Week 20  
1. let menuJson = `[{

"composition": "meat",

"weight": 300,

"price": 10

},

{

"composition": "potato",

"weight": 200,

"price": 5

},

{

"composition": "salad",

"weight": 300,

"price": 8

}]`

console.log(menuJson);

let menu = JSON.parse(menuJson);

console.log(menu);

let staffJson = `[{

"waiter": "Danny",

"chef": "Ralf",

"cleaner": "Endy"

},

{

"waiter": "Dina",

"chef": "Rosa",

"cleaner": "Endry"

},

{

"waiter": "Mila",

"chef": "Donald",

"cleaner": "Maya"

}]`

console.log(staffJson);

let staff = JSON.parse(staffJson);

console.log(staff);

let flatJson = `[{

"habitant": [

"mother",

"father",

"son",

"daughter"

],

"facilities": {

"square":82,

"floor": 2,

"view": "garden",

"room\_number": 4 },

"dog": null,

"neighbour": {

"name": "Jone",

"age": 28,

"character": "calm"

}

},

{

"habitant": [

"brother",

"sister"

],

"facilities": {

"square":43,

"floor": 3,

"view": "garden",

"room\_number": 2 },

"dog": true,

"neighbour": {

"name": "Jane",

"age": 23,

"character": "nervous"

}

}]`

console.log(flatJson);

let flat = JSON.parse(flatJson);

console.log(flat);

2. JSON:   
Легко писать и манипулировать.   
Поддерживает все фреймворки JavaScript, а также все браузеры.   
Потребляет меньше памяти.   
Открытый исходный код и бесплатное использование  
  
XML:   
Он более эффективно обрабатывает сложные структуры данных.  
XML описывает данные на языке разметки.  
Он имеет возможность управлять данными в древовидной структуре, имеющей только один корневой элемент.

JSON — это нотация объектов JavaScript для форматирования данных, тогда как XML — это язык разметки. Ключевое различие между JSON и XML заключается в том, что JSON имеет меньший размер файла и эффективно передает данные в Интернет по сравнению с XML. JSON обращается к данным через объекты JSON, тогда как XML требует анализа данных. JSON легко читается, поскольку имеет более организованную структуру кода. С другой стороны, XML трудно интерпретировать из-за его сложной структуры.  
Хотя JSON и XML нельзя полностью сопоставить. JSON хорош для передачи данных, поскольку он не требует обработки, тогда как XML может быть сложным, поскольку он позволяет не только передавать данные, но также обрабатывать и форматировать файлы. JSON наименее безопасен в отличие от XML. JSON не может поддерживать пространства имен, но способен полностью поддерживать инструментарий Ajax. Напротив, XML поддерживает пространства имен, но не может поддерживать инструменты Ajax.  
Более того, XML хранит данные иначе, чем JSON. Поскольку JSON хранит данные как карту, XML, напротив, хранит данные как древовидную структуру. Кроме того, JSON использует массивы, но не имеет закрывающих тегов.

dogJson = `[{  
"Breed": "Beagle",  
"Size": "large",  
"Color": "orange",  
"Age": "6 years"  
}]`

<dog>

<breed> Beagle </breed>

<size> large </size>

<color> orange </color>

<age> 6 years </age>

</dog>

3.   
Сериализация: процесс преобразования структуры данных или объекта в двоичную строку.  
Если нам нужно отправить какие-то данные на сервер, то мы должны сначала упаковать их в строку JSON-формата (сериализовать) и только потом отправлять.

Для сериализации объекта javascript в json применяется функция \*\*JSON.stringify()\*\*  
  
Десериализация: процесс преобразования двоичной строки, созданной в процессе сериализации, в структуру данных или объект.  
Для обратной операции - десериализации или парсинга json-объекта в javascript применяется метод JSON.parse().

Проще говоря, сериализация - это процесс преобразования объекта в байтовый поток, а десериализация - это процесс восстановления байтового потока в объект.

4. Мы можем выбрать только один метод чтения ответа.

Если мы уже получили ответ с response.text(), тогда response.json() не сработает, так как данные уже были обработаны.

let text = await response.text(); // тело ответа обработано

let parsed = await response.json(); // ошибка (данные уже были обработаны)

5. Понятие асинхронности довольно сложное. Но если коротко - это возможность JS выполнять параллельно несколько операций (а не строго друг за другом). Асинхронность позволяет сложным интерфейсам работать быстро, без длительного ожидания выполнения каждого действия по очереди и без постоянных перезагрузок страницы.  
Асинхронные запросы происходят параллельно с работой других функций, при этом скорость обработки информации не снижается.  
  
6. AJAX означает асинхронный JavaScript и позволяет асинхронно обновлять веб-страницы, обмениваясь данными с веб-сервером во время работы приложения. Короче говоря, это означает, что вы можете обновлять части веб-страницы без перезагрузки всей страницы (их URL-адрес остается прежним). Это значит, что наша страничка будет работать гораздо быстрее - ведь перегружаем только ее кусочек, а не всю целиком.

При использовании AJAX нет необходимости обновлять каждый раз всю страницу, так как обновляется только ее конкретная часть. Это намного удобнее, так как не приходится долго ждать, и экономичнее, так как не все обладают безлимитным интернетом. Правда в этом случае, разработчику необходимо следить, чтобы пользователь был в курсе того, что происходит на странице. Это можно реализовать с использованием индикаторов загрузки, текстовых сообщений о том, что идёт обмен данными с сервером. Необходимо также понимать, что не все браузеры поддерживают AJAX (старые версии браузеров и текстовые браузеры). Плюс Javascript может быть отключен пользователем. Поэтому, не следует злоупотреблять использованием технологии и прибегать к альтернативным методам представления информации на Web-сайте.

Обобщим достоинства AJAX:

* Возможность создания удобного Web-интерфейса
* Активное взаимодействие с пользователем
* Частичная перезагрузка страницы, вместо полной
* Удобство использования

7. fetch("https://api.github.com/users/NataliNovi")

.then(response => response.json())

.then(user => console.log(user))

.catch(error => console.log(error));

8. Single Page Application – сокращенно SPA, в переводе на русский язык означает “Приложение одной страницы”. Другими словами SPA – это web-приложение, размещенное на одной web-странице, которая для обеспечения работы загружает весь необходимый код вместе с загрузкой самой страницы. Приложение такого типа появились сравнительно недавно, с началом эры HTML5 и SPA является типичным представителем приложений на HTML5.

Как мы знаем, HTML5 это нечто иное как HTML + CSS3 + JavaScript + [несколько новых тегов]. Таким образом, SPA - это приложения написанные на языке JavaScript. И, следовательно, немного перефразировав предыдущие определение получаем:

“SPA – это web-приложение, размещенное на одной странице, которая для обеспечения работы загружает все javascript-файлы (модули, виджиты, контролы и т.д.) , а также файлы CSS вместе с загрузкой самой страницы.”

Если приложение достаточно сложное и содержит богатый функционал, как например, система электронного документооборота, то количество файлов со скриптами может достигать нескольких сотен, а то и тысяч. А “…загрузка всех скриптов…” никоим образом не означает, что при загрузке сайта будут загружены сразу все сотни и тысячи файлов со скриптами. Для решения проблемы загрузки большого количества скриптов в SPA призван [API](http://ru.wikipedia.org/wiki/API) под названием [AMD](http://ru.wikipedia.org/wiki/Asynchronous_module_definition). AMD реализует возможность загрузки скриптов по требованию. То есть, если для “главной станицы” одностраничного портала потребовалось 3 скрипта, они будут загружены стразу перед стартом программы. А если пользователь кликнул на другую страницу одностраничного портала, например, “О программе”, то принцип AMD загрузит модуль (скрипт + разметка) только перед тем как перейти на эту страницу.