Javascript

Array Methods

- forEach
- map
- filter
- split
- some
- every
- slice vs splice
- sort
- reduce

- forEach

```
const formParamsExample = [
 { key: "first_name", value:
"Amirmahdi" },
 { key: "last_name", value: "Digbari"
},
 { key: "gender", value: "Male" },
];
```

```
const formData = new FormData();
formParamsExample.forEach(({ key,
value }) => {
 formData.append(key, value);
});

// [
//   ["first_name", "Amirmahdi"],
//   ["last_name", "Digbari"],
//   ["gender", "Male"],
// ];
```

map

```
const categories = [
 {
   id: 1,
   name: "Category 1",
   children: [
     { id: 2, name: "Subcategory 1.1" },
     { id: 3, name: "Subcategory 1.2" },
   ],
 },
 {
   id: 4,
   name: "Category 2",
   children: [
     { id: 5, name: "Subcategory 2.1" },
     { id: 6, name: "Subcategory 2.2" },
   ],
 },
];
```

```
const categoryIds = categories.map((user) => user.id);
  // [1, 4]
```

• map

```
const categoriesWithChildNames =
categories.map((category) => ({
 ...category,
 childNames: category.children
   .map((subcategory) =>
subcategory.name)
   .join(", "),
}));
```

```
// [
//   {
//     id: 1,
//     name: "Category 1",
//     children: [...],
//     childNames: "Subcategory 1.1, Subcategory 1.2",
//   },
//   {
//     id: 4,
//     name: "Category 2",
//     children: [...],
//     childNames: "Subcategory 2.1, Subcategory 2.2",
//   },
// ];
```

- # filter

```
const tickets = [
 {
   id: 1,
   messages: ["Hi", "Hey", "Bye"],
   state: "closed",
 },
 {
   id: 2,
   messages: ["Hi", "Hey"],
   state: "open",
 },
];
```

```
const closedTickets =
tickets.filter((ticket) => ticket.state
=== "open");
// [
//   {
//     id: 2,
//     messages: ["Hi", "Hey"],
//     state: "open",
//   },
// ];
```

- split

```
const subcategoryNames = "Subcategory 1.1, Subcategory 1.2";

const subcategories = subcategoryNames.split(",").map((name) =>
name.trim());
// ["Subcategory 1.1", "Subcategory 1.2"]
```

- some + every

```
nst userTickets = [
 {
   id: 1,
   messages: ["Hi", "Hey", "Bye"],
   state: "closed",
 },
 {
   id: 2,
   messages: ["Hi", "Hey"],
   state: "open",
 },
];
```

```
const hasAtLeastOneOpenTicket =
userTickets.some(
 (ticket) => ticket.state === "open"
); // true,

const allTicketClosed = userTickets.every(
 (ticket) => ticket.state === "closed"
); // false
```

- every

```
[].every(Boolean);
 // true
```

- slice vs splice

```
const alphabets = ["a", "b", "c", "d", "e", "f"];
```

```
const slicedAlphabet =
alphabets.slice(1, 4);

// {
// alphabets: ["a", "b", "c", "d",
"e", "f"],
// slicedAlphabet: ["b", "c", "d"],
// }
```

```
const splicedAlphabet =
alphabets.splice(1, 4);

// {
// alphabets: ["a", "f"],
// splicedAlphabet: ["b", "c", "d",
"e"],
// }
```

- sort

```
const times = ["12:00", "2:00", "1:30", "12:25", "00:56"];
```

```
const firstApproachSortedList = [...times].sort((a, b) => {
 const [aHour, aMinute] = a.split(":").map(Number);
 const [bHour, bMinute] = b.split(":").map(Number);

 if (aHour !== bHour) {
   return aHour > bHour ? 1 : -1;
 }
 if (aMinute !== bMinute) {
   return aMinute > bMinute ? 1 : -1;
 }
 return 0;
});
// ["00:56", "1:30", "2:00", "12:00", "12:25"];
```

```
const secondApproachSortedList = [...times].sort((a, b) => {
 const [aHour, aMinute] = a.split(":").map(Number);
 const aAllMinutes = aHour * 60 + aMinute;

 const [bHour, bMinute] = b.split(":").map(Number);
 const bAllMinutes = bHour * 60 + bMinute;

 return aAllMinutes - bAllMinutes;
});

// ["00:56", "1:30", "2:00", "12:00", "12:25"];
```

```
const numbers = [3,44,55,33, -33, -4,2, -3, -44].sort()

// [-3, -33, -4, -44, 2, 3, 33, 44, 55]
```

- reduce

```
const ticketsAgain = [
 {
   id: 1,
   messages: ["Hi", "Hey", "Bye"],
   state: "closed",
 },
 {
   id: 2,
   messages: ["Hi", "Hey"],
   state: "open",
 },
];
```

```
const closedTicketsAgain = filter(tickets,
(ticket) => ticket.state === "open");

// [
//   {
//     id: 1,
//     messages: ["Hi", "Hey", "Bye"],
//     state: "closed",
//   },
// ];

const ticketIds = map(tickets, (ticket) =>
ticket.id);

// [1, 2]
```

```
function filter(array, condition) {
 return array.reduce(
   (result, item) =>
(condition(item) ? [...result,
item] : result),
   []
 );
}
```

```
function map(array, mapper) {
 return array.reduce((result,
item) => [...result,
mapper(item)], []);
}
```

- StructuredClone

- Makes a deep copy of the value
- Works with recursion
- Doesn't work with
  - Functions
  - DOM
  - Prototype chain (later in the slides)
- `JSON.parse(JSON.stringify(...))`

- Shallow Copy

- Reference of items remain the same
- How:
  - `Array.from(...)`
  - `[...items]`
  - `{...items}`
  - etc

Pass by Reference vs Pass by Value

```
const object1 = { field_a: "a",
field_b: "b" };
const list1 = [object1];
console.log(list1);
// [{field_a: “a”, field_b: “b”}]
console.log(object1);
// {field_a: “a”, field_b: “b”}
list1[0].field_a = "b";
console.log(list1);
// [{field_a: “b”, field_b: “b”}]
console.log(object1);
// {field_a: “b”, field_b: “b”}
```

```
const value1 = "a";
const list2 = [value1];
console.log(list2);
 // [“a”]
console.log(value1);
 // “a”
list2[0] = "b";
console.log(list2);
// [“b”]
console.log(value1);

// “a”
```

Object Methods

- keys
- values
- entries
- ***** (after challenge :D)

- keys + values

```
const object1 = {
 first_name: "Amirmahdi",
 last_name: "Digbari",
 gender: "Male",
};
Object.keys(object1); // ["first_name", " last_name", " gender"]
Object.values(object1); // ["Amirmahdi", "Digbari", "Male"]
```

- entries

```
const objectFormParamsExample = {
 first_name: "Amirmahdi",
 last_name: "Digbari",
 gender: "Male",
};
```

```
const formDataAgain = new FormData();
Object.entries(formParamsExample).for
Each(([key, value]) => {
 formData.append(key, value);
});

// [
//   ["first_name", "Amirmahdi"],
//   ["last_name", "Digbari"],
//   ["gender", "Male"],
// ];
```

Challenge

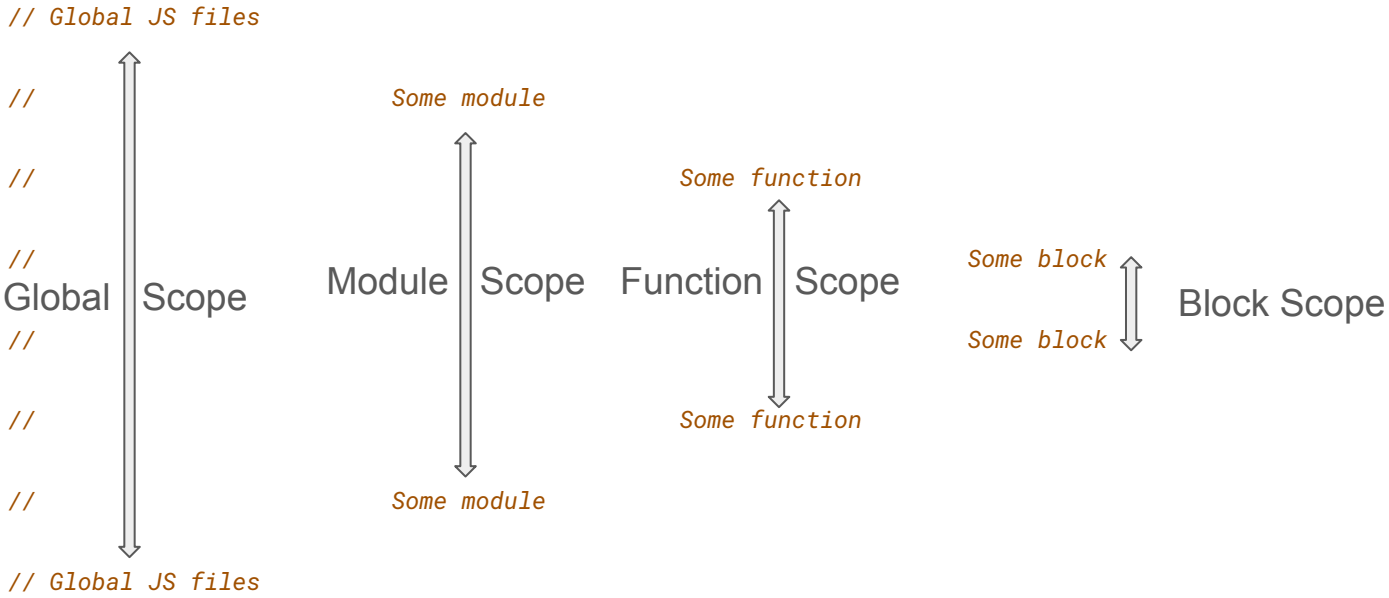
```
const workingHours = [
 {
   day_of_week: "Saturday",
   start_time: "9:00",
   end_time: "14:00",
 },
 {
   day_of_week: "Friday",
   start_time: "9:00",
   end_time: "16:00",
 },
 {
   day_of_week: "Saturday",
   start_time: "16:00",
   end_time: "22:00",
 },
 {
   day_of_week: "Sunday",
   start_time: "9:00",
   end_time: "22:00",
 },
];
```

```
// {
//   "Saturday": ["9:00-14:00", "16:00-22:00"],
//   "Sunday": ["9:00-16:00"],
//   "Friday": ["9:00-22:00"]
// }
```

- groupBy
  - Try Challenge with this method

Q/A

Scopes


// Global JS files
//
//
//
Global Scope
//
//
//
// Global JS files
Some module
Module Scope
Some module
Some function
Function Scope
Some function
Some block
Some block
Block Scope

- Examples

```
const global_const = "GLOBAL_CONST";   // Global
let global_let = "GLOBAL_LET";  // Global
var global_var = "GLOBAL_VAR";  // Global

// file.js
const module_const = "MODULE_CONST"; // Module
let module_let = "MODULE_LET";  // Module
var module_var = "MODULE_VAR";  // Global

function someFunction() {
 const inner_function_const = "INNER_FUNCTION_CONST";   // Function
 let inner_function_let = "INNER_FUNCTION_LET";   // Function
 var inner_function_var = "INNER_FUNCTION_VAR";   // Function

 if (true) {
   const inner_block_const = "INNER_BLOCK_CONST";  // Block
   let inner_block_let = "INNER_BLOCK_LET";   // Block
   var inner_block_var = "INNER_BLOCK_VAR";   // Function
 }
}
```

Hoisting

- JS has compiler

Closure

```
function add(count) {
 return function _add(number) {
   return number + count;
 };
}
const addTwo = add(2);
console.log(addTwo(5)); // 7
```

• Debounce + throttling

```
function debounce(callback, delay) {
 let timeoutRef = undefined;

 return function delayedCallback(...args) {
   clearTimeout(timeoutRef);

   timeoutRef = setTimeout(function () {
     timeoutRef = undefined;
     callback(...args);
   }, delay);
 };
}
```

```
const debouncedLog = debounce(console.log,
2_000);
for (let i = 0; i <= 1_000_000; i++) {
 debouncedLog(i);
}
console.log("Done");

// Done
// After 2 seconds ====> 1000000
```

• once

```
function once(callback) {
 let isExecuted = false;

 return function onceCall(...args) {
   if (isExecuted) {
     return;
   }
   isExecuted = true;
   return callback(...args);
 };
}
```

```
const onceLog = once(console.log);
for (let i = 0; i <= 1_000_000; i++) {
 onceLog(i); // 0
}
```

• memoize

```
function fibonacci(n) {
 if (n === 1) return 1;
 if (n === 2) return 2;

 return fibonacci(n - 1) + fibonacci(n - 2);
}
console.log(fibonacci(50)); // Won't answer soon:)
```

```
function memoize(callback) {
 const cache = {};

 return function memoizedCallback(...args) {
   const argsString = JSON.stringify(args);
   if (!(argsString in cache)) {
     cache[argsString] = callback(...args);
   }

   return cache[argsString];
 };
}
fibonacci = memoize(fibonacci);
console.log(memoize(fibonacci)(50)); // Answers pretty fast
```

Experimental Decorators

```
function catcher(_, propertyKey, descriptor) {
 const originalMethod = descriptor.value;
 descriptor.value = function (...args) {
   try {
     return originalMethod.apply(this, args);
   } catch (error) {
     console.log(`Error in ${propertyKey}:`,
error);
   }
 };
 return descriptor;
}
```

```
class SomeClass {
 @catcher
 static someMethod() {
   throw new Error("Hahahaha");
 }
}
SomeClass.someMethod();
// Error in someMethod: Hahahaha
```

Challenge

```
add(1, 2); // 3
add(1, 2)(3); // 6
add(1, 2)(3)(4, 5); // 15
add(1, 2)(3)(4, 5)(6); // 21
add(1, 2)(3)(4, 5)(6)(7, 8, 9);// 45
```

Challenge

Implement Throttle that can call n times within each p period

Q/A

Promise

- An Object :D
- states = Pending | Fulfilled | Rejected
- onFulfilled, onRejected methods
- Can be nested/chained (.then chain)
- Parallel processing

```
function submitFormWithAttachment() {
 fetch("/token")
   .then((res) => res.json())
   .then((res) => {
     const headers = { Authorization: res.token };
     fetch("/submit-attachment", { method: "POST", headers })
       .then((res) => res.json())
       .then((res) => {
         fetch("submit-data", {
           headers,
           body: JSON.stringify({ attachments: res }),
         })
           .then((res) => res.json())
           .then((res) => {
             visualize(res);
           });
       });
   });
}
```

```
function submitFormWithAttachmentAndCatch() {
 fetch("/token")
   .then((res) => res.json())
   .then((res) => {
     const headers = { Authorization: res.token };
     fetch("/submit-attachment", { method: "POST", headers })
       .then((res) => res.json())
       .then((res) => {
         fetch("submit-data", {
           headers,
           body: JSON.stringify({ attachments: res }),
         })
           .then((res) => res.json())
           .then((res) => {
             visualize(res);
           })
           .catch((error) => {
             console.log(error);
             return;
           });
       })
       .catch((error) => {
         console.log(error);
         return;
       });
   })
   .catch((error) => {
     console.log(error);
     return;
   });
}
```

Promise.all vs Promise.race

```
Promise.all([fetch("1"), fetch("2")]).then((res) => console.log(res));
// [1 response, 2 response]


Promise.race([fetch("1"), fetch("2")]).then((res) => console.log(res));
// sometimes [1 response], sometimes [2 response]
```

Parallel Processing

- What Single thread means in JS.

Async-Await

- Flatten Promise
- (Beautiful. Argue with me :D)

```
async function asyncSubmitFormWithAttachmentAndCatch() {
 try {
   const token = await fetch("/token").then((res) => res.json());
   const headers = { Authorization: token };

   const attachments = await fetch("/submit-attachment", {
     method: "POST",
     headers,
   }).then((res) => res.json());

   const result = await fetch("submit-data", {
     headers,
     body: JSON.stringify({ attachments }),
   }).then((res) => res.json());

   visualize(result);
 } catch (error) {
   console.log(error);
   return;
 }
}
```

```
async function asyncSubmitFormWithAttachmentAndCatch() {
 try {
   const token = await fetch("/token").then((res) => res.json());
   const headers = { Authorization: token };

   try {
     const attachments = await fetch("/submit-attachment", {
       method: "POST",
       headers,
     }).then((res) => res.json());

     try {
       const result = await fetch("submit-data", {
         headers,
         body: JSON.stringify({ attachments }),
       }).then((res) => res.json());

       visualize(result);
     } catch (error) {
       console.log(error);
       return;
     }
   } catch (error) {
     console.log(error);
     return;
   }
 } catch (error) {
   console.log(error);
   return;
 }
}
```

Generator

```
function* generator(n) {
 for (let i = 1; i <= n; i++) {
   yield i;
 }
}

const gen = generator(10);
let value = gen.next();

while (!value.done) {
 console.log(value.value); // 1 2 3 4 5 6 7 8 9 10
 value = gen.next();
}
```

Async-await with Generator

```
function asyncFunctionExecutor(generatorFunction) {
 const generator = generatorFunction();

 function handle(result) {
   if (result.done) return result.value;

   return Promise.resolve(result.value)
     .then((res) => handle(generator.next(res)))
     .catch((err) => handle(generator.throw(err)));
 }

 return handle(generator.next());
}
```

```
function* asyncFunction() {
 const data = yield fetchData();

 const processedData = yield processData(data);
 return processedData;
}
asyncFunctionExecutor(asyncFunction);
```

Web Workers

- Run in background
- Run in separate thread
- Use cases
- Shared Workers.

Service Worker

- Type of Web Worker
- Proxy between application and network
- Some use cases are caching and mocking.

Challenge

Q/A

Prototype

```
const greeter = {
 greet() {
   console.log("Hi");
 },
};

const welcomingPerson = Object.create(greeter);
welcomingPerson.greet();
console.log(Object.getPrototypeOf(welcomingPerson));
// welcomingPerson.__proto__
// {greet: f}
```

Inheritance

```
const greeter = {
 greet() {
   console.log("Hi");
 },
};

const welcomingPerson =
Object.create(greeter);
welcomingPerson.greet();
console.log(Object.getPrototypeOf(welcomingP
erson)); // welcomingPerson.__proto__
// {greet: f}
```

```
const extroverted = Object.create(welcomingPerson);
console.log(Object.getPrototypeOf(extroverted)); //
extroverted.__proto__
 // {}

console.log(Object.getPrototypeOf(Object.getPrototypeOf(ex
troverted))); // extroverted.__proto__.__proto__
 // {greet: f}

 console.log(
  Object.getPrototypeOf(
   Object.getPrototypeOf(Object.getPrototypeOf(extroverted))
  )
 ); // extroverted.__proto__.__proto__.__proto__

// {__defineGetter__: f, __defineSetter__: f, hasOwnProperty:
f, __lookupGetter__: f, __lookupSetter__: f, …}
```

Class

```
class Greeter {
 greet() {
   console.log("Hi");
 }
}
const g = new Greeter();
g.greet();
```

```
export class TokenStorage {
 static _accessTokenKey = 'access-token'

 @windowIsAvailable
 static set(value) {
   Cookies.set(this._accessTokenKey, value);
 }

 @windowIsAvailable
 static get() {
   return Cookies.get(this._accessTokenKey);
 }

 @windowIsAvailable
 static delete() {
   Cookies.remove(this._accessTokenKey);
 }
}
```

```
export const windowIsAvailable = (_target,
_propertyKey, descriptor) => {
 const originalMethod = descriptor.value;

 descriptor.value = function (...args) {
   if (typeof window !== "undefined") return
originalMethod.apply(this, args);
   else throw new Error(`You should not use
${_propertyKey} in server`);
 };

 return descriptor;
};
```

Inheritance

```
class Shape {
 constructor() {}
 area() {
   throw new Error("Method area not implemented");
 }
}
```

```
class Circle extends Shape {
 constructor(r) {
   this.r = r;
 }
 area() {
   return Math.PI * Math.pow(this.r, 2);
 }
}
class Square extends Shape {
 constructor(d) {
   this.d = d;
 }
 area() {
   return Math.pow(this.d, 2);
 }
}
```

Inheritance

```
const shape = () => ({
 area() {
   throw new Error("Method area not implemented");
 },
});
```

```
const circle = (r) => {
 const result = Object.create(shape);
 result.r = r;
 result.area = function () {
   return Math.PI * Math.pow(this.r, 2);
 };

 return result;
};

const square = (d) => {
 const result = Object.create(shape);
 result.d = d;
 result.area = function () {
   return Math.pow(this.d, 2);
 };

 return result;
};
```

This

- Context that code is executing in.
- Contextes can be:
  - global
  - function / method
  - object / class
  - etc
- Runtime determination

```
function getThis() {
 return this;
}

const obj1 = { name: "obj1" };
obj1.getThis = getThis;

console.log(obj1.getThis()); // { name: 'obj1', getThis: [Function: getThis] }
console.log(getThis()); // global
```

```
const obj2 = Object.create(obj1);
obj2.name = "obj2";

console.log(obj2.getThis()); // { name: 'obj2' }
```

```
const obj3 = { name: "obj3" };
obj3.getThis = obj1.getThis;

console.log(obj3.getThis());
// { name: 'obj3', getThis: [Function: getThis] }
```

```
const obj4 = {name: "obj4"}
obj4.getThis = getThis()

console.log(obj4.getThis.apply(obj2));
// { name: 'obj2' }
```

Arrow Function vs Function

- Arrow Functions: refers to lexical context. Valued in definition time.

```
const obj = {
 name: 'Obj',
 greet: function() {
     console.log(this.name);
 }
};
obj.greet()  // Obj
```

```
const obj = {
 name: 'Obj',
 greet: () => {
     console.log(this.name);
 }
};
obj.greet()  // global
```

Strict mode

- this → can be undefined / can be primitive.

```
function nonStrictMethod() {
  const getThis = () => {
   return this;
 };
 console.log(getThis());// global
}
```

```
function strictMethod() {
 "use strict";
  const getThis = () => {
    return this;
  };
  console.log(getThis()); // undefined
}
```

```
function Person(name) {
 this.name = name;
}
const person = new Person('Ali');
person.name // Ali
```

```
function Timer() {
 this.seconds = 0;
 setInterval(() => {
   this.seconds++;
   console.log(this.seconds);
 }, 1000); // Works
}
```

```
button.addEventListener("click",
function () {
 console.log(this); // button
});
```

```
button.addEventListener("click", ()
=> {
 console.log(this); // global
});
```

```
class Counter {
 constructor() {
   this.count = 0;
 }

 increment() {
   this.count++;
   console.log(this.count);
 }

 incrementArrow = () => {
   this.count++;
   console.log(this.count);
 };
}

const counter = new Counter();
setTimeout(counter.increment, 1000);  // Nan
setTimeout(counter.incrementArrow, 1000);  // 1
```

Challenge

No Challenge. Let’s talk about my error handling idea

Q/A

Open Conversation

Ask me anything