працює | Працює |

Іноді в коді розробників-початківців можна побачити багато *Function Expression*. Чомусь, мабуть, не дуже розуміючи, що відбувається, функції вирішують створювати як *var func = function (),* але в більшості випадків звичайне оголошення функції - краще.

Якщо немає явної причини використовувати *Function Expression* - віддавайте перевагу *Function Declaration*.

Порівняйте 2 приклади:

// Function Expression

var f = function() { ... }

// Function Declaration

function f() { ... }

*Function Declaration* коротше і краще читається. Додатковий бонус - такі функції можна викликати до того, як вони оголошені.

Використовуйте *Function Expression* тільки там, де це дійсно потрібно і зручно.