JavaScript Расширенный

Этот курс об применении JavaScript в браузере.

Модуль 1. Углубленная работа с функциями (3 ак. ч.)

Ключевые термины:

- **Контекст** вызова функции - значение переменной this внутри функции в момент вызова
- Function.prototype.**apply(obj, arr)** - метод функций, при его вызове функция вызывается с нужным контекстом вызова obj и аргументами массива arr
- Function.prototype.**call(obj, a1, a2, …)** - *метод функций аналогичный apply*, но аргументы передаются не массивом, а просто перечнем
- Function.prototype.**bind(obj)** - вызов метода возвращает функцию с контекстом вызова obj
- **Замыкание** - функция первого класса (например, возвращаемая функция), в теле которой есть переменные не объявленные в самой функции
- **Рекурсия** - вызов функцией самой себя
- **Функция высшего порядка** - функция *принимающая в качестве аргументов другие функции* или возвращающая другую функцию в качестве результата
- **Каррирование** - преобразование вызова функции из fn(a, b) в fn(a)(b)

Пример методов функций

```
let car = { model: 'Kia'};

function changeColor(color){
   this.color = color
}

changeColor.apply(car, ['red']);
changeColor.call(car, 'green');
const setColor = changeColor.bind(car);
setColor('blue');

console.log(car);
```

Пример замыкания

```
function fn(){
  let i = 0;

  return function (){
     return ++i;
  }
}
const counter = fn();
console.log( counter() ); //1
console.log( counter() ); //2
console.log( counter() ); //3
```

Пример рекурсии

```
function next(n){
  if(n > 0) next(n-1);
  console.log(n);
}
next(10);
```

Пример каррирования

```
//function sum(a,b){ return a + b; }
function sum(a){
  return function (b){
    return a + b;
  };
}
console.log( sum(2)(3) ); //5
```

Модуль 2. Методы массивов (1 ак. ч.)

Ключевые термины:

- Array.prototype.**map**(fn) - возвращает новый массив на основе каждого элемента старого массива и функции fn
- Array.prototype.**filter**(fn) - возвращает новый массив, на основе элементов прежнего массива, для которых функция fn вернула true-значение
- Array.prototype.**reduce**(fn) - применяет fn к каждому элементу массива слева направо и возвращает одно результирующее значение
- Array.prototype.**find**(fn) - возвращает значение первого найденного в массиве элемента, для которого fn возвращает true-значение
- Array.prototype.**findIndex**(fn) - как и find, но возвращает индекс
- Array.prototype.**forEach**(fn) - применяет fn каждому элементу массива
- Array.prototype.**includes**(value) - возвращает true, если массив содержит value
- Array.prototype.**some**(fn) - возвращает true, если в массиве есть элемент для которого fn вернет true
- Array.prototype.**every**(fn) - как и some, но вернет true если fn вернула true для каждого элемента массива

Пример метода map

```
[1, 3, 11, 45, 23].map( function(item){
  return item * 100;
} ); // [100, 300, 1100, 4500, 2300]
```

Пример метода filter

```
[1, 1, 2, 3, 5].filter( function(item){
  return item % 2;
} ); //[1, 1, 3, 5]
```

Пример метода reduce

```
[10, 20, 30].reduce( function(prev, curr){
  return prev + curr;
} ); //60
```

Пример метода find

```
[2, 4, 7, 9].find( function( item ){
  return item % 2;
} ); //7
```

Пример метода findIndex

```
[2, 4, 7, 9].findIndex( function( item ){
  return item % 2;
} ); //2
```

Пример метода forEach

```
[2, 4, 7, 9].forEach( item => {
  console.log( item )
} );
```

Пример метода includes

```
[2, 4, 7, 9].includes( 4 ); //true
```

Пример метода some

```
[2, 4, 7, 9].some( item => item % 2); //true
```

Пример метода every

```
[2, 4, 7, 9].every( item => item % 2); //false
```

Модуль 3. Объектная модель браузера (4 ак. ч.)

Ключевые термины:

- **BOM** (Browser Object Model) - *дополнительные объекты JavaScript* предоставляемые браузером
- Объект **window** - объект BOM отвечающий за текущую вкладку в браузере
- **интерфейс** - совокупность свойств, методов и правил взаимодействия элементов системы
- **DOM** (объектная модель документа) - программный интерфейс для HTML и других документов, которая рассматривает эти документы как древовидные структуры
- Объект **document** - отвечает за работу с документом страницы
- **onclick** - название атрибута HTML-элемента. Содержит JS-код, который будет вызван, когда на этот элемент будет выполнено нажатие.
- **таймеры** - методы setTimeout, setInterval, clearTimeout, clearInterval для отложенного вызова кода
- **requestAnimationFrame** - метода объекта window для анимации
- **HTMLCollection** - коллекция элементов
- document.**querySelector**(selector) - принимает строку с селектором, а возвращает отдельный HTML-узел
- className -
- document.**querySelectorAll**(selector) - принимает селектор, а возвращает коллекцию HTML-узлов
- **NodeList** - коллекция узлов возвращаемых методами Node.childNodes и document.querySelectorAll

Пример: свойства window

```
for(let prop in window){
  console.log(prop, window[prop])
}
console.log(location)
console.log(document)
console.log(localStorage)
```

Пример интерфейса HTMLElement

```
interface HTMLElement : Element {
  [HTMLConstructor] constructor();

  // metadata attributes
  [CEReactions] attribute DOMString title;
  [CEReactions] attribute DOMString lang;
  ...
  // user interaction
  [CEReactions] attribute boolean hidden;
  undefined click();
```

```
  [CEReactions] attribute DOMString accessKey;

  ...
};
```
https://html.spec.whatwg.org/multipage/indices.html#elements-3

Пример: свойств document

```
for(let prop in document){
  console.log(prop, document[prop])
}
console.log(document.documentElement)
console.log(document.title)
console.log(document.body )
```

Пример: обработчик onclick

```
<body onload="(()=>{ console.log('loaded') })()"></body>
document.onclick =() => {  console.log('Нажатие document') }
document.body.onclick =() => {  console.log('Нажатие body') }
```

Пример таймера setTimeout

```
const fn = () => {
  console.log('Таймер сработал');
}
let t1 = setTimeout(fn, 2000);
//clearTimeout(t1) ;
```

Пример requestAnimationFrame

```
let counter = 10;
function step() {
  counter--
  console.log(counter)
  if ( counter ) {
    requestAnimationFrame(step);
  }
}
requestAnimationFrame(step);
```

Пример HTMLCollection

```
console.log(document.links)
console.log(document.links.length)
```

```
console.log(document.links.item(0))
console.log(document.links instanceof HTMLCollection)
console.log(document.links.constructor)
```

Пример document.querySelector

```
const a = document.querySelector('a')
console.log(a)
console.log(a instanceof HTMLElement)
a.style.color = 'orange'
a.href = '#'
a.innerHTML = 'Новый текст'
```

Пример document.querySelectorAll

```
const a = document.querySelectorAll('a')
console.log(a[0])
console.log(a[1])
console.log(a[2])
console.log(a.constructor)
```

Модуль 4. DOM (5 ак. ч.)

Ключевые термины:

- **DOM узел** - фрагмент DOM-документа со своими свойствами и методами
- **Тип узла** - один из доступных видов узлов в DOM
- Node.**nodeType** - числовое целое значение у DOM-узла, которое описывает тип узла (https://developer.mozilla.org/ru/docs/Web/API/Node/nodeType)
- **Связь между объектами** - свойства DOM-узлов, указывающие на другие узлы
- интерфейс **Node** - описывает свойства и методы DOM-узлов
- интерфейс **Element** - описывает св-ва и методы общие для всех (HTML) элементов
- **Обход набора/коллекции** - использование циклов JS и методов объектов для получения ссылок на отдельные элементы коллекции и работы с ними
- Node.**cloneNode**() - метод узла, при вызове возвращает клон/копию узла
- <template> - HTML-элемент для описания шаблона контента
- template.**content** - ссылка на содержимое шаблона (клонирование выполняется так - template.content.cloneNode())
- Element.**classList** - свойство представляющее собой псевдомассив со всеми классами элемента. Имеет полезные методы add, remove, item, toggle, contains
- Element.**getBoundingClientRect**() - возвращает размер элемента и его позицию относительно видимой части страницы
- document.**createElement**(tag) - метод который создаёт DOM-узел как HTML элемент по названию тега tag

Номер типа	Тип узла	Описание	Пример
1	ELEMENT_NODE	Элемент	…
3	TEXT_NODE	Текстовый узел	слово
8	COMMENT_NODE	Комментарий	<!-- комм -->
9	DOCUMENT_NODE	Узел документа	document
10	DOCUMENT_TYPE_NODE	Декларация типа	<!doctype html>
11	DOCUMENT_FRAGMENT_NODE	Фрагмент док-та	создаётся программно

Пример: DOM-узел

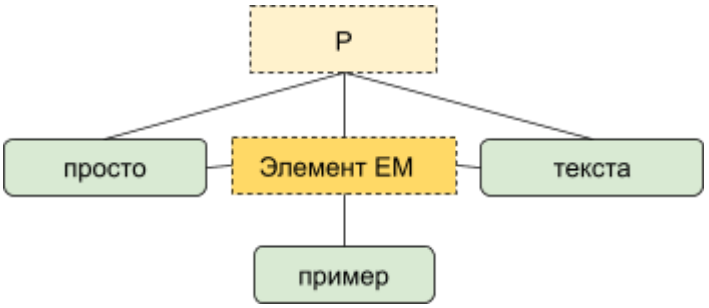
```
const a = document.querySelector('a')

for(let prop in a){ console.log(prop, a[prop]) }

console.log(a.nodeType); //1
console.log(a.nodeName); //A
console.log(a.nodeValue); //null
```

Пример: фрагмент DOM

```
<p>просто <em>пример</em> текста</p>
```



Примеры: связи DOM

let p = document.querySelector('p')

Пример связи	Название	На что ссылается на
p.childNodes	дочерние узлы	коллекцию из двух текстовых узлов и узла EM
p.firstChild	первый дочерний узел	текстовый узел “просто”
p.lastChild	последний дочерний узел	текстовый узел “текста”
p.childNodes[1]	второй дочерний элемент	узел EM
em.parentNode	родительский элемент	узел P
em.nextSibling	ссылка на следующий элемент	текстовый узел “текста”
em.previousSibling	ссылка на предыдущий элемент	текстовый узел “просто”
em.firstChild	ссылка на первый дочерний	текстовый узел “пример”

Пример: методы узлов Node

p.hasChildNodes() - *true*, если есть дочерние элементы
let clone = p.cloneNode(true) - клонирует узел
p.appendChild(ch) - добавляет в качестве дочернего элемент *ch*
p.insertBefore(el,p) - вставляет элемент *el* перед *p*
p.replaceChild(el, p) - заменяем *p* элементом *el*
p.removeChild(el) - удаляет дочерний элемент *el*

Пример: методы Element

p.getElementsByTagName('em') - получает коллекцию по названию тега
p.hasAttribute('data-test') - true, если есть атрибут *data-test*
p.getAttribute('class') - возвращает значение атрибута *class*
p.setAttribute('data-test',1) - устанавливает значение атрибута *data-test* в 1
p.removeAttribute('style') - удаляет атрибут *style*

Пример: обход коллекции

```
let p = document.querySelectorAll('p')

p.forEach( (el, index) => {
  console.log(index, el.nodeName.toLowerCase())
})

for(let i = 0; i < p.length; i++){ console.log(i, p[i]) }

for(let i in p){ console.log(i, p[i]) } //осторожно
```

Пример элемента <template>

```
<template id="card">
   <!-- HTML-разметка карточки  →
   <p>...</p>
</template>
```

Пример клонирования template

```
const template = document.querySelector('#card')
let clone  = template.content.cloneNode(true)
let p = clone.querySelector('p')
p.innerHTML = 'Текст карточки'
document.body.appendChild( clone )
```

Пример работы с classList

```
let p = document.querySelector('p')

p.classList.add('test1')
p.classList.add('test2')
p.classList.remove('test2')
p.classList.toggle('test3')

console.log( p.classList.contains('test3') )
console.log( p ); //<p class="test1 test3">
```

Пример getBoundingClientRect

```
let p = document.querySelector('p')
let rect = p.getBoundingClientRect()
console.log( rect.x, rect.y )
console.log( rect.width, rect.height )
console.log( rect.top, rect.left )
```

Пример создания элемента

```
let p = document.createElement('p')
p.innerHTML = 'содержимое'
document.body.appendChild(p)
```

Модуль 5. События (5 ак. ч.)

Ключевые термины:

- Базовая модель событий - старая модель основанная на обработчиках типа onclick
- **W3C модель** - современная модель событий использующая методы addEventListener() и removeEventListener()
- Объект-событие - объект, который передаётся в обработчик при возникновении события
- element.**addEventListener** - метод элементов для навешивания обработчика события на элементе
- event.**preventDefault**() - метод объекта события для отмены обработчика по умолчанию (типа нажатия на ссылку или отправку формы)
- event.**stopPropagation()** - метод предотвращающий всплытие события и дальнейшую его обработку

Пример: базовая модель

```
let p = document.querySelector('p')
function fn(ev){ console.log(ev) }
p.onclick = fn
```

Пример: W3C модель событий

```
let p = document.querySelector('p')

function fn(ev){ console.log(ev) }
p.addEventListener('click', fn)
```

Пример удаления обработчиков события

```
p.onclick = null;
p.removeEventListener('click', fn)
```

Пример: свойства события

```
let p = document.querySelector('p')
function fn(ev){
  const {type, target, button, altKey, clientX, clientY, keyCode} = ev
  console.log(type, target, button, altKey, clientX, clientY, keyCode)
}
p.addEventListener('click', fn)
```

Пример: отмена действия по умолчанию

```
let a = document.querySelector('a')
a.addEventListener('click', ev => {
   ev.preventDefault();
   //… остальной код
})
```

Пример предотвращения всплытия события

```
let div = document.querySelectorAll('div')
div[0].addEventListener('click', ev => {
  ev.stopPropagation();
  console.log(0)
})
div[1].addEventListener('click', ev => {
  ev.stopPropagation();
  console.log(1)
})
```

Модуль 6. Практическая работа (4 ак. ч.)

Задачи формулирует преподаватель

Модуль 7. Введение в асинхронный JS (4 ак. ч.)

Ключевые термины:

- **Асинхронность** - возможность выполнять разные задачи одновременно или не дожидаться окончания одной задачи перед стартом другой
- **Коллбеки** - функции передаваемые в качестве аргументов в другие функции
- **Промисы** (Promise - обещание) - новый стиль асинхронного кода
- async/await - новый новый стиль асинхронного кода
- **async** - ключевое слово для создания асинхронной функции
- **await** (в переводе - "ждите") - оператор заставляющий дожидаться выполнения промиса внутри асинхронных функций
- **fetch API** - интерфейс BOM для получения информации по сети, по сути - AJAX

Пример работы с промисом

```
const promise1 = new Promise((resolve, reject) => {
  setTimeout(() => { resolve('foo'); }, 300);
});

promise1.then((value) => {
  console.log(value); //"foo"
});

console.log(promise1); //[object Promise]
//https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise
```

Пример асинхронной функции

```
async function f() {
  return 'foo';
}
f().then( console.log )
```

Пример await

```
async function f() {
  const promise = new Promise((resolve, reject) => {
    setTimeout(() => { resolve('foo'); }, 300);
  });
  let str = await promise
  console.log(str); //foo
}
f()
```

Пример Fetch API

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

Модуль 8. Введение в модули и Node.js (5 ак. ч.)

Ключевые термины:

- **Модули** - отдельный структурные фрагменты приложения. В JS - JS-файлы, написанные по определённым правилам. Могут экспортировать и импортировать функции, классы, переменные
- **import** - ключевое слово для импорта данных в модуль
- **export** - ключевое слово для экспорта данных из модуля
- **Node.js** - программная платформа для выполнения JavaScript в различных средах (Linux, MacOS, Windows)
- **npm** (Node package manager) - сайт, утилита и самый большой реестр пакетов/модулей на JavaScript
- **package.json** - файл с конфигурацией проекта для npm, записанная в формате JSON (https://docs.npmjs.com/cli/v8/configuring-npm/package-json)
- **npx** - утилита выполнения двоичных файлов npm

Пример ES6-модуля в JS

```
export const sum = (...args) => {
  return args.reduce( (p, c) => p + c )
}
export default function sub (a,b) {
  return a - b;
}
```

Пример импорта модуля ES6

```
import {sum} from './sum.mjs'
import sub from './sum.mjs'

console.log( sum(10,3) )
console.log( sub(10,3) )
```

Пример запуска с консолью Node.js

```
node -v
node -h
node index.js
```

Пример работы с npm

```
npm init
npm init -y
npm install commander
npm install vue@next
npm i --save-dev jest
npm i --save-dev parcel
npm list
```

Пример package.json

```
{
  "name": "test-module",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "commander": "^8.3.0"
  },
  "devDependencies": {
    "jest": "^27.4.7",
    "parcel": "^2.2.0"
  }
}
```

Пример работы с npx

```
npx -v
npx create-react-app my-app
cd my-app
npm start
```

Модуль 9. Тестирование и сборка (5 ак. ч.)

Ключевые термины:

- **Тестирование** - проверка кода на корректность
- **Jest.js** - один из фреймворков тестирования JS-кода
- **matchers** (сопоставители) - методы для проверки значений разными способами
- **setup** и **teardown** - подготовка начальных величин для тестирования и сброс величин в начальное значение
- **Мок-функции** (mock functions) - функции заглушки, позволяют проверять связи между кодом, удаляя фактическую реализацию функции и фиксируя вызовы функции
- **Сборка проекта** - набор действий по подготовке файлов проекта для финишного использовани. Например, удаление комментариев, минификация кода и объединение файлов модулей
- **Parcel.js** - один из инструментов сборки

Пример: Старт Jest.js на CommonJS-модулях

sum.js:

```
const sum = (...args) => args.reduce( (p, c) => p + c )
module.exports = sum
```

sum.test.js:

```
const sum = require('./sum.js');

test('сумма 3 и 5 будет равна 8', () => {
  expect(sum(3, 5)).toBe(8);
});
```

package.json:

```
  …
  "scripts": {  "test": "jest"  }
  …
```

Пример запуска тестирования

npm run test

```
> test-module@1.0.0 test
> jest

 PASS  ./sum.test.js
  √ сумма 3 и 5 будет равна 8 (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.3 s
```

Ran all test suites.

Пример: Parcel.js первые шаги

src/index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <title>My First Parcel App</title>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

npx parcel src/index.html

http://localhost:1234/

Пример: Parcel.js - CSS и JS

src/styles.css:

```
h1 {
  color: lightblue;
  font-family: cursive;
}
```

src/app.js:

```
console.log('Hello world!');
```

src/index.html:

```
    <link rel="stylesheet" href="styles.css" />
    <script type="module" src="app.js"></script>
```

Пример: package.json и Parcel.js

package.json **для webapp**

```
"source": "src/index.html",
"scripts": {
    "start": "parcel",
 }
```

package.json **для библиотеки**

```
  "source": "src/app.js",
  "main": "dist/main.js",
  "scripts": {
    "build": "parcel build"
}
```

Модуль 10. Практическая работа (4 ак. ч.)

Задачи формулирует преподаватель