МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

М. С. ДУТЧАК, І. М. ЛАЗАРОВИЧ

Лабораторний практикум із дисципліни “Програмування Інтернет”

Івано-Франківськ
2023 р

УДК 004.51

Д 84

Рекомендовано до друку Вченою радою факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника» (протокол №7 від 22.08.2023 р.)

Рецензенти:

Пікуляк М.В. – кандидат технічних наук, доцент кафедри інформаційних технологій факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника;

Ткачук В.М. – кандидат фізико-математичних наук, доцент кафедри інформаційних технологій факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника.

Дутчак М.С.

Д 84 Дутчак М.С, Лазарович І.М. Лабораторний практикум із дисципліни “Програмування Інтернет”, Івано-Франківськ: ПНУ ім. В. Стефаника, 2023, 131 с.

У лабораторному практикумі наведено теоретичні відомості, завдання та рекомендації щодо виконання лабораторних робіт з дисципліни «Програмування Інтернет», що дозволяє здобувачам вищої освіти освоїти технології Web-програмування мовою програмування JavaScript та бібліотеки Jquery.

Лабораторний практикум рекомендовано для здобувачів вищої освіти спеціальності 121 «Інженерія програмного забезпечення» та інших спеціальностей галузі 12 Інформаційні технології.

УДК 004. 51

Д 84

Зміст

Передмова

Web-сервіси дедалі ширше знаходять впровадження практично в усіх сферах людської діяльності, тому освоєння інструментальних засобів та методології Web-програмування є актуальним етапом підготовки сучасного фахівця у сфері Web-розробки. Однією із основних мов Web-програмування є мова JavaScript, а також численні її бібліотеки, зокрема бібліотека JQuery. Саме цим інструментам Web-програмування і присвячений даний лабораторний практикум, який є важливою складовою частиною навчального процесу здобувачів вищої освіти спеціальності 121 «Інженерія програмного забезпечення», а також здобувачів вищої освіти інших спеціальностей ІТ-галузі.

Лабораторний практикум містить теоретичний матеріал, який супроводжується значною кількістю прикладів і завдання до виконання лабораторних робіт, які включають завдання для індивідуального виконання та спрямовані на закріплення теоретичного матеріалу. У лабораторному практикумі розглянуто основні можливості мова програмування JavaScript, а також бібліотеки JQuery, зокрема описано загальний синтаксис, основні мовні конструкції, роботу з функцій, обробники подій, об'єктну модель документа (DOM), методи обробки рядків та особливості синтаксису та застосування регулярних виразів для формування шаблонів пошуку. Значну увагу приділено обробці масивів, створенню анімацій та динамічних Web-вебсторінок, зокрема описано можливості зміни значень властивостей HTML-елементів засобами JavaScript та JQuery.

Наведений матеріал призначений для студентів, які вивчають веб-програмування, може бути використаний ними для самостійної роботи, а також для підготовки до виконання лабораторних і практичних робіт. Для розуміння та засвоєння поданих відомостей необхідно володіти навиками розробки веб-сторінок на основі HTML, CSS та Javascript.

Тема 1. Основи JavaScript. Використання функцій. Події.

JavaScript – мова програмування для створення скриптів, що розроблена фірмою Netscape. За допомогою JavaScript можна створювати інтерактивні Web-сторінки. Щоб запускати скрипти, що написані з використанням мови JavaScript необхідний браузер, що має змогу працювати з JavaScript. Код скрипта JavaScript розміщується безпосередньо на HTML-сторінці з використанням тегу <script>. Події та їх обробники є одними з важливих питань для програмування на мові JavaScript. Події, головним чином викликаються діями користувача. Наприклад, клік по кнопці або текстовому полю приводить до виникнення події "Click". Якщо курсор миші перетинає певний елемент Web-сторінки – відбувається подія MouseOver та ін. Існує кілька різних типів подій, розглянемо спочатку подію onClick. Наведений нижче код є прикладом простої програми обробки події onClick:

```
<form>
<input type="button" value="Click me" onClick="alert('Hello!!!')">
</form>
```

Однією з найвідоміших функцій є alert(), що дозволяє видати користувачеві інформацію або попередження у окремому діалоговому вікні. При її виклику, у якості параметра передається деякий рядок, у наведеному прикладі це “Hello!!!”. Таким чином, коли користувач клікає на кнопці або іншому елементі форми скрипт створює вікно, що містить текст ”Hello!!!”.

Функції на мові JavaScript.

Функції

В JavaScript є вбудовані функції, які можна використовувати в програмі, але їхній код не можна ні змінити, ні подивитись. Крім цих функцій, користувач може створювати свої власні .

Вбудовані функції

parseInt(рядок, основа) — перетворює вказаний рядок на ціле число у системі числення за вказаною основою (8, 10, 16); якщо основа не вказана, то за замовчуванням використовується 10.

parseFloat(рядок, основа) — перетворює вказаний рядок у число з плаваючою крапкою.

isNaN(значення) — повертає true, якщо вказане значення не є числом, і false в противному випадку.

eval(рядок) — обчислює вираз у вказаному рядку; вираз повинен бути написаний на мові JavaScript (не містити дескрипторів HTML).

escape(рядок) — повертає рядок у вигляді %XX, де XX — ASCII-код вказаного символу.

unescape(рядок) — здійснює обернене перетворення.

typeOf(об'єкт) — повертає тип вказаного об’єкта у вигляді символьного рядку.

prompt(text|[,value])— виводить повідомлення у вікні з текстовим полем і двома кнопками: "ОК" і "ВІДМІНА". Повертає введене значення або null, якщо відвідувач натиснув на кнопку "ВІДМІНА". Відвідувач не може робити нічого іншого, поки не вибере одну з кнопок.

Аргумент text: текст повідомлення, value: рядок, введений в текстове поле за замовчуванням. Необов'язковий параметр.

Приклад:

```
let years = prompt ('Скільки вам років?', 100)
alert ('Вам' + years + 'років!')
```

Функції користувача

У більшості програм на мові JavaScript, як і у випадках використання інших мов програмування, широко використовуються функції. Для прикладу розглянемо скрипт, що друкує деякий текст три рази поспіль:

```
<html>
<script language="JavaScript">
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");
</script>
</html>
```

Такий скрипт напише:

“Ласкаво просимо на мою сторінку!

Це JavaScript!”

тричі.

Як бачимо поставлена задача вирішена за допомогою того, що відповідна частина коду була дубльована три рази. Такий підхід не є ефективним. Наступний скрипт виконує ту ж саму задачу з використанням функції.

```
<html>
<script language="JavaScript">
function myFunction() {
document.write("Ласкаво просимо на мою сторінку!<br>");
document.write("Це JavaScript!<br>");}
myFunction();
myFunction();
myFunction();
</script>
</html>
```

І хоча кількість рядків у даному випадку не змінилась структура програми стала більш зрозумілою, читабельною, та придатною до подальшої модифікації. Отже, функції користувача — це функції, які ви можете створити самі, за власним вибором, для вирішення своїх задач. Функція задається своїм описанням, яке починається словом “function” і має наступний синтаксис:

function *ім'я_функції*(*параметри*) {

код

}

Через необов'язковий елемент “параметри” можна зчитати передані у функцію змінні. Якщо функція повинна повернути деяке значення, то в її тілі використовується оператор return, справа від якого вказується те, що необхідно повернути. Цей оператор може міститись в тілі функції декілька разів.

Ініціалізація в тілі функції змінної з ключовим словом let створює (ініціалізує) локальну змінну, навіть якщо вона вже була оголошена у зовнішній програмі; якщо вона створена лише з допомогою оператора присвоювання, то вона є глобальною (створення даного типу змінних не рекомендується).

Приклад:

```
function result(a,b){
c=a+b
return c
}
```

Приклад оголошення та виклику функції:

1. Оголошуємо функцію

```
<script>
function buttonOff() {  alert("Sorry, button disabled.") ; }
</script>
```

2. Задаємо в коді HTML атрибут

```
 <INPUT id=btn TYPE="button" VALUE="Calculate" onClick=" buttonOff() ">
```

або в JavaScript

```
<script>
document.getElementById('btn').onclick = buttonOf
</script>
```

Видимість

```
 let b=1;
function fun1()
{
let a;
// тут існують змінні a та b
}
// тут існує лише змінна b (змінна a не існує )
```

Змінні на мові JavaScript.

Найчастіше застосункам на JavaScript потрібно працювати з інформацією. Ось два приклади:

- онлайн-магазин – інформацією можуть бути товари, які продаються, і вміст кошика;
- застосунок для чату – інформація може включати користувачів, повідомлення та багато іншого.

Змінні використовуються для зберігання цієї інформації.

Змінна - це “іменована частина сховища”, в якій зберігаються дані. Ми можемо використовувати змінні, щоб зберігати дані про товари, відвідувачів, дати тощо.

Щоб створити змінну, використовуйте ключове слово let. Цей рядок нижче створить (оголосить, ініціалізує) змінну з ім’ям “message”:

```
let message;
```

Тепер ми можемо присвоїти значення цій змінній, використовуючи оператор присвоєння “=”:

```
message = 'Привіт'; // збереження рядка 'Привіт' у змінній `message`
```

Тепер рядок збережено в частину пам’яті, яка зв’язана з цією змінною. Ми можемо отримати доступ до даних, використовуючи ім’я змінної:

```
alert(message); // показує вміст змінної
```

Щоб писати менше коду, ми можемо поєднувати оголошення змінної та її присвоєння в одному рядку:

```
let message = 'Привіт!'; // оголошення змінної і присвоєння значення
alert(message); // Привіт!
```

Ми також можемо оголосити декілька змінних в одному рядку:

```
let user = 'Іван', age = 25, message = 'Привіт';
```

Таке оголошення може виглядати коротшим, але заради кращої читабельності, оголошуйте змінні з нового рядка.

```
let user = 'Іван';
let age = 25;
let message = 'Привіт';
```

Дехто також оголошує змінні в такому багаторядковому стилі:

```
let user = 'Іван',
age = 25,
message = 'Привіт';
```

Або навіть у стилі “кома спочатку”:

```
let user = 'Іван'
  , age = 25
  , message = 'Привіт';
```

Технічно, всі ці способи роблять одне й те ж. Тому це питання особистого смаку та естетики.

var замість let

У старих скриптах ви можете знайти інше ключове слово: var замість let:

```
var message = 'Привіт';
```

Ключове слово var майже таке, як let. Воно теж оголошує змінну, але дещо іншим, “застарілим” способом.

Іменування змінних

В JavaScript є два обмеження, які стосуються імен змінних:

1. Ім’я має містити лише букви, цифри або символи $ і _.
2. Перший символ не має бути числом.

Ось приклади допустимих імен:

```
let userName;
let test123;
```

Для написання імені, яке містить декілька слів, зазвичай використовують “верблюжий регістр” (camelCase). Тобто слова йдуть одне за одним, і кожне слово пишуть із великої букви й без пробілів: myVeryLongName. Зауважте, що перше слово пишеться з маленької букви.

Що цікаво – знак долара '$' і знак підкреслення '_' також можна використовувати в іменах. Це звичайні символи, такі ж, як і букви, без будь-якого особливого значення.

Ці імена також допустимі:

```
let $ = 1; // оголошено змінну з ім’ям "$"
let _ = 2; // а тепер змінна з ім’ям "_"
alert($ + _); // 3
```

Приклади недопустимих імен змінних:

```
let 1a; // не може починатися з цифри
let my-name; // дефіс '-' недопустимий в імені
```

Регістр має значення. Змінні з іменами apple і APPLE – це дві різні змінні.

Нелатинські букви дозволені, але не рекомендуються. Можна використовувати будь-яку мову, включно з кирилицею або навіть ієрогліфами, наприклад:

```
let назва = '...';
let 我 = '...';
```

Технічно тут немає помилки. Такі імена дозволені, проте є міжнародна традиція використовувати латинський алфавіт, зокрема і англійську мову в іменах змінних (наприклад, iLoveUkraine).

Зарезервовані слова

Є писок зарезервованих слів, які не можна використовувати як імена змінних, тому що ці слова використовує сама мова.

Наприклад: let, class, return і function зарезервовані.

Такий код видаватиме синтаксичну помилку:

```
let let = 5; // не можна назвати змінну "let", помилка!
let return = 5; // також не можна назвати змінну "return", помилка!
```

Створення змінної без використання use strict

Зазвичай нам потрібно оголосити змінну перед її використанням. Але в старі часи було технічно можливим створити змінну простим присвоєнням значення, без використання let. Це все ще працює, якщо не

вмикати суворий режим у наших скриптах для підтримання сумісності зі старими сценаріями.

// "use strict" в цьому прикладі не використовується

num = 5; // якщо змінна "num" не існувала, її буде створено, при чому вона буде глобальною

alert(num); // 5

Це погана практика, яка призведе до помилки в суворому режимі: "use strict";

num = 5; // помилка: num не оголошено

Константи

Щоб оголосити константу (незмінювану змінну), використовуйте ключове слово const замість let:

const myBirthday = '18.04.1982';

Змінні, оголошені за допомогою const, називаються “константами”. Їхні значення не можна перепризначити. Спроба це зробити призведе до помилки:

const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // помилка, не можна перевизначати константу!

Коли програміст впевнений, що змінна ніколи не буде змінюватися, він може оголосити її через const, що гарантує постійність і буде зрозумілим для кожного.

Константи в верхньому регістрі

Широко поширена практика використання констант як псевдонімів для значень, які важко запам’ятати і які відомі до початку виконання скрипту.

Такі константи пишуться в верхньому регістрі з використанням підкреслень.

Наприклад, створімо константи, в які запишемо значення кольорів у так званому “Web-форматі” (в шістнадцятковій системі):

const COLOR_RED = "#F00";

const COLOR_GREEN = "#0F0";

const COLOR_BLUE = "#00F";

```
const COLOR_ORANGE = "#FF7F00";
// ...коли потрібно вибрати колір
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Переваги:

COLOR_ORANGE набагато легше запам’ятати, ніж "#FF7F00" .

Набагато легше припуститися помилки в "#FF7F00", ніж під час введення COLOR_ORANGE.

Під час читання коду COLOR_ORANGE набагато зрозуміліше, ніж #FF7F00.

Коли ми маємо використовувати для констант великі букви, а коли звичайні? Давайте це з’ясуємо.

Назва “константа” лише означає, що змінна ніколи не зміниться. Але є константи, які відомі нам до виконання скрипту (наприклад, шістнадцяткове значення для червоного кольору), а є константи, які *вираховуються* в процесі виконання скрипту, але не змінюються після їхнього початкового присвоєння.

Наприклад:

```
const pageLoadTime = /* час, витрачений на завантаження Web-сторінки */;
```

Значення pageLoadTime невідоме до завантаження сторінки, тому її ім’я записано звичайними, а не великими буквами. Але це все ще константа, тому що вона не змінює значення після присвоєння.

Інакше кажучи, константи з великими буквами використовуються як псевдоніми для “жорстко закодованих” значень.

Придумуйте правильні імена

Імена змінних повинні мати чіткий і зрозумілий сенс, який описує дані, що в них зберігаються.

Іменування змінних – одна із важливих навичок у програмуванні. Швидкий погляд на імена змінних може показати, який код був написаний початківцем, а який досвідченим розробником.

У реальному проєкті більшість часу тратиться на змінення і розширення наявної кодової бази, а не на написання чогось цілком нового. Коли ми повертаємося до якогось коду після виконання чогось іншого впродовж тривалого часу, набагато легше знайти інформацію, яку добре позначено. Або, інакше кажучи, коли змінні мають хороші імена.

Декілька хороших правил:

Використовуйте імена, які легко прочитати, як-от userName або shoppingCart.

Уникайте використання абревіатур або коротких імен, таких як a, b та c, окрім тих випадків, коли ви точно знаєте, що так потрібно.

Робіть імена максимально описовими і лаконічними. Наприклад, такі імена погані: data і value. Такі імена нічого не говорять. Їх можна використовувати лише тоді, коли з контексту очевидно, на які дані або значення посилається змінна.

Погоджуйте з вашою командою (та з самим собою), які терміни будуть використовуватися у проєкті. Якщо відвідувач сайту називається “user”, тоді ми маємо давати відповідні імена іншим пов’язаним змінним: currentUser або newUser, замість currentVisitor або newManInTown.

Повторно використовувати чи створювати нові?

Є ліниві програмісти, які замість оголошення нових змінних повторно використовують наявні.

У результаті їхні змінні схожі на коробки, в які люди кидають різні речі, не змінюючи на них наклейки. Такі програмісти економлять трішки часу на оголошенні змінних, але втрачають вдесятеро більше під час налагодження.

Додаткова змінна – це добро, а не зло.

Сучасні браузери оптимізують код досить добре, тому додаткові змінні не погіршують продуктивність. Використання різних змінних для різних значень може навіть допомогти рушію оптимізувати ваш код.

Оператори

Числові

+ Додавання для чисел та з'єднання для рядків

- Віднімання

* Множення

/ Ділення

% Остача від ділення (12%5 повертає 2, УВАГА 12.3%5 повертає 2.3)

++ інкремент

-- декремент

Об'єкт Math

Об'єкт Math дозволяє виконувати різні математичні дії. Об'єкт Math не є конструктором. Це означає, що всі властивості і методи об'єкту Math можуть викликані без його попереднього створення.

Синтаксис

let x = Math.PI; // Поверне число «пі»
let y = Math.sqrt (16); // Поверне квадратний корінь числа 16.

Властивості об'єкта Math

Властивість	Опис
E	Повертає число Ейлера (близько 2.718)
LN2	Повертає натуральний логарифм числа 2 (близько 0.693)
LN10	Повертає натуральний логарифм числа 10 (близько 2.302)
LOG2E	Логарифм E за основою 2 (близько 1.442)
LOG10E	Логарифм E по підставі 10 (близько 0.434)
PI	Число "пі" (близько 3.14159)

Методи об'єкта Math

Методи	Опис
abs (x)	Абсолютне значення числа x

ceil (x)	Округлення числа x в більшу сторону
floor (x)	Округлення числа x в меншу сторону
max (x, y, z, ..., n)	Обчислення максимуму з n чисел
min (x, y, z, ..., n)	Обчислення мінімуму з n чисел
pow (x, y)	Зводить число x в ступінь y
random ()	Повертає випадкове число від 0 до 1
round (x)	Округлення чисел до найближчого цілого числа
sqrt (x)	Обчислення квадратного кореня з числа x

Присвоєння

a += b a = a + b

a -= b a = a - b

a *= b a = a * b

a /= b a = a / b

a %= b a = a % b

a &= b a = a & b

a |= b a = a | b

a ^= b a = a ^ b

Логічні

! NOT

&& AND (якщо перший операнд = false, другий операнд не обчислюється)

|| OR (якщо перший операнд = true, другий операнд не обчислюється)

Оператори порівняння

Багато з операторів порівняння нам відомі з математики. В JavaScript вони записуються ось так:

Більше/менше: a > b, a < b.

Більше/менше або дорівнює: a >= b, a <= b.

Дорівнює: a == b. Зверніть увагу, для порівняння потрібно використовувати два знаки рівності ==. Один знак рівності a = b означав би присвоєння.

Не дорівнює: a != b.

Результат порівняння має логічний тип. Всі оператори порівняння повертають значення логічного типу:

true – означає “так”, “правильно” або “істина”.

false – означає “ні”, “неправильно” або “хибність”.

Наприклад:

```
alert( 2 > 1 );  // true (правильно)
alert( 2 == 1 ); // false (неправильно)
alert( 2 != 1 ); // true (правильно)
```

Результат порівняння можна присвоїти змінній, як і будь-яке інше значення:

```
let result = 5 > 4; // присвоїти результат порівняння змінній result
alert( result ); // true
```

Порівняння рядків

Щоб визначити, чи один рядок більший за інший, JavaScript використовує так званий “алфавітний” або “лексикографічний” порядок. Інакше кажучи, рядки порівнюються посимвольно.

Наприклад:

```
alert( 'Я' > 'А' ); // true
alert( 'Соки' > 'Сода' ); // true
alert( 'Комар' > 'Кома' ); // true
```

Алгоритм порівняння рядків досить простий: порівнюються перші символи обох рядків.

Якщо перший символ першого рядка більший (менший) за перший символ другого рядка, то перший рядок більший (менший) за другий. Порівняння закінчено.

В іншому випадку, якщо перші символи обох рядків рівні, то таким самим чином порівнюються вже другі символи рядків.

Порівняння продовжується до того часу, доки не закінчиться один з рядків.

Якщо два рядки закінчуються одночасно, то вони рівні. Інакше, довший рядок вважатиметься більшим.

В прикладах вище, порівняння 'Я' > 'А' завершиться на першому кроці.

Проте друге порівняння слів 'Соки' і 'Сода' буде виконуватися посимвольно:

С дорівнює С.

о дорівнює о.

к більше ніж д.

На цьому кроці порівняння закінчується. Перший рядок більший.

Використовується кодування Unicode, а не справжній алфавіт. Такий алгоритм порівняння схожий на алгоритм сортування, який використовується в словниках і телефонних довідниках, проте вони не зовсім однакові.

Наприклад, в JavaScript має значення регістр символів. Велика буква "A" не рівна маленькій "a". Але яка з них більше? Маленька буква "a". Чому? Тому що маленькі букви мають більший код у внутрішній таблиці кодування, яку використовує JavaScript (Unicode).

Порівняння різних типів

Коли порівнюються значення різних типів, JavaScript конвертує ці значення в числа. Наприклад:

```
alert( '2' > 1 ); // true, рядок '2' стає числом 2
alert( '01' == 1 ); // true, рядок '01' стає числом 1
```

Логічне значення true стає 1, а false — 0. Наприклад:

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

Дивний наслідок. Можлива наступна ситуація:

Два значення рівні. Одне з них true як логічне значення, а інше — false.

Наприклад:

```
let a = 0;
alert( Boolean(a) ); // false
let b = "0";
```

```
alert( Boolean(b) ); // true
alert(a == b); // true!
```

З погляду JavaScript, результат очікуваний. Порівняння перетворює значення на числа (тому "0" стає 0), тоді як явне перетворення за допомогою Boolean використовує інший набір правил.

Строге порівняння

Використання звичайного оператора порівняння == може викликати проблеми. Наприклад, він не відрізняє 0 від false:

```
alert( 0 == false ); // true
```

Така ж проблема станеться з пустим рядком:

```
alert( '' == false ); // true
```

Це відбувається тому, що операнди різних типів перетворюються оператором порівняння == на числа. Пустий рядок, так само як false, стане нулем.

Як тоді відрізнити 0 від false?

Оператор строгої рівності === перевіряє рівність без перетворення типів.

Іншими словами, якщо a і b мають різні типи, то перевірка a === b негайно поверне результат false без спроби їхнього перетворення.

Перевіримо:

```
alert( 0 === false ); // false, тому що порівнюються різні типи
```

Є також оператор строгої нерівності !==, аналогічний до !=.

Оператор строгої рівності довше писати, проте він робить код більш зрозумілим і залишає менше місця для помилок.

Порівняння з null і undefined

Поведінка null і undefined під час порівняння з іншими значеннями — особливе:

при строгому порівнянні === ці значення різні, тому що різні їхні типи.

```
alert( null === undefined ); // false
```

При не строгому порівнянні == ці значення рівні. Водночас ці значення не рівні значенням інших типів. Це спеціальне правило мови.

alert(null == undefined); // true

Під час використання математичних операторів та інших операторів порівняння < > <= >= значення null/undefined конвертуються в числа: null стає 0, тоді як undefined стає NaN.

Тепер глянемо, які кумедні речі трапляються, коли ми застосовуємо ці правила. І, що більш важливо, як уникнути помилок під час їхнього використання.

Дивний результат порівняння null і 0:

alert(null > 0); // (1) false

alert(null == 0); // (2) false

alert(null >= 0); // (3) true

З погляду математики це дивно. Результат останнього порівняння показує, що “null більше або дорівнює нулю”, в такому випадку результат одного з порівнянь вище повинен бути true, але вони обидва false.

Причина в тому, що нестроге порівняння == і порівняння > < >= <= працюють по-різному. Останні оператори конвертують null в число, розглядаючи його як 0. Ось чому вираз (3) null >= 0 дає true, а вираз (1) null > 0 — false.

З іншого боку, для нестрогого порівняння == значень undefined і null діє окреме правило: ці значення не перетворюються на інші типи, вони рівні один одному і не рівні будь-чому іншому. Ось чому вираз (2) null == 0 повертає результат false.

Не порівнюйте значення undefined. Значення undefined не має порівнюватись з іншими значеннями:

alert(undefined > 0); // false (1)

alert(undefined < 0); // false (2)

alert(undefined == 0); // false (3)

Чому тоді порівняння undefined з нулем завжди повертає false? Ми отримуємо такі результати, тому що: порівняння (1) і (2) повертає false, тому що undefined під час порівняння з “не null” значеннями завжди конвертується в NaN, а NaN — це спеціальне числове значення, яке завжди повертає false під час будь-яких порівнянь.

Нестроге порівняння (3) повертає false, тому що undefined рівне тільки null, undefined і жодним іншим значенням.

Як уникати проблем

Можна уникнути проблем, якщо дотримуватися надійних правил:

Будьте пильні під час порівняння будь-якого значення з undefined/null, за винятком строгого порівняння ===.

Не використовуйте порівняння >= > < <= зі змінними, які можуть приймати значення null/undefined, хіба що ви цілком впевнені в тому, що робите. Якщо змінна може приймати ці значення, то додайте для них окремі перевірки.

Підсумки:

- Оператори порівняння повертають значення логічного типу.
- Рядки порівнюються посимвольно в лексикографічному порядку.
- Значення різних типів під час порівняння конвертуються в числа.
- Винятками є порівняння за допомогою операторів строгої рівності/нерівності.
- Значення null і undefined рівні == один одному і не рівні будь-якому іншому значенню.
- Будьте обережні, використовуючи оператори порівняння на зразок > чи < зі змінними, які можуть приймати значення null/undefined. Хорошою ідеєю буде зробити окрему перевірку на null/undefined для таких значень.

Умовний оператор

(a>b)?a : b

Якщо a>b, повернути a, інакше повернути b.

Умовні вирази

Структура умовного оператора в JavaScript має наступний вигляд:

if(<логічний_вираз>) <оператор_1> else <оператор_2>

У даному випадку <логічний_вираз> представляє собою повний або скорочений варіант певного виразу. При умові коли <логічний_вираз> приймає значення true виконується <оператор_1>, у іншому разі, тобто коли його значення дорівнює false <оператор_2>. При побудові логічних виразів здебільшого застосовуються операції порівняння.

Слід зазначити, що у більшості випадках для скорочення програмного коду та підвищення рівня його читабельності використовуються наступні спрощення при записі логічних виразів:

$a == 0$	$!a$
$a != 0$	a
$str == ""$	$!str$
$str != ""$	str

Структурний елемент else умовного оператору if у більшості випадках використовується для оптимізації програмного коду та удосконалення його читабельності. Слід зазначити, що у випадках коли варіанти перебору умовного оператору if є взаємно виключаючими, використання оператору else дозволяє скоротити об'єм перевірок і як результат пришвидшити роботу програми. У випадку коли варіанти перебору не доповнюють один одного використання структурного елементу else може спровокувати виникнення помилки. Нижче наведені два фрагменти коду, що демонструють не тільки оптимізаційні властивості умовних операторів, але й варіанти їх використання:

Взаємовиключаючі умови	Взаємодоповнюючі умови

if(a==1) b=a;
else if(a==2) b=a*a;
else if(a==3) b=a*a*a;
else if(a==4) b=1/a;
else if(a==5) b=1/a*a;
	if(str.length) flag=1;
	if(((str[2]==”a”)||(str[2]==”k”))&&(str[0]=” ”)) flag=0;
	if(str.length>=5) flag=str.length
	if(str==”twenty”) {number= 20; flag=0;}

switch (expression){

case label :

```
statement;
break;
case label :
statement;
break;
...
default : statement;
}
```

Цикли

```
for (let i=0; i < 20; i++) {
…
}

do {
i+=1;
document.write(i);
}

while (i<5);
 n = 0
x = 0
while( n < 3 ) {
n ++
x += n
}
for (let i in obj) { result += obj_name + "." + i + " = " + obj[i] + "<BR>" }
markLoop: while (theMark == true) { …… }
```

label - оператор з ідентифікатором, що дозволяє звернутися до нього в програмі.

break використовується для переривання циклу, або операторів switch або label.

1. break
2. break [label]

continue

continue - для рестарта циклу (while, do-while, for) або label.

```
checkiandj : while (i<4) {
checkj : while (j>4) {
…
if ((j%2)==0) continue check j;
…
}
}
```

Ітерація по всім властивостям об'єкта

```
for (let i in obj) { result += obj_name + "." + i + " = " + obj[i] + "<br>" }
```

Коментар

// Це однорядковий коментар.

/ * Це багаторядковий коментар. Він може бути будь-якого розміру, і Ви може помістити в нього що завгодно. * /

Події

Сценарії в документі HTML призначені, зокрема, для обробки подій: натискання мишкою на елементі документа, наведення стрілки миші на елемент чи забирання її з нього, натиснення клавіші тощо. Більшість дескрипторів HTML мають спеціальні атрибути, що визначають події, на які можуть реагувати відповідні елементи. Список допустимих подій наведений нижче; він досить великий і розрахований на всі випадки життя. Значенням такого атрибуту-події є рядок, що містить сценарій — код-обробник події.

Зазвичай обробники подій оформлюються у вигляді функцій, визначення яких розміщують у дескрипторі <script>. Розглянемо наступні приклади:

Приклад 1

```
<html>
<script>
function clickimage(){
alert ("Hello!")
}
</script>
<img src="pict.gif" onclick="clickimage()">
</html>
```

Приклад 2

```
<html>
<img src="morgan.jpg" onclick="alert('Hello!')" >
</html>
```

Є ще один метод, за допомогою якого можна визначати обробники подій. Майже для всіх дескрипторів HTML можна вказати атрибут ID — ідентифікатор. Його значенням є будь-який рядок, який відіграє роль індивідуального імені елемента в об’єктній моделі документа (англ. Document Object Model, DOM). З використанням цього атрибута для задання обробника події можна не використовувати атрибути-події; замість цього досить в контейнері <script> написати визначення функції-обробника події, ім'я якої створюється за шаблоном значення_ID.подія(). Розглянемо наступний фрагмент коду:

Приклад 3

```
<html>
<h1 id= "Myheader">Привіт усім!</h1>
<script>
function Myheader.onclick(){
alert("Привіт")
```

```
}
</script>
</html>
```

Браузер, зустрічаючи в HTML-документі дескриптор з визначеним ідентифікатором id, створює в об’єктній моделі цього документа об’єкт з тим же іменем. Для цього об’єкта існує метод обробки події. Назва метода співпадає з назвою події, однак синтаксис використання метода вимагає, щоб його ім'я було написане в нижньому регістрі. Елемент документа повинен бути завантажений раніше, ніж функція-обробник події. В таблиці 2.1 наведено основні події для документів HTML.

Основні події для документів HTML

Подія	**До яких елементів застосовна**	**Коли відбувається**	**Обробник події**
abort	Зображення	Користувач перериває завантаження, наприклад, натисканням на нове посилання чи кнопку зупинки	onabort
blur	Вікна, фрейми та всі елементи форм	Користувач переносить фокус введення з вікна, фрейму чи елемента форми	onblur
click	Всі кнопки, перемикачі, прапорці та посилання	Натискання мишею на елементі	onclick
change	Текстові поля та області, списки	Користувач змінює значення елемента	onchange
error	Зображення, вікна	Завантаження документа чи зображення призвело до помилки	onerror

focus	Вікна, фрейми та всі елементи форм	Користувач переносить фокус введення у вікно, фрейм чи елемент форми.	onfocus
load	Тіло документа	Користувач завантажує сторінку у браузер	onload
mouseout	Області, посилання	Користувач пересуває стрілку миші за межі області чи посилання	onmouseout
mouseover	Посилання	Користувач поміщає стрілку миші на посилання	onmouseover
reset	Форми	Користувач повертає форму у вихідний стан (натисканням кнопки Reset)	onreset
select	Текстові поля та області	Користувач виділяє поле вводу елемента форми	onselect
submit	Кнопки Submit	Користувач передає форму на сервер	onsubmit
unload	Тіло документа	Користувач закриває сторінку	onunload
keyup	Текстові поля та області	Користувач відпускає нажату клавішу	onkeyup

Тема 2. Робота з масивами в JavaScript

Масиви - це тип даних, який може містити багато значень. Іншими словами, змінна типу масив - це змінна, яка містить в собі не одне значення (як було раніше), а багато значень одночасно.

Створення

Створення порожнього масиву та задання значень перших двох елементів:

```
let myArray = new Array ()
myArray [0] = 'Київ'
myArray [1] = 'Івано-Франківськ'
```

Створення нового масиву і резервування місця для 100 змінних:

```
let myArray = new Array (100)
```

Створення нового масиву і явна ініціалізація першого елемента:

```
let myArray = new Array ('Елемент');
```

Непряме створення масиву з двох елементів:

```
let myArray = ['перший елемент', 'другий елемент']
```

Всі масиви, незалежно від способу створення, являють собою екземпляри класу (об'єкта) Array. Додавання елементів проводиться простою ініціалізацією відповідного елемента:

```
let myArray = new Array ()
myArray [0] = 'Івано-Франківськ'
```

Масиви в JavaScript не обов'язково повинні містити всі елементи. При необхідності можна створювати так звані "розріджені" масиви:

```
let myArray = new Array ();
myArray [0] = 'Київ'
// пропустимо myArray [1]
myArray [2] = 'Львів'
// пропустимо myArray [3]
```

```
myArray [4] = 'Луцьк'
myArray [5] = 'Суми'
for (i = 0; i <6; i + +)
alert (myArray [i])
```

Цей код створює масив і заповнює тільки необхідні елементи. Тепер якщо ми спробуємо отримати значення не ініціалізованих елемента (у прикладі це перший і третій), то отримаємо "undefined". Використання розріджених масивів іноді дуже зручно, але вимагає досвіду та уважності, тому на початковому етапі краще від них відмовитися. Крім того, розріджені масиви не дають вигоди з використання пам'яті, тому що місце резервується для всіх елементів, у тому числі не ініціалізованих. Елементами масиву можуть бути змінні будь-якого типу. Цікава особливість JavaScript - масив може одночасно містити елементи різних типів, у тому числі масиви/

Масив з елементами різних типів:

```
let myArrayS = new Array ();
myArrayS [0] = 'Дубно';
myArrayS [1] = 10000;
myArrayS [2] = 'Івано-Франківськ';
myArrayS [3] = 5000;
for (i = 0; i <myArrayS.length; i+=2)
alert ('Місто:' + myArrayS [i] + '\ n' +
        'Населення:' + myArrayS [i+1] + '\ n')
```

Масив з елементами-масивами:

```
let myArrayA = new Array ()
myArrayA [0] = new Array (' Дубно ', 10000);
myArrayA [1] = new Array (' Івано-Франківськ', 5000);
for (i = 0; i <myArrayA.length; i++)
alert ('Місто:' + myArrayA [i] [0] + '\ n' +
        'Населення:' + myArrayA [i] [1] + '\ n')
```

Клас Array містить властивість length, яка дозволяє дізнатися поточну довжину масиву:

```
let myArray = new Array ();
...
// Дізнатися кількість елементів
alert (myArray.length)
// Додати елемент останнім
myArray [myArray.length] = 'останнє значення'
```

Зверніть увагу, що значення Array.length на одиницю більше номера останнього елемента масиву, тому що нумерація в масиві починається з **нуля**, а властивість length показує загальну кількість елементів.

Методи

push

Синтаксис:

```
array.push (elem1, elem2, ...)
```

Аргументи:

elem1, elem2, …

Ці елементи будуть додані в кінець масиву

Опис, приклади:

Метод push корисний для додавання значень у масив.

Він додає елементи, починаючи з поточної довжини length і повертає нову, збільшену довжину масиву.

Приклад додавання двох елементів:

```
// array.length = 2
let array = [ "one", "two" ]
// додати елементи "three", "four"
let pushed = array.push("three", "four")
// тепер array = [ "one", "two", "three", "four" ]
// array.length = 4
// pushed = 4
```

---

shift

Синтаксис:

let elem = arrayObj.shift ()

Опис, приклади:

Видаляє елемент з індексом 0 і зміщує інші елементи на одну позицію. Повертає видалений елемент:

```
let arr = ["мій","маленький","масив"]
let my = arr.shift() // => "мій"
alert(arr[0]) // => "маленький"
// тепер arr = ["маленький", "массив"]
```

join

Синтаксис:

arrayObj.join ([glue])

Аргументи:

glue - рядковий аргумент, за допомогою якого будуть з'єднані в рядок всі елементи масиву. Якщо аргумент не заданий, елементи будуть сполучені комами.

Опис, приклади:

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

concat

Синтаксис:

let newArray = array.concat (value1, value2, ..., valueN)

Аргументи:

value1, value2, … - масиви або значення для приєднання

Опис, приклади:

Приклад об'єднання двох масивів:

```
let alpha = ["a", "b", "c"];
let numeric = [1, 2, 3];
```

let alphaNumeric = alpha.concat(numeric); // створює масив ["a", "b", "c", 1, 2, 3];

Приклад об'єднання трьох масивів:

let num1 = [1, 2, 3];

let num2 = [4, 5, 6];

let num3 = [7, 8, 9];

// створює масив [1, 2, 3, 4, 5, 6, 7, 8, 9];

let nums = num1.concat(num2, num3);

Приклад додавання значень у масив:

let alpha = ['a', 'b', 'c'];

let alphaNumeric = alpha.concat(1, [2, 3]); // створює масив ["a", "b", "c", 1, 2, 3]

pop

Синтаксис:

arrayObj.pop ()

Опис, приклади:

Цей метод змінює вихідний масив.

myFish = ["angel", "clown", "mandarin", "surgeon"];

popped = myFish.pop();

// тепер popped = "surgeon"

// a myFish = ["angel", "clown", "mandarin"]

unshift

Синтаксис:

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

Аргументи:

elem1, elem2, ..., elemN

Додаються в початок масиву елементи. Додані елементи збережуть порядок проходження.

Опис, приклади:

Даний метод додає в масив нього аргументи і повертає довжину отриманого масиву.

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

slice

Синтаксис:

arrayObj.slice (start [, end])

Аргументи:

start -індекс елемента в масиві, з якого буде починатися новий масив.

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

Якщо start негативний, то він буде трактуватися як arrayObj.length + start (тобто start'ний елемент з кінця масиву). Якщо end від'ємний, то він буде трактуватися як arrayObj.length + end (тобто end'ний 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:

Даний метод не змінює вихідний масив, а просто повертає його частину.

Приклад елементів з середини:

```
let 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 відліку з кінця:

```
arr = [1, 2, 3, 4, 5]
```

arr.slice(-2) // => [4, 5]

arr.slice(arr.length - 2) // теж саме

arr.slice(-3, -1) // [3, 4]

arr.slice(arr.length-3, arr.length-1) // теж саме

reverse

Синтаксис:

arrayObj.reverse ()

Опис, приклади:

Даний метод змінює вихідний масив. Після його застосування порядок елементів у масиві змінюється на зворотний.

Він повертає посилання на змінений масив.

arr = [1,2,3]

a = arr.reverse() // [3, 2, 1]

for(let i=0; i<a.length; i++) {

alert(a[i]) // 3, 2, 1

}

Зверніть увагу, ніякого нового масиву не створюється. Змінюється сам масив arr і повертається посилання на змінений масив.

arr = [1,2,3]

a = arr.reverse() // [3, 2, 1]

alert(a === arr) // true

sort

Синтаксис:

arrayObj.sort ([sortFunction])

Аргументи:

sortFunction - необов'язковий параметр - функція сортування. Якщо вказана функція, то елементи масиву будуть відсортовані згідно значень, що повертаються функцією. Умови на функцію можна записати таким чином:

function sortFunction(a, b){

if(a менше ніж b по деякому критерію)

```
    return -1 // або будь-яке число, меньше 0
  if(a більше ніж b по деякому критерію)
    return 1  // або будь-яке число, більше 0
  // у випадку a = b повернути 0
  return 0
 }
```

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

Опис, приклади:

Даний метод змінює вихідний масив. Одержаний масив також повертається в якості результату.

```
arr = [1,-1, 0]
a = arr.sort()
// => arr = [ -1, 0, 1 ]
alert(a === arr) // => true, це то й же посортований масив
```

splice

Синтаксис:

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

Аргументи:

start - індекс в масиві, з якого починати видалення.

deleteCount - кількість елементів, яке потрібно видалити, починаючи з індексу start.

elem1, elem2, ..., elemN - елементи, які додаються в масив. Додавання починається з позиції start.

Опис, приклади:

Мабуть, самий комплексний метод для роботи з масивом. Він об'єднує в собі дві різні функціональності: видаляє частину масиву і додає нові елементи на місце видалених. При цьому можна звести до нуля кількість видаляються елементів - 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"]
// arr = ["a", "d", "e"] (ті що залишились)
```

Приклад видалення одного елемента:

```
arr = [ "a", "b", "c", "d", "e" ]
// видалити з індексом 2 один елемент
arr.splice(2,1)
// arr = ["a", "b", "d", "e"]
```

Приклад додавання елементів:

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

Приклад видалення з кінця:

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

Тема 3. Метод console

Метод console.log() дозволяє вивести налагоджувальну інформацію не заважаючи користувачеві. Також об'єкт console має ще безліч інших не менш корисних методів.

Примітка: використання коду налагодження може негативно вплинути на продуктивність. Видаляйте його із продакшн версії (кінцевої версії продукту).

Розглянемо функціонал console.log детальніше.

Приклад можливості передачі будь-якого числа аргументів:

let foo = {baz: "tubular", goo: "rad"}, bar = "baz";

console.log("string",1,foo.goo,bar,foo.baz); // string 1 rad baz tubular

В результаті всі передані аргументи будуть виведені та розділені пробілом. Якщо Ви знайомі з функцією printf() в інших мовах, то console.log() теж може виводити дані подібним чином. Візьмемо останній приклад і передаємо перший аргумент у трохи зміненому вигляді.

let foo = {baz: "tubular", goo: "rad"}, bar = "baz";

console.log("%s theory is %d %s concept. I can only describe it as %s and %s", "string", 1, foo.goo, bar, foo.baz);

// string theory is 1 rad concept. I can only describe it as baz and tubular

%s, %d - це керуючі послідовності, які замінюються на відповідні їм значення (у порядку черговості). %s означає "трактувати значення як рядок", %d - як число.

Кожне входження такої послідовності буде замінено на наступний порядок. Перше входження - на перший (за першим аргументом-рядком, звичайно), ну і так далі.

Порядок використання аргументів можна змінити вручну, використавши символ $ і приписавши перед ним номер аргументу, який хочемо вивести:

let foo = {baz: "tubular", goo: "rad"}, bar = "baz";

console.log("%2$d theory is %d %s concept. I can only describe it as %s and %s", "string",1,foo.goo,bar,foo.baz);

// 1 theory is 0 baz concept. I can only describe it as tubular and %s string

Ця команда ніби каже: «Я почну з іншого аргументу і продовжу, починаючи з наступного». Як Ви бачите, послідовності, яким не вистачило аргументів, залишилися незаповненими.

Перший аргумент (рядок формату) також бере участь у нумерації (вона починається з нуля) аргументів. Таким чином, у нашому прикладі останній аргумент матиме номер 5. Але ми задали лише 5 аргументів і, при цьому, почали з іншого. Тому останній керуючій послідовності аргументів не дісталося, і вона не змінилася.

Щоб це виправити, можна змінити формат рядка таким чином, щоб «показник» на наступний елемент синхронізувався зі списком аргументів у певний момент. Тоді все працюватиме як належить.

let foo = {baz: "tubular", goo: "rad"}, bar = "baz";

console.log("%2$d theory is %1$s %3$s concept. I can only describe it as %s and %s", "string",1,foo.goo,bar,foo.baz);

// 1 theory is the string rad concept. I can only describe it as baz and tubular

Для того щоб аргументи були виведені правильно, нам потрібно змінити порядок виведення другого та третього елементів. Інші елементи і так у правильному положенні, так що немає потреби вказувати їх позиції. Аргументи будуть використані в наступному порядку: 2, 1, 3, 4, 5.

Форматування рядків – потужний інструмент, і тут охоплено лише його малу частину.

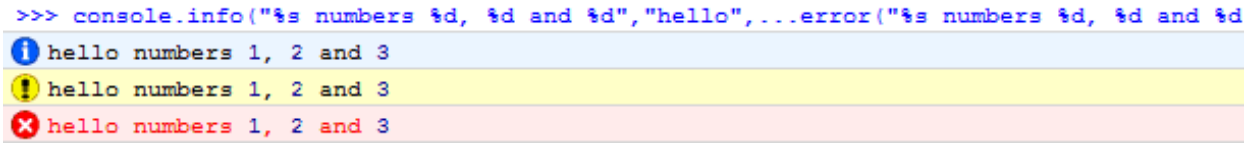
Різні типи повідомлення

Є ще пара методів, подібних до log, але відрізняються зовні. А саме: console.info(), console.warn() та console.error().

console.info("%s numbers %d, %d and %d","hello",1,2,3); // hello numbers 1, 2 and 3

console.warn("%s numbers %d, %d and %d","hello",1,2,3);

console.error("%s numbers %d, %d and %d","hello",1,2,3);



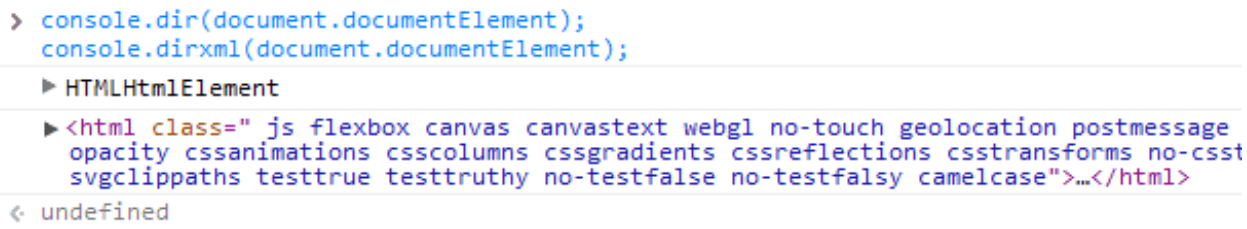
console.info(), console.warn() і console.error() у Firebug's.

Усі три методи вміють виводити рядки у відповідність до формату і приймати будь-яку кількість аргументів.

Логи DOM'а

Методи console.dir() або console.dirxml() можуть перерахувати властивості елемента або вивести HTML код елемента.

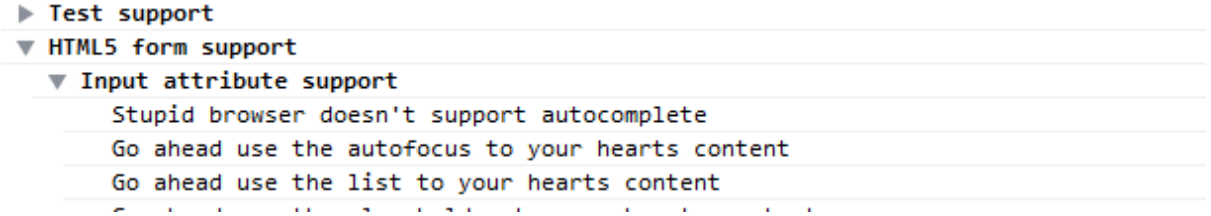
```
console.dir(document.documentElement);
console.dirxml(document.documentElement);
```



Групування

Іноді корисно згрупувати логи для спрощення роботи з ними. Для цього є методи console.group(), console.groupCollapsed() та console.groupEnd().

```
console.group("Overlord");
console.log("Overlord stuff");
console.group("Lord");
console.log("Overlord stuff");
console.group("Minion");
console.log("Minion stuff");
console.groupEnd();
console.groupCollapsed("Servant");
console.log("Servant stuff");
```



Як Ви можете побачити, виклики group, що йдуть поспіль, створюють вкладені папки. Для закриття папки використовуйте метод console.groupEnd(). Метод console.groupCollapsed() аналогічний console.group() за тим винятком, що група зі всім вмістом буде спочатку згорнута.

Профілювання та виміри

Також консоль дозволяє точно визначити годину, використовуючи метод console.time() та console.timeEnd(). Розташуйте виклик першого з них перед кодом, годину виконання якого хочете визначити, а іншого після.

```
console.time("Execution time took");
// Some code to execute
console.timeEnd("Execution time took");
```

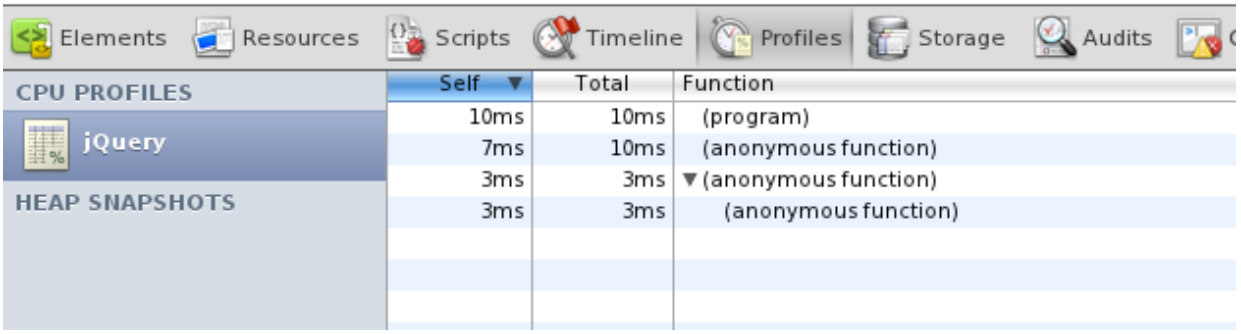
```
Testing webforms took: 9ms
Modernizr took: 286ms
```

Приклад роботи console.time() і console.timeEnd()

Таймери пов'язані між собою мітками (передаються першим аргументом і можуть бути будь-яким рядком), так що Ви можете запустити кілька таймерів одночасно. Коли спрацює console.timeEnd(), буде виведено повідомлення з міткою та годиною у мілісекундах.

Крім виміру часу можна профільтрувати Ваш код і вивести стек профілювання, який детально показує, де і скільки годин витратив браузер.

```
console.profile(); // Some code to execute
console.profileEnd();
```



Профілювання в Chrome.

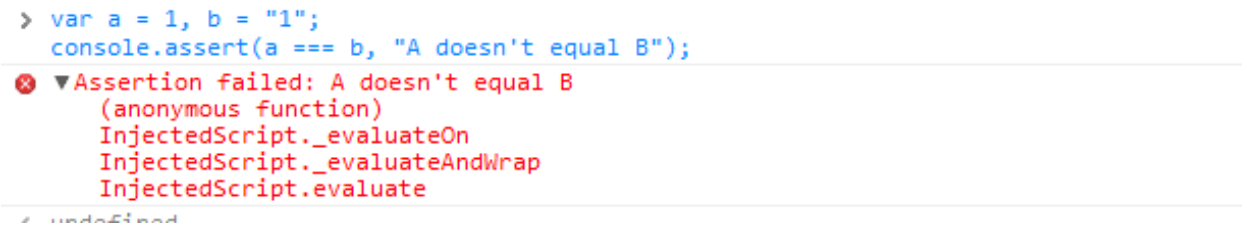
Assert'и

Коли ви працюєте над складним проектом, важливо покривати код юніт тестами. Це дозволить уникнути помилок та можливих регресій. Для цього консоль включає assert'и.

Assert'и дозволяють забезпечувати дотримання правил у коді та бути впевненим, що результати виконання цього коду відповідають очікуванням. Метод console.assert() дозволяє проводити елементарне тестування коду:

якщо щось піде не так, буде виведене повідомлення. Першим аргументом може бути все, що завгодно: функція, перевірка рівності чи перевірка існування об'єкта.

```
let a = 1, b = "1";
console.assert(a === b, "A doesn't equal B");
```



Assert у Chrome

Метод assert приймає умову, яка є обов'язковою до виконання (у даному випадку проста сувора перевірка на рівність) та іншим аргументом повідомлення, яке буде виведено в консоль, якщо перша умова не буде виконана.

Деякі особливості

log не вміє змінювати порядок аргументів для підстановок, але самі підстановки реалізовані.

При виклику методу trace у консоль буде виведено стек викликів (всі методи нічого не повертають, а просто пишуть у консоль). Працює це також як мінімум у Chrome та Opera.

У Opera і Chrome, крім перерахованих, реалізовані такі методи:

count - виводить скільки разів уже виконувався рядок, у якому розташований виклик методу. Аргументом передається рядок, який буде виведений перед кількістю викликів.

debug – нічим не відрізняється від log.

Тема 4. Об'єктна модель документа (DOM)

Об'єктна модель документа (англ. Document Object Model, DOM) — специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами. Визначається ця специфікація консорціумом W3C.

Разом із поширенням та розвитком вебтехнологій і вебпереглядачів почали з'являтись різні, часто несумісні інтерфейси роботи із HTML документами в інтерпретаторах JavaScript, вбудованих у вебпереглядачі. Це спонукало World Wide Web Consortium (W3C) узгодити та визначити низку стандартів, які отримали назву W3C Document Object Model (W3C DOM). Специфікації W3C не залежать від платформи або мови програмування.

W3C — це об'єднання розробників технологій та організацій, відповідальних за стандарти HTTP, HTML, XML, XSL, DOM та інші важливі стандарти Web та Internet, і більшість розробників браузерів реалізували її у останніх версіях своїх продуктів. Основна ідея полягає у використанні спільного інтерфейсу API, який розробники Web-сторінок можуть застосовувати для обробки вмісту документа HTML (або XML), а також ресурсів самого браузера.

При використанні об'єктної моделі документа програми та сценарії можуть динамічно отримувати доступ та оновлювати зміст документа, його структуру та стиль. Потім документ може оброблятись браузером, і результати цієї обробки можуть фіксуватись на відображуваній сторінці. За такої архітектури браузер відповідає як за відображення сторінки HTML, яка може бути змінена після її отримання з сервера, так і за виконання сценаріїв та компільованих програм в документі.

Назва моделі DOM пов'язана з тим, що документи розглядаються як об'єкти, що мають дані та поведінку. Взаємодія цих об'єктів представляє структуру документа.

Об'єкти JavaScript

Основою HTML-документа є теги. Відповідно до об'єктної моделі документа ("Document Object Model", DOM), кожен HTML-тег є об'єктом. Вкладені теги є "дітьми" (дочірніми елементами) батьківського елементу. Текст, який знаходиться всередині тегу, також є об'єктом.

Всі ці об’єкти доступні за допомогою JavaScript і ми можемо використовувати їх для зміни сторінки. Наприклад, document.body це об’єкт, що представляє тег <body>.

Запуск цього коду зробить <body> червоним протягом 3 секунд:

document.body.style.background = 'red'; // зробити фон червоним

setTimeout(() => document.body.style.background = '', 3000); // повернути назад

Тут ми використовували style.background, щоб змінити фоновий колір document.body, але також є багато інших властивостей, таких як:

innerHTML – вміст HTML елемента.

offsetWidth – ширина елемента (у пікселях).

…і так далі.

Об'єктна модель документа, головне джерело об’єктів для JavaScript, забезпечує об’єктний інтерфейс не лише для документів HTML, але й для броузера. Сценарій JavaScript може взаємодіяти з браузером для завантаження нової сторінки, перевіряти журнал браузера (список завантажених раніше Web-сторінок) або взаємодіяти з іншими сторінками у сусідніх фреймах.

Головним об'єктом при роботі з документами є document. Посилання на цей об'єкт можна отримати за допомогою атрибута цього об’єкта window, який є глобальним для будь-якої функції JavaScript. Для звернення до атрибутів в JavaScript використовується оператор „крапка” (.).

В об’єктній моделі документи згруповані у так звані колекції. Колекцію можна розглядати як проміжний об'єкт, що містить об’єкти власне документа. З іншого боку, колекція є масивом об’єктів, відсортованих у порядку слідування відповідних елементів в HTML-документі. Синтаксис звертання до елементів колекції такий же, як і для елементів масиву. Колекція має атрибут length — кількість всіх її документів. Колекція всіх елементів документа називається all; є також тематичні колекції: images, forms, links тощо (колекції усіх зображень, форм, посилань відповідно). Об'єкт може належати до якоїсь тематичної колекції, але обов’язково входить до колекції all. Загальні правила звертання до атрибутів всіх об’єктів такі:

document.колекція.id_об’єкта

document.колекція["id_об’єкта"]

document.колекція[індекс_об’єкта]

Об'єкти користувача JavaScript

Хоча JavaScript не є повноцінною об’єктно-орієнтованою мовою, вона забезпечує механізм для інкапсуляції. *Інкапсуляція* (encapsulation) — це можливість розміщення даних всередині об’єкта. У мові JavaScript можна створити екземпляр родового об’єкта (generic object) і призначити йому атрибути і навіть методи. Наприклад, у наступному фрагменті коду на JavaScript створюється екземпляр родового об’єкту з іменем cartLineItem (один з товарів у віртуальному кошику покупця). За допомогою оператора „крапка” (.) задаються значення чотирьох вказаних користувачем властивостей.

Спеціальні оператори:

new - створити новий об'єкт

delete – знищити об'єкти або властивість об'єкта

this – звернення до поточного об'єкта

typeof – повертає тип аргумента

void - «не повертати значення»

Приклад:

```
LineItem= new Object();
LineItem.productID = ‘MG1234’;
LineItem.productName = ‘MGB Roadster (1935)’;
LineItem.qty =1;
LineItem.unitPrice =36000;
```

Екземпляр LineItem можна використовувати в подальшому в будь-якому сценарії JavaScript з відповідним посиланням:

```
alert(‘В вашому кошику є ’ + cartLineItem.qty + ‘ ’ + LineItem.productName);
```

Для об’єктів можна також визначати методи. Наприклад, наступну функцію total() можна використовувати як метод. Вона звертається до свого об’єкту за допомогою оператора this:

```
function total(){
        return (this.qty * this.unitPrice);
            }
```

Доступ до функції здійснюється таким же чином як і до атрибутів.

LineItem.total = total;

Тепер цю функцію можна викликати безпосередньо для об’єкта LineItem:

LineItem= new Object();

LineItem.productID = ‘MG1234’;

LineItem.productName = ‘MGB Mk I Roadster’;

LineItem.qty = 2;

LineItem.unitPrice = 12500;

LineItem.total = total;

document.write(‘ ’ + LineItem.qty + ‘ ’ + LineItem.productName +‘коштує $’ + LineItem.total() + ‘ ’);

Прототип в JavaScript нагадує конструктор в C++. *Прототип* (prototype) — це функція, що створює визначений користувачем об'єкт та ініціалізує його атрибути. Прототип для об’єкта JavaScript повинен також містити функції об’єкта. Прототип об’єкта cartLineItem повинен мати такий вигляд:.

```
function cartLineItem(id, name, qty, price){
    this.productID = id;
    this.productName = name;
    this.qty = qty;
    this.unitPrice = price;
    this.total = total;
}
```

Якщо визначений прототип, то масив екземплярів LineItem можна створити за допомогою наступного фрагменту коду JavaScript:

```
let LineItem = new Array();
LineItem[0] = new cartLineItem (‘MG123’, ‘MGB Mk I Roadster’, 1, 36000);
LineItem[1] = new cartLineItem (‘AH736’, ‘Austin-Healey Sprite’, 1, 9560);
LineItem[2] = new cartLineItem (‘TS225’, ‘Triumph Spitfire Mk I’, 1, 11000);
```

Детальніше про DOM дерево тут

Посилання на об'єкт DOM найпростіше отримати за допомогою функції document.getElementById(), яка отримує посилання на об'єкт з допомогою значення його атрибуту id;

Наприклад, якщо в коді HTML ви маєте фрагмент

visible text

то відповідний об'єкт DOM можна отримати як

let mylink=document.getElementById('mylink');

Отримавши посилання на об'єкт DOM можна отримати значення атрибута:

let mylinkHref=mylink.href;

або встановити значення атрибута:

mylink.href="http://www.pnu.edu.ua/contacts";

Видимий текст всередині мітки встановлюється і читається з допомогою атрибута innerHTML:

let oldText=mylink.innerHTML;

mylink.innerHTML='new text';

Особливим атрибутом є **style**, який керує властивостями CSS. Наприклад, продовжуючи початий вище приклад, можна встановити колір шрифта і тла:

mylink.style.color='red';

mylink.style.backgroundcolor='yellow';

Приклад переведення температури за Фаренгейтом та за Цельсієм:

```
<html>
<body>
<table cellspacing="0px" cellpadding="3pt" style="">
<tbody><tr>
  <td> Температура за Фаренгейтом </td>
  <td><input type="text" id=f onkeyup="f2c()"></td>
</tr>
<tr>
  <td> Температура за Цельсієм </td>
```

```
  <td><input type="text" id=c onkeyup="c2f()"></td>
</tr>
</tbody></table>
<script>
function f2c(){
  let c=document.getElementById('c');
  let f=document.getElementById('f');
  c.value=(f.value-32)/1.8;
}
function c2f(){
  let c=document.getElementById('c');
  let f=document.getElementById('f');
  f.value=c.value*1.8+32;
}
</script>
</body>
</html>
```

Приклад коду для створення клавіатури:

```
let language = "ua";
const letters_ua = ["а", "б", "в", "г", "ґ", "д", "е", "є", "ж", "з", "и", "і", "ї", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ь", "ю", "я", " "];
const letters_en = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", " "]
 function createKeyboard(letters = letters_ua) {
        let keys = ""
        letters.map((key, index) => {
           if (index % 11 === 0) {
              keys += "<br>"
           }
        keys += `<button onclick="addSymbol('${key}')">${key}</button>`
```

```
        })
        screenKeyboard.innerHTML = keys;
    }
        function clean() {
        document.getElementById('display').value = "";
    }
    function addSymbol(key) {
        document.getElementById('display').value += key;
    }
…..
<body onload="createKeyboard()">
….
 <div id="screenKeyboard"></div>
```

Приклад розгортаючого меню:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <script>
      let toggleSubMenuValue = false;
         function init(){
         let str = "";
         str += "<a href='javascript:toggleSubMenu()'>Головна</a>";
         str += "<div id='subMain'></div>";
         document.getElementById("container").innerHTML=str;
         console.log(str);
      }
      function toggleSubMenu(){
         let str = ""
         if (!toggleSubMenuValue){
            str+="Про нас<br>";
```

```
            str+="Товари <br>";
            str+="Послуги <br>";
            str+="Контакти <br>";
            str+="Різне";
                    }
        toggleSubMenuValue = !toggleSubMenuValue;
        document.getElementById("subMain").innerHTML = str;
    }
</script>
</head>
<body onload="init()">
<div id="container"></div>
</body>
</html>
```

Тема 5. JavaScript. Регулярні вирази. Методи регулярних виразів та рядків.

Регулярні вирази

Регулярні вирази – потужний засіб пошуку та заміни тексту в рядках.

В JavaScript регулярні вирази реалізовані окремим об’єктом RegExp та інтегровані у методи рядків.

Регулярні вирази використовуються для операцій з рядками (пошук, заміна, порівняння)

Регулярний вираз складається з шаблону (також кажуть “патерн”) і необов’язкових прапорів.

Існує два синтаксиси для створення регулярного виразу.

“Довгий” синтаксис:

```
regexp = new RegExp("шаблон", "прапори");
```

Та “короткий” синтаксис, в якому використовуються слеши "/":

```
regexp = /шаблон/; // без прапорів
regexp = /шаблон/gmi; // з прапорами g, m та i (будуть описані далі)
```

Слеші /.../ говорять JavaScript про те, що це регулярний вираз. Вони відіграють таку саму роль, як і лапки для позначення рядків.

Регулярний вираз regexp в обох випадках є об’єктом вбудованого класу RegExp. Основна різниця між цими двома синтаксами полягає в тому, що слеші /.../ не допускають жодних вставок змінних (на зразок тих, що прописуються через ${...}). Вони повністю статичні.

Слеші використовуються, коли ми на момент написання коду точно знаємо, яким буде регулярний вираз – і це більшість ситуацій. А new RegExp – коли ми хочемо створити регулярний вираз “на льоту” з динамічно згенерованого рядка, наприклад:

```
let tag = prompt("Який тег ви хочете знайти?", "h2");
let regexp = new RegExp(`<${tag}>`); // те саме, що /<h2>/ при відповіді "h2" на запит вище
```

Прапори

Регулярні вирази можуть мати прапори, які впливають на пошук. У JavaScript їх всього шість:

i - з цим прапором пошук не залежить від регістру: немає різниці між A та a;

g - з цим прапором пошук шукає всі збіги, без нього – лише перше;

m - багаторядковий режим;

s - вмикає режим “dotall”, при якому крапка . може відповідати символу нового рядка \n.

u - вмикає повну підтримку Юнікоду.). Прапор дозволяє коректну обробку так званих “сурогатних пар” (докладніше про це у розділі Юнікод: прапорець "u" та клас \p{...})

y - режим пошуку на конкретній позиції в тексті (описаний у розділі Липкий прапорець "y", пошук на заданій позиції)

Алфавіт регулярних виразів

Символ	Пояснення
\	1. Якщо наступний символ – не літера і не цифра, відміняє спеціальне значення наступного символу. Наприклад символ "*" в шаблоні записується як "*"; символ "\" – як "\\" 2. Якщо наступний символ – літера або цифра, надає наступному символу спеціального значення
^	1. Символ початку рядка 2. Комбінація “[^” починає список заборонених символів
$	Символ кінця рядка
.	Будь-який символ (крім розриву рядка)
[	Початок множини символів
]	Кінець множини символів
|	Початок альтернативного шаблона
(	Початок підшаблона
)	Кінець підшаблона
?	1. Після дужки – модифікатор її значення

Символ	Пояснення
	2. 0 або 1 повтор попереднього підшаблона 3. Мінімізує «жадібність» регулярного виразу
*	0 або більше повторів попереднього підшаблона
+	1 або більше повторів попереднього підшаблона
{	Початок обмежувача кількості повторів
}	Кінець обмежувача кількості повторів
-	В квадратних дужках між двома символами позначає всі проміжні символи

Деякі спеціальні символи

Символ	Пояснення
\d	Будь-яка цифра
\D	Не цифра
\s	Пробіл
\S	Не пробіл
\w	Символ, який зустрічається в словах (літера, цифра або знак підкреслення)
\W	Символ, який не зустрічається в словах
\b	Границя слова
B	Не границя слова

Квадратні дужки

Шаблон	Підходять рядки	Не підходять рядки
[au]	“a”; “u”	В яких немає символів “a” та “u”
[^au]	В яких немає символів “a” та “u”	“a”; “u”
[а-г]	“а”,”б”,”в” ,”г”	В яких немає символів “а”,”б”,”в” ,”г”

Вертикальна риска

Шаблон	Підходять рядки	Не підходять рядки
f|g	“f”, “g”	В яких немає символів “f”, “g”
слово|назва	“слово”; “назва”	
((red|white) (king|queen))	"red king", " white king ", "red queen ", " white queen ",	

Підшаблони

Частину рядка, яка підходить до частини шаблона, взятої в дужки, зберігається окремо в результатах пошуку

Шаблон	Рядок	Результат пошуку містить
((red|white) (king|queen))	"the red king"	“red”, “king”, “red king”

Якщо підшаблон починається комбінацією «(?:», він не зберігається в результаті пошуку

Кількість повторів

Обмежувач кількості повторів	Приклад шаблона	Підходять рядки
* - 0 і більше	ф*	«», «ф», «фф», … «ффффффффф», …
+ - 1 і більше	ф+	«ф», «фф», … «ффффффффф» , …
? - 0 або 1	ф?	«», «ф»
{2,} - 2 і більше	ф{2,}	«фф», … «ффффффффф» , …
{2,4} - від 2 до 4	ф{2,4}	«фф», «ффф», «фффф»

Методи регулярних виразів та рядків

Використання exec

Метод об'єкта RegExp, який виконує пошук в рядку. Повертає масив інформації.

```
<SCRIPT  LANGUAGE="JavaScript">
let reg=/(\d+).(\d+).(\d+)/
```

```
let arr=reg.exec("Я народився 15.09.1980")
document.write("Дата народження: " +arr[0]+ "<br>")
document.write("День народження: " +arr[1]+ "<br>")
document.write("Місяць народження: " +arr[2]+ "<br>")
document.write("Рік народження: " +arr[3]+ "<br>")
</SCRIPT>
```

Результат:

Дата народження: 15.09.1980
День народження: 15
Місяць народження: 09
Рік народження: 1980

test

Метод об'єкта RegExp, який перевіряє, чи підходить рядок до шаблона. Повертає true або false.

```
<script language="JavaScript">
        let str="JavaScript"
        let reg=/PHP/
        let result=reg.test(str)
        document.write(result)
 </script>
```

Виведе false. Результат можна перетворити в будь-який інший рядок:

```
<script language="JavaScript">
        let str="JavaScript"
        let reg=/PHP/
        let result=reg.test(str) ? "Рядок співпав" : "Рядок не співпав"
        document.write(result)
 </script>
```

match

Метод об'єкта String, який перевіряє, чи підходить рядок до шаблона. Повертає масив інформації, або null.

Приклад перевірки коректності введеної електронної адреси і виводу її складових на екран:

```
<script>
let s="dwbyu@mail.ua";
let r=/^([a-z0-9]+)(\.[a-z0-9]+)*@([a-z0-9]+)(\.[a-z0-9]+)+$/i;
let a=s.match(r);
for (i=0;i<a.length;i++){
document.writeln("<hr> a["+i+"] "+a[i]);
}
</script>
```

Результат:

```
a[0]= dwbyu@mail.ua
a[1]= dwbyu
a[2]= undefined
a[3]= mail
a[4]= .ua
```

Приклад виводу слів рядка у зворотному порядку:

```
let re = /(\w+)\s|(\w+)$/ig;
let str = "John Smith  men";
let newstr=str.match(re)
for (i=newstr.length-1;i>-1;i--){
document.writeln(newstr[i]);
}
```

Результат: men Smith John

search

Метод об'єкта String, перевіряє, чи підходить рядок до шаблона. Повертає індекс знайденої частини або -1, якщо пошук завершився невдачею.

```
<script type="text/javascript">
    let str="Visit W3Schools!";
    document.write(str.search(/w3schools/i));
</script>
```

Результат: 6

replace

Метод об'єкта String, який виконує пошук та заміну.

Приклад заміни:

```
str = "тест і ще раз тест"
str.replace("тест","пройшов") // = "пройшов і ще раз тест"
```

Приклад глобальної заміни рядка:

```
str = "тест і ще раз тест"
str.replace(/тест/g," пройшов ") // = "пройшов і ще раз пройшов"
// або так
str.replace(new RegExp("тест",'g'),"пройшов")
```

Приклад заміни із ссилками:

```
let re = /(\w+)\s(\w+)/;
let str = "John Smith";
let newstr = str.replace(re, "$2, $1") // "Smith, John"
```

split

Метод об'єкта String, який розділяє рядок на частини, вважаючи шаблон границею частини.

```
str="1 2 3 4";
```

```
arr = str.split(' ')
document.writeln("<p>arr=" +arr+"<p>");
for (i=0;i<arr.length;i++){
document.writeln("<p>arr["+i+"]= "+arr[i]);
}
```

Результат:

arr=1,2,3,4

arr[0]= 1

arr[1]= 2

arr[2]= 3

arr[3]= 4

Тема 6. JavaScript: Робота з об'єктами. Вбудований об’єкт Date.

В JavaScript дозволено використовувати деякі заздалегідь визначені об’єкти. Прикладами таких об’єктів можуть бути Date, Array або Math. Об’єкт Date дозволяє працювати як з поточним часом, так і з датою. Це необхідно, наприклад, для того щоб відобразити на поточному html-документі час та дату. Для створення об’єкта потрібно використати оператор new:

today = new Date()

У даному випадку створюється новий об’єкт Date, з іменем today. Якщо при створенні даного об’єкта ви не вказали час та дату, то вона буде встановлена у значення поточної. Після виконання команди today = new Date() об’єкт today буде містити дату та час у який дана команда була виконана. Об'єкт Date() має декілька методів, що можуть застосовуватись для об’єкту today. Наприклад це методи – getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth() тощо.

new Date(milliseconds)

Створює об’єкт Date з часом, що дорівнює кількості мілісекунд (1/1000 секунди), що минули після 1 січня 1970 UTC+0.

```
let Jan01_1970 = new Date(0); // 0 означає 01.01.1970 UTC+0
alert( Jan01_1970 );
// тепер додамо 24 години, отримаємо 02.01.1970 UTC+0
let Jan02_1970 = new Date(24 * 3600 * 1000);
alert( Jan02_1970 );
```

Ціле число, яке являє собою кількість мілісекунд, що пройшли з початку 1970 року, називається міткою часу (timestamp).

Це числове представлення дати. Можна створити дату з timestamp за допомогою new Date(timestamp) і перетворити об’єкт Date, що існує, до timestamp за допомогою методу date.getTime().

Дати до 01.01.1970 р. мають негативний timestamp, наприклад:

```
let Dec31_1969 = new Date(-24 * 3600 * 1000); // 31 грудня 1969 року
alert( Dec31_1969 );
```

new Date(datestring)

Якщо є єдиний аргумент, і це рядок, то він автоматично аналізується.

```
let date = new Date("2017-01-26");
alert(date);
// Час не встановлений, тому припускається, що це буде північ за GMT і
// регулюється відповідно до того часового поясу, де запускається код
// Тому результат може бути
// Thu Jan 26 2017 11:00:00 GMT+1100 (Australian Eastern Daylight Time)
// або Wed Jan 25 2017 16:00:00 GMT-0800 (Pacific Standard Time)
```

Щоб зафіксувати будь-яку іншу дату та час ми можемо скористатись зміненим конструктором:

```
today = new Date(1997, 0, 1, 17, 35, 23)
```

При цьому буде створений об’єкт, у якому буде зафіксоване перше січня 1997 року 17:35 і 23 секунди. Тобто ви встановлюєте дату та час за наступним шаблоном:

new Date(year, month, date, hours, minutes, seconds, ms)

Створює дату з заданими компонентами у місцевому часовому поясі. Тільки перші два аргументи обов’язкові.

year має містити 4 цифри. Для сумісності також допускаються 2 цифри, які вважаються 19xx, напр. 98 тут те саме, що й 1998, але наполегливо рекомендується завжди використовувати 4 цифри.

Рахунок місяців починається з 0 (січня), до 11 (грудня).

Параметр date це місяця, якщо він відсутній, то береться 1.

Якщо hours/minutes/seconds/ms відсутні, вони вважаються рівними 0.

Наприклад:

```
new Date(2011, 0, 1, 0, 0, 0, 0); // 1 січня 2011 року, 00:00:00
new Date(2011, 0, 1); // те ж саме
```

Максимальна точність становить 1 мс (1/1000 сек):

```
let date = new Date(2011, 0, 1, 2, 3, 4, 567);
alert( date ); // 1.01.2011, 02:03:04.567
```

Тема 7. Бібліотека jQuery

Введення в jQuery

jQuery - бібліотека JavaScript, що містить в собі готові функції мови JavaScript, всі операції jQuery виконуються з коду JavaScript.

Бібліотека jQuery проводить маніпуляції з html-елементами, керуючи їх поведінкою і використовуючи DOM для зміни структури веб-сторінки. При цьому вихідні файли HTML і CSS не змінюються, зміни вносяться лише в відображення сторінки для користувача.

Для вибору елементів використовуються селектори CSS. Вибір здійснюється за допомогою функції $(). При виклику функція $() повертає новий екземпляр об'єкта JQuery, який повертає нуль або більше елементів DOM і дозволяє взаємодіяти з ними різними способами.

Виконання різних сценаріїв можливе тільки після закінчення завантаження структури документа document, коли браузер перетворює html-код сторінки в дерево DOM. Управління процесом завантаження забезпечує конструкція JavaScript:

```
jQuery(document).ready(function() { ... });
```

Спочатку проводиться вкладення примірника document в функцію jQuery(), після застосовується метод ready(), якому передається функція function(){...}, що виконується після завантаження документа.

На практиці зазвичай використовується скорочена форма такого запису jQuery(function(){...});, Або $(function(){...}); .

Для зберігання інформації при роботі з бібліотекою jQuery використовуються змінні JavaScript. У змінних можуть зберігатися елементи. Імена змінних, які призначені для зберігання повернутих jQuery-об'єктів, починаються зі знака $, наприклад:

JavaScript:

```
$h = $(".list").parent().parent().detach();
```

Для зберігання декількох елементів використовуються масиви JavaScript:

JavaScript:

```
$k[3] = 15;
```

Знак $ є дозволеним символом в імені змінної на мові Javascript, але його не прийнято ставити на початку імені змінних, які не містять jQuery-об'єкт, щоб чітко відрізняти змінні, які містять jQuery-об'єкт.

jQuery використовує знак $ як псевдонім (скорочення) для ідентіфіктора jQuery-об'єкту.

Наприклад:

$tab_title_input - просто ім'я, яке включає знак долара,

$ ("#tab_title") - виклик функції jQuery.

// погано, але синтаксично все вірно:

var first = $('# first');

var second = $('# second');

var value = first. text ();

// добре - перед об'єктами, які управляються jQuery, ми ставимо символ $:

var $first = $('# first');

var $second = $('# second'),

var value = $first. text();

Правила роботи з бібліотекою jQuery

1. Як додати jQuery на веб-сторінку

Додати бібліотеку jQuery на свою веб-сторінку можна двома способами:

Використовувати версію файлу jQuery, розміщену на jQuery.com. Після переходу на сайт ресурсу вам буде потрібно всього лише скопіювати посилання на jQuery-файл і додати її на свою веб-сторінку між тегами <script> ... </ script>:

<script type="text/javascript" src="https://code.jquery.com/jquery-3.7.1.min.js"> </script>

Щоб завантажити останню версію бібліотеки jQuery з офіційного сайту необхідно перейти за адресою jQuery.com і вибрати необхідну версію бібліотеки. Для завантаження пропонується дві версії jQuery-файлу - мінімізований і не мінімізований. Розмір не мінімізованого файлу більший від мінімізованого, він містить коментарі, тому його краще використовувати з метою розробки і налагодження коду.

Мінімізована версія файлу за розміром менша, в ній видалені всі коментарі і непотрібні пропуски, що прискорює завантаження файлу браузером.

Для завантаження потрібно перейти за посиланням, і у вікні, натиснути правою кнопкою миші і вибрати «Зберегти як ...». Після цього помістити файл в потрібну папку (зазвичай для цього використовується папка «scripts») і додати його на вашу сторінку:

HTML:

<script type="text/javascript" src="jquery-3.7.1.min"></script>

2. Правила додавання jQuery на сторінку

- Розміщуйте посилання на jQuery-файл всередині тега <head>.
- Розміщуйте посилання на jQuery-файл після посилань на стилі CSS, так як часто бібліотека jQuery проводить маніпуляції зі стилями елементів веб-сторінки.
- Розміщуйте інші теги <script> ... </ script> тільки після посилання на файл jQuery, якщо даний скрипт використовує бібліотеку jQuery.

3. Як створити новий html-елемент

Створити порожній html-елемент, наприклад, блок, можна кількома способами:

1) за допомогою короткої форми запису $(" ")

2) за допомогою інструкції $(" ")

3) за допомогою інструкції $(" ")

Всі три способи робочі, але, тим не менш, рекомендується включати відкриваючі та закриваючі теги, щоб показати, що даний елемент може містити інші елементи.

При створенні нового елемента методу $() можна передати другий параметр у вигляді об'єкта JavaScript, що визначає додаткові атрибути елемента:

```
 $("<img/>",
{src: "images/flower.jpg",
title: "Rose_in_garden",
click: function() {...}
}).appendTo("body");
```

Таким чином створюється елемент <img> з заданими атрибутами і включається в дерево DOM.

Завантажити актуальну версію бібліотеки можна на офіційному сайті http://jquery.com/ безкоштовно.

Події і їх обробка

У документації до бібліотеки jQuery тема подій і їх обробки займає значне місце. Нище наведені «основні» (умовно звичайно) методи - ready, bind, one, trigger і triggerHandler, unbind, а у другій половині хелпери - так звані помічниками, використовуючи які, Ви зможете як викликати певні події, так і пов'язувати з ними деякі дії.

Page Load:

ready (fn)

Призначає функції, які будуть виконуватися щоразу, коли об'єктна модель документа (DOM) готова до використання. Це мабуть найважливіша функція, включена в розділ «Події», оскільки вона може поліпшити час реакції веб-додатків, і хороша альтернатива використанню події window.onload, оскільки при використанні цього методу, Ваші функції будуть викликані вже в той момент, коли об'єктна модель документа (DOM) готова до роботи. Тут не потрібно чекати, коли сторінка буде повністю завантажена.

Аргумент, що передається обробнику події, є посиланням на функцію jQuery. Можна використовувати jQuery або $ без ризику виникнення колізій у просторі імен. Однак необхідно переконатися в тому, що не має обробника події onload, в іншому випадку $ (document). Ready () може не виконатися.

На сторінці можна мати як завгодно багато $ (document).ready подій. У цьому випадку функції, пов'язані з ними, будуть виконуватися в порядку їх слідування.

У цьому прикладі ми покажемо повідомлення про те, що DOM завантажена і готова до роботи.

```
$(document).ready(function(){
$("p:first")
.text("DOM завантажена і готова до роботи!");
}); // ще раз те саме
```

```
jQuery(document).ready(function(){
  jQuery("p:last")
    .text("Бібліотека jQuery готова до роботи!");
});
```

Event Handling:

bind (type, [data], fn)

Пов'язує обробник з одним або більше подіями (наприклад click) для кожного елемента набору. Може також пов'язувати користувальницькі події. Можливі події: blur, focus, load, resize, scroll, unload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error. Обробник події приймає об'єкт подія, що робить можливим запобігання поведінки за замовчуванням. Щоб запобігти і поведінку за замовчуванням і «спливання» події (тобто передачу події від одного елемента іншому), обробник повинен повертати false.

У більшості випадків можна визначити обробники події як анонімні функції, як показано в наступному прикладі. Всякий раз, коли здійснений клік на параграфі, в alert виводиться текст цього параграфа.

```
$("p").bind("click", function(){
alert( $(this).text() );
});
```

У разі, якщо це неможливо, слід передавати додаткові дані в другому параметрі [data] (див. приклад).

```
function handler(event) {
alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler);
```

Щоб скасувати дію за замовчуванням і запобігти «спливання», повертаємо false:

```
  $("form").bind("submit", function() { return false; }
```

Щоб скасувати тільки дія за замовчуванням використовується метод preventDefault.

```
$("form").bind("submit", function(event){
event.preventDefault();
});
```

Для запобігання «спливання» події використовується метод stopPropagation.

```
$("form").bind("submit", function(event){
event.stopPropagation();
});
```

Також можна пов'язати деякі користувальницькі події.

Также можно связать некие пользовательские события.

```
$("p").bind("myCustomEvent", function(e,myName,myValue){
$(this).text(myName + ", hi there!");
$("span").stop().css("opacity", 1)
.text("myName = " + myName)
.fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
$("p").trigger("myCustomEvent", [ "John" ]);
});
```

У наступному прикладі демонструється обробка подій click і double-click на параграфі.

```
$("p").bind("click", function(e){
var str = "( " + e.pageX + ", " + e.pageY + " )";
$("span").text("Клікнуто! " + str);
});
$("p").bind("dblclick", function(){
$("span").text("Подвійний клік зроблений на "+this.tagName);
});
$("p").bind("mouseenter mouseleave", function(e){
$(this).toggleClass("over");
});
```

one (type, [data], fn)

Пов'язує обробник з одним або більше подіями, які будуть виконані тільки один раз, для кожного елемента в наборі.Оброблювач виповнюється тільки один раз для кожного елемента. В іншому діють ті ж правила, описані при застосуванні bind (). Обробник події приймає об'єкт подія, що робить можливим запобігання поведінки за замовчуванням.Щоб запобігти і поведінку за замовчуванням і «спливання» події (тобто передачу події від одного елемента іншому), обробник повинен повертати false.

trigger (type, [data])

Викликає подія (передане в type) для кожного елемента набору. Проте це також викличе виконання браузером дій за замовчуванням для цієї події. Наприклад, передаючи 'submit' у функцію trigger () Ви змусите браузер відправити форму. Дії браузера за замовчуванням можна запобігти, повертаючи false для однієї з функцій, пов'язаних з подією.

triggerHandler (type, [data])

Це особливий метод виклику всіх пов'язаних з елементом обробників подій БЕЗ виконання браузером дій за замовчуванням. Подія викликається тільки для першого елемента, включеного в набір. Якщо набір елементів порожній, цей метод поверне 'undefined'.

unbind ([type], [data])

Це метод - протилежність bind, він видаляє всі події пов'язані з елементом для кожного елемента набору. Можна також видалити користувальницькі події, зареєстровані в bind ().Якщо переданий аргумент type - видалені будуть всі події цього типу.

Хелпери.

Використовуючи хелпери, можна як викликати певні події, так і пов'язувати з ними деякі дії ...

Interaction Helpers:

hover (over, out)

Симулює hovering - відстеження входу покажчика миші в межі об'єкта і виходу за його межі. Всякий раз, коли покажчик миші виявляється поверх об'єкту, викликається функція, передана в якості першого аргументу. Аналогічно, коли покажчик миші виходить за межі об'єкта - викликається функція, яка є другим аргументом.

Event Helpers:

blur ()

Викликає подія blur для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією blur і також викличе дію браузера за замовчуванням для цієї події.Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією blur. Подія blur зазвичай викликається, коли елемент втрачає фокус через дію вказівного пристрою або клавіші Tab.

blur (fn)

Пов'язує функцію з подією blur для всіх елементів набору.

change ()

Викликає подія change для кожного елемента набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією change і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією change. Подія change зазвичай викликається, коли елемент управління втрачає фокус і його значення було змінено з моменту отримання ним фокусу.

change (fn)

Пов'язує функцію з подією change для всіх елементів набору.

click ()

Викликає подія click для кожного елемента набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією.

click (fn)

Пов'язує функцію з подією click для всіх елементів набору. Подія click настає, коли кнопка вказівного пристрою (наприклад миша) клікнути на елементі. Клік визначається як mousedown + mouseup в деякому місці екрана.

dblclick ()

Викликає подія dblclick для кожного елемента набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією dblclick і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією dblclick.

dblclick (fn)

Пов'язує функцію з подією dblclick для кожного елемента набору. Подія dblclick зазвичай викликається, коли кнопка вказівного пристрою двічі швидко клікнути на елементі.

error ()

Викликає подія error для кожного елемента набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією error і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією error. Подія error зазвичай викликається, коли елемент втрачає фокус через дію вказівного пристрою або клавіші Tab.

error (fn)

Пов'язує функцію з подією error для кожного елемента набору.
Для події error немає загальних стандартів. У більшості браузерів, подія error об'єкта window викликається, коли на сторінці виявлена помилка JavaScript. Для об'єкта image подія error викликається при некоректному вмісті атрибуту src - наприклад, неіснуючий файл.

Якщо подія відбулася в об'єкті window, обробник події передає три параметри:

- повідомлення, яке описує подію («varName is not defined», «missing operator in expression», і т.д.);
- повний URL документа, який містить помилку;
- номер рядка, в якій сталася помилка;

Якщо обробник події поверне true, це означає, що подія було оброблено і браузер не показує ніяких помилок.

focus ()

Викликає подія focus для кожного елемента набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією focus.

focus (fn)

Пов'язує функцію з подією focus для кожного елемента набору.Подія focus викликається, коли елемент отримує фокус через вказівник миші або клавішу Tab.

keydown ()

Викликає подія keydown для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією keydown і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією keydown.

keydown (fn)

Пов'язує функцію з подією keydown для кожного елемента набору. Подія keydown зазвичай викликається, коли на клавіатурі натиснута яка-небудь клавіша.

keypress ()

Викликає подія keypress для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією keypress і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох

функцій, пов'язаних з цією подією keypress. Подія keypress зазвичай викликається, коли на клавіатурі була натиснута яка-небудь клавіша.

keypress (fn)

Пов'язує функцію з подією keypress для кожного елемента набору. Подія keypress зазвичай викликається, коли на клавіатурі була натиснута яка-небудь клавіша. Подія keypress визначається як keydown + keyup на який-небудь клавіші.

keyup ()

Викликає подія keyup для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією keyup і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією keyup.

keyup (fn)

Пов'язує функцію з подією keyup для кожного елемента набору. Подія keyup зазвичай викликається, коли на клавіатурі була відпущена небудь клавіша.

load (fn)

Пов'язує функцію з подією load для кожного елемента набору.Будучи пов'язаним з елементом window, це подія викликається тільки тоді, коли агент користувача завершить завантаження всього вмісту документа (фрейми, об'єкти, графіком). Для елементів це викликається, коли цільової елемент і весь його вміст повністю завантажено.

Необхідно пам'ятати, що load буде працювати тільки в тому випадку, якщо він визначений перед елементом, повного завантаження якого ми очікуємо. Якщо ж його визначити після такого елемента - нічого не відбудеться.

mousedown (fn)

Пов'язує функцію з подією mousedown для кожного елемента набору. Подія mousedown зазвичай викликається, коли клавіша вказівного пристрою (миші) була натиснута на елементі.

mousemove (fn)

Пов'язує функцію з подією mousemove для кожного елемента набору. Подія mousemove зазвичай викликається, коли покажчик миші переміщається поверх елемента. Обробник події приймає один аргумент - об'єкт події, у властивостях clientX і clientY якого представлені координати покажчика миші.

mouseout (fn)

Пов'язує функцію з подією mouseout для кожного елемента набору. Подія mouseout зазвичай викликається, коли покажчик миші виходить за межі елемента.

mouseover (fn)

Пов'язує функцію з подією mouseover для кожного елемента набору. Подія mouseover зазвичай викликається, коли покажчик миші знаходиться в межах елемента.

mouseup (fn)

Пов'язує функцію з подією mouseup для кожного елемента набору. Подія mouseup зазвичай викликається, коли клавіша вказівного пристрою (миші) була відпущена на елементі.

resize (fn)

Пов'язує функцію з подією resize для кожного елемента набору. Подія resize зазвичай викликається, коли змінюються розміри області перегляду документа.

scroll (fn)

Пов'язує функцію з подією scroll для кожного елемента набору.Подія scroll зазвичай викликається, коли прокручується область перегляду документа.

select ()

Викликає подія select для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією select і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією select.
Не варто плутати подія «select» з подією «change», яке викликається, коли HTML-елемент select використовується для вибору який (їх)-або опцій.

select (fn)

Пов'язує функцію з подією select для кожного елемента набору. Подія select зазвичай викликається, коли користувач виділяє небудь текст у текстовому полі, наприклад input і textarea.

submit ()

Викликає подія submit для всіх елементів набору. У цьому випадку будуть викликані всі функції, пов'язані з цією подією submit і також викличе дію браузера за замовчуванням для цієї події. Поведінка браузера за замовчуванням можна попередити, повертаючи false з однієї або декількох функцій, пов'язаних з цією подією submit.

submit (fn)

Пов'язує функцію з подією submit для кожного елемента набору. Подія submit зазвичай викликається, коли форма відправляється.

unload (fn)

Пов'язує функцію з подією unload для кожного елемента набору.

Додаткові теоретичні відомості

Про jQuery

Лабораторна робота №1. Основи JavaScript. Використання функцій. Події. Оператори умови та циклу.

Мета: набути навичок з написання скриптів та використання функцій на мові JavaScript.

Тривалість: 4 год.

1.1 Теоретичні відомості

1. Тема 1. Основи JavaScript. Використання функцій. Події
2. Тема 2. Робота з масивами

Додаткові теоретичні відомості

1. Сучасний підручник з JavaScript

1.2 Завдання для виконання лабораторної роботи

Завдання 1

Розробити Web-сторінку, де поряд із звичайним html текстом буде міститись прізвище, ім’я, по батькові студента, надрукованого засобами Javascript. Розмістити на сторінці кнопку, при кліку на яку користувачеві на екран з’являтиметься повідомлення про дату та місце народження студента.

Приклад:

```
<html>
<body>
<h3> Завдання 1</H3>
<p> Це звичайний HTML абзац.</p>
<script language="JavaScript">
document.write("Привіт! Мене звати …., студент групи ….")
</script>
<p>Знову звичайний HTML абзац.</p>
<form>
<input type="button" value="Click me" onClick="alert('Я народився 13.10.2006 року в місті Івано-Франківськ ')">
</form>
```

</body>

</html>

Завдання 2

Доповнити Інтернет сторінку із попереднього завдання заголовком із номером завдання (для наступних завдань теж додавайте заголовок із їх номером) та однією кнопкою, при кліку на яку призначити виконання функції розрахунку арифметичної операції вибраної за номером студента в списку групи:

№ п/п	F	№ п/п	F
1	$F = x + 2y$	16	$F = \frac{1}{x + y + z}$
2	$F = 2x - 3y^2$	17	$F = \frac{x - y}{2xz}$
3	$F = \frac{1 - x^2}{y^3}$	18	$F = 21xy - xz + xyz$
4	$F = 2x + 3y - z$	19	$F = x^3 + x^2 + x + xy^2$
5	$F = 3xy - y + 2x$	20	$F = x + y^2 + z^{0.5}$
6	$F = \frac{x}{y} + \frac{y^2}{1 - xy}$	21	$F = x^{-0.5}$
7	$F = x - xy + y^2$	22	$F = x^6 + x^5 y + 0.5z + x^2$
8	$F = x + \sqrt{y}$	23	$F = \frac{xz}{y}$
9	$F = xy + xz - yz$	24	$F = \frac{\sqrt{x}}{y + z}$
10	$F = x^{y^z} + 2xz$	25	$F = \frac{x - y^{0.5}}{xz}$

11	$F = x^6 + x^5 y + 0.5z + x^2$	26	$F = -3xz + \sqrt{x^3}$
12	$F = \left(1 - \frac{x}{y}\right)^2 + z$	27	$F = x^{y^z} + 2xz$
13	$F = x^y + y^z + z^x$	28	$F = \frac{\sqrt{y}}{\sqrt[3]{xz}}$
14	$F = 1 - x - y - z$	29	$F = 2x + y - \frac{z}{xy}$
15	$F = x + y^2 + z^{0.5}$	30	$F = \frac{\sqrt{x}}{y + z}$

Приклад:

```
<html>
<head>
<script language="JavaScript">
function calculation() {
let x= 12;
let y= 5;
let result= x + y;
alert(result);
}
</script>
</head>
<body>
<form>
<input type="button" value="Calculate" onClick="calculation()">
</form>
</body>
</html>
```

Результати виконання:

Завдання 3

Доповнити Інтернет сторінку із попереднього завдання як мінімум двома текстовими полями форми. Обробити події згідно номеру варіанта, для цього призначити реакцію на відповідну події над першим елементом, саму реакцію задати у вигляді відображення текстових повідомлень у другому текстовому полі (див. приклад виконання після таблиці із варіантами).

№ п/п	Події	№ п/п	Події
1	onClick, onMouseOut	30	onChange, onFocus
2	onChange, onClick	16	onFocus, onMouseOut
3	onClick, onSelect	17	onBlur, onClick
4	onBlur, onMouseOver	18	onClick, onMouseOver
5	onFocus, onClick	19	onChange, onSelect
6	onSelect onMouseOut	20	onFocus, onChange
7	onBlur, onChange	21	onSelect onMouseOut
8	onChange, onMouseOver	22	onClick, onFocus
9	onFocus, onSelect	23	onChange, onSelect
10	onClick, onChange	24	onBlur, onSelect
11	onFocus, onSelect	25	onFocus, onMouseOver
12	onClick, onMouseOut	26	onChange, onFocus

13	onSelect, onMouseOver	27	onSelect, onMouseOut
14	onBlur, onSelect	28	onClick, onSelect
15	onChange, onMouseOut	29	onBlur, onMouseOver

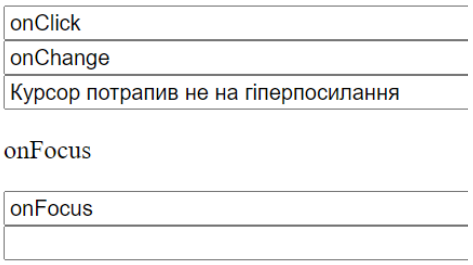
Приклад:

```
<html>
<head>
<script>
function fun(a)
{
main_form.t_res.value = a;
}
function fun1(a)
{
main_form.t_res2.value = a;
}
</script>
</head>
<body onLoad = "fun('Відбулось завантаження сторінки');">
<form id = "main_form">
<input type="text" size = "100" value="onClick" onClick="fun('Відбувся клік на текстовому елементі форми');"><br>
<input type="text" size = "100" value="onChange" onChange = "fun('Зміст текстового поля форми змінений');"><br>
<input type = "text" size = "100" id = "t_res" onMouseOver = "fun('Курсор потрапив на гіперпосилання');" onMouseOut = "fun('Курсор потрапив не на гіперпосилання');">
<p>onFocus </p>
<input type="text" size = "100" value="onFocus" onFocus = "fun1('Текстовий елемент форми отримав фокус');" onSelect = "fun('В текстовому полі форми виділений текст);"><br>
```

<input type = "text" size = "100" id = "t_res2";">
</form>

</body>

</html>



Результати виконання:

Завдання 4

Доповнити Інтернет сторінку із попереднього завдання реалізацією розгалуженого обчислювального процесу згідно варіанту завдання, передбачити введення значень змінних у відповідні текстові поля:

№ п/п	Варіант завдання	№ п/п	Варіант завдання
1	$F=\begin{cases} x+y, xy>0 \\ x, x^2>100 \\ x^3+y, інакше \end{cases}$	16	$F=\begin{cases} x+y, xy\neq 0 \\ x, x^3>100 \\ x^3+y, інакше \end{cases}$
2	$F=\begin{cases} x+y, (x>0) та (y>0) \\ x^2+y^2, (x<3) та (y<4) \\ x^3, інакше \end{cases}$	17	$F=\begin{cases} 2x+y, (x>0) та (y>0) \\ x^2+2y^2, (x<3) та (y>4) \\ x^3, інакше \end{cases}$
3	$F=\begin{cases} x^2+y^2, (x<10) або (y>3) \\ x-y+x^2, x>2 \\ x^3+3xy, інакше \end{cases}$	18	$F=\begin{cases} 2x^2-3y^2, (x<10) або (y\neq 3) \\ 2x-3y/x+x^2, x\neq 2 \\ x^3+3xy, інакше \end{cases}$

4	$F=\begin{cases} x/y+xy^2,(x>0)та(y>0) \\ y/x,(-x+y)/2>0 \\ x^2,інакше \end{cases}$	19	$F=\begin{cases} 2x/y+xy^2,(x>0)та(y\neq 0) \\ y/2x,(2x+y)/2\neq 0 \\ x^2,інакше \end{cases}$
5	$F=\begin{cases} x-y^2,x-y>2 \\ x/y,x>0 \\ x^3-y^2,інакше \end{cases}$	20	$F=\begin{cases} x-2y^2,x-3y\neq 2 \\ x-y/y,x>0 \\ 2x^2-3y,інакше \end{cases}$
6	$F=\begin{cases} x/2y,xy<-100 \\ x^2-xy,xy>20 \\ x^3+2,інакше \end{cases}$	21	$F=\begin{cases} 2x/y,xy\neq -100 \\ x^3-2xy,xy\neq 20 \\ x^2+4,інакше \end{cases}$
7	$F=\begin{cases} 2xy-3,x/y>0 \\ xy/3,(x>0)та(y<3) \\ 2x-3y,інакше \end{cases}$	22	$F=\begin{cases} xy-1,x/y>0 \\ 2xy/2,(x\neq 0)та(y\neq 3) \\ x-y,інакше \end{cases}$
8	$F=\begin{cases} 2x+3y,x<3y \\ xy+x/3y,(y^2>2xy)або(y>0) \\ x^2+y,інакше \end{cases}$	23	$F=\begin{cases} 10x+y,x\neq y \\ 2xy+4x/y,(y^2\neq 2xy)або(y<0 \\ x+y^2,інакше \end{cases}$
9	$F=\begin{cases} x^3-\sqrt{x},x^2>3 \\ xy,(x>0)та(y<0) \\ x+y+xy-x/y,інакше \end{cases}$	24	$F=\begin{cases} x^3,x^3\neq 3 \\ xy,(x\neq 0)та(y<0) \\ 2x+xy-x/2y,інакше \end{cases}$
10	$F=\begin{cases} 3xy-x,xy>0 \\ 2x-3y/x,(7x>2y)або(y>2) \\ x-y,інакше \end{cases}$	25	$F=\begin{cases} xy-x,2x/y\neq 0 \\ (x-2y)/x,(14x\neq y)або(2y< \\ x-y,інакше \end{cases}$

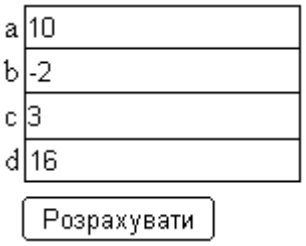
11	$F=\begin{cases}3xy, x>0\\ 2x/y, (x<0) та (y>3/2)\\ x^2+y^3, інакше\end{cases}$	26	$F=\begin{cases}x/y, x\neq 0\\ 2x/y, (x\neq 0) або (y>3/2)\\ -x^3+2y^2, інакше\end{cases}$
12	$F=\begin{cases}x-y, xy>0\\ x^2-y/x, (y<0) та (x>3y)\\ 3xy-x^2y^2, інакше\end{cases}$	27	$F=\begin{cases}2x-y, 2xy\neq 0\\ x^2+2y/x, (y\neq 0) або (x\neq 3y)\\ x/y-xy, інакше\end{cases}$
13	$F=\begin{cases}2\sqrt{x}\cdot y, (y>3) та (x>0)\\ x/y^2, (yx^2/2>3x) або (x^2>2y)\\ x^2/y, інакше\end{cases}$	28	$F=\begin{cases}(3/5)\sqrt{x}\cdot y, (y\neq 5) та (x\neq 0)\\ x/y^2, (yx^2\neq 3x) або (x^3\neq 5xy)\\ x/y^2, інакше\end{cases}$
14	$F=\begin{cases}-3x+y^3, (x>0) та (y<0)\\ x+y-x^2y, xy-x<3\\ x^2+y^2, інакше\end{cases}$	29	$F=\begin{cases}x+y^2, (x\neq 0) та (y\neq 0)\\ x-y+x^3y^2, xy-x>=3\\ x^2-y^2, інакше\end{cases}$
15	$F=\begin{cases}3xy-x, xy>0\\ 2x-3y/x, (7x>2y) або (y>2)\\ x-y, інакше\end{cases}$	30	$F=\begin{cases}2x-y, 2xy\neq 0\\ x^2+2y/x, (y\neq 0) або (x\neq 3y)\\ x/y-xy, інакше\end{cases}$

Приклад:

```
<html>
<head>
<script>
function fun()
{
let a, b, c, d;
a = parseInt(main_form.t_a.value);
```

```
b = parseInt(main_form.t_b.value);
c = parseInt(main_form.t_c.value);
if(a*b*c>0) d=a*b-c; else
if(a*b*c<0) d=a-b*c; else
d=a*c-Math.sqrt(b);
main_form.t_d.value = "" + d;
}
</script>
</head>
<body>
<form id = "main_form">
a<input type="text" id = "t_a" value="10"><br>
b<input type="text" id = "t_b" value="-2"><br>
c<input type="text" id = "t_c" value="3"><br>
d<input type="text" id = "t_d" value=""><br>
<input type = "button" onClick = "fun();" value = "Розрахувати">
</form>
</body>
</html>
```

Результати виконання:



Завдання 5

Доповнити Інтернет сторінку із попереднього завдання реалізацією розрахунку суми ряду заданого формулою згідно варіанту, передбачити введення значень змінних у відповідні текстові поля:

№ п/п	Варіант завдання	№ п/п	Варіант завдання
1	$\sum_{i=1}^{5}(-1)^i \frac{x+y-i!}{(i-1)!}$	15	$\sum_{i=2}^{6}(-1)^{i+1} \frac{2xy-x^2}{(i+1)!}$
2	$\sum_{i=3}^{5}(-1)^{i+1} \frac{x^2-y^3+3}{i!-5x}$	16	$\sum_{i=2}^{7}(-1)^i \frac{i!-xy}{(i+1)!}$
3	$\sum_{i=7}^{11}(-1)^i \frac{x+y-i!}{(i-1)!}$	17	$\sum_{i=3}^{5}(-1)^{i+1} \frac{2xy-y^2+x}{3x} i!$
4	$\sum_{i=5}^{8}(-1)^{i+1} \frac{x/y+x^2/3}{x^2y^2(i+1)!}$	18	$\sum_{i=6}^{12}(-1)^i \frac{-2x+y^2}{i!}$
5	$\sum_{i=1}^{5}(-1)^i \frac{2xy-i!}{1-x}$	19	$\sum_{i=3}^{7}(-1)^{i+1}\left(\frac{7i-xy}{i+12} i^2-(i+2)!\right)$
6	$\sum_{i=3}^{6}(-1)^{i+1} \frac{x-y}{i!}$	20	$\sum_{i=2}^{4}(-1)^{i+1} \frac{-x+xy^2}{2x} i!$
7	$\sum_{i=2}^{6}(-1)^i \frac{i+1}{i-2} i!$	21	$\sum_{i=2}^{7}(-1)^i \frac{i!-2xy}{(i+2)!}$
8	$\sum_{i=7}^{9}(-1)^{i+1} \frac{i!/x+y}{(i+1)!}$	22	$\sum_{i=1}^{5}(-1)^i \frac{3y/x+y^2}{(i+1)!}$
9	$\sum_{i=4}^{6}(-1)^i \frac{2x-y}{i!}$	23	$\sum_{i=2}^{7}(-1)^{i+1} \frac{0.85i+xy/(1+x^2)}{i!}$
10	$\sum_{i=1}^{12}(-1)^{i+1} \frac{i+3}{(x+y)!}$	24	$\sum_{i=3}^{5}(-1)^i \frac{3xy-x}{2y(i+x)!}$
11	$\sum_{i=3}^{5}(-1)^i \frac{2xy-x^2}{i+2} i!$	25	$\sum_{i=7}^{11}(-1)^{i+1} \frac{+-y}{2+x}(i+3)!$
12	$\sum_{i=2}^{4}(-1)^{i+1} \frac{-x+xy^2}{2x} i!$	26	$\sum_{i=5}^{8}(-1)^i \frac{12y/2x-x^2y^2(i+1)!}{i!}$

13	$\sum_{i=6}^{12}(-1)^i \frac{x^y - \sqrt{x}}{i!}$	27	$\sum_{i=1}^{5}(-1)^{i+1} \frac{3xy-5}{i!}$
14	$\sum_{i=2}^{5}(-1)^{i+1} \frac{x+2xy-y^2}{(i+1)!}$	28	$\sum_{i=7}^{10}(-1)^i \frac{y-x+xy+\sqrt{x}}{(i+y)!}$

Приклад:

```
<html>
<head>
<script>
function fun()
{
let a, b, sum=0.0, i=0.0, cur=0.0;
a = parseInt(main_form.t_a.value);
b = parseInt(main_form.t_b.value);
let to4nost = 0.001;
do
{
i++;
let factorial = 1;
for(let j=1;j<=i;j++)
factorial*=j;
cur = (2*a*i*i+i/b)/(factorial*factorial);
sum+=cur;
}while(cur>to4nost)
main_form.t_c.value = "" + sum;
}
</script>
</head>
<body>
<form id = "main_form">
a=<input type="text" id = "t_a" value="10"><br>
```

b=<input type="text" id = "t_b" value="-2">

Результат:<input type="text" id = "t_c" value="">

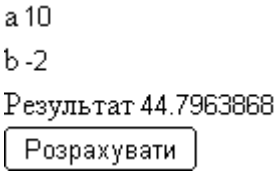
<input type = "button" onClick = "fun();" value = "Розрахувати">

</form>

</body>

</html>

Результати виконання:



Завдання 6

Створіть нову Web-сторінку, на якій користувач міг би ввести у текстове поле назву товару, а після кліку на кнопку «Наявна кількість і вартість» отримати дані про наявну на складі кількість і вартість вказаного товару ***(fun1).*** Зробіть так, щоб ця інформація виводилася на сторінці в окремих полях. Назви товарів мають починатися на першу літеру вашого прізвища або імені.

Створіть кнопку «Загальна сума», при кліку на яку безпосередньо на сторінку виводиться добуток кількості товару на його вартість із відповідним супроводжуючим текстом ***(fun2).***

Вказівки до виконання: створіть три масиви, які містять назви товарів, доступна кількість на складі, вартість за одиницю товару. Однаковий порядковий номер в масивах відповідає даним одного товару. Змінна, яка відповідає даному номеру, повинна бути видимою в обох функціях ***(fun1 і fun2)***.

Частина коду:

<HTML>

<HEAD>

<SCRIPT LANGUAGE='JavaScript'> <!-- дескриптор опису мови сценарію -->

let product = new Array(6); <!-- опис масиву product, виділення місця в пам'яті -->

product = ["Зошит", "Ручка"] <!-- задання значень елементів масиву -->

quantity = ["105", "200"]

cost = ["26", "10.5"]

function fun1() {

<!-- функція викликається при кліку на кнопку «Наявна кількість і вартість» і знаходить елементи, які співпадають з введеною назвою товару і виводить наявну кількість на складі і вартість за одиницю товару-->

…}

function fun2() {

<!-- функція викликається при кліку на кнопку «Загальна сума» і виводить добуток наявної кількість товару на складі і вартості одиниці товару, який визначений у функції fun1() (змінну, яка відповідає за визначений товар зробіть видимою в обох функціях) -->

…}

</SCRIPT>

</ HEAD>

<BODY>

<FORM NAME="form1"> / / відкриття дескриптора об'єкта форми.

Введіть назву планети:

<INPUT TYPE="text" NAME="name_cont"> / / створення текстового поля для введення назви планети

<INPUT TYPE="button" VALUE="Густина і площа" onClick=" fun1()">> / / створення кнопки, при натисканні на якій в обробнику подій onClick викликається функція fun1()

<INPUT TYPE="text" SIZE="70" NAME="output_density"> / / створення текстового поля з розміром 70 символів для виведення результату.

<INPUT TYPE="text" SIZE="70" NAME="output_area"> / / створення текстового поля з розміром 70 символів для виведення результату.

</ FORM> / / закриття дескриптора об'єкта форми.

</ BODY>

</ HTML>

Завдання 7

Модифікуйте попереднє завдання, передбачивши вибір товару із випадаючого списку.

1.3 Контрольні запитання

1. Які існують способи додавання скриптів JavaScript до HTML-сторінки?
2. Які допустимі символи у назві змінних JavaScript?
3. Які ви знаєте вбудовані функції JavaScript?
4. Яка різниця між var і let?
5. Яка різниця і для чого призначені наступні оператори: =, ==, ===?
6. Які ви знаєте обробники подій і для чого вони використовуються?
7. Які ви знаєте способи створення масивів?
8. Які ви знаєте методи роботи із масивами?

Лабораторна робота №2. Створення годинника, калькулятора та динамічних інтерфейсів засобами Javascript

Мета: набути навичок з написання скриптів та використання функцій на мові JavaScript.

Тривалість: 2 год.

2.1 Теоретичні відомості

1. Тема 1. Основи JavaScript. Використання функцій. Події
2. Тема 2. Робота з масивами
3. Тема 3. Метод console

Додаткові теоретичні відомості

1. Сучасний підручник з JavaScript

2.2 Завдання для виконання лабораторної роботи

Завдання 1

Створити простий калькулятор з кнопками, при нажатті на які будуть виводитися в поле виводу відповідні символи та виконуватимуться обчислення. Для обчислення результату можете використати функцію **eval()**.

Завдання 2

Розробити html документ, в якому існуватиме два текстових елемента “кількість рядків” та “кількість стовпців”. Згідно із введеним їх значенням створити таблицю текстових елементів (див. приклад, значення циклу починати із 1) кожний елемент якого заповнити відповідно індивідуального варіанту (порядковий номер у списку групи). Для варіантів 1, 6, 11, 16, 21, 26 підрахувати суму елементів отриманого двовимірного масиву. Для варіантів 2, 7, 12, 17, 22, 27 підрахувати суму елементів другого рядка. Для варіантів 3, 8, 13, 18, 23, 28 підрахувати суму елементів третього стовпця. Для варіантів 4, 9, 14, 19, 24 підрахувати суму елементів порядковий номер строки та стовпця яких співпадають. Для варіантів 5, 10, 15, 20, 25 підрахувати добуток елементів останнього стовпця.

№ п/п	Варіант завдання	№ п/п	Варіант завдання
1	$A_{ij} = 2i + j$	15	$A_{ij} = 7j + 2i$
2	$A_{ij} = 2i / j$	16	$A_{ij} = 2j - i$
3	$A_{ij} = 11i + j$	17	$A_{ij} = -5j + 3i$
4	$A_{ij} = 2i + 13j$	18	$A_{ij} = 3i - 2j$
5	$A_{ij} = 3i + 2j$	19	$A_{ij} = 11j - i$
6	$A_{ij} = 2i / 5j$	20	$A_{ij} = 2j / 5i$
7	$A_{ij} = 1/(2 + j)$	21	$A_{ij} = 2j + i$
8	$A_{ij} = 11j + i$	22	$A_{ij} = -7j / 3i$
9	$A_{ij} = 5i + 3j$	23	$A_{ij} = 11i - j$
10	$A_{ij} = 2i - j$	24	$A_{ij} = 2j / i$
11	$A_{ij} = -2i / 7j$	25	$A_{ij} = 3j + 2i$
12	$A_{ij} = 3i + j$	26	$A_{ij} = i - j$
13	$A_{ij} = 3j - 2i$	27	$A_{ij} = 5j + 3i$
14	$A_{ij} = -5i + 3j$	28	$A_{ij} = (3 - i)/(j + 5)$

Приклад:

<html>

<head>

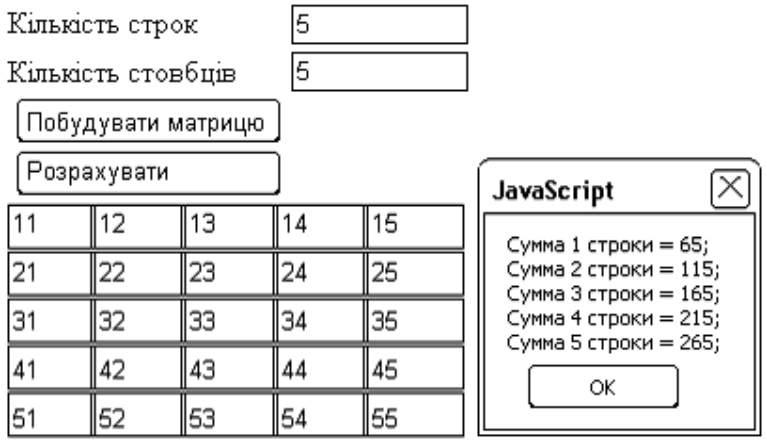
<script>

let str, stb;

```
function fun()
{
str = parseInt(main_form.t_str.value);
stb = parseInt(main_form.t_stb.value);
let res_str = "<table>\n";
for(let i=1;i<=str;i++)
{
res_str+="<tr>\n";
for(let j=1;j<=stb;j++)
{
res_str += "<td>";
res_str += "<input type = \"text\" id = \"_" + i + "_" + j + "\" value = \"" + i +
"" + j + "\">";
res_str += "<\/td>\n";
}
res_str+="<\/tr>\n";
}
res_str += "<\/table>";
main_div.innerHTML = res_str;
}
function fun_build()
{
let res_str = "";
let str_report = "";
for(let i=1;i<=str;i++)
{
let sum = 0;
for(let j=1;j<=stb;j++)
{
res_str = "sum += parseInt(main_form._" + i + "_" + j + ".value);";
eval(res_str);
}
```

```
str_report += "Сумма " + i + " строки = " + sum + ";\n"
}
alert(str_report);
}
</script>
</head>
<body>
<form id = "main_form">
<table>
<tr>
<td>Кількість строк</td>
<td> <input type="text" id = "t_str" value="5"></td>
</tr>
<tr>
<td> Кількість стовбців</td>
<td><input type="text" id = "t_stb" value="5"></td>
</tr>
<tr>
<td> <input type = "button" onClick = "fun();" value = "Побудувати матрицю"><br> <td>
</tr>
<tr>
<td> <input type = "button" onClick = "fun_build();" value = "Розрахувати"> <td>
</tr>
</table>
<div id = "main_div"></div>
</form>
</body>
</html>
```

Результати виконання:



Завдання 3

Примітка 1.

Закладка Console знаходиться на панелі DevTools браузера. Її можна викликати клікнувши на пункт “Перевірити” контекстного меню вікна браузера або F12.

Реалізувати і дослідити роботу методу: console.log() для таких випадки:

− вивід остачі від ділення одного числа на інше;

− вивід логічного значення (true/false) перевірки рівності нулю остачі від ділення одного числа на інше;

− комбінований вивід тексту і значень деякого наперед заданого масиву, використовуючи керуючі послідовності %2, %d, %s).

Приклад виводу значення змінної:

```
let a1 =5
console.log("a1="+a1)
console.log("a1=", a1)
```

Примітка 2:

a%2 як арифметична операція — «остача від ділення a на 2»,

%2 на початку послідовності виведення означає «вивід починати з другого елементу»

Завдання 4

Реалізувати і дослідити роботу наступних методів: console.info(), console.warn(), console.error(), .console.dir(), console.dirxml(), console.time(), console.timeEnd(), console.profile(), console.profileEnd(); console.assert().

2.3 Контрольні запитання

1. Які ви знаєте способи створення масивів?
2. Які ви знаєте методи роботи із масивами?
3. Коли потрібно і не потрібно використовувати метод console.log()?
4. Для чого використовуються %s, %d при виклику метод console.log()?
5. Які ви знаєте console-методи?

Лабораторна робота №3. Javascript. Об'єктна модель документа (DOM). Використання формату JSON

Мета: ознайомитися із об'єктною моделлю документа (DOM) в JavaScript.

Тривалість: 2 год.

3.1 Теоретичні відомості

1. Тема 4. Об'єктна модель документа (DOM)

Додаткові теоретичні відомості

1. DOM дерево
2. Використання формату JSON
3. Пошук елементів в DOM
4. Як використовувати функцію JS .map () (метод масиву)

3.2 Завдання для виконання лабораторної роботи

Завдання 1

Використовуючи посилання на об'єкт DOM, створити код JavaScript "Розрахунок площ геометричних фігур", який за внесеними у текстове поле параметрами обчислює площу геометричної фігури відповідно до варіанту, який рівний остачі від ділення на 10 вашого номеру у списку групи. Сигналом для початку обчислення служить зміна числа в текстовому полі.

1. Формули площі плоских фігур:
2. Формули площі трикутника
3. Формули площі квадрата
4. Формула площі прямокутника
5. Формули площі паралелограма
6. Формули площі ромба
7. Формула площі трапеції
8. Формули площі опуклого чотирикутника
9. Формули площі круга
10. Формула площі еліпса

Орієнтовний вигляд веб-сторінки:

Довжина 2

Ширина 3

Площа 6

Завдання 2

Створити код JavaScript "Екранна клавіатура", який містить кнопки з цифрами та літерами українського алфавіту (перелік літер задається як масив у форматі JSON (наприклад, let a=['а','б','в','г','д','е','є','и','і','ї']) та велике текстове поле, текст в якому можна набирати, натискаючи мишею на відповідні кнопки. Цифри розмістити у верхньому рядку, далі літери розмістити в три рядки. Додати кнопку «Очистити поле»

Завдання 3

Модифікуйте Завдання 2, додавши кнопку «Alt+Shift», при кліку на яку відбудеться перемикання розкладки клавіатури між українською та англійською.

Завдання 4.

Створити код JavaScript для перегляду мінімум чотирьох зображень, який в параметрах отримує імена (шлях до) файлів, як масив у форматі JSON (наприклад, let a=['file1.jpg','file2.gif','file3.gif']) та показує одне вибране зображення. Вибір зображень відбувається за допомогою двох кнопок: «Вперед» та «Назад». Передбачити циклічність зміни зображень, тобто після останнього зображення про кліку на «Вперед» має відображатися перше зображення, а перед першим зображення про кліку на «Назад» має відображатися останнє зображення.

Завдання 5

Створити JavaScript, який перевіряє знання результату арифметичних операцій між двома випадковими числами. Веб-сторінка містить текстовий напис для показу завдання, текстове поле для вводу відповіді, кнопку «перевірити», кнопку «наступне завдання», текстовий напис для виводу

результатів перевірки: “Правильно” чи “Помилка, правильна відповідь «….»”, текстовий напис для показу загального рахунку правильних відповідей із за вказанням загальної кількості питань. Арифметичні операції (+, -, /, *) і числа для обчислення вибирати випадковим чином з діапазону [0;9], передбачити виконання 5-ти завдань.

Орієнтовний зовнішній вигляд веб-сторінки:

Загальний рахунок 80% (4 правильних відповідей з 5)

3 × 4 = 11

Помилка, правильна відповідь «12»

Завдання 6

Створити за допомогою JavaScript дерево для показу ієрархії.

fruits:

- apple:
- pineapple;
- apricot;
- pear;
- lemon;

vegetables:

- potatoes;
- beetroot;
- carrots;
- pear;

Натискання на посилання згортає або розгортає перелік підрозділів.

3.3 Контрольні запитання

1. Що таке “Об'єктна модель документа (DOM)”?
2. Що таке “W3C Document Object Model (W3C DOM)”?
3. Відповідно до об’єктної моделі документа (DOM) чим є HTML-тег та вкладені HTML-теги?
4. Які ви знає функції для отримання посилання на об'єкт DOM?

Лабораторна робота №4. Javascript. Опрацювання даних форми

Мета: набути навичок з написання скриптів опрацювання даних форми на мові JavaScript.

Тривалість: 2 год.

4.1 Теоретичні відомості

1. Тема 4. Об'єктна модель документа (DOM)

Додаткові теоретичні відомості

1. DOM дерево
2. Пошук елементів в DOM

4.2 Завдання для виконання лабораторної роботи

Завдання 1

Скласти сторінку тестування, яка складається із 7 запитань, у кожному по 4-ри варіанти відповіді у кожному (окрім однорядкового текстового поля): 2 групи радіокнопок (input type="radio"), 2 групи прапорців (input type="checkbox"), 2 списки (select): один одинарного і один множинного вибору, 1 однорядкове текстове поле. Для запитань у вигляді прапорців і множинного вибору задати два правильних і два неправильних варіанти відповіді. Для кожного із запитань задати по 1 балу для правильно обраних варіантів відповіді, зокрема, для запитань у вигляді прапорців і множинного вибору, якщо всього обрано три або чотири варіанти відповіді, то запитання оцінити в 0 балів, якщо обрано 2-ва варіанти відповіді і два правильних, то оцінити в 2 бали, якщо з двох обраних варіантів тільки один вірний, то 1 бал. Після кліку на кнопку “Завершити тестування” вивести під запитаннями у вигляді таблиці поставлені запитання, обрані варіанти відповіді, отриманий бал за кожне запитання і під таблицею загальний бал.

Зразок опрацювання радіокнопок:

```
<body>
….
Result0 <span id="rezultatRadio0"></span><p>
….
```

```
</body>

function Result(form) {
let bal= new Array(10)
bal[0] = document.getElementsByName('pol0')
for (let i=0;i<bal[0].length;i++) {
if (form.pol0[i].type === 'radio' && form.pol0[i].checked) {
rezultatRadio0 = parseFloat(form.pol0[i].value)  }else rezultatRadio0=0
 }}
document.getElementById('rezultatRadio0').innerHTML = rezultatRadio0
}
```

4.3 Контрольні запитання

5. Які ви знаєте елементи Web-форми?
6. Що таке “Об'єктна модель документа (DOM)”?
7. Що таке “W3C Document Object Model (W3C DOM)”?
8. Відповідно до об’єктної моделі документа (DOM) чим є HTML-тег та вкладені HTML-теги?
9. Які ви знає функції для отримання посилання на об'єкт DOM?

Лабораторна робота №5. Створення вкладок за допомогою CSS і JavaScript

Мета: набути навичок з написання скриптів для створення вкладок за допомогою CSS і JavaScript.

Тривалість: 2 год.

5.1 Теоретичні відомості

1. Довідник по JavaScript
2. Пошук елементів в DOM

Завдання для виконання

Одним з найпоширеніших проблем веб дизайнерів є розташування великої кількості інформації на одній сторінці без втрати зручності. Одним з найкращих рішень цієї проблеми є використання вкладок.

Завдання 1

Створити меню вкладок відповідно до ходу роботи.

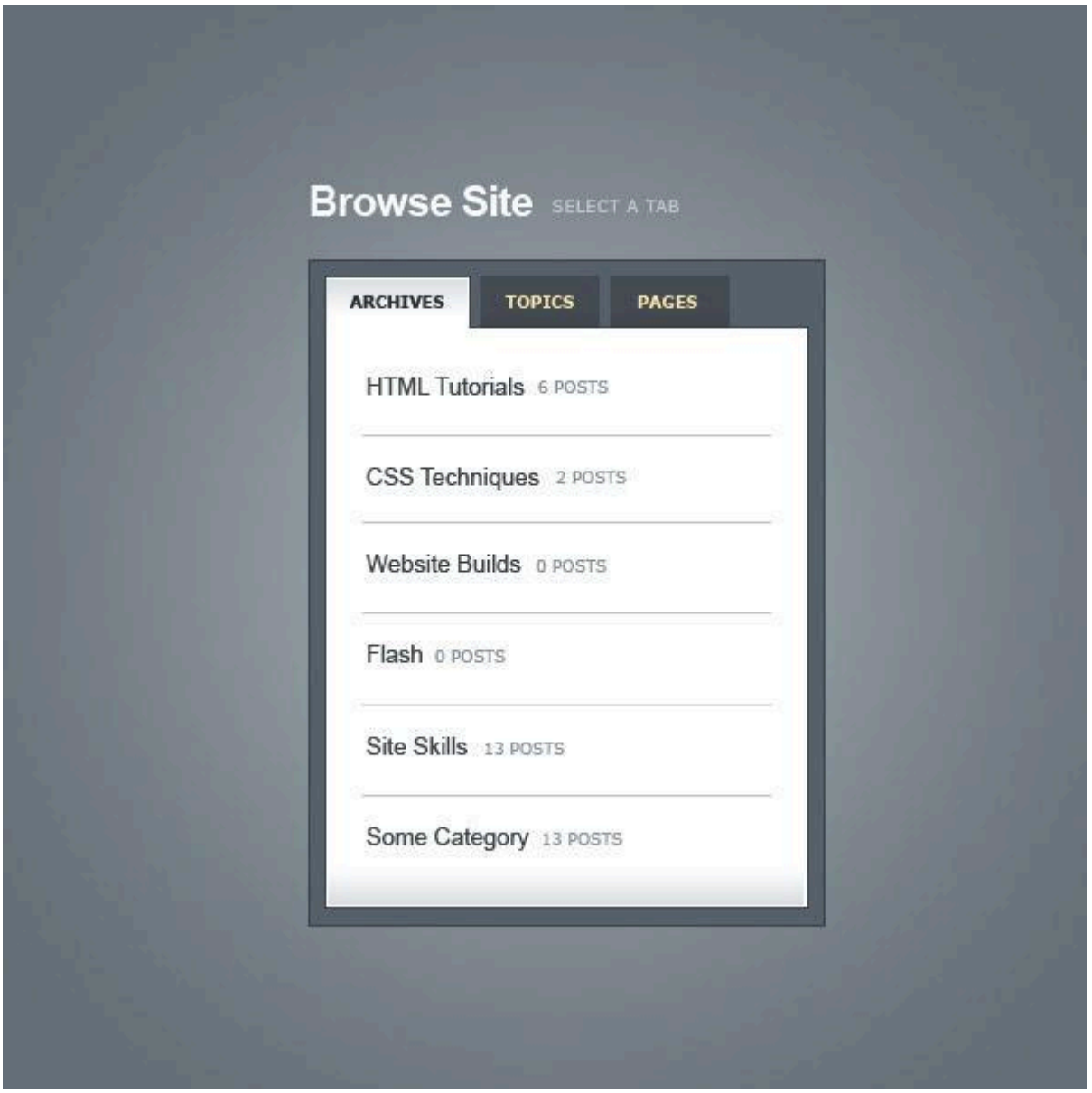
Завдання 2

На основі методу реалізації вкладок, розглянутому в ході роботи, створити **вертикальне** меню із вкладеними підпунктами меню таким чином, щоб зовнішній вигляд активних і неактивних пунктів і підпунктів меню відповідав наступному зображенню:

Хід роботи

Крок 1

По перше нам потрібно створити в графічному редакторі (Photopea.com, Adobe Photoshop тощо) кінцевий результат нашої роботи, тобто підібрати кольори, розміри, розташування, щоб надалі на ходу нічого не потрібно було вигадувати. Ось те що потрібно зробити (і здати) в PSD форматі:



Крок 2

Наступним кроком нам потрібно створити Html верстку, яка виглядає наступним чином:

```
<div>
    <h4>Heading</h4>
    <div>
```

```
<ul>
  <li><a>Tab</a></li> <li> <a> Tab </ a> </ li>
  <li><a>Tab</a></li> <li> <a> Tab </ a> </ li>
  <li><a>Tab</a></li> <li> <a> Tab </ a> </ li>
</ul>
<div>Content for Tab 1</div> <div> Content for Tab 1 </ div>
<div>Content for Tab 2</div> <div> Content for Tab 2 </ div>
<div>Content for Tab 3</div> <div> Content for Tab 3 </ div>
</div>
</div>
```

Тут ми уклали все в тег для подальшого переміщення його в будь-яке місце нашого сайту. Також використовували для заголовка тег <h4>, а для вкладок використовували список, далі знаходяться три div в них буде розташовувати контент вкладок.

Крок 3

Добавляем до нашого html-файлу CSS. Після чого виділяємо і зберігаємо фон з PSD документа як окреме зображення в тій же директорії де і html-файли, у прикладі фонову картинку назвали background.jpg. Далі прописуємо для body наступне:

```
body {
background-image:url(background.jpg);
    background-repeat:no-repeat;
    background-position:top center;
    background-color:#657077;
    margin:40px;
}
```

Тут ми вказали для body фон, розташування фону, прибрали повторення, також вказали фоновий колір тобто фон за картинкою.

Крок 4

Далі ми додаємо до кожного елементу свої класи та ідентифікатори для подальшої маніпуляції ними за допомогою CSS і JavaScript тобто class і id, де це необхідно:

```
<div id="tabbed_box_1" class="tabbed_box">
<h4>Browse Site <small>Select a Tab</small></h4>
<div class="tabbed_area">
<ul class="tabs">
<li><a href="" id="tab_1" class="active">Archives</a></li>
<li><a href="" id="tab_2">Topics</a></li>
<li><a href="" id="tab_3">Pages</a></li> </ul>
<div id="content_1" class="content">Content for Tab 1</div>
<div id="content_2" class="content">Content for Tab 2</div>
<div id="content_3" class="content">Content for Tab 3</div>
</div> </div>
```

Клас active будуть мати активні вкладки. Зараз наша робота виглядає таким чином:

Крок 5

Тепер ми почнемо описувати всі класи та ідентифікатори, а почнемо ми з самого верхнього рівня це пропишемо йому позицію, так щоб він стояв у центрі нашого екрану:

```
#tabbed_box {
    margin: 0px auto 0px auto;
    width:300px;
}
```

Крок 6

Далі описуємо наш заголовок прописуємо йому шрифти і т.д. :

```
.tabbed_box h4 {
    font-family:Arial, Helvetica, sans-serif;
    font-size:23px;
    color:#ffffff;
    letter-spacing:-1px;
    margin-bottom:10px;
}
.tabbed_box h4 small {
    color:#e3e9ec;
    font-weight:normal;
    font-size:9px;
    font-family:Verdana, Arial, Helvetica, sans-serif;
    text-transform:uppercase;
    position:relative;
    top:-4px;
    left:6px;
    letter-spacing:0px;
}
```

Крок 7

Далі описуємо внутрішній div з класом tabbed_area:

```
.tabbed_area {
    border:1px solid #494e52;
    background-color:#636d76;
    padding:8px;
}
```

Тут ми описали межі розміром в 1 піксель і кольору # 494e52, також описали для нього сірий фон і внутрішній відступ у 8 пікселів тобто padding.

Крок 8

Тепер ми нарешті то перейдемо до вкладок. Потрібно буде виконати велику роботу перед тим як елементи li стануть схожі на вкладки:

```
ul.tabs {
    margin:0px; padding:0px;
```

```
}
ul.tabs li {
    list-style:none;
    display:inline;
}
```

У цьому коді описано те що ul з класом tabs не повинен мати ні внутрішніх ні зовнішніх відступів, також що li елементи, які знаходяться в списку з класом tabs не мають картинок ліворуч, які з'являються за замовчуванням, також для цих елементів описано таку властивість як display, значення за замовчуванням якого block, але для того щоб елементи списку тобто li розташовувалися горизонтально ми display прирівнюємо inline.

Крок 9

Звичайно наші вкладки виглядають поки не дуже, але ми це зараз виправимо, для їх позиціонування ми використовували <li> елемент, але для їх стилізації будемо використовувати елемент <a> які знаходяться в li:

```
ul.tabs li a {
    background-color:#464c54;
    color:#ffebb5;
    padding:8px 14px 8px 14px;
    text-decoration:none;
    font-size:9px;
    font-family:Verdana, Arial, Helvetica, sans-serif;
    font-weight:bold;
    text-transform:uppercase;
    border:1px solid #464c54;
}
ul.tabs li a:hover {
    background-color:#2f343a;
    border-color:#2f343a;
}
ul.tabs li a.active {
    background-color:#ffffff;
    color:#282e32;
```

```
    border:1px solid #464c54;
    border-bottom: 1px solid #ffffff;
}
```

Тут ми описали елементи <a> встановили їм фон, визначили шрифти, прибрали підкреслення, описали відступи і т.д. Також описали для них Hover тобто при наведенні на вкладку курсору фон і межі вкладки будуть змінювати свій колір. Далі описаний клас active він потрібен, для виділення активної вкладки. Зараз все наша робота має виглядати наступним чином:

Крок 10

Далі потрібно зробити так, щоб відображався тільки той конвент, вкладка якого зараз активна для цього нам потрібно контенту інших вкладок прописати display: none після чого вони будуть невидимі, потім ми з допомогою JavaScript'a будемо нальоту міняти dispaly: none на dispaly: block. Зараз пишем:

```
.content {
    background-color:#ffffff;
    padding:10px;
    border:1px solid #464c54;
}
 #content_2, #content_3 {
    display:none;
}
```

Зараз наші вкладки виглядають наступним чином:

Крок 11

Змінюємо ul.tabs наступним чином для того, щоб вони не перетиналися з контентом:

```
ul.tabs {
    margin:0px; padding:0px;
    margin-top:5px;
    margin-bottom:6px;
}
```

Крок 12

Тепер наповнюємо наші вкладки осмисленим конвентом (списками посилань). Міняємо html код на:

```
<div id="tabbed_box_1" class="tabbed_box">
  <h4>Browse Site <small>Select a Tab</small></h4>
  <div class="tabbed_area">
  <ul class="tabs">
   <li><a href="" id="tab_1" class="active">Topics</a></li>
   <li><a href="" id="tab_2">Archives</a></li>
   <li><a href="" id="tab_3">Pages</a></li>
  </ul>
  <div id="content_1" class="content">
   <ul>
   <li><a href="">HTML Techniques <small> 4 Posts </small> </a> </li>
   <li><a href="">CSS Styling <small> 32 Posts </small></a></li>
```

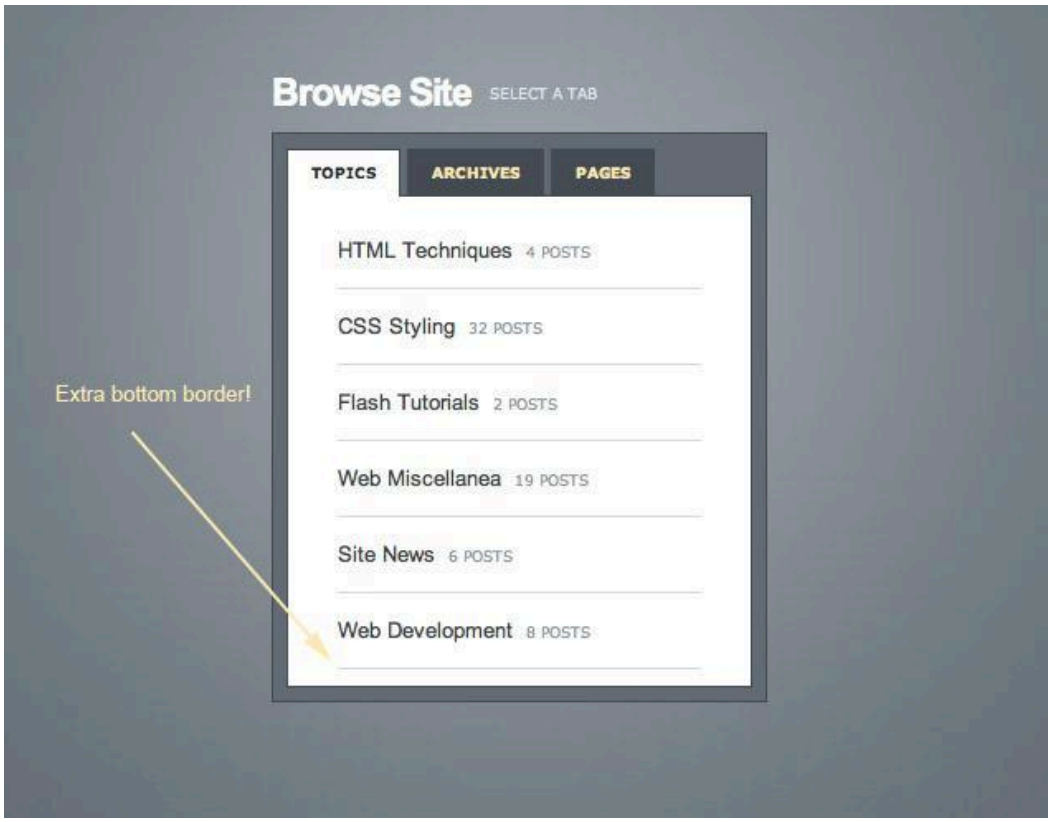
```
<li><a href="">Flash Tutorials <small> 2 Posts </small></a></li>
<li><a href="">Web Miscellanea <small> 19 Posts </small> </a> </li>
<li><a href="">Site News <small> 6 Posts </small></a></li>
<li><a href="">Web Development <small> 8 Posts </small> </a> </li>
</ul>
</div>
<div id="content_2" class="content">
<ul>
<li><a href="">December 2008 <small>6 Posts</small></a></li>
<li><a href="">November 2008 <small> 4 Posts </small> </a> </li>
<li><a href="">October 2008 <small>22 Posts</small></a></li>
<li><a href="">September 2008 <small> 12 Posts </small> </a> </li>
<li><a href="">August 2008 <small>3 Posts</small></a></li>
<li><a href="">July 2008 <small>1 Posts</small></a></li>
</ul>
</div>
<div id="content_3" class="content">
<ul>
<li><a href="">Home</a></li>
<li><a href="">About</a></li>
<li><a href="">Contribute</a></li>
<li><a href="">Contact</a></li>
</ul>
</div>
</div>
</div>
```

Для задання зовнішнього вигляду пишемо наступні стилі:

```
.content ul {
    margin:0px;
    padding:0px 20px 0px 20px;
}
.content ul li {
```

```
        list-style:none;
        border-bottom:1px solid #d6dde0;
        padding-top:15px;
        padding-bottom:15px;
        font-size:13px;
    }
    .content ul li a {
        text-decoration:none;
        color:#3e4346;
    }
    .content ul li a small {
        color:#8b959c;
        font-size:9px;
        text-transform:uppercase;
        font-family:Verdana, Arial, Helvetica, sans-serif;
        position:relative;
        left:4px;
        top:0px;
    }
    .content ul li:last-child {
      border-bottom:none;
    }
```

З допомогою content ul li: last-child ми забираєм останню лінію в списку, а так як це CSS3, то в старих браузерах і деяких нових вона все таки буде відображатися:



Крок 13

З CSS все, тепер нам потрібно створити всього лише одну ось таку JavaScript- функцію:

```
function tabSwitch(new_tab, new_content) {
    document.getElementById('content_1').style.display = 'none';
    document.getElementById('content_2').style.display = 'none';
    document.getElementById('content_3').style.display = 'none';
    document.getElementById(new_content).style.display = 'block';

    document.getElementById('tab_1').className = '';
    document.getElementById('tab_2').className = '';
    document.getElementById('tab_3').className = '';
    document.getElementById(new_tab).className = 'active';
}
```

Як ви помітили нашій функції потрібно передати два значення це new_tab і new_content, так от передати ми їй повинні id натиснутої вкладки і відповідний контент цієї вкладки, далі функція забирає у всіх вкладок клас active, навіть у натиснутої так як ми не знаємо яка вкладка активна, і ставимо клас active переданому id, тобто new_tab, після чого вона стає білого кольору

і т.д. Відповідно теж саме робимо для контенту тільки ми йому ставимо не клас, а робимо його видимим, а решту контенту робимо невидимим з допомогою dispaly: none;

Останній крок 14

Тепер нам потрібно зробити так, щоб на кліку по вкладці виконувалася функція tabSwitch () і передавати їй id поточної вкладки і id відповідного вкладці контенту, для цього змінюємо список вкладок наступним чином:

```
<ul class="tabs">
<li><a href="javascript:tabSwitch('tab_1', 'content_1');" id="tab_1" class="active"> Topics </a></li>
<li><a href="javascript:tabSwitch('tab_2', 'content_2');" id="tab_2">Archives</a></li>
<li><a href="javascript:tabSwitch('tab_3', 'content_3');" id="tab_3">Pages</a></li> </ul>
```

5.3 Контрольні запитання

1. Для чого використовується функція getElementById?
2. Поясніть що виконує такий код:
   ```
   <a href="javascript:tabSwitch('tab_1', 'content_1');" id="tab_1" class="active"> Topics </a>
   ```
3. До якого елементу HTML-документа звертається такий селектор “.content ul li:last-child”?
4. Поясніть що виконує такий код:
   ```
   document.getElementById('content_3').style.display = 'none';
   document.getElementById(new_content).style.display = 'block';
   ```
5. Поясніть що виконує такий код:
   ```
   document.getElementById('tab_3').className = '';
   document.getElementById(new_tab).className = 'active';
   ```

Лабораторна робота №6. JavaScript. Регулярні вирази. Методи обробки регулярних виразів та рядків.

Мета: набути навичок з написання скриптів для створення і застосування регулярних виразів на мові JavaScript.

Тривалість: 2 год.

6.1 Теоретичні відомості

1. Тема 5. JavaScript. Регулярні вирази. Методи регулярних виразів та рядків

Додаткові теоретичні відомості

1. Регулярні вирази
2. Методи регулярних виразів та рядків

6.2 Завдання до виконання лабораторної роботи

Завдання 1

Розібрати і реалізувати приклад виводу елементів e-mail:

```
<script>
let s="dwbyu@mail.ua";
let r=/^([a-z0-9]+)(\.[a-z0-9]+)*@([a-z0-9]+)(\.[a-z0-9]+)*$/i;
let a=s.match(r);
for (i=0;i<a.length;i++)
{
document.writeln("<hr> a["+i+"] "+a[i]);
}
</script>
```

Завдання 2.1

Створити форму, яка міститиме наступні поля для заповнення:

- адреса електронної пошти (e-mail).

- ім’я та прізвище;
- логін;
- пароль;
- поштовий індекс;

Завдання 2.2

Використовуючи регулярні вирази JavaScript по кліку на кнопку виконати перевірку валідності інформації, введеної в поля форми:

- e-mail на наявність символу @, після якого йде pnu.edu.ua;
- ім’я та прізвище: повинні містити лише літери, у першому слові перша літера велика, наступні маленькі, у другому слові всі букви великі, дозволено кирилиця і латина;
- логін тільки латина;
- пароль: мінімум 8 символів, з яких мінімум по 1 букві, цифрі і спецсимволу: _-!@#$%^&*;
- поштовий індекс повинен складатися з 5 цифр.

Якщо поле заповнене вірно, після кліку на кнопку, колір рамки поля стає зеленим і справа появляться позначка (наприклад, зображення галочки), інакше колір тексту і рамки стає червоним, справа від поля з’являється вимога до заповнення (Використати об’єктну модель).

6.3 Контрольні запитання

1. Що таке “Регулярні вирази” і для чого вони використовуються?
2. Які ви знаєте прапори для роботи із регулярними виразами і для чого вони призначені?
3. Що таке алфавіт регулярних виразів, наведіть приклади і призначення символів із цього алфавіту?
4. Які ви знаєте спеціальні символи регулярних виразів і для чого вони призначені?
5. Для чого використовуються наступні символи у регулярному виразі: *, +, ?, {2,}, {2,4}, поясніть їх роботу?
6. Які ви знаєте методи для роботи із рядками та регулярними виразами і для чого вони призначені?

Лабораторна робота №7. JavaScript. Робота з рядками

Мета: набути навичок з написання скриптів опрацювання рядків на мові JavaScript.

Тривалість: 2 год.

7.1 Теоретичні відомості

1. Тема 5. JavaScript. Регулярні вирази. Методи регулярних виразів та рядків

Додаткові теоретичні відомості

1. Методи регулярних виразів та рядків

7.2 Завдання до виконання лабораторної роботи

Завдання 1

Реалізувати наступні функції:

1.1. Функція, що виводить задану послідовність чисел в оберненому порядку (split).

1.2. Створити функцію no_zeros(), аргументом якої є масив чисел, а результатом дії — модифікований вхідний масив, який не містить нульових значень (replace).

1.3. Створити функцію reverser(), аргументом якої є текстовий рядок, а результатом дії — відображення вхідного рядку у протилежному порядку.

1.4. Створити функцію replace(), аргументом якої є текстовий рядок «Я не люблю морозиво», а результатом дії:

a) друге і третє слово поміняти місцями;

b) пропуски замінені на коми.

Завдання 2

Реалізувати наступні функції для роботи із рядками:

2.1 Функція, яка приймає 2 рядки та порівнює їх довжину. Функція повертає 1, якщо в першому рядку більше символів, ніж у другому; -1 - якщо

у другому більше символів, ніж у першому; 0 - якщо рядки однакової довжини.

2.2 Функція, яка приймає рядок і символ і виводить індекси, за якими знаходиться цей символ у рядку. Також виведіть, скільки разів зустрічається цей символ у рядку.

2.3 Функція, яка переводить у верхній регістр перший символ

переданого рядка.

2.4 Функція, яка підраховує кількість голосних літер у переданому рядку.

2.5 Функція, для перевірки спаму в переданому рядку. Функція повертає true, якщо рядок містить спам. Спамом вважати наступні слова: 100% безкоштовно, збільшення продажів, тільки сьогодні, не видаляйте.

2.6 Напишіть функцію скорочення рядка. Функція приймає рядок та його максимальну довжину. Якщо довжина рядка більша, ніж максимальна, необхідно відкинути зайві символи, додавши замість них три крапки. Три крапки рахуються до максимальної довжини. Наприклад: truncate("Hello, world!", 8) має повернути "Hello...".

Завдання 3

Використати регулярні вирази, реалізуйте наступні завдання:

3.1 Напишіть функцію toCamelCase(), яка приймає значення імені змінної в стилі “Snake case”, та повертає це ж ім’я в стилі «Lower camel case». Вхідні і вихідні назви змінних розмістіть у масивах. Виведіть значення вхідного і вихідного масиву.

Для прикладу: значення, що передається в функцію: my_variable, new_brand_product Результат роботи функції: myVariable, newBrandProduct

3.2 Напишіть функцію toSnakeCase(), яка працює як обернена до функції toCamelCase().

3.3 У тексті замініть всі дати формату yyyy/mm/dd на формат dd.mm.yyyy. Текст може бути як завгодно великий і одна з вимог - використати пошук і заміну з використанням регулярних виразів. Текст вводить користувач.

7.3 Контрольні запитання

1. Які ви знаєте методи для роботи із рядками та регулярними виразами і для чого вони призначені?
2. Що таке стилі “Snake case” та «Lower camel case», яки між ними різниця?

Лабораторна робота №8. JavaScript. Робота з об'єктами. Вбудований об’єкт Date

Мета: набути навичок з написання скриптів для роботи з об'єктами на мові JavaScript.

Тривалість: 2 год.

8.1 Теоретичні відомості

1. Тема 6. JavaScript: Робота з об'єктами. Вбудований об’єкт Date.

Додаткові теоретичні відомості

1. Об'єкти
2. Вбудований об’єкт Date

8.2 Завдання до виконання лабораторної роботи

Завдання 1

Засобами JavaScript створити годинник або календар (див. індивідуальний варіант) у форматі згідно номеру за списком:

№п/п	Формат
1	hour/min/sec+year*moun*day
2	year:moun:day:hour:min:sec
3	year#moun#day-hour#min#sec
4	year-moun-day-hour-min-sec
5	hourminsec-year$moun$day
6	hour:min:sec-year:moun:day
7	hour/min/sec+year*moun*day
8	year:moun:day:hour:min:sec

9	year#moun#day-hour#min#sec
10	year-moun-day-hour-min-sec

Приклад:

```
<html>
<head>
<meta charset=ut8></meta>
</head>
<body onLoad="clock()">
<form name="clock">
Час:
<input type="text" name="time" size="8" value=""><br>
Дата:
<input type="text" name="date" size="8" value="">
</form>
<script Language="JavaScript">
let timeStr, dateStr;
function clock() {
now= new Date();
hours= now.getHours();
minutes= now.getMinutes();
seconds= now.getSeconds();
timeStr= "" + hours;
timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
timeStr+= ((seconds < 10) ? ":0" : ":") + seconds;
document.clock.time.value = timeStr;
date= now.getDate();
month= now.getMonth()+1;
year= now.getYear();
dateStr= "" + month;
dateStr+= ((date < 10) ? "/0" : "/") + date;
```

```
dateStr+= "/" + year;
document.clock.date.value = dateStr;
Timer= setTimeout("clock()",1000);
}
</script>
</body>
</html>
```

Завдання 2

Створіть об'єкт, що описує час (години, хвилини, секунди), і наступні функції для роботи з цим об'єктом.

2.1. Функція виведення поточного часу на екран.

2.2. Функція зміни часу на передану через поле введення кількість секунд.

2.3. Функція зміни часу на передану через поле введення кількість хвилин.

2.4. Функція зміни часу на передану через поле введення кількість годин.

Враховуйте те, що в останніх 3 функціях при зміні однієї частини часу, може змінитися й інша. Наприклад: якщо до часу «20:30:45» додати 30 секунд, то має вийти «20:31:15», а не «20:30:75».

Завдання 3

3.1 Написати функцію, яка виводить на сторінку поточну дату у вигляді:

Дата: 17 жовтня 2023 року

День тижня: вівторок

Час: 15:08

Реалізувати запуск цієї функції при натисканні на кнопку з написом “Виконати”, перед якою напишіть “Завдання 2.1 Виведення поточної дати, дня тижня і часу”.

3.2 Написати функцію, яка приймає об’єкт Date і повертає інформацію про день тижня у вигляді об’єкта, що має поля:

- dayNumber - номер дня тижня для дати у звичайному форматі (1 – понеділок, 2 – вівторок, 3 – середа і т.д);

- dayName – назва дня тижня українською мовою.

Реалізувати запуск цієї функції при натисканні на кнопку з написом “Виконати”, перед якою напишіть “Завдання 2.2 Виведення дня і номера тижня”.

Наприклад: Номер тижня: 2

Назва дня тижня: вівторок.

3.3 Знайти дату, яка була N днів назад чи вперед (значення вводить користувач у текстове поле; N може бути від’ємним, якщо дата назад і додатнім, якщо дата вперед). При введенні вказати чи це минула чи майбутня дата.

3.4 Напишіть функцію, яка за заданим роком та номером місяця визначає останній день місяця.

3.5 Реалізуйте функцію, яка повертає об’єкт з двома полями:

- кількість секунд, яка пройшла від початку сьогоднішнього дня;

- кількість секунд до початку наступного дня.

3.6 Напишіть функцію, яка за введеним користувачем рядом у форматі “дд.мм.рррр год:хв” формує об’єкт *date* і яка:

- якщо з часу *date* пройшло менше хвилини, то "n сек. назад";

- інакше, якщо пройшло менше години, то "m хв. назад";

- інакше, повна дата у форматі "дд.мм.рррр год:хв".

8.3 Контрольні запитання

1. Назвіть приклади об’єктів JavaScript.
2. З чим дозволяє працювати Об’єкт Date?
3. Яку дату створює об'єкт new Date(0)?
4. Які ви знаєте методи об'єкту Date() і для чого вони призначені?

Лабораторна робота №9. JavaScript-анімації

Мета: набути навичок із створення JavaScript-анімації

Тривалість: 2 год.

9.1 Теоретичні відомості

1. JavaScript-анімації

9.2 Завдання до виконання лабораторної роботи

Завдання 1

Розібратися і реалізувати приклади створення JavaScript-анімацій, наведених у теоретичних відомостях.

Завдання 2

Створити анімований логотип спеціальності 121 Інженерія програмного забезпечення, який активізується і зупиняється при кліку кнопки миші або через певний час. Під час створення анімації обов'язково задати зміну розміру і напряму руху об'єктів, динамічну появу та зникання текстової інформації, зміну властивостей об'єктів, появу нових об'єктів, кілька анімацій, запущених одночасно.

9.3 Контрольні запитання

1. У що потрібно вносити зміни, щоб отримати анімацію елементів HTML-документа?
2. Для чого використовується метод requestAnimationFrame?
3. Які ви знаєте JavaScript-функції для створення анімацій елементів HTML-документа?
4. Яка JavaScript-функція для для створення анімації елементів HTML-документа вам найбільше сподобалася і чому?

Лабораторна робота №10. Бібліотека jQuery. Основні поняття і можливості

Мета: набути навичок з написання скриптів з допомогою бібліотеки jQuery.

Тривалість: 2 год.

10.1 Теоретичні відомості

1. Тема 7. Бібліотека jQuery

Додаткові теоретичні відомості

1. Про jQuery

10.2 Завдання до виконання лабораторної роботи

Завдання 1

Проробіть завдання наведені в наступній інструкції:

Початок роботи з бібліотекою jQuery

Якщо ви ще не завантажили бібліотеку, скачайте її з офіційного сайту http://jquery.com/ (або додайте пряме посилання на потрібну бібліотеку)

Помістіть завантажений файл у папку js папки дисципліни. У папці lab10_lastname створіть файл lab10_lastname.html. Скопіюйте нижче наведений код у файл lab10_lastname.html і змініть значення властивості src тегу *script* на шлях до завантаженого Вами файлу.

```
<html>
<head>
<script type="text/javascript" src="jquery-3.7.1.min"></script>
<script src="jquery-3.7.1.js"></script>
<script>
$(document).ready(function(){
alert("Jquery ready!!!");
 });
</script>
```

```
</head>
<body>
Контент сторінки
</body>
</html>
```

Запустіть скрипт, і переконайтеся, що jQuery вже працює. При оновленні сторінки ми отримуємо напис "Бібліотека jQuery готова до роботи!"**.** Це означає, що ми все зробили правильно. Якщо ви не отримуєте таке повідомлення, то уважно подивіться на назву та розташування файлу з бібліотекою, можливо потрібно відкоригувати рядок ініціалізації файлу:

```
<script type = "text / javascript" src = " jquery-3.7.1.min"> </script>
```

Подивіться на вміст наведеного коду, і ви побачите не характерний для звичайного Javascript скрипта запис:

```
$ (document).ready(function() {
alert ("Бібліотека jQuery готова до роботи!");
});
```

Подія **ready** - означає, що виконання вмісту буде виведено тільки після завантаження всієї сторінки і остаточному формуванні DOM**.** Детальніше події бібліотеки jQuery описані у теорії начального центру jQuery. Якби ми не використовували функціонал бібліотеки, а працювали з чистим Javascript нам замість цього довелося б використовувати такий запис:

```
window.onload = function() {...}
```

Перший скрипт на jQuery

Тепер доповнимо код, і напишемо перший простий плагін:

```
<html>
<head>
<script type = "text/javascript" src = "jquery-3.7.1.min"> </script>
<script>
$ (document).ready(function() {
var $text = $('#hide_text') ;// присвоюємо змінній text блок з id = hide_text
$text.hide() ;// приховуємо отриманий блок
```

```
$ ('.button').click(function() {// при кліці на HTML елемент з класом button
$text.show(200) ;// показуємо прихований блок із затримкою 200 мілісекунд
});
});
</script>
</head>
<body>
<H1> Контент сторінки </h1>
<a href = "#" class = "button"> Натисніть, щоб показати текст </a>
<div id = "hide_text"> Демонстраційний прихований текст </div>
</body>
</html>
```

Перезавантажте сторінку і подивіться, що у нас вийшло. Коли DOM сформований, у події **ready** ми отримуємо вміст блоку з id= hide_text**,** та одразу ж приховуємо його методом **hide().**

Потім при натисканні на посилання класу **button,** спрацьовує обробник подій **click()** бібліотеки jQuery, який подією **show ()** показує нам прихований текст:

```
$ ('.button').Click (function() {
text.show (200);
});
```

У цьому скрипті, написаному за допомогою бібліотеки jQuery ми з легкістю отримали об'єкти DOM'а простими конструкціями:

```
$('#hide_text'); // отримуємо блок з id = hide_text
$('.button'); // отримуємо елементи де class = button
```

Порівняння з чистим Javascript

А тепер подивіться, як би ми отримали той же самий результат, якби не використовували jQuery:

```
<script>
window.onload = function() {
```

```
document.getElementById ('hide_text').style.display = 'none';
}
function visibleText () {
document.getElementById ('hide_text').style.display = 'block';
};
</script>
<a href = "#" class = "button" onclick = "visibleText();"> Натисніть, щоб показати текст </a>
```

Як бачимо, робота з бібліотекою jQuery робить процес програмування більш швидким, а код більш читабельним.

Завдання 2

2.1 Модифікуйте код із завдання 1 таким чином, щоб виводилося два повідомлення на сторінці розміщені у першому і останньому абзаці.

Функції:

```
$("p:first").text("DOM first!");
jQuery("p:last").text("DOM last!");
```

2.2 Додайте стиль на свій вибір для div-об’єкта, над яким знаходиться покажчик миші (**hover, addClass, removeClass**).

Завдання 3

3.1 Додайте спеціальний стиль для div-об’єкта: при одинарному кліку задайте появу рамки і заливки, при подвійному — повернення до початкових значень. (**bind: click, dblclick, addClass, removeClass**)

3.2 Створіть 2-ві кнопки. Перша при одинарному кліку виводить повідомлення «Одинарний клік», друга при подвійному кліку виводить повідомлення «Подвійний клік». (**bind: click, dblclick, alert**)

3.3 Додайте надпис «Показати зображення», при кліку на який появиться зображення. При кліку на зображення воно пропадає, а при наведенні покажчика миші на зображення з’являється спливаюче вікно, де вказано підпис до зображення (**mouseout, mouseover**).

3.4 Задайте форму для перевірки коректності введених даних, яка містить однорядкове текстове поле для введення і кнопку, для відправки даних . На сторінці напишіть «Введіть в текстове поле слово 'correct'». Якщо

слово введене правильно, виведіть повідомлення «Успішно!», інакше під текстовим полем виведіть «Неправильно!»

```
$("form").submit(function() {
    if ($("input:first").val() == "correct") {
      $("span").text("Перевіряєм...").show();
      return true;
    }
    $("span").text("Неправильно!").show().fadeOut(1000);
    return false;
  });
<p> Введіть в текстове поле слово 'correct'.</p>

 <form action="javascript:alert('Успішно!');">
   <div>
     <input type="text" />
     <input type="submit" />
   </div>
 </form>
 <span></span>
```

10.3 Контрольні запитання

1. Що таке бібліотека jQuery?
2. Де можна завантажити актуальну версію бібліотеки jQuery?
3. Що забезпечує конструкція JavaScript:
   jQuery(document).ready(function() { ... });
4. Які різниця між “jquery-3.7.1” і “jquery-3.7.1.min”?
5. Назвіть три правила додавання jQuery на сторінку.
6. Назвіть «основні» методи бібліотеки jQuery, поясніть їх призначення.
7. Назвіть «хелпери» бібліотеки jQuery, поясніть їх призначення.

Лабораторна робота №11. Бібліотека jQuery. Маніпуляції з HTML-елементами. jQuery переміщення.

Мета: набути навичок з написання скриптів засобами бібліотеки jQuery.

Тривалість: 2 год.

11.1 Теоретичні відомості

1. Тема 7. Бібліотека jQuery

Додаткові теоретичні відомості

1. Про jQuery
2. Ітерація об’єктів jQuery та об’єктів, що не належать до jQuery
3. Дії

11.2 Завдання до виконання лабораторної роботи

Завдання 1

Розробити сценарії, які демонструють виконання усіх наступних методів:

- .html();
- .css();
- .val();
- .attr (ім'я атрибута), .attr (ім'я атрибута, значення), .attr (атрибути), .attr (ім'я атрибута, функція), .removeAttr()

Завдання 2

Розробити сценарії, які демонструють результат виконання наступних методів згідно варіанту:

№	Додавання	Заміни і видалення	Розмір і координати
1.	.append()	.replaceAll()	.width(функция); .position()
2.	.appendTo()	.replaceWith()	.width(функция); .outerWidth()

3.	.prepend()	.clone()	.height(функция); .outerHeight()
4.	.prependTo()	.replaceAll()	.innerHeight(функция); .offset()
5.	.before()	.replaceWith()	.width(функция); .position()
6.	.insertBefore()	.empty()	.height(функция); .outerWidth()
7.	.after()	.remove()	.innerHeight(функция); .outerHeight()
8.	.insertAfter()	.detach()	.width(функция); .offset()
9.	.wrap()	.unwrap()	.height(функция); .position()
10.	.wrapInner()	.clone()	.innerHeight(функция); .outerHeight()

Завдання 3

Розробити вебсторінку, яка містить кнопки із значеннями згідно варіанту, при кліку на які демонструється виконання відповідних методів:

№	Метод				
1.	.add()	.parent()	.addBack()	.has()	.map()
2.	.addBack()	.has()	.children()	.last()	.parents()
3.	.children()	.is()	.parents()	.closest()	.parentsUntil()
4.	.closest()	.last()	.parentsUntil()	.is()	.parent()
5.	.contents()	.map()	.prev()	.each()	.nextAll()
6.	.each()	.next()	.prevAll()	.contents()	.nextUntil()
7.	.end()	.nextAll()	.prevUntil()	.eq()	.prev()
8.	.eq()	.nextUntil()	.siblings()	.next()	.end()
9.	.find()	.not()	.first()	.add()	.offsetParent()
10.	.first()	.offsetParent()	.find()	.slice()	.not()

Завдання 4

Використовуючи метод **.filter()**, задати ввімкнення деяких стилів і додавання тексту “modify” :

- для всіх елементів деякого класу;
- для кожного третього елементу <li>.

Ввімкнення стилів відбувається після кліку на відповідну кнопку.

Завдання 5

Розробити сценарії, які демонструють виконання методу відповідно до вашого варіанту і трьох наступних:

1. $.contains(),
2. $.data(),
3. $.extend(),
4. $.fn.extend(),
5. $.globalEval(),
6. $.grep(),
7. $.inArray(),
8. $.isArray(),
9. $.isEmptyObject(),
10. $.isNumeric(),
11. $.isPlainObject(),
12. $.makeArray(),
13. $.map(),
14. $.merge(),
15. $.noop(),
16. $.now(),
17. $.parseHTML(),
18. $.removeData(),
19. $.trim(),
20. $.type(),(вивести мінімум 5 різних значень виводу)
21. $.uniqueSort(),
22. $.contains(),
23. $.data(),
24. $.extend(),
25. $.fn.extend(),

26. $.globalEval(),
27. $.grep(),
28. $.inArray()

11.3 Контрольні запитання

1. Яке призначення методу .html()?
2. Яке призначення методу .css()?
3. Яке призначення методу .val()?
4. Яке призначення методі .attr (ім'я атрибута), .attr (ім'я атрибута, значення), .attr (атрибути), .attr (ім'я атрибута, функція), .removeAttr()?
5. Яке призначення методу .filter()?
6. Який із використаних вами методі вам найбільше сподобався і чому?

Лабораторна робота №12. jQuery-анімація. jQuery-ефекти

Мета: набути навичок з написання скриптів для написання jQuery-анімацій та jQuery-ефектів

Тривалість: 2 год.

12.1 Теоретичні відомості

1. jQuery анімація

12.2 Завдання до виконання лабораторної роботи

Завдання 1

Розробити сценарії, які демонструють виконання методу відповідно до вашого варіанту і мінімум 3-х методів на власний вибір:

1. .clearQueue()
2. .delay()
3. .dequeue()
4. .fadeIn()
5. .fadeOut()
6. .fadeTo()
7. .fadeToggle()
8. .finish()
9. .hide()
10. .fx.off
11. .queue()
12. .show()
13. .slideDown()
14. .slideToggle()
15. .slideUp()
16. .stop()
17. .toggle()
18. .clearQueue()

19. .delay()
20. .dequeue()
21. .fadeIn()
22. .fadeOut()
23. .fadeTo()
24. .fadeToggle()
25. .finish()
26. .hide()
27. .fx.off
28. .queue()

Завдання 2

Створити анімований логотип спеціальності 121 Інженерія програмного забезпечення або кафедри інформаційних технологій з допомогою методу .animate()**,** використовуючи як мінімум наступні параметри: properties (мінімум три властивості), duration, easing, complete, options (мінімум три опції).

12.3 Контрольні запитання

1. Який із використаних вами методі вам найбільше сподобався і чому?
2. Для чого призначений методу .animate()?
3. Для чого призначений параметр properties методу .animate()?
4. Для чого призначений параметр duration методу .animate()?
5. Для чого призначений параметр easing методу .animate()?
6. Для чого призначений параметр complete методу .animate()?
7. Для чого призначений параметр options методу .animate()?

Список літератури

1. Довідник по JavaScript
2. Сучасний підручник з JavaScript
3. Використання формату JSON
4. Як використовувати функцію JS .map () (метод масиву)
5. Про jQuery