МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТІТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт та самостійної роботи студентів спеціальності 122 «Комп'ютерні науки» з курсу «Інтернет технології комп'ютерної графіки та анімації»

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від17.05.2019

Харків 2021

Методичні вказівки до виконання лабораторних робіт та самостійної роботи студентів спеціальності 122 «Комп'ютерні науки» з курсу «Інтернет технології комп'ютерної графіки та анімації»/ Уклад. О. Г. Сімонова, І. Б. Шеліхова. Харків: «НТМТ», 2021. – 28 с.

Укладачі: Ольга Сімонова,
Інеса Шеліхова

Рецензент

Олександр Ніцин, д-р техн. наук, проф. кафедри «Геометричного моделювання та компютерної графіки» Національного техничного університету «ХПІ»

Кафедра геометричного моделювання та комп'ютерної графіки

ВСТУП

Для отримання і відправки даних в мережі Internet використовуються формати даних JSON і XML.

JSON – простий текстовий формат обміну даними, заснований на мові програмування JavaScript. Як і багато інших текстові формати JSON легко читається людьми.

XML — мова розмітки, який визначає набір правил для кодування документів в форматі, який читається людиною і читається машиною. В XML застосовуються теги, що задаються користувачами. Саме тому, що кожен користувач здатний і вільний створити розмітку, яка йому потрібна в конкретній ситуації, мова і називається розширюваним. За структурою цей тип документа складається з дерева елементів, де ці елементи мають вміст і атрибути. За рахунок того, що файл містить текстову інформацію, він редагується в багатьох текстових редакторах.

Документи в форматах HTML і XML містять дані, укладені в теги, але на цьому схожість між двома мовами закінчується. У форматі HTML теги визначають оформлення даних – розташування заголовків, початок абзацу і т. д. У форматі XML теги визначають структуру і зміст даних – те, чим вони є.

При описі структури та змісту даних стає можливим їх повторне використання декількома способами. Можна використовувати одну систему для генерації даних та позначки їх тегами в форматі XML, а потім обробляти ці дані в будь-яких інших системах незалежно від клієнтської платформи або операційної системи. Завдяки такій сумісності XML є основою однієї з найпопулярніших технологій

Але, JSON має ряд переваг:

• XML вимагає відкриття та закриття тегів, а JSON використовує пари ім’я/значення, чітко позначені «{*″i″*}» для об’єктів, «[*″i″*]» для масивів, «,» (кому) для поділу пари і «:» (двокрапка) для відділення імені від значення;

• при однаковому обсязі інформації JSON майже завжди значно менше, що призводить до більш швидкої передачі і обробці;

• JSON є підмножиною JavaScript, тому код для його аналізу та упаковки цілком природно вписується в код JavaScript.

Вважається, що формат JSON був розроблений американським програмістом Дугласом Крокфордом, який відомий як постійний учасник розвитку мови JavaScript, творець текстового формату обміну даними JSON і автор різних пов'язаних з JavaScript інструментів. Дуглас Крокфорд зробив цей формат популярним в 2001 році.

За іншою версією, формат не винайшли, а просто «відкрили». У презентації розробникам з «Yahoo!» Крокфорд розкрив секрет про те, що JSON використовувався ще раніше в браузері Netscape ще в 1996 році.

Мета роботи: набути навичок створення програм на мові JavaScript з використанням формату JSON і методів JavaScript для роботи з JSON.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Формат JSON

JSON (JavaScript Object Notation) – це загальний формат для представлення значень і об'єктів. Його опис задокументовано в стандарті RFC 4627. Спочатку він був створений для JavaScript, але в теперішній час багато інших мов також мають бібліотеки, які можуть працювати з ним. Таким чином, JSON легко використовувати для обміну даними, коли клієнт використовує JavaScript, а сервер написаний на Ruby, PHP, Python, Java або будь-якому іншому мовою, наділених функціями і спеціальними інструментами для читання і редагування файлу JSON.

Загальні приклади використання JSON:

- збереження даних;
- генерація структур даних з призначеного для користувача введення;
- передача даних із сервера на клієнт, з клієнта на сервер і з сервера на сервер;
- конфігурація і перевірка даних.

Припустимо, є складний об'єкт, і потрібно перетворити його в рядок, щоб відправити в мережі або просто вивести для перевірки. Такий рядок повинен включати в себе всі важливі властивості.

Можна реалізувати перетворення наступним чином:

```
let user = {
  name: "John",
  age: 30,

  toString() {
    return `{name: "${this.name}", age: ${this.age}}`;
  }
};

alert(user); // {name: "John", age: 30}
```

Але в процесі розробки додаються нові властивості, старі властивості перейменовуються і видаляються. Оновлення такого toString кожного разу може стати проблемою. Ми могли б спробувати перебрати властивості в ньому, але якщо об'єкт складний, і в його властивостях містить додаткові об'єкти? Ми повинні були б здійснити їх перетворення теж. Однак немає необхідності писати код для обробки всього цього. У завдання є просте рішення – використати формат JSON.

У JSON, на відміну від XML, дані організовані як в об'єкті JavaScript. Але JSON – це не об'єкт, а рядок. При цьому не будь-який об'єкт JavaScript може бути переведений один до одного в JSON. Наприклад, якщо в об'єкта є методи, то вони при перетворенні в рядок JSON будуть проігноровані і не включені в неї.

Структура рядка JSON практично нічим не відрізняється від запису JavaScript об'єкта. Вона складається з набору пар ключ-значення. У цій парі ключ відділяється від значення за допомогою знака двокрапки (:), а одна пара від іншого – за допомогою коми (,). При цьому ключ в JSON, на відміну від об'єкта обов'язково повинен бути укладений в подвійні лапки. Це перша відмінність JSON від JavaScript об'єкта. В об'єкті ключ (ім'я властивості) не обов'язково слід укладати в подвійні лапки.

Наприклад:

```
// Рядок JSON
var jsonPerson = {"name":"Іван","age":25};
```

```
// Об'єкт JavaScript
var person = {
  name: "Іван",
  age: 25
};
```

Щоб не ускладнювати доступ до даних, при завданні ключам імен краще дотримуватися тих же правил, що і при іменуванні змінних.

Друга відмінність полягає в тому, що значення ключа в JSON можна задати тільки в одному з таких форматів: string (рядком), number (числом), object (об'єктом), array (масивом), boolean (логічним значенням true або false) або null (спеціальним значенням JavaScript). Наприклад, значення ключа в JSON не може бути функцією або датою (об'єктом типу Date).

```
// Об'єкт JavaScript
var person = {
  name: "Іван",
  birthDay: new Date(Date.UTC(1985, 03, 05)),
  getAge: function () {
    var ageDate = new Date(Date.now() - this.birthDay.getTime());
    return Math.abs(ageDate.getUTCFullYear() - 1970);
  }
};

// Рядок JSON
var jsonPerson = {"name":"Іван","birthDay":"1985-04-
05T00:00:00.000Z"};
```

Приклад JSON рядка, що складається з різних типів даних:

```
{
  "name": "Іван",
  "age": 37,
  "mother": {
    "name": "Ольга",
    "age": 58
  },
  "children": [
    "Маша",
    "Ігор",
    "Тетяна"
  ],
  "married": true,
  "dog": null
}
```

У ній ключ «name» має в якості значення рядок, «age» — число, «mother» — об'єкт, що складається з «name» і «age», «children» — масив, «married» — логічний тип, «dog» — null.

При створенні ключів найкраще використовувати рядок з латинськими символами без пробілів, можна в Camel-синтаксисі або з нижнім підкресленням, наприклад: "companyName", "company_name", але не "company name".

Формат JSON є підмножиною синтаксису мови JavaScript, тому в якості значень в JSON можуть бути використані типи даних:

- **об'єкт** — це неврегульована безліч пар **ключ:значення**, укладену в фігурні дужки *«{ }»*. Ключ описується **рядком**, між ним і значенням стоїть символ *«:»*. Пари *ключ-значення* відокремлюються один від одного комами;
- **масив** (одновимірний) — це впорядкована множина значень. Масив полягає в квадратні дужки *«[ ]»*. Значення розділяються комами. Масив може бути порожнім, тобто не містити жодного значення.
- **число** (ціле або дійсне);
- **літерали** *true* ("істина"), *false* («хибність») і *null*;
- **рядок** — це впорядкована множина з нуля або більше символів юнікода, укладену в подвійні лапки. Символи можуть бути вказані з використанням escape-послідовностей, що починаються з зворотної косої межі *«\»* (підтримуються варіанти *\ ', \ ", \\, \ /, \ t, \ n, \ r, \ f і \ b)*, або записані шістнадцятковим кодом в кодуванні Unicode у вигляді *\uFFFF*.

Приклад синтаксису формату JSON:

```
"employees":[
{"firstName":"Lev", "lastName":"Tolstoy"},
{"firstName":"Anna", "lastName":"Karenina"},
{"firstName":"Aleksey", "lastName":"Vronsky"}
]
```

У наведеному прикладі об’єкт "employees" являє собою масив, що містить три об’єкти. Кожен об’єкт являє собою запис людини (з ім’ям і прізвищем).

Виконання синтаксичних правил JSON:

- дані вказуються в парах *ім’я/значення*, що розділяються двокрапкою "firstName": "Lev";
- дані розділяються комами
  "firstName": "Anna", "lastName": "Karenina";
- фігурні дужки визначають об’єкти
  {"firstName": "Lev", "lastName": "Tolstoy"};
- квадратні дужки визначають масиви.

При звертанні до елементів формату JSON необхідно ставити крапку між назвою об’єкту і ключем, наприклад:

```
employees.firstName;
```

Перший запис масиву об’єктів JavaScript можна отримати наступним чином:

```
// returns Lev Tolstoy
employees[0].firstName + " " + employees[0].lastName;
```

або

```
employees[0].["firstName"] + " " + employees[0].["lastName"];
```

Для перебору елементів масиву можна використовувати наступний цикл:

```
for (var i=0; i<=employees.length-1; i++) {
   //i = индекс
   //значение = employees[i].firstName+ " " + employees[i].lastName
   console.log("Элемент [ "+ i +" ] = " + employees[i].firstName+ " " +
                                                    employees[i].lastName);
 }
```

JSON простий для розуміння і використання, він є дуже корисним і гнучким інструментом для передачі даних між додатками і комп'ютерами.

Особливості використання JSON

- Тип файлу для JSON-файлів ".json".
- JSON — це чисто формат даних — він містить тільки властивості, без методів.
- JSON вимагає **подвійних лапок**, які будуть використовуватися навколо рядків і імен властивостей. Одиничні лапки недійсні.
- Навіть одна недоречна кома або двокрапка можуть привести до збою JSON-файлу і не працювати. Можна перевірити JSON за допомогою програми JSONLint.
- JSON може приймати форму будь-якого типу даних, допустимого для включення в JSON, а не тільки масивів або об'єктів. Так, наприклад, один рядок або номер будуть дійсним об'єктом JSON.
- На відміну від коду JavaScript, в якому властивості об'єкта можуть не укладатися в подвійні лапки, в JSON в якості властивостей можуть використовуватися тільки рядки укладені в подвійні лапки.
- JSON не підтримує коментарі. Додавання коментаря в JSON робить його недійсним.

Посилання на Інтернет-сервіс, за допомогою якого можна легко створювати JSON файли:

http://www.jsoneditoronline.org/

Перевага в використанні цього варіанта в тому, що сервіс вказує на всі синтаксичні помилки, які можна допустити.

Методи JavaScript для роботи з JSON

JavaScript має вбудовані методи для роботи з JSON:

- **JSON.stringify** для перетворення об'єктів JavaScript в рядок JSON.
- **JSON.parse** для перетворення JSON назад в об'єкт JavaScript.

Метод JSON.stringify()

Метод **JSON.stringify()** перетворює значення JavaScript в рядок JSON.

Особливості перетворення:

- Значення *undefined*, функція або значення типу *symbol*, зустрінуті під час перетворення, будуть або опущені (якщо вони знайдені в об'єкті), або перетворені в *null* (якщо вони знайдені в масиві).
- Члени прототипу і неперелічувані властивості ігноруються.

Синтаксис

```
JSON.stringify(value[, replacer[, space]])
```

Параметри:

value – значення, яке буде перетворено в рядок JSON.

replacer (необов'язковий) – значенням параметра може бути функція, масив або *null*.

Масив визначає набір властивостей об'єкта, які будуть включені в JSON-рядок. Як значення масиву вказуються рядки, відповідні іменам властивостей об'єкта, що перетворюється.

Функція дозволяє замінити значення властивостей об’єкта в JSON-рядку. Функція повинна містити два параметри: ім’я властивості і значення властивості. Ім’я властивості є рядком. Функція повинна повертати нове значення властивості.

Якщо значенням параметра є null, то JSON-рядок буде включати всі властивості об’єкта.

space (необов’язковий) — робить JSON-рядок більш зрозумілим, додаючи відступу для кожного рівня вкладеності. Значенням параметра може бути рядок або число.

Число вказує кількість пробілів, які використовуються в якості відступу для кожного рівня вкладеності. Кожен наступний рівень вкладеності доповнюється новими відступами. Наприклад, якщо в якості значення параметра використовується число 2, то на першому рівні вкладеності відступ становитиме два пробілу, на наступному рівні вкладеності відступ становитиме 4 пробілу і т. д. Максимальна кількість пробілів, яке можна зазначити – 10. Якщо вказати більшу кількість , воно автоматично зменшиться до 10.

Рядок визначає символ, який використовується в якості відступу для кожного рівня вкладеності. Довжина рядка обмежена 10 символами, якщо вказати рядок довший, він обрізається до 10 символів. Використання рядка також дозволяє використовувати табуляцію ("\ t") в якості відступу. Кожен наступний рівень вкладеності доповнюється новими символами відступу. Наприклад, якщо в якості значення параметра вказано символ — (дефіс), то на першому рівні вкладеності як відступу буде використовуватися один дефіс, на наступному рівні вкладеності буде використовуватися 2 дефіса і т. д.

Повертаєме значення – JSON-рядок.

Приклади використання в JSON.stringify() необов'язкових параметрів

```
let person = {
  name: "Гомер",
  age: 40,
  work: {
    place: "Атомна станція",
    location: "Спрінгфілд"
  }
}
```

```
// Приклад з одним параметром
console.log(JSON.stringify(person));   //
'{"name":"Гомер","age":40,"work":{"place":"Атомна
станція","location":"Спрінгфілд"}}'
```

```
// Приклад з двома параметрами (масив)
console.log(JSON.stringify(person, ["name", "age"]));
                                                    // '{"name":"Гомер","age":40}'
```

```
// Приклад з двома параметрами (функція)
console.log(JSON.stringify(person, function (key, value) {
  switch (key) {
    case "name":
      return "Барт";
    case "age":
      return 10;
    case "work":
      return undefined;
    default:
      return value;
  }
}));
        // '{"name":"Барт","age":10}'
```

Наведені вище JSON-відформатовані об’єкти не мали відступів і зайвих пробілів. Це нормально, якщо необхідно відправити об’єкт по мережі. Аргумент ***space*** використовується виключно для виведення в зрозумілому людині вигляді.

В наступному прикладі значення параметру space = 2 вказує JavaScript відображати вкладені об’єкти в кілька рядків з відступом в 2 пробіли всередині об’єкта:

```
// Приклад з трьома параметрами
console.log(JSON.stringify(person, null, 2));
/* {
                                    *   "name": "Гомер",
                                    *   "age": 40,
                                    *   "work": {
                                    *     "place": "Атомна станція",
                                    *     "location": "Спрінгфілд"
                                    *   }
                                    * }
                                    */
```

Для JSON.stringify(person, null, 4) результат містить більше відступів.

Отриманий рядок json називається *JSON-форматованим,* **або серіалізованим об’єктом**. Ми можемо відправити його по мережі або помістити в звичайне сховище даних.

Серіалізацією називається перетворення рідного об’єкта в рядок таким чином, щоб він міг бути переданий в мережу, а перетворення рядка в рідній об’єкт називається **десеріалізацією**.

Зверніть увагу, що об’єкт в форматі JSON має кілька важливих відмінностей від об’єктного літерала:

- Рядки використовують подвійні лапки. Ніяких одинарних лапок або зворотних лапок в JSON. Так ‘John’ стає "John".
- Імена властивостей об’єкта також полягають в подвійні лапки. Це обов’язково. Так age: 30 стає "age": 30.

Метод toJSON

Як і toString для перетворення рядків, об'єкт JSON може надавати метод toJSON для перетворення в JSON. Якщо перетворений в рядок об'єкт має властивість з ім'ям toJSON і значенням властивості, встановленим в функцію, то цей метод toJSON() змінить стандартну поведінку перетворення в JSON: замість перетворення об'єкта буде використовуватися значення, що повертається методом toJSON(). JSON.stringify автоматично викликає його, якщо він є.

Так, примірники об'єкта Date посилаються на певний момент часу. Виклик методу toJSON() поверне рядок, відформатований в JSON (за допомогою методу toISOString ()), що представляє значення об'єкта Date. Цей метод, як правило, призначений для серіалізації об'єктів Date в JSON.

Приклад:

```
var jsonDate = (new Date()).toJSON();
var backToDate = new Date(jsonDate);
console.log('Сериализованный объект даты: ' + jsonDate);
```

Ще один приклад:

```
let room = {
  number: 23
};

let meetup = {
  title: "Conference",
  date: new Date(Date.UTC(2017, 0, 1)),
  room
};

alert( JSON.stringify(meetup) );
```

```
/*
  {
    "title":"Conference",
    "date":"2017-01-01T00:00:00.000Z",  // (1)
    "room": {"number":23}               // (2)
  }
*/
```

Як бачимо, date (1) став рядком. Це тому, що всі об’єкти мають вбудований метод toJSON, який повертає такий рядок.

Тепер додамо власну реалізацію методу toJSON в наш об’єкт room (2):

```
let room = {
  number: 23,
  toJSON() {
    return this.number;
  }
};

let meetup = {
  title: "Conference",
  room
};

alert( JSON.stringify(room) ); // 23

alert( JSON.stringify(meetup) );
/*
  {
    "title":"Conference",
    "room": 23
  }
*/
```

Як бачимо, toJSON використовується як при прямому виклику JSON.stringify (room), так і коли room вкладений в інший серіалізований об'єкт.

Метод JSON.parse()

Парсинг (Parsing) – це прийняте в інформатиці визначення синтаксичного аналізу. Для цього створюється математична модель порівняння лексем з формальної граматикою, описана одним з мов програмування.

Програма (скрипт), що дає можливість комп'ютеру «читати» — порівнювати запропоновані слова з наявними у Всесвітній мережі, називається **парсером**.

Так парсером в JavaScript є глобальна функція **parse()** об'єкта JSON, призначена для аналізу (парсингу) рядків у форматі JSON. При необхідності перетворює і повертає значення, отримані в ході аналізу.

Метод **JSON.parse()** перетворює рядок JSON у відповідне значення JavaScript і має деякі особливості перетворення: якщо підчас перетворення в рядку JSON зустрінеться значення undefined, то воно буде опущено (не буде включено в результат).

Синтаксис

```
JSON.parse(str[, reviver])
```

Параметри:

str — JSON-рядок, який буде перетворений у відповідне значення JavaScript.

reviver (необов'язковий) — значенням параметра повинна бути функція. Функція дозволяє замінити значення властивості JSON-рядка перед його поверненням. Як і JSON.stringify(), метод JSON.parse() може бути викликаний з другим необов'язковим параметром reviver, який являє собою виклик

функції виду function (key, value), послідовно перебирає всі значення JSON-рядків. Функція повинна містити два параметри: key — ім'я властивості і value — значення властивості. Ім'я властивості є рядком. Функція повинна повертати нове значення властивості.

У функції можна за якоюсь умовою впорядкувати значення ключів або самі ключі і вивести трохи змінений варіант JSON-даних. Важливим моментом тут є те, що функція в кінці перебирає всі рядки, тому бажано в функції відстежувати ключ у вигляді порожнього рядка (key === ''), щоб уникнути неприємностей:

Повертаєме значення

Відповідне значення JavaScript (примітивне значення, об'єкт або масив).

Приклади

```
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

```
// Рядковий масив
let numbers = "[0, 1, 2, 3]";
numbers = JSON.parse(numbers);
alert( numbers[1] ); // 1
```

```
// Для вкладених об'єктів
let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends":
[0,1,2,3] }';
user = JSON.parse(user);
alert( user.friends[1] ); // 1
```

JSON може бути настільки складним, наскільки це необхідно, об’єкти і масиви можуть включати інші об’єкти і масиви. Але вони повинні бути також в форматі JSON.

Ось типові помилки в написаному від руки JSON (іноді доводиться писати його для налагодження):

```
let json = `{
  name: "John",                // Помилка: ім’я властивості без лапок
 "surname": 'Smith',           // Помилка: одинарні лапки в значенні
                               // (повинні бути подвійними)
 'isAdmin': false              // Помилка: одинарні лапки в ключі
                               // (повинні бути подвійними)
 "birthday": new Date(2000, 2, 3), // Помилка: не допускається
                                   // конструктор "new", тільки значення.
 "friends": [0,1,2,3]          // Тут все в порядку
}`;
```

Крім того, JSON не підтримує коментарі. Додавання коментаря в JSON робить його недійсним.

Робота з даними JSON після парсинга

Робота з даними JSON після парсинга здійснюється як з об’єктом JavaScript.

```
//JSON
var personData =
'{"name":"Іван","age":37,"mother":{"name":"Ольга","age":58},"children":
["Маша","Ігор","Тетяна"],"married": true,"dog": null}';

//Объект JavaScript person
var person = JSON.parse(personData);
```

Види робіт з даними JSON після парсинга:

```
//  Отримати значення ключа (властивості) name
person.name;
person["name"];

// Отримати значення ключа (властивості) name, що знаходиться
в // об’єкті mother
person.mother.name;

//Видалити елемент age
delete(person.age)

//Додати (або оновити) ключ (властивість)
person.eye = "карії";
```

При роботі з масивами необхідно використовувати методи, призначені для роботи саме з масивами

```
// Видалити 1 елемент з масива (метод splice)
person.children.splice(1,1)
// Додати елемент в масив (метод push)
person.children.push("Катя");
```

Висновки

- JSON Робота з даними JSON після парсинга здійснюється як з об’єктом JavaScript. це формат даних, який має власний незалежний стандарт і бібліотеки для більшості мов програмування.
- JSON є природним форматом для використання в JavaScript і має багато реалізацій, доступних для використання в багатьох популярних мовах програмування.
- JSON простий для розуміння і використання, він є дуже корисним і гнучким інструментом для передачі даних між додатками і комп’ютерами.
- JSON підтримує прості об’єкти, масиви, рядки, числа, логічні значення і null.
- JavaScript надає методи JSON.stringify для серіалізації в JSON і JSON.parse для читання з JSON.
- Обидва методи підтримують функції перетворення для інтелектуального читання/запису.
- Якщо об’єкт має метод toJSON, то він викликається через JSON.stringify.

Контрольні запитання

1. Що таке JSON і для чого він призначений?
2. Які типи даних можна використовувати в JSON?
3. Які особливості використання JSON?
4. Які методи надає JavaScript для роботи з JSON?
5. Які особливості перетворення значення JavaScript в рядок JSON?
6. Скільки параметрів має метод JSON.stringify() і які?
7. Що таке серіалізація в JSON?
8. Які параметри методу JSON.stringify() обов'язкові і які необов'язкові?
9. Які значення може приймати параметр *space* методу JSON.stringify()?
10. У яких випадках використовується метод toJSON?
11. Що таке парсинг і парсер?
12. Що таке десеріалізація в JSON?
13. Скільки параметрів має метод JSON.parse()?
14. Які значення може приймати параметр *reviver* методу JSON.parse()?
15. Які види робіт з даними JSON можуть відбуватися після парсинга?

2 ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Завдання 1. Перетворити об’єкт в JSON, а потім назад в звичайний об’єкт, використовуючи типи даних: запис. рядок, число, масив. Наприклад, перетворіть *user* в JSON, потім прочитайте цей JSON в іншу змінну.

```
let user = {
  name: "Василь Іванович",
  age: 35
};
```

Розв’язання

```
let user = {
  name: "Василь Іванович",
  age: 35
};

let user2 = JSON.parse(JSON.stringify(user));
```

Опис виконання завдання внести в загальний звіт про роботу.

Завдання 2. Вивчити можливість виключення циклічних посилань

У простих випадках циклічних посилань ми можемо виключити властивість, через яку вони виникають, з серіалізації за його ім'ям.

Але іноді ми не можемо використовувати ім'я, оскільки можуть бути і інші, потрібні властивості з цим ім'ям у вкладених об'єктах. Тому можна перевіряти властивість за значенням.

Напишіть функцію *replacer* для JSON-перетворення, яке видалить властивості, що посилаються на *meetup*:

```
let room = {
  number: 23
};

let meetup = {
  title: "  Нарада ",
  occupiedBy: [{name: "Іванов"}, {name: "Петров"}],
  place: room
};

// циклічні посилання
room.occupiedBy = meetup;
meetup.self = meetup;

alert( JSON.stringify(meetup, function replacer(key, value) {
  /* ваш код */
}));

/* в результаті повинно бути:
{
  "title":"Нарада",
  "occupiedBy":[{"name":"Іванов"},{"name":"Петров"}],
```

```
  "place":{"number":23}
}
*/
```

Розв'язання

```
let room = {
  number: 23
};

let meetup = {
  title: "Нарада",
  occupiedBy: [{name: "Іванов"}, {name: "Петров"}],
  place: room
};

room.occupiedBy = meetup;
meetup.self = meetup;

alert( JSON.stringify(meetup, function replacer(key, value) {
  return (key != "" && value == meetup) ? undefined : value;
}));

/*
{
  "title":" Нарада ",
  "occupiedBy":[{"name":"Іванов"},{"name":"Петров"}],
  "place":{"number":23}
}
*/
```

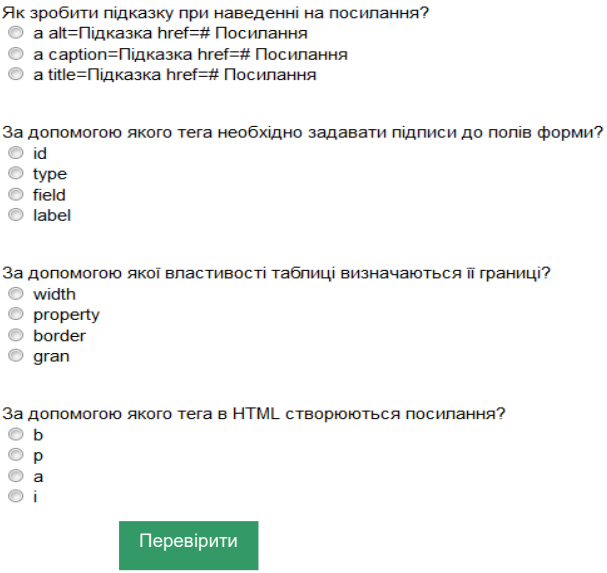
Тут нам також потрібно перевірити *key == ""*, щоб виключити перший виклик, де значення *value* равно *meetup*.

Опис виконання завдання внести в загальний звіт про роботу.

Завдання 3. Створити тест з використанням JSON

Для виконання завдання 3 необхідно спочатку продумати алгоритм виконання завдання та сформувати необхідні файли.

Сформувати сторінку з тестовими завданнями та варіантами відповідей у вигляді елементів RadioButton. Для формування питань та варіантів відповідей необхідно використати файл test.json. Початковий вигляд сторінки повинен мати наступний вигляд:



Реалізувати кнопку «Перевірити», при натисканні на яку підраховується та виводиться кількість вірних відповідей, що надав користувач. Правильні відповіді виділити зеленим кольором, помилкові — червоним.

Після завершення виконання програмної частини всіх завдань необхідно скласти загальний звіт з докладним описом кожного завдання. Програмну частину необхідно супроводжувати докладними коментарями.

Навчальне видання

СІМОНОВА Ольга Геннадіївна
ШЕЛІХОВА Інеса Борисівна

Методичні вказівки до виконання лабораторних робіт та самостійної роботи студентів спеціальності 122 «Комп'ютерні науки» з курсу «Інтернет технології комп'ютерної графіки та анімації»

В авторській редакції

Роботу до видання рекомендувала проф. Ольга Шоман

План 2019 р., поз. 160

Підп. до друку 17.09.2020 Формат 60x84 1/16. Папір офісний.
Друк цифровий. Гарнітура Times New Roman. Ум. друк. арк.1,75
Наклад 50 прим. Зам. № 21/05-02 Ціна договірна.

Видавництво ТОВ фірма «НТМТ»
Свідоцтво внесення до державну реєстру видавців ДК № 1748 від 15.04.2004 р.
61103, м. Харків, вул. Дерев'янка, 16

Самостійне електронне видання