Розробка клієнтських сценаріїв з використанням JavaScript та бібліотеки jQuery

STEP computer ACADEMY

Урок 2-1

Об'єкт. Масиви. Рядки. Об'єкт Date. Об'єкт Math

Contents

Об'єкти

Що таке об'єкт?

Ви вже стикалися з поняттям об'єктів у раніше вивчених мовах програмування. Проте, зайвий раз трохи нагадати не завадить.

Об'єкт — це деяка конкретна реалізація певної сутності. Наприклад, у нас є деяка загальна сутність «яблуко». Об’єктом буде конкретне яблуко, що лежить перед нами. Або, наприклад, сутність «автомобіль», червоний автомобіль марки AUDI, який ми бачимо перед собою в автосалоні, — це об'єкт. Ну і звичайно, сутності з точки зору об'єктно-орієнтованого програмування — це класи.

У JavaScript є підтримка об'єктів і об'єктно-орієнтованого програмування. Ми обов'язково поговоримо ще про це, але почнемо ми трохи з іншого боку. JavaScript дає нам можливість подивитися на об'єкти з двох сторін: перша — це об’єктно-орієнтований стиль, друга — об’єкти, як асоціативні масиви.

І почнемо ми з другого підходу. Поняття асоціативного масиву вам вже зустрічалися раніше. Нагадаємо, це масив, який складається з пар «ключ-значення». Наприклад, в якості ключа може бути назва країн, а в якості значення — назви столиць цих країн. Головне обмеження на такий масив полягає в тому, що ключі мають бути унікальними. Подивимося, як ця концепція реалізована в JavaScript.

Перший крок для нас — створення об’єкту. Зробимо це таким чином:

```
// створюємо порожній об'єкт
// ми використовуємо ключове слово new для створення
// об'єкту
var obj = new Object();
// другий варіант для створення об'єкту
var obj2 = {};
```

Ви можете використати той або інший механізм залежно від своїх уподобань. Для додавання пар «ключ-значення» в об’єкт можна використати два підходи. Перший підхід через стандартний синтаксис масиву. У прикладі нижче ми створюємо об'єкт «студент» і заповнюємо його властивостями.

```
// створюємо порожній об'єкт
var student=new Object();

// Додаємо властивість в об'єкт використовуючи
// звичайний синтаксис масивів
// ["ключ"]
student["Name"] = "Vasya";

// замість подвійних лапок можна використовувати
// одинарні
student['Age'] = 23;
alert(student["Name"]);
alert(student['Age']);
```

Використовуючи другий метод ви додаєте характеристики в об'єкт, як ви вже вказували раніше.

```
// створюємо порожній об'єкт
var firm={};
// створюємо властивість всередині об'єкта
firm.Name = "Star Inc";
```

```
firm.Address = 'Somewhere street 5';
alert(firm.Name);
alert(firm.Address);
```

Ви можете використовувати будь-який з двох способів, але якщо у вас виникне необхідність зберігати в якості ключа рядок, що містить пробіли, то другий синтаксис не підійде.

```
// створюємо порожній об'єкт
var dog = {};

// нижче наведене правильне використання синтаксису
dog['Name of dog'] = 'Caesar';
alert(dog['Name of dog']);
// рядок нижче це синтаксична помилка
// dog.Name Of dog = 'Pit';
```

Видалення властивостей

Додані до об'єкта властивості можна видалити. Для цього використовується delete. Після видалення властивості ключ і значення безповоротно зникнуть.

```
// створюємо порожній об'єкт
var cat = {};
// задаємо значення властивостями
// синтаксис через крапку
cat.Name = "Vasiliy";
// синтаксис, як у роботі з масивами
cat["Age"] = 2;

// відображаємо значення
alert(cat.Name);
```

```
// можемо звернутися у будь-якому синтаксисі
alert(cat.Age);
alert(cat["Age"]);

// видаляємо властивості
delete cat.Name;
delete cat["Age"];

// при спробі показу значення ми побачимо значення
// undefined
alert(cat.Name);
alert(cat.Age);
```

Перевірка існування властивості всередині об'єкта

Ви вже знаєте, що при відображенні неіснуючих властивостей відображається значення undefined. А як перевірити чи існує властивість у коді?

Для цього можна використати конструкцію in. Вона повертає true, якщо властивість є всередині об'єкта.

```
// створили об'єкт
var obj={};
obj.Name = "Oleg";

// перевіряємо чи є властивість Age
// in поверне false, тому що цієї властивості немає
if("Age" in obj){
    alert("Exists");
}
else {
    alert("Not exists");
}
```

Ще один спосіб створення об'єкта з властивостями

Є ще один спосіб створення об'єктів, окрім вже відомих вам механізмів. Ви ж не думали, що їх лише два? Третій варіант створення об'єкта дозволяє відразу створити велику кількість властивостей всередині об'єкта. Покажемо на прикладі:

```
// створили об'єкт студента
var student = {
    name: "Daria",
    lastName: "Kislicina",
    age:23
};

/*
  Те саме, що й вище
  var student = {};
  student.name = "Daria";
  student.lastName = "Kislicina";
  student.age = 23;
*/
alert(student.age);
```

Цей шлях є дуже зручним, коли вам потрібно створити описати об'єкт відразу. При цьому у вас є можливість додати властивість пізніше, коли вона вам знадобиться. Під час створення об'єкта можна вставити інший об'єкт.

```
// створили об'єкт студента
var student = {
    name: "Daria",
    lastName: "Kislicina",
```

```
    age:23,
    address:{
        street: "Tiraspolskaya 5",
        city: "Odessa",
        country: "Ukraine"
    }
};

alert(student.lastName);
alert(student.address.street);
alert(student.address.city);
```

Перегляд всіх властивостей всередині об'єкту

У вас є можливість дізнатися про всі властивості об'єкта. Для цього необхідно використовувати цикл із використанням конструкції in.

```
var rect={
    x:0,
    y:0,
    endX:10,
    endY:10
};

// tempProperty потраплятиме назва властивості
// таке як x,y,endX,endY

for(var tempProperty in rect){
    // відображаємо назву властивості
    alert(tempProperty);
    // значення властивості
    alert(rect[tempProperty]);
}
```

Масиви

Що таке масиви?

Ми сподіваємося, що ви читаєте дані рядки і це питання у вас не стоїть гостро, тому що ви вже знайомі з цим поняттям з інших мов програмування, вивчених раніше. Проте, все одно зробимо крок назад і пригадаємо, що таке масив.

Масив — це структура даних, яка групує набір деяких значень під іменем. Для доступу до конкретного значення використовується індекс. Індексація в масивах JavaScript починається з нуля. Відразу звертаємо вашу увагу, що JavaScript в масиві може зберігати значення різних типів.

JavaScript-масив може бути створений двома способами. Почнемо із конструкції Array.

Об'єкт Array

Перший спосіб створення масиву полягає у використанні конструкції new Array.

Нижче наводимо можливі форми створення:

```
// створюємо порожній масив
var arrayName = new Array();

// створюємо масив заданої довжини
var arrayName = new Array (Number length);

// створюємо масив і відразу ініціалізуємо його
// значеннями
var arrayName = new Array(element1, element2,...
                          elementN);
```

Всі три конструкції досить прості та не несуть жодних складнощів для програміста. Розглянемо приклад запису значень масивів для кожної з форм.

```
// створили порожній масив
var arr = new Array();

// заповнили його значеннями
arr[0] = 34;
arr[1] = 99;
arr[2] = 100;

// створили масив із довжиною 3
var arr2 = new Array(3);

// заповнили його значеннями
arr2[0] = 111;
arr2[1] = 56;
arr2[2] = 73;

// тепер довжина масиву збільшилася на 2 елементи
arr2[3] = 333;
arr2[4] = 999;

// створили масив і відразу записали в нього три
// значення
var arr3 = new Array("music","guitar","apple");

// додали ще один елемент
arr3[3] = "lemon";
```

Зверніть увагу, масиви JavaScript автоматично збільшуються в розмірі, коли це необхідно. Це означає, що вам не доведеться замислюватися про динамічне виділення пам'яті.

Для того, щоб дізнатися довжину масиву, потрібно скористатися вбудованою властивістю масиву під назвою length.

```
var arr = new Array (10,20);

// відображаємо значення нульового елемента
// 10
alert(arr[0]);

// за допомогою alert можна показувати весь масив відразу
// елементи масиву будуть перераховані через кому
alert(arr);

// показуємо довжину масиву
// 2
alert(arr.length);
```

При використанні властивості length слід пам'ятати про один важливий момент. Властивість length — це не кількість елементів масиву, а значення останнього індексу +1.

```
var arr = new Array();

// реальних заповнених елементів один
arr[499] = 86;

// на екрані 500
// length це останній індекс +1
alert(arr.length);

// при зверненні до неініціалізованого елемента
// відобразиться undefined
alert(arr[0]);
```

Ви вже знаєте, що масиви зростають у розмірі самостійно, коли це потрібно кодом скрипту. А що робити, якщо вам потрібно зменшити розмір масиву? Відповідь очевидна! Потрібно зменшити значення length до потрібної довжини.

```
var arr = new Array (11, 74, 35);

// робимо розмір масиву рівним 2
// значення 35 втрачено назавжди
arr.length = 2;

// на екрані 11,74
alert(arr);

// а тепер розмір масиву 5, але
// задані значення лише двом елементам
// arr[0] = 11
// arr[1] = 74
arr.length = 5;
alert(arr);

// тепер розмір масиву 0, всі значення втрачені назавжди
arr.length = 0;
```

Створення масиву другий спосіб

Перший спосіб створення масивів трохи громіздкий. Другий спосіб звичний для нас. Розглянемо його:

```
// створюємо порожній масив
var arrayName = [];

// створюємо масив із набором значень
var arrayName = [element1, element2,..., elementN];
```

Цікавим моментом є використання [] для створення масиву. Раніше ви стикалися з використанням {} для тієї ж мети. Тепер спробуємо на прикладі нові конструкції.

```
// створюємо порожній масив
var arr = [];

// Записали в нього 2 елементи
arr[0] = 11;
arr[1] = 12;
// 11,12
alert(arr);

// створюємо масив із трьома елементами
var arr = [88,99,111];

// 88,99,11
alert(arr);

// створюємо масив із трьома елементами
var cars = ["BMW", "Audi", "Toyota"];

// "BMW", "Audi", "Toyota"
alert(cars);
// 3
alert(cars.length);
```

За цим кодом можна зробити простий висновок: така форма створення масиву зручніша за перший спосіб з Array.

Звернення до елементів масиву

Ми вже вміємо звертатися до елементів масиву, використовуючи індекс. Додаємо до нашого багажу приклад перебору масиву за допомогою циклу:

```
var arr = [2,9,33,1];
var amt = 0;

// підрахуємо у циклі суму елементів масиву
for(var i = 0; i<arr.length; i++){
    amt+=arr[i];
}

// результат 45
alert(amt);
```

І ще приклад. Відобразимо вміст масиву з елементами різного типу:

```
var arr = [33, "sun", 12, "planet"];

for(var i = 0; i<arr.length; i++){
    alert(arr[i]);
}
```

Тепер настав час створити двовимірний масив:

```
// створили двовимірний масив: 2 рядки 3 стовпці
// 1 3 5
// 2 7 8
var arr = [
    [1,3,5],
    [2,7,8]
];

// 1
alert(arr[0][0]);

// 8
alert(arr[1][2]);
```

Звичайні змінні JavaScript передаються всередину функції за значенням. Це означає, що при передаванні змінної у функцію створюється копія змінної. Якщо ви зміните значення цієї копії, це ніяк не вплине на оригінальну змінну. Масиви передаються в функцію за посиланням. Так відбувається, тому що масив — це об'єкт, а об'єкти завжди передаються всередину функцій за посиланням. Звідси можна дійти висновку, що зміни значень елементів масиву зберігаються під час виходу з функції.

```
// функція записує нове значення за вказаним
// індексом
function SetValue(arr, index, newValue){
    arr[index] = newValue;
}
var arr = [88,11,3];

// 88,11,3
alert(arr);
SetValue(arr,0,999);

// 999,11,3
alert(arr);
```

Властивості та методи масивів

У масивів JavaScript є велика кількість вбудованих методів. Ми познайомимося з деякими із них.

Почнемо з методів пошуку. Це indexOf і lastIndexOf. Перший шукає збіг у масиві зліва направо. Сигнатура методу:

```
name_of_array.indexOf(what_to_search[, fromIndex])
```

- what_to_search — значення, яке ми шукаємо у масиві
- fromIndex — необов'язковий параметр, який використовується для вказівки на стартовий індекс. Якщо ми не вказуємо його, пошук почнеться з нульового індексу.

Якщо шукане значення знайдено метод повертає індекс знайденого значення, якщо значення немає в масиві, то метод поверне -1.

Продемонструємо роботу цього методу на прикладах:

```
var arr = [1,45,-3,78,1];

// шукаємо значення 45
var index = arr.indexOf(45);

// на екрані індекс 1
alert(index);

// шукаємо значення, якого немає в масиві
index = arr.indexOf(99);
/
/ на екрані -1, тому що 99 немає в масиві
alert(index);
```

А тепер давайте подивимося приклад підрахунку, скільки разів деяке шукане значення зустрічається в масиві.

```
var arr = [12,45,-3,82,12,78,12];
// лічильник для підрахунку кількості разів входження
// шуканого значення масив
// будемо шукати значення 12
var counter = 0;
```

```
var index = arr.indexOf(12);
while (index! = -1) {
      counter++;
      // рухаємося далі масивом за рахунок зміни
      // індексу на значення індекс+1
      index = arr.indexOf(12,index+1);
}

// на екрані 3
alert(counter);
```

Метод lastIndexOf працює за схожим індексом, але шукає справа наліво. Це означає, що пошук починається з останнього доступного індексу. Сигнатура методу:

```
name_of_array.indexOf(what_to_search[, fromIndex])
```

- what_to_search — значення, яке ми шукаємо у масиві
- fromIndex — необов'язковий параметр, який використовується для вказівки на стартовий індекс. Якщо ми не вказуємо його, то пошук почнеться з останнього індексу.

Якщо шукане значення знайдено метод повертає індекс знайденого значення, якщо значення немає в масиві, то метод поверне -1.

Давайте на прикладі розберемо відмінності цього методу.

```
var arr = [12,45,-3,82,12,78,12];

// шукаємо 12 справа наліво
var index = arr.lastIndexOf(12);
```

```
// на екрані 6
alert(index);
index = arr.lastIndexOf(77);

// на екрані -1
alert(index);
```

А тепер порахуємо скільки разів зустрічається шукане значення в масиві за допомогою lastIndexOf.

```
var arr = [12,45,-3,82,12,78,12];

// лічильник для підрахунку кількості разів входження
// шуканого значення масив
// будемо шукати значення 12
var counter = 0;
var index = arr.lastIndexOf(12);
while (index! = -1) {
          counter++;

  // ми перевіряємо на нуль
  // тому що нижче починаємо з index-1
  // за 0 ми отримаємо старт -1
  // для методу lastIndexOf негативний індекс
  // означає шукати з кінця масиву
  if(index == 0)
           break;

  // рухаємося далі масивом за рахунок зміни індексу
  // на значення індекс-1
  index = arr.lastIndexOf(12,index-1);
}

// на екрані 3
alert(counter);
```

При роботі з даними дуже важливо мати механізм сортування. І такий механізм представлений у вигляді методу sort, який має таку сигнатуру:

```
name_of_array.sort([compareFunc])
```

- what_to_search — значення, яке ми шукаємо у масиві
- compareFunc — необов'язковий параметр, ім'я функції користувача, яка буде використана для сортування масиву. Якщо її не вказувати, дані в масиві будуть відсортовані за правилами сортування рядків.

Почнемо у наших прикладах із сортування за замовчуванням.

```
var arr = [10,1,3,33,6];
arr.sort();
// 1 10 3 33 6
alert(arr);
```

В результаті сортування за замовчуванням, 10 стоїть перед 33, так вийшло через рядкове сортування. Якщо вам потрібне числове сортування, то потрібно буде реалізувати функцію сортування. Можливо, ви вже стикалися з таким рішенням в інших мовах програмування. Загальна форма цієї функції:

```
function name_of_function(name_of_var1, name_of_var2)
{
    body_of_function
}
```

Ви можете назвати функцію будь-яким іменем. Функція має приймати два параметри. Це два значення маси-

ву, які у цей час порівнюються алгоритмом сортування. Припустимо, ви хочете сортувати елементи масиву за зростанням, тоді правила сортування для вас такі:

- якщо name_of_var1 > name_of_var2, значення, що повертається з функції позитивне (зазвичай використовують 1);
- якщо name_of_var1 < name_of_var2, значення, що повертається з функції негативне (зазвичай використовують -1);
- якщо рівні, зазвичай повертають 0.

При сортуванні за спаданням, у першому випадку повертайте негативне значення, а у другому — позитивне. Розглянемо приклад сортування даних за зростанням:

```
function compareFunc(a,b){
  if(a>b)
    return 1;
  else if(b>a)
    return -1;
  else
    return 0;
}
var arr = [10, 1, 3, 33, 6];
// 1 3 6 10 33
arr.sort(compareFunc);
```

А тепер, сортування за спаданням:

```
function compareFunc(a,b){
   if(a>b)
      return -1;
```

```
    else if(b>a)
       return 1;
    else
      return 0;
}
var arr = [10,1,3,33,6];

// 33 10 6 3 1
arr.sort(compareFunc);
alert(arr);
```

Розглянемо наступне завдання: у нас є рядок і нам потрібно його конвертувати в масив, розбивши на елементи масиву на основі якогось роздільника. Для того, щоб це зробити, використовуємо метод рядка під назвою split. Його сигнатура:

```
name_of_string.split(separator)
```

- separator — сепаратор, за його допомогою ми розбиваємо рядок на елементи масиву.

В результаті роботи методу повертається масив результату. Цей метод можна використовувати, наприклад, під час роботи з текстом. Розіб'ємо рядок за допомогою виклику split.

```
var str = "apple,onion,strawberry";

// розбиваємо на підставі,
// у масиві буде три елементи: apple, onion, strawberry
var arr = str.split(',');
// apple на екрані
alert(arr[0]);
```

Зауважте, що split це рядковий метод. Масив має зворотний метод join. Він використовується для конвертування масиву в рядок. Наприклад:

```
var arr = ["bmw", "audi", "opel"];

// створюємо рядок
// як розділовий символ між елементами
// масиву вказуємо *
var str = arr.join("*");

// bmw*audi*opel
alert(str);

// якщо не вказати роздільник, то буде використано ,
var str2 = arr.join();

// bmw,audi,opel
alert(str2);
```

Ми навели лише частину прикладів методів масиву JavaScript. З рештою, ви можете ознайомитись, ретельно вивчивши MSDN або документацію за посиланням.

Рядки

Об'єкт String

У JavaScript немає окремих типів для рядків і символів. Є тільки рядковий тип і якщо вам потрібно працювати з одним символом, ви створюєте рядок з одним символом. Вміст рядка може бути укладено у подвійні або одинарні лапки. Для JavaScript між ними немає різниці. Ви можете вибрати будь-який стиль, хоча на наш погляд використання подвійних лапок більш звичне. Приклад створення рядків:

```
// використовуємо подвійні лапки
var str="Test string";
alert(str);

// використовуємо одинарні лапки
var str2 = 'New string';
alert(str2);
```

Внутрішнім форматом зберігання рядків у JavaScript є Unicode. І при цьому абсолютно неважливо, яке у вас кодування вашої сторінки. Ви можете включати спеціальні символи у свій рядок. Наприклад, escape-послідовності. Приклад рядків із такими символами:

```
// \t - табуляція
var str = "Sun\t is going \\down\";

// Sun is going \down\
alert(str);
```

```
// для вставки лапок у рядок треба використовувати \"
var str2 = "Yes";

// на екрані "Yes"
alert(str2);
```

Для доступу до конкретного елемента рядка необхідно використати відомі вам [].

```
var str="trees and fruits";

// на екрані trees and fruits
alert(str);

// на екрані t
alert(str[0]);

// на екрані s
alert(str[4]);
```

Рядки в JavaScript є незмінними об'єктами. Це означає, що після створення рядка ви не можете змінити і-ий елемент рядка (наприклад, елемент за індексом 0 або 5). Ви можете перетворити рядок повністю!

```
var str="trees and fruits";

// на екрані trees and fruits
alert(str);
str[0] = "Z";

// на екрані однаково t
alert(str[0]);
str[4] = 'W'
```

```
// на екрані s
alert(str[4]);
str = "Here is a car";

// на екрані Here is a car
alert(str);
```

Для конкатенації рядків використовується оператор +

```
var first = "boat";
var second= "river";
var result = first + "and" + second;

// boat and river
alert(result);
```

Властивості та методи String

У рядку є велика кількість властивостей та методів, які дуже корисні для вирішення того чи іншого завдання. Почнемо з очевидного: властивість length використовується для отримання довжини рядка:

```
var str="gold";
// 4
alert(str.length);
```

Ви не можете зменшити або збільшити довжину рядка, задавши значення length явно. Це не надасть вам бажаного результату.

Метод charAt є аналогом [] і використовується для доступу до і-му елементу масиву. Відмінність між charAt

і [] полягає в тому, що при зверненні до неіснуючого індексу charAt повертає порожній рядок, а [] значення undefined.

```
var str="gold";
// g
alert(str.charAt(0));
// порожній рядок
alert(str.chartAt(15));
// undefined
alert(str[15]);
```

Методи toLowerCase та toUpperCase змінюють регістр літер. Метод toLowerCase змінює регістр на нижній, а toUpperCase змінює регістр на верхній. Зверніть увагу, що нове значення повертається з методів, але вміст самого рядка не змінюється.

```
var str="FootballL";
var newStr = str.toLowerCase();
// Football
alert(newStr);
// FootballL
alert(str);
```

Раніше з уроку ви вже дізналися про методи indexOf та lastIndexOf, які використовувалися для пошуку за масивом. У рядка є свої версії цих методів, які поводяться так само, як і їх аналоги в масиві. Приклад пошуку значення у рядку:

```
var str = "earth and sun";
var index = str.indexOf("sun");
```

```
// значення індексу дорівнює 10
alert(index);
index=str.indexOf("moon");
// значення індексу дорівнює -1
// тому що moon немає в рядку
alert(index);
```

А тепер порахуємо, скільки разів певне слово зустрічається у рядку:

```
var str = "test it is test sun test no";
var counter = 0;
var wordToFind="test";
var index = str.indexOf(wordToFind);
while (index! = -1) {
          counter++;
          index = str.indexOf(wordToFind,index+1);
}
// 3
alert(counter);
```

Подібний приклад пошуку ви вже бачили у розділі про масиви.

Методи substr, substring використовуються для отримання рядка. Почнемо з substring(start, [end]). Метод повертає підрядок, починаючи з індексу start, але не включаючи індекс end. Якщо end не вказано, то повертаємо підрядок до кінця рядка.

```
var str="Some value";

// end - необов'язковий параметр, який можна опустити
var newStr = str.substring(2);
```

```
// me value
alert(newStr);
newStr = str.substring(1,3);
// om
alert(newStr);
```

Метод substr(start, [length]) працює трохи інакше. Він повертає підрядок починаючи зі start, при цьому можна вказати довжину підрядка у другому параметрі. Якщо ж довжина не вказана, повертається підрядок до кінця оригінального рядка.

```
var str="Some value";
// length - необов'язковий параметр, який можна опустити
var newStr = str.substr(2);
// me value
alert(newStr);
newStr = str.substr(1,3);
// ome
alert(newStr);
```

Для порівняння рядків з урахуванням locale, використовують метод localeCompare(compareValue). Якщо рядки між собою рівні метод поверне 0, якщо рядок, який викликав метод більше рядка у параметрі, метод поверне 1, інакше -1. Принципи порівняння такі ж, як у функції strcmp з відомої вам мови С.

```
var str = "cheese";
// потрапимо в if, тому що рядки рівні між собою
if(str.localeCompare("cheese")==0){
        alert("Strings are equal! Cheese!");
}
```

```
else {
    alert("Strings are not equal! Not cheese!");
}

str = "Fb";
// перший рядок більше, оскільки порівняння
// відбувається посимвольно до першої розбіжності.
// F більше за числовим кодом, ніж f
// тому перший рядок більше.
if(str.localeCompare("fb")>0){
    alert("First is greater");
}
else {
    alert("Equal or less than second string");
}
```

Затримки та інтервали

Періодичний виклик функцій

Іноді виникає необхідність викликати функцію через певну кількість часу. Наприклад, якщо ви реалізуєте онлайн гру і дали гравцеві 60 секунд на вирішення пазла, вам потрібно мати механізм, який відрахує ці 60 секунд, після чого гра повідомить користувачеві, що час вийшов. У звичайному житті такий механізм називається таймером. Ви можете щодня спостерігати роботу таймера в мікрохвильовій печі, коли обираєте функцію «Розігрів».

Для реалізації таймера JavaScript існує набір функцій. Почнемо з функції встановлення таймера.

```
setTimeout(функція/код, затримка, аргумент1, аргумент2, …)
```

Функція setTimeout використовується для встановлення таймера. Функція/код — функція або код, який запуститься через вказаний проміжок часу.

Затримка — час затримки таймера. Після закінчення часу буде запущено функцію/код з першого аргументу. Затримка вказується в мілісекундах. *Для довідки: 1000 мілісекунд = 1 секунда.*

Аргумент1, аргумент2, … — аргументи, які можна передати у функцію, яка спрацює, коли зазначена затримка закінчиться.

Функція setTimeout повертає ідентифікатор встановленого таймера. Його можна використовувати для того, щоб скасувати спрацювання таймера.

Почнемо знайомство з таймером із найпростіших прикладів. У цьому прикладі коду таймер спрацює за секунду і буде викликана функція HelloWorld, вказана у параметрі.

```
function HelloWorld(){
    alert("Hello world!");
}
setTimeout(HelloWorld, 1000);
```

Спробуємо передати аргументи у функцію:

```
function Sum(a,b){
    alert(a+b);
}
setTimeout(Sum, 1000,1,2);
```

Ми передали значення 1 (потрапило до параметра a) і 2 (потрапило до параметра b). Функція порахувала суму двох чисел і вивела її у вікно повідомлення.

Коли ми описували параметри для setTimeout, ми говорили, що перший параметр може бути функцією або просто кодом. Наприклад:

```
setTimeout("alert('hi')", 1000);
```

Тут alert використовується у якості коду. Вказувати код у такому вигляді не рекомендується, тому що він не є читабельним і мінімізатори коду під час обробки такого фрагмента можуть неправильно його інтерпретувати. Альтернативою є використання безіменних (анонімних) функцій.

```
setTimeout(function(){
    alert("Hi!");
}, 1000);
```

У цьому прикладі ми створили анонімну функцію з одним рядком коду для відображення вікна повідомлення. Такі анонімні функції можна створювати з параметрами.

```
setTimeout(
    function(a,b){
       alert(a*b);
    },
    1000, 3, 7
);
```

Після встановлення таймера повертається ідентифікатор. Його можна використовувати для скасування виклика функції.

```
var id = setTimeout(function(){
            alert("Boom!")},3000
);
alert("id timer: "+id);
```

У цьому прикладі ми відобразили ідентифікатор таймера і використовуємо отриманий ідентифікатор для його скасування. Функція скасування таймера clearTimeout.

```
clearTimeout(ідентифікатор_таймера)

var id = setTimeout(function(){
```

```
    alert("Boom!")},50000
);
clearTimeout(id);
```

Ми встановили таймер на 50 секунд, а потім скасували його. Звичайно, в результаті наших дій роботу таймера буде зупинено.

А що ж робити, якщо нам потрібне спрацювання певної функції через певні інтервали часу? Механізм, який ми розібрали вище, дозволяє створити функцію, яка буде викликана один раз. Аби вирішити цю проблему, можемо скористатися одним із двох способів. Перший спосіб — використання функції setInterval.

```
setInterval(функція/код, інтервал_часу, аргумент1,
            аргумент2, …)
```

Подивившись на функцію, можна легко зробити висновок про те, що вона повністю схожа з setTimeout. Відмінність лише у принципі дії. На відміну від setTimeout, функція/код, що вказана в першому параметрі, буде регулярно викликатися через вказаний інтервал часу. Для зупинки виклику функції потрібно буде викликати функцію clearInterval. Спробуємо використати інтервальний механізм:

```
setInterval(
    function(){
        alert("Boom!");
    },
    2000);
```

Функція, вказана у першому параметрі, буде викликатися кожні дві секунди. У цьому коді ми не викликали clearInterval, тому вікно повідомлення з написом Boom буде викликатися кожні дві секунди, поки вікно або вкладка браузера не будуть закриті. Тепер додаємо виклик clearInterval.

```
var id = setInterval(IntervalFunc,2000);
var counter = 0;

function IntervalFunc(){
    if(counter == 3){
       clearInterval(id);
       return;
    }
    counter++;
    alert("Boom");
}
```

Вікно повідомлення буде показано три рази, після чого зупинимо інтервальну функцію. Альтернативою використання setInterval є рекурсивний виклик setTimer. Наприклад:

```
var id = setTimeout(TimeOutFunc, 2000);
var counter = 0;

function TimeOutFunc() {
       // якщо таймер спрацював вже тричі, зупиняємо
       // процес
       if (counter == 3) {
           clearTimeout(id);
           return;
       }
```

```
        counter++;
        alert("Boom Timer");

        // наново ставимо таймер на дві секунди
        id = setTimeout(TimeOutFunc, 2000);
}
```

Принцип використання таймера замість інтервалів простий: всередині таймерної функції ми знову ставимо таймер. У нашому прикладі ми наново встановлювали таймер на дві секунди всередині таймерної функції. Коли наш цикл роботи виконається три рази, ми викличемо clearTimeout і вийдемо з таймерної функції. При установці нового таймера ми повинні вказувати ту ж затримку, як і у першому виклику. Наприклад:

```
var id = setTimeout(TimeOutFunc, 2000);
var counter = 1;

function TimeOutFunc() {
    alert("Boom Timer");
    switch(counter){
        case 1:
           id = setTimeout(TimeOutFunc, 5000);
           break;
        case 2:
           id = setTimeout(TimeOutFunc, 10000);
           break;
        case 3:
           clearTimeout(id);
           return;
    }
    counter++;
}
```

У нашому випадку кожен з таймерів має свою затримку: 2, 5, 10 секунд.

Для створення інтервальної функції можна використовувати setInterval або setTimeout. Вибір за вами. Ми можемо додати лише те, що setTimeout дещо гнучкіше, тому що у вас є можливість щоразу вказувати нову тривалість затримки.

Використання математичних можливостей

Об'єкт Math

Вирішуючи завдання з області програмування, вам можуть знадобитися математичні можливості, вбудовані в JavaScript. Як можна скористатися ними? Чи складно це зробити? Відповідь: ні. Для того, щоб використовувати математичні можливості JavaScript, вам необхідно звернутися до вбудованого об'єкту Math. Всередині цього об'єкта ви знайдете цілу міріаду функцій для вирішення тієї чи іншої математичної проблеми, що стоїть перед вами. Почнемо знайомство з внутрішнім улаштуванням цього об'єкта.

Властивості та методи об'єкту Math

Ми не розглядатимемо всі властивості та методи об'єкта Math. Почнемо з деяких властивостей:

- Math.PI — повертає число пі (приблизно 3.14);
- Math.E — повертає число Ейлера (приблизно 2.718);
- Math.SQRT2 — повертає квадратний корінь із двох (приблизно 1.414);
- Math.SQRT1_2 — повертає квадратний корінь із однієї другої (приблизно 0.707).

Приклад використання:

```
alert(Math.PI);
alert(Math.E);
```

Тепер давайте розглянемо деякі методи:

- Math.ceil(x) — повертає округлення параметра x вгору до найближчого цілого;
- Math.floor(x) — повертає округлення параметра x вниз до найближчого цілого;
- Math.round(x) — повертає округлене значення параметра x до найближчого цілого (якщо дробова частина більша або дорівнює 0,5 тоді заокруглення вгору, інакше вниз);
- Math.pow(x, y) — повертає значення x, зведене в ступінь y;
- Math.sqrt(x) — повертає квадратний корінь з x;
- Math.min() і Math.max — повертають відповідно мінімум і максимум із переданих параметрів;
- Math.abs(x) — повертає модуль переданого параметра x.

Приклад використання функцій округлення та відтинання значень:

```
var res = Math.ceil(4.3);
// 5
alert(res);
res = Math.round(4.3);
// 4
alert(res);
res = Math.round(4.5);
// 5
alert(res);
res = Math.floor(4.5);
// 4
alert(res);
```

Приклад використання функцій з пошуку максимуму та мінімуму:

```
var res = Math.min(1, 4, -1,-9, 20);

// -9
alert(res);
res = Math.max(21, 3, 4, 5, 6, 7);

// 21
alert(res);
```

Ми впевнені, що знайомство та використання інших математичних методів не викличе у вас складнощів.

Випадкові числа

У житті будь-якого програміста настає момент, коли доводиться мати справу з випадковими числами. Наприклад, вам необхідно створити програмне забезпечення для лотереї або гральних автоматів. У цьому випадку вам потрібен механізм, який генеруватиме випадкові числа. Для генерації псевдовипадкових чисел JavaScript використовується метод random об'єкта Math.

- Math.random() — повертає псевдовипадкове число у діапазоні від 0 до 1;

Приклад використання:

```
var res = Math.random();

// псевдовипадкове число
alert(res);
res = Math.random();
```

```
// псевдовипадкове число
alert(res);
```

Зверніть увагу, що число псевдовипадкове. Це означає, що число, яке буде повернене, буде згенеровано алгоритмічно. Зазвичай основу алгоритмів, які генерують випадкові числа, складають поняття початкової точки. За замовчуванням, зазвичай при першому виклику функції для генерації випадкового числа, у якості початкової точки, беруть кількість мілісекунд, що пройшли з 1 січня 1970 року. У деяких мовах можна змінити початкову точку. У JavaScript ця можливість прихована. Як можна згенерувати псевдовипадкове число в іншому діапазоні, відмінному від 0 і 1. Для вирішення цієї задачі використовується зв'язка random і floor. Наприклад:

```
// випадкове значення від 0 до 9
alert(Math.floor(Math.random()*10));

// випадкове значення від 0 до 11
alert(Math.floor(Math.random()*11));

// випадкове значення від 1 до 10
alert(Math.floor(Math.random()*10) + 1);
```

Як ми бачимо з прикладу вище, використання випадкових чисел не викликає складнощів.

Об'єкт Date. Обробка дати та часу

Об'єкт Date в JavaScript використовується для керування датою та часом. За допомогою цього об'єкта можна отримувати дані про поточну дату, а також дату в минулому або майбутньому.

Створюється екземпляр об'єкта Date за допомогою ключового слова new. Якщо у дужках ви не передаєте дані, то змінна ініціалізується поточними даними дати та часу з комп'ютера користувача, що завантажив файл зі скриптом.

```
const today = new Date();
console.log(today);
```

При виведенні в консоль ви побачите дату у форматі «День тижня, місяць, число, рік, години:хвилини:секунди, часовий пояс за Гринвічем» приблизно так:

```
Sat Aug 07 2021 20:22:50 GMT+0300
```

Рисунок 1

Отримати поточну дату можна, написавши:

```
Date.now()
```

У цьому випадку ви отримаєте, наприклад, число 1628357359903 — це кількість мілісекунд, що пройшли з 1 січня 1970 р. Ця дата вважається «точкою відліку» для

всіх дат в JavaScript. Також кількість мілісекунд можна отримати за допомогою методу getTime() або valueOf().

Якщо вас цікавить певна дата, то при створенні змінної у дужках потрібно вказати її в одному з рядкових форматів:

```
let date1 = new Date("2021-05-17"); // рік-місяць-день
console.log(date1);
let date2 = new Date("06/25/2021"); // місяць/день/рік
console.log(date2);
const date3 = new Date('November 2, 1999 13:25:00');
console.log(date3);
const date4 = new Date('1999-11-02T13:25:00');
console.log(date4);
const date5 = new Date('02 November 1999 13:25:00');
console.log(date5);
```

Якщо ви вкажете неправильну дату, то в такому випадку JavaScript під час спроби використати дату поверне рядок '*Invalid Date*'.

Ви також можете задавати дату у вигляді кількох чисел через кому:

```
const date6 = new Date(2022, 0, 12, 03, 45, 12, 500);
// рік, місяць 0-11, день, години, хвилини, секунди,
// мілісекунди
console.log(date6);
```

Зверніть увагу на те, що рік потрібно задавати у вигляді 4-х цифр, місяці вказуються від 0 (січень) до 11 (грудень), день від 1 до 31 (якщо день НЕ вказано, то буде 1), години, хвилини, секунди та мілісекунди можна не вказувати, тоді замість них буде підставлено 0.

Створений будь-яким із перерахованих методів об'єкт Date містить число мілісекунд, що пройшли з 1 січня 1970 р. Тому будь-які екземпляри об'єкта Date можна віднімати один від одного, отримуючи різницю між ними в мілісекундах, а потім переводити в години, дні, роки тощо, наприклад:

```
console.log('Різниця між датами в мілісекундах',
            date2 - date1);
console.log('Різниця між датами в днях',
            Math.round((date2 - date1)/24/60/60/1000));
```

У консолі ми побачимо такий результат для оголошених вище змінних та date2:

Різниця між датами у мілісекундах 3358800000

Різниця між датами у днях 39

Для будь-якого з представлених способів ви можете дізнатись кількість мілісекунд з 1 січня 1970 року, використовуючи метод Date.parse(), вказавши у дужках дату у вигляді рядка.

```
console.log(Date.parse('11/08/2025'));
```

Ще один спосіб створення дати передбачає, що у дужках ви вказуєте кількість мілісекунд:

```
const date7 = new Date(1728357351109); //Tue Oct 08
                                       //2024 06:15:51
console.log(date7);
```

Різні способи створення дат дозволяють маніпулювати ними у різних завданнях.

Наприклад, нам необхідно отримати змінну поточної дати, тобто поточну, а також змінні зі вчорашньою та завтрашньою датами. Найпростіше це зробити на основі останнього способу створення екземпляра об'єкта Date.

```
let today = new Date();
let yesterday = new Date(today - 24 * 60 * 60 * 1000);
let tomorrow = new Date(today + 24 * 60 * 60 * 1000);
console.log(yesterday, today, tomorrow);
```

Об'єкт Date має ряд методів, за допомогою яких ви можете отримати доступ до окремих даних зі змінної-дати:

- getFullYear() — повертає рік, що складається із 4 цифр;
- getMonth() — повертає номер місяця, від 0 (січень) до 11 (грудень);
- getDate() — повертає день місяця, від 1 до 31;
- getMilliseconds(), getSeconds(), getMinutes(), getHours() — повертають відповідно мілісекунди, секунди, хвилини або години;
- getDay() — повертає день тижня: 0 відповідає неділі, 1 — понеділку, 2 — вівторку тощо.

Типове завдання на використання одного з методів — це визначення того, на який день тижня припадав день народження користувача.

```
let userdata = prompt("Enter your birthday in
                       year-month-day format",
                      "2002-08-14");
```

```
let birthday = new Date(userdata) == 'Invalid Date' ?
                new Date() : new Date(userdata);
console.log(birthday);
let days = ['sunday', 'monday', 'tuesday', 'wednesday',
            'thursday', 'friday', 'saturday'];
alert("You were born on " + days[birthday.getDay()]);
```

Методи об'єкта Date також дозволяють встановити дату та час. Для цього існують такі методи:

- setFullYear() — встановлює рік, що складається із 4 цифр. Необов'язковими параметрами є місяць та день місяця.
- setMonth() — встановлює номер місяця, від 0 (січень) до 11 (грудень). Необов'язковим параметром є день місяця.
- setDate() — встановлює день місяця від 1 до 31.
- setMilliseconds(), setSeconds(), setMinutes(), setHours() — встановлюють відповідно кількість мілісекунд, секунд, хвилин або годин.
- getDay() — повертає день тижня: 0 відповідає неділі, 1 — понеділку, 2 — вівторку тощо.

Всі методи зі списку вище повертають результат для місцевої часової зони, в якій часто присутня позначка у вигляді GMT+0300 тощо, це означає, що до часу нульового (Гринвіцького) меридіана додається або віднімається певна кількість годин. При створенні екземплярів об'єкта Date необхідно враховувати, що поверхня Землі поділена на 24 часові пояси. У JavaScript існує всього два часові пояси:

- Local Time — часовий пояс, в якому знаходиться ваш комп'ютер;
- UTC (*Coordinated Universal Time*) — те саме, що і час за Гринвічем (GMT).

Тому в JavaScript існують UTC-варіанти методів, що повертають день, місяць, рік тощо, для зони GMT+0 (UTC), тобто Гринвіцький меридіан: getUTCFullYear(), getUTCMonth(), getUTCDay(). Тобто відразу після «get» ставиться «UTC».

Крім того, в об'єкті Date є ще методи, що дозволяють вивести дату у тому вигляді, у якому вона представлена на пристрої користувача, тобто в локальному форматі з урахуванням зсуву часу відносно Гринвіцького меридіана:

- toLocaleString() — повертає рядок дати та часу у локальному форматі, тобто у форматі тієї часової зони, в якій знаходиться операційна система користувача.
- Наприклад, для україномовної зони формат дати буде таким: 21.09.2021, 10:39:47.
- toLocaleDateString() — повертає рядок дати у локальному форматі, наприклад, 21.09.2021.
- toLocaleTimeString() — повертає рядок часу у локальному форматі, наприклад, 10:39:47.

У наступному прикладі ми встановимо дату, а потім з'ясуємо, в яких ще місяцях цей день випадає на той же день тижня. У прикладі це понеділок. У випадку, якщо ця дата припадає на понеділок в іншому місяці, ми виведемо цю дату в консоль.

```
let d = new Date('02/07/2022');
let month = d.getMonth(), weekDay = d.getDay();
console.log(weekDay, d.toLocaleDateString());
for(let i=0; i<12; i++){
    if(i ==  month) continue;
    d.setMonth(i);
    if(weekDay == d.getDay())
       console.log(d, d.toLocaleDateString());
}
```

Домашнє завдання

Завдання 1

За допомогою методів setTimeout() або setInterval() виведіть у тіло документа методом document.write() перший рядок з монологу Гамлета *'To be, or not to be, that is the question...'* так, щоб букви з'являлися по одній через 200-300 мілісекунд, а потім із нового рядка, також методом document.write(), потрібно вивести *'William Shakespeare, from «Hamlet»'*.

Подивитися, як це має виглядати, можна у файлі *hamlet.gif*.

Рисунок 2

Завдання 2

Розрахуйте скільки днів, годин, хвилин і секунд залишилося до Нового року. Виведіть ці значення відповідним чином, використовуючи метод document.write() з тегами <p> та та класами для них. Стилі

можна записати в окремому CSS-файлі. Якщо одне з чисел буде менше ніж 10, його потрібно вивести з провідним 0, наприклад, так:

Рисунок 3

Зробіть свій скрипт універсальним, поставивши дату наступного Нового року відповідно до поточної дати за допомогою методів об'єкта Date.

Завдання 3

Створіть об'єкт list, задайте для нього:

- властивість values, яка містить масив схожих значень, наприклад, будь-яких продуктів;
- метод printList(), який сортує всі елементи масиву values в алфавітному порядку та виводить їх у вигляді нумерованого списку в тіло документа методом document.write(),;
- метод add(product), який додає до values ще один елемент.

Виведіть спочатку масив початкових значень об'єкта list за допомогою методу printList(). Наприклад, це буде список продуктів:

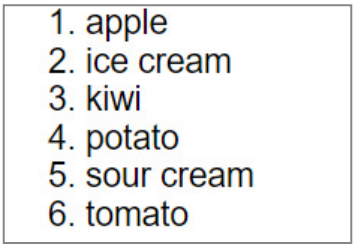

Рисунок 4

Потім додайте ще один елемент за допомогою методу add() і знову виведіть всі значення об'єкта list методом printList().

Рисунок 5

Потім змініть всі значення властивості list.values на інший масив і знову виведіть його методом printList(). Наприклад, так:

Рисунок 6

Урок 2-1.
Об'єкт. Масиви. Рядки. Об'єкт Date. Об'єкт Math