Практична робота №2

Тема: ***Методи alert, confirm, prompt. Масиви.***

Завдання0

Опрацювати теоретичні відомості:

пункти 5.4 – 5.5 за посиланням https://learn.javascript.ru

Завдання1. *Методи alert(), confirm() и prompt()*

1) Запитайте у користувача ціле число.
2) Якщо це число просте, то через **alert()** повідомте про це. Робіть це до тих пір, поки користувач не введе 0.
3) Розрахуйте суму простих введених користувачем чисел і виведіть на екран у вигляді, 2 + 3 + 11 + ... + введене_число = сума

Завдання2

Знайдіть та виправте синтаксичні помилки в створенні масивів:

let a [12, 30 50];

1) let b = [sunday, monday, tuesday];
2) let = [1, 2, 3];
3) let myArray = (5);
4) let colors = "red", "green", "blue";

Завдання3

1) Створіть масив seasons з назвами пір року.
2) Виведіть масив, його довжину та окремо всі його елементи.

let seasons = [];

Завдання4

1) Створіть масив daysOfWeek з назвами днів тижня
2) Використовуючи цикл, виведіть елементи даного масиву в окремих рядках
3) Зробіть вивід елементів масиву в зворотному порядку в одному рядку
   let daysOfWeek = [];
4) Використовуючи методи для роботи з масивами, здійсніть циклічний зсув в масиві daysOfWeek: перенесіть останній елемент на початок

Завдання5.

1) Для кожної пори року створіть масив, що містить відповідні назви місяців.
2) Створіть масив seasons та об’єднайте в нього всі 4 масиви

Завдання6.

Двовимірні масиви

1) Створіть двовимірний масив 3 на 3 з будь-якими числами.
2) Виведіть цей масив в тезі <table>, щоб кожне число було в своїй комірці.
3) Напишіть скрипт, який діагональні елементи зробить рівними 1, а решта 0. Тобто елементи [0] [0], [1] [1], [2] [2] - 1, а решта 0.
4) Виведіть через <table> отриманий масив.

Методичні рекомендації

Базові UI операції: alert, prompt і confirm

Дозволяють працювати з даними, отриманими від користувача.

alert

синтаксис:

alert (повідомлення)

alert виводить на екран вікно з повідомленням і призупиняє виконання скрипта, поки користувач не натисне «ОК».

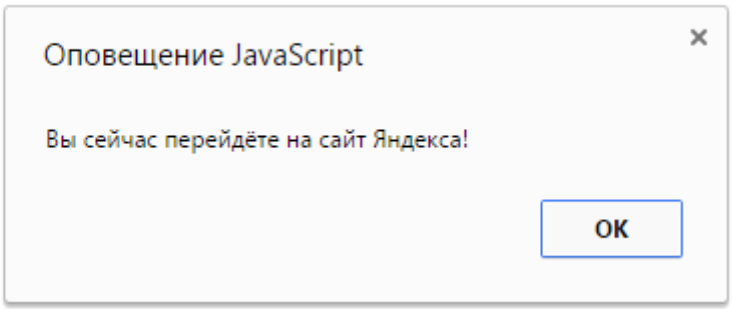
```
alert ( "Привіт");
```

Вікно повідомлення, яке виводиться, є модальним вікном. Слово «модальне» означає, що відвідувач не може взаємодіяти зі сторінкою, натискати інші кнопки і т.п., поки не розбереться з вікном. В даному випадку - поки не натисне на «ОК».

Наприклад, виведемо для користувача сайту при натисканні на посилання попереджувальне діалогове вікно:

```
1. <a href="http://yandex.ru" onclick="alert('Вы сейчас перейдёте на сайт Яндекса!')">
2.   Перейти на сайт
3. <a/>
```

Перейти на сайт



prompt

Функція **prompt** приймає два аргументи:

```
result = prompt (title, default);
```

Вона виводить модальне вікно з заголовком **title**, полем для введення тексту, заповненим рядком за замовчуванням **default** і кнопками OK / CANCEL.

Користувач повинен або щось ввести і натиснути OK, або скасувати введення кліком на CANCEL або натисканням Esc на клавіатурі.

Виклик **prompt** повертає те, що ввів відвідувач - рядок або спеціальне значення null, якщо введення скасоване.

Як і у випадку з **alert**, вікно **prompt** модальне.

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

*Приклад: скласти 2 числа і результат вивести за допомогою **alert:***

```
let x=prompt("Введіть 1 число");  //наприклад, x = 5
```

let y=prompt("Введіть 2 число"); //наприклад, y = 4
alert(x + "+" + y + "=" + (x+y)); // результат: 5 + 4 = 54, тому що prompt повертає рядок

Щоб вивести правильну суму, потрібно скористатись функцією Number:
let x=Number(prompt("Введіть 1 число")); //наприклад, x = 5
let y=Number(prompt("Введіть 2 число")); //наприклад, y = 4
alert(x + "+" + y + "=" + (x+y)); // результат: 5 + 4 = 9

confirm

синтаксис:

result = confirm (question);

confirm виводить вікно з питанням **question** з двома кнопками: OK і CANCEL.

Результатом буде true при натисканні OK і false - при CANCEL (Esc).

наприклад:

```
let isAdmin = confirm ( "Ви - адміністратор?");
alert (isAdmin);
```

Масиви в JavaScript

1) Оголошення масивів в JavaScript

```
let seasons = ["Winter", "Spring", "Summer", "Autumn"];
let grades = [5, 5, 4, 5, 8];
let products = [];
```

2) Об'єктний спосіб створення масивів

```
//Оголошення із заданням значень
var colors = new Array("red", "yellow", "green");

//Оголошення із вказанням довжини
var weekendDays = new Array(2);

weekendDays[0] = "Saturday";
weekendDays[1] = "Sunday";
```

3) Масиви можуть містити елементи будь-якого типу

```
let user = [
  "John",
  new Date(1980, 10, 25),
  {
    "city": "Zhytomyr",
    "street": "Kyivska"
  },
]

console.log(user[2].city);
```

4) Простий спосіб додавання нових елементів масиву

```
//Додавання нових елементів масиву
let months = [];
months[1] = "January";
months[2] = "February";
months[4] = "April";
console.log(months);
```

*5) Властивість **length** визначає довжину масиву*

```
var colors = new Array("red", "yellow", "green");
console.log(colors.length);
```

*6) **length** можна перевизначати:*

```
let months = ["December", "January", "February"];
months.length = 12;
console.log(months);
```

```
▶ (12) ["December", "January", "February", empty × 9]
```

7) Прохід по масиву

```
//З використанням циклу з параметром
let seasons = ["Winter", "Spring", "Summer", "Autumn"];
for (let i = 0; i < seasons.length; i++) {
  console.log(seasons[i]);
}
```

*Метод **forEach***

Метод **arr.forEach** позволяет запускать функцию для каждого элемента массива.

Его синтаксис:

```
arr.forEach(function(item, index, array) {
  // ... делать что-то с item
});
```

Например, этот код выведет на экран каждый элемент массива:

```
// Вызов alert для каждого элемента
["Bilbo", "Gandalf", "Nazgul"].forEach(alert);
```

А этот вдобавок расскажет и о своей позиции в массиве:

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} имеет позицию ${index} в ${array}`);
```

```
});
```

Результат функции (если она вообще что-то возвращает) отбрасывается и игнорируется.

```
//Об'єктно-орієнтовний спосіб: використання методу forEach
seasons.forEach(function(season, i, seasons){
  console.log(season);
})
```

*8) Методи **push/pop***

```
let courses = ["JS", "HTML", "CSS", "Pascal"];
//pop() - отримує останній елемент та видаляє його з масиву
let lastElement = courses.pop();
//push() - додає елемент в кінець масиву та отримує його нову довжину
let len = courses.push("PHP");
console.log(courses);
```

*9) Методи **shift/unshift***

```
let courses = ["Perl", "JS", "HTML", "CSS", "PHP"];

//shift() - отримує перший елемент та видаляє його з масиву
let firstElement =  courses.shift();
//unshift() - додає новий елемент в початок масиву та отримує його нову довжину
let len =  courses.unshift("NodeJS");

console.log(courses);
```

*10) Метод **indexOf()***

```
//indexOf() - отримує індекс елемента масиву по значенню
let courses = ["NodeJS", "JS", "HTML", "CSS", "PHP"];
console.log(courses.indexOf("HTML")); //2
console.log(courses.indexOf("Perl")); //-1
```

Методы **arr.indexOf**, **arr.lastIndexOf** и **arr.includes** имеют одинаковый синтаксис и делают по сути то же самое, что и их строковые аналоги, но работают с элементами вместо символов:

- **arr.indexOf(item, from)** ищет **item**, начиная с индекса **from**, и возвращает индекс, на котором был найден искомый элемент, в противном случае -1.
- **arr.lastIndexOf(item, from)** – то же самое, но ищет справа налево.
- **arr.includes(item, from)** – ищет **item**, начиная с индекса **from**, и возвращает **true**, если поиск успешен.

Например:

```
let arr = [1, 0, false];
alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
```

```
alert( arr.indexOf(null) ); // -1
alert( arr.includes(1) ); // true
```

Обратите внимание, что методы используют строгое сравнение ===. Таким образом, если мы ищем **false**, он находит именно **false**, а не ноль.

Если мы хотим проверить наличие элемента, и нет необходимости знать его точный индекс, тогда предпочтительным является **arr.includes**.

*11)Метод **concat()***

```
//concat() - об'єднує два масиви і отримує третій масив
let courses_1 = ["HTML", "CSS", "JS"];
let courses_2 = ["PHP", "WordPress"];
let courses = courses_1.concat(courses_2);
console.log(courses);
```

```
▶ (5) ["HTML", "CSS", "JS", "PHP", "WordPress"]
```

*12) Метод **slice()***

```
let fruits = ["Banana", "Orange", "Lemon", "Apple", "Pear"];
//slice() - заносить частину масиву в інший масив
let favouriteFruits = fruits.slice(3); //З 3-го елемента і до кінця
let citrus = fruits.slice(1, 3); //З 1-го по 3-тій не включно

console.log(favouriteFruits);
console.log(citrus);
```

```
▶ (2) ["Apple", "Pear"]
▶ (2) ["Orange", "Lemon"]
```

```
// Два останніх елементи:
  var twoLast = produce.slice(-2);

  // Елементи з 3-го по 2-гий з кінця
  var favorite = produce.slice(3, -2);
```

*13) Метод **splice()***

```
//splice() - може видаляти частину масиву та додавати нові елементи
let arr = ["I", "know", "JS"];
//Починаючи з 1-го, видалити 1 елемент
// ["I", "JS"]
arr.splice(1, 1);

//Починаючи з 0-го, видалити 1 елемент і додати інші
// ["We", "are", "learning", "JS"]
arr.splice(0, 1, "We", "are", "learning");

//Починаючи з 3-го, видалити 0 елементів і додати інші
// ["We", "are", "learning", "very", "hard", "JS"]
arr.splice(3, 0, "very", "hard");

//Від'ємний номер позиції (відраховуємо з кінця)
//["We", "are", "learning", "very", "hard", "language", "JS"]
arr.splice(-1, 0, "language");
```

*14) Метод **join()***

```
//join() – формує з елементів масиву рядок з розділювачем
let fruits = ["Banana", "Orange", "Lemon", "Apple", "Pear"];
let fruitsStr = fruits.join("; "); //По замовчуванню, розділювач ","
console.log(fruitsStr);
```

```
Banana; Orange; Lemon; Apple; Pear
```

*15) Метод **sort()***

```
let numbers = [4, 8, 5, 9, -3, 2, 1];
//Компаратор - функція, що порівнює два елементи
//В прикладі задає сортування по зростанню
function compare(a, b) {
  if (a > b) return 1;
  if (b > a) return -1;
  return 0;
}
//В метод sort() можна передати функцію-компаратор
//для визначення порядку сортування
numbers.sort(compare);

console.log(numbers);
```

```
▶ (7) [-3, 1, 2, 4, 5, 8, 9]
```

*16) Метод **find и findIndex***

Представьте, что у нас есть массив объектов. Как нам найти объект с определённым условием?

Здесь пригодится метод **arr.find**.

Его синтаксис таков:

```
let result = arr.find(function(item, index, array) {
  // если true - возвращается текущий элемент и перебор прерывается
  // если все итерации оказались ложными, возвращается undefined
});
```

Функция вызывается по очереди для каждого элемента массива:

- **item** – очередной элемент.
- **index** – его индекс.
- **array** – сам массив.

Если функция возвращает **true**, поиск прерывается и возвращается **item**. Если ничего не найдено, возвращается **undefined**.

Например, у нас есть массив пользователей, каждый из которых имеет поля **id** и **name**. Попробуем найти того, кто с **id == 1**:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];
let user = users.find(item => item.id == 1);
alert(user.name); // Вася
```

*17) Метод **filter***

Метод **find** ищет один (первый попавшийся) элемент, на котором функция-колбэк вернёт **true**.

На тот случай, если найденных элементов может быть много, предусмотрен метод arr.filter(fn).

Синтаксис этого метода схож с **find**, но **filter** возвращает массив из всех подходящих элементов:

```
let results = arr.filter(function(item, index, array) {
  // если true - элемент добавляется к результату, и перебор продолжается
  // возвращается пустой массив в случае, если ничего не найдено
});
```

Например:

```
let users = [
  {id: 1, name: "Вася"},
  {id: 2, name: "Петя"},
  {id: 3, name: "Маша"}
];


// возвращает массив, состоящий из двух первых пользователей
let someUsers = users.filter(item => item.id < 3);


alert(someUsers.length); // 2
```

*18) Метод **reverse***

Метод **arr.reverse** меняет порядок элементов в **arr** на обратный.

Например:

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

arr.reverse();

alert( arr ); // 5,4,3,2,1
```

Он также возвращает массив **arr** с изменённым порядком элементов.

*19) Метод **str.split(delim)***

Он разбивает строку на массив по заданному разделителю **delim**.

В примере ниже таким разделителем является строка из запятой и пробела.

```
let names = 'Вася, Петя, Маша';
let arr = names.split(', ');
for (let name of arr) {
  alert( `Сообщение получат: ${name}.` ); // Сообщение получат: Вася (и другие имена)
}
```

У метода **split** есть необязательный второй числовой аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано, то остаток массива будет отброшен. На практике это редко используется:

```
let arr = 'Вася, Петя, Маша, Саша'.split(', ', 2);
alert(arr); // Вася, Петя
```

Разбивка по буквам

Вызов **split(s)** с пустым аргументом s разбил бы строку на массив букв:

```
let str = "тест";

alert( str.split('') ); // т,е,с,т
```

Итого

Шпаргалка по методам массива:

- Для добавления/удаления элементов:
  - **push (...items)** – добавляет элементы в конец,
  - **pop()** – извлекает элемент с конца,
  - **shift()** – извлекает элемент с начала,
  - **unshift(...items)** – добавляет элементы в начало.
  - **splice(pos, deleteCount, ...items)** – начиная с индекса **pos**, удаляет **deleteCount** элементов и вставляет **items**.
  - **slice(start, end)** – создаёт новый массив, копируя в него элементы с позиции **start** до **end** (не включая **end**).
  - **concat(...items)** – возвращает новый массив: копирует все члены текущего массива и добавляет к нему **items**. Если какой-то из **items** является массивом, тогда берутся его элементы.
- ***Для поиска среди элементов:***
  - **indexOf/lastIndexOf(item, pos)** – ищет **item**, начиная с позиции **pos**, и возвращает его индекс или **-1**, если ничего не найдено.
  - **includes(value)** – возвращает **true**, если в массиве имеется элемент **value**, в противном случае **false**.
  - **find/filter(func)** – фильтрует элементы через функцию и отдаёт первое/все значения, при прохождении которых через функцию возвращается ***true***.
  - **findIndex** похож на **find**, но возвращает индекс вместо значения.
- Для перебора элементов:
  - **forEach(func)** – вызывает **func** для каждого элемента. Ничего не возвращает.
- Для преобразования массива:
  - **map(func)** – создаёт новый массив из результатов вызова **func** для каждого элемента.
  - **sort(func)** – сортирует массив «на месте», а потом возвращает его.
  - **reverse()** – «на месте» меняет порядок следования элементов на противоположный и возвращает изменённый массив.
  - **split/join** – преобразует строку в массив и обратно.
  - **reduce(func, initial)** – вычисляет одно значение на основе всего массива, вызывая **func** для каждого элемента и передавая промежуточный результат между вызовами.
- ***Дополнительно***:
  - **Array.isArray(arr)** проверяет, является ли **arr** массивом.

Обратите внимание, что методы **sort**, **reverse** и **splice** изменяют исходный массив.