JavaScript Course

Tuesday, 5 July 2022 12:26 pm

Credentials

zeshan@bitsol.tech
zeshan@123

== and ===

```
const number = 1234
const stringNumber = '1234'

console.log(number == stringNumber) //true
console.log(number === stringNumber)  //false
```

The value of `number` and `stringNumber` looks similar here. However, the type of `number` is `Number` and type of `stringNumber` is `string`. Even though the values are same, the type is not the same. Hence a `==` check returns `true`, but when checked for value and type, the value is `false`.

- **Variables**
- **Data types**
- **Basic operations**
- **Decision Statements**
- **Statements and Expressions**
- **Functions and its types**
- **Arrays**
- **Objects**
- **Loops**
- **Get Started with Node.js**
- **Projects**
- **Selecting and Manipulating Elements**
- **Handling Click Events**
- **Implementing Game Logics**
- **Working with Classes**
- **JavaScript Behind the Scenes**
- **Data Structures, Modern Operators and Strings**
  - **Destructing Arrays**
    - **Const [a,b] = mystructure;**

DRY PRINCIPLE (DON'T REPEAT YOURSELF)

JAVASCRIPT BEHIND THE SCENES

1. High-Level
2. Garbage Collected (Collected the garbage e.g. the unused objects etc.) from the memory to speed up the language and make the memory allocation in better way.
3. Interpreted
4. Multi Paradigm.
   a. Procedural Programming
   b. OOP
   c. FP
5. Prototype-based object-oriented
6. First-Class functions (Means functions are treated as variables) .. Can pass into other functions and return them from functions.
7. Dynamic (Dynamically Typed)
8. Non-blocking event loop
   a. Concurrency model
   b. Multi tasking
   c. EVENT LOOP
      i. Execute the process in the background.
9. COMPILATION vs INTERPRETATION

AT ONCE

LINE BY LINE

1.

TODAY, I MADE THE DICE GAME in javaScript, played with the scoring system.... How to add/remove classes in the elements using javascript... How to add actionlisteners to the buttons onclick etc... How to get keyboard events etc.... Change the styles of the elements using JS... Use functions inside a functions... Toggle the classes....

Returning multiple values from the structure... Arrays.... Indexing ..
Theoretical background of JS.

NOTES

1. Next -> Graphql & RestAPI

```
const {
  fri: { open, close },
} = openingHours;
```

Attribute from which we are getting the Value.

Attributes of "Fri" will become new variable with same name. if we want to make new variables, we will declare them using ':' operator. i.e

{ open: op, close: cl}

SPREAD OPERATOR (...)

Separate all the elements of an array by the operator '.'

```
let r1 = [2, 3, 4];
let r2 = [...r1];
```

```
// Making Objects
const newRes = {
  foundedIn: 1998,
  ...restaurant,
  founder: 'Guiseppe',
};
```

Here 'restaurant' is the separate object, its all the properties/attributes will be copied to newRes.

&& operator is like a if statements

// REST PATTERNS AND PARAMETERS (Opposite to Spread Operator)
// [It used at the left side of the equality operator.]

```
const [a5, b5, ...others] = [1, 2, 3, 4, 5];
console.log(others);
```

SHORT CIRCUITING:

```
console.log(3 || 'Zeshan'); // 3
```

^ it returns first truthy value or last falsy value(in case of no truth false there) !

In case of && operator : it will return first false value or last true value (in case of no false statement there)

FEW PROPERTIES OF LOGICAL OPERATORS:

1. Use any data type and return any data type.

What Are the Truthy values?

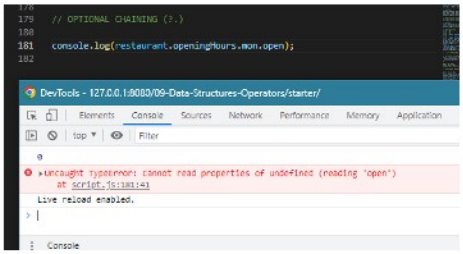
In JavaScript, a truthy value is **a value that is considered true when encountered in a Boolean context**. All values are truthy unless they are defined as falsy. That is, all values are truthy except false , 0 , -0 , 0n , "" , null , undefined , and NaN

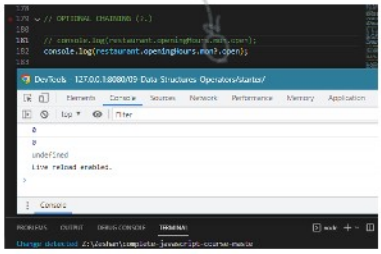
From <https://www.google.com/search?q=Truthy+value+means&oq=Truthy+value+means&aqs=chrome..69i57.2890j0j7&sourceid=chrome&ie=UTF-8>

```
// Nullish Coalescing Operator (??)
( Difference is that : Here falsy values are only NULL
and UNDEFINED values.)
```

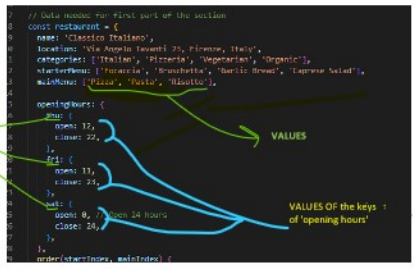
```
let x4 = 0;
console.log(x4 || 'Its true!'); // Its true
console.log(x4 && 'Its true!'); // 0
```

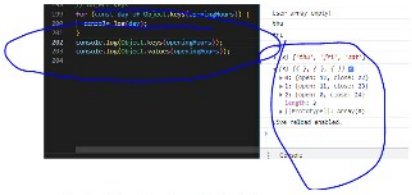
OPTIONAL CHAINING : It will help in NOT producing error when any element in the chain doesn't exits.






```
// optional chainging for methods
console.log(restaurant.orderrrrrr?.(0, 1) ?? 'Method does not exist');

for (const [1, ele] of game.players[0].entries()) console.log(1, ele);
```



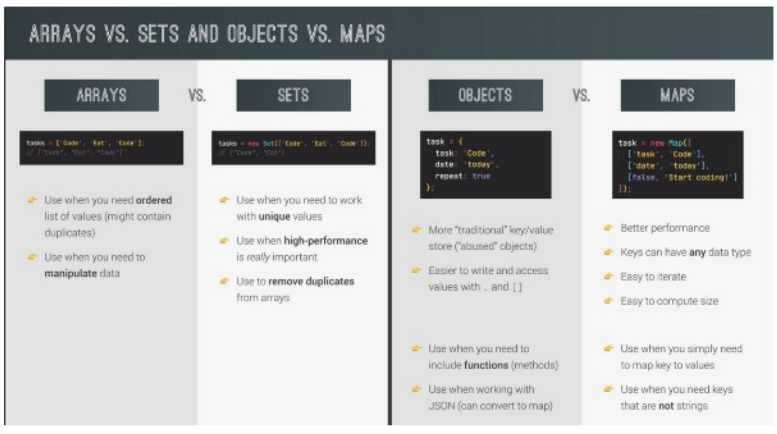


```
console.log(Object.keys(openingHours));
console.log(Object.values(openingHours));
console.log(Object.entries(openingHours));
```

===

MAPS

```
// MAPS [ Keys can be arrays, numbers, booleans NOT ONLY STRINGS like we have in objects]
const rest = new Map();
rest.set('name', 'Classico Italiano');
rest.set(1, 'Zeshoooooooooo');
rest.set('open', 11).set('close', 23);
console.log(rest.get(1));
// consecutively!
rest
  .set(5, 'S')
  .set(6, 'umair00')
  .set(true, 'WE ARE OPEN!')
  .set(false, 'WE ARE CLOSE!');
console.log(rest.set(2, 'Umairrrr1111'));
const time = 21;
console.log(rest.get(time > rest.get('open') && time < rest.get('close')));
rest.delete('name');
console.log(rest);
rest.clear();
const arr2 = [1, 2, 3];
const arr4 = [1, 2, 3];
rest.set(arr2, 'ARRAY');
console.log(rest.get(arr2));
rest.set(document.querySelector('h1'), 'Heading');
console.log(rest);
// Iterating the MAPS
const question = new Map([
  ['question', 'What is the best programming language in the world?'],
  [1, 'C'],
  [2, 'Java'],
  [3, 'JS'],
  [true, 'Correct 🎉'],
  [false, 'Try Again 🤓'],
  ['correct', 3],
]);
console.log(question);
// Convert Object to Map
console.log(...Object.entries(openingHours));
const hoursMap = new Map(Object.entries(openingHours));
console.log(hoursMap);
console.log(question.get('question'));
for (const [key, value] of question) {
  if (typeof key == 'number') {
    console.log(`Option ${key}: ${value}`);
  }
}
const answer = Number(prompt('Enter Answer: 1/2/3'));
console.log(question.get(question.get('correct') === answer));
```



ABOUT STRINGS

```
// ------------ WORKING WITH STRINGS --------------
let airline = 'TAP Air Pakistan';
const plane = 'A320';
console.log(plane[0]);
console.log(plane[1]);
console.log(plane[2]);
console.log('Pakistan'.length);
console.log(airline.indexOf('a'));
console.log(airline.lastIndexOf('a'));
console.log(airline.indexOf('Pakistan'));
console.log(airline.slice(4, 7));
```

NUMBER SHOWS THAT IT IS MAKING THE TOAL LENGTH OF SUCH NUMBER LENGTH INCLUDING THE PADDING CHARACTER i.e + in this case.

```
let s = 'My';
let t = 'Laptop';

let st = [s, t];
let joined = st.join(' ');
console.log(joined);

let message = '__';
message = message.padStart(9, '+');
```

```
No Issues
You got the middle seat 🎉                    script.js:368
TAP AIR PAKISTAN                              script.js:374
2..,976$                                      script.js:380
2F1ghtF1ght,976$                              script.js:382
true                                          script.js:384
false                                         script.js:385
                                              script.js:394
▶ (6) ['My', 'Name', 'is', 'Muhammad', 'Zeshan', 'Tahir']
```

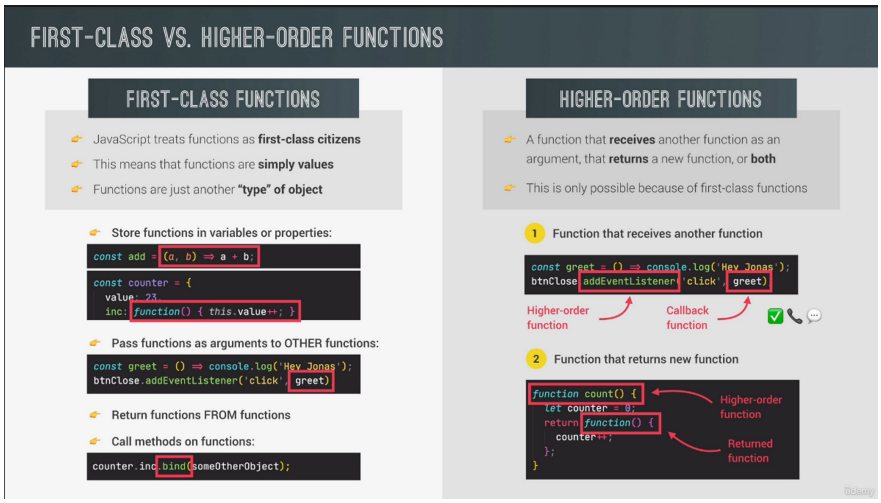
```
console.log(plane[0]);
console.log(plane[1]);
console.log(plane[2]);
console.log('Pakistan'.length);
console.log(airline.indexOf('a'));
console.log(airline.lastIndexOf('a'));
console.log(airline.indexOf('Pakistan'));
console.log(airline.slice(4, 7));
console.log(airline.slice(-5));
// console.log(airline.);
// console.log(airline.);
const checkMiddleSeat = function (seat) {
  const s = seat.slice(-1);
  if (s === 'B' || s === 'E') console.log('You got the middle seat 😬');
};
checkMiddleSeat('11B');
checkMiddleSeat('2R');
airline = airline.toUpperCase();
console.log(airline);
// replacing
const priceUS = '288,976$';
const pricePK = priceUS.replace('$', 'Pkr');
console.log(priceUS.replace(/8/g, '.')); // Making it global
const pricePK2 = priceUS.replaceAll('8', 'Eight');
console.log(pricePK2);
console.log(plane.includes('A320'));
console.log(plane.includes('Air'));
if (plane.startsWith('Airbus') && plane.endsWith('neo')) {
  console.log('HURAAHHHH!');
}
// Paractice Excercise
const checkBabbage = function (items) {};
```

```
let st = [s, t];
let joined = st.join(' ');
console.log(joined);

let message = '-_-';
message = message.padStart(9, '(');
message = message.padEnd(15, ')');
console.log(message);
```

```
2EightEight,976$                    script.js:382
true                                script.js:384
false                               script.js:385
                                    script.js:394
▶ (6) ['My', 'Name', 'is', 'Muhammad', 'Zeshan', 'Tahir']
My Laptop                           script.js:401
((((((-_-))))))                     script.js:406
Live reload enabled.                (index):64
>
Console
```

Javascript doesn't have Functions passing by reference but only by values.

Functions in Js are objects .... And objects have methods/



FUNCTIONS RETURNING FUNCTIONS (CALLING FUNCTION IN A FUNCTION)

```
};
['a', 'b', 'c'].forEach(high5);

// Functions returning Functions

const greet = function (greeting) {
  return function (name) {
    console.log(`${greeting}, ${name}!`);
  };
};

const getGreet = greet('Hello');
console.log(getGreet('Zeshan'));

const greetArr = greeting => name => console.log(`${greeting}, ${name}!`); // Functions Returning functions in a single line.
greetArr('Hello')('Zeshan');
```

```
Original String: Muhammad Zeshan Tahir
Transformed String:  MUHAMMAD Zeshan Ta
Transformed By:  upperFirstWord()
undefined
Hello, Zeshan!
undefined
Hello, Zeshan!
Live reload enabled.
```

Used to tell the public function to access the specific 'this' from a reference to specific function.

CALL, APPLY AND BIND FUNCTIONS

Bind is used to fix few arguments values to a certain value to associate specific arguments to a variable.

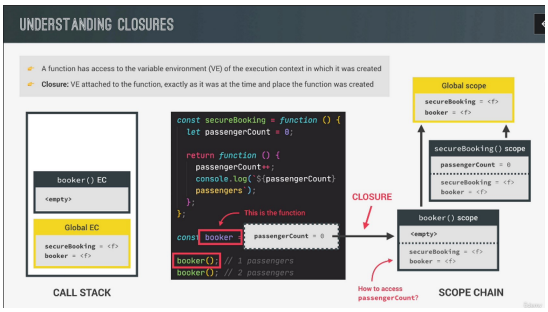
```
// The call and apply Methods

const book = function (flightNum, name) {
  console.log(
    `${name} booked a seat on ${this.airline} flight ${this.iataCode} ${flightNum}`
  );
  this.bookings.push({ flight: `${this.iataCode}${flightNum}`, name: name });
};

const lufthansa = {
  airline: 'Lufthansa',
  iataCode: 'LH',
  bookings: [],
};

book.call(lufthansa, 239, 'Zeshan Tahir');
book.call(lufthansa, 423, 'Umair Qaisar');
console.log(lufthansa);

const eurowings = {
  airline: 'Rurowings',
```

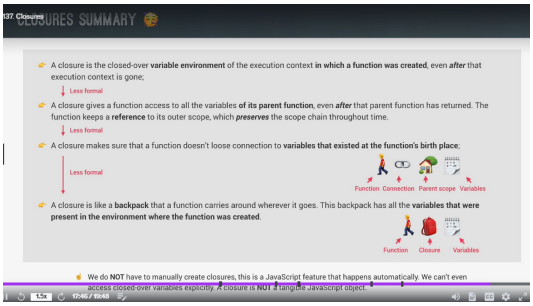
```
// The call and apply Methods

const book = function (flightNum, name) {
  console.log(
    `${name} booked a seat on ${this.airline} flight ${this.iataCode} ${flightNum}`
  );
  this.bookings.push({ flight: `${this.iataCode}${flightNum}`, name: name });
};

const lufthansa = {
  airline: 'Lufthansa',
  iataCode: 'LH',
  bookings: [],
};

book.call(lufthansa, 239, 'Zeshan Tahir');
book.call(lufthansa, 423, 'Umair Qaisar');
console.log(lufthansa);

const eurowings = {
  airline: 'Rurowings',
  iataCode: 'EW',
  bookings: [],
};

// Not Working
// book(23, 'Sarah Williams');

// As we used 'this' Keyword inside the 'Book()' function. So, we need to use the 'Call" property to tell that it is calling by which reference.
book.call(eurowings, 23, 'Sarah WIlliams');
// 'Apply' method is not used in modern js, as we have the '...' operator
let arr3 = [23, 'Sarah Williams'];
// book.apply(eurowings, arr3);
book.call(eurowings, ...arr3); // Same Functioning

// Bind Function
const bookEW = book.bind(eurowings);
bookEW(23, 'NEWWWW');
const bookLT = book.bind(lufthansa, 45); // Next argument '45' also binded to the bookLT.
bookLT('LT');
```

Var keyword is not scope specific .... Its global assignment!!!!!

E.g : WHENEVER WE RETURN THE FUNCTION FROM FUNCTION

```
// CLOSURES
const secureBooking = function () {
  let passengerCount = 0;
  return function () {
    passengerCount++;
    console.log(`${passengerCount} passengers`);
  };
};
const booker = secureBooking();
booker();
booker();
```





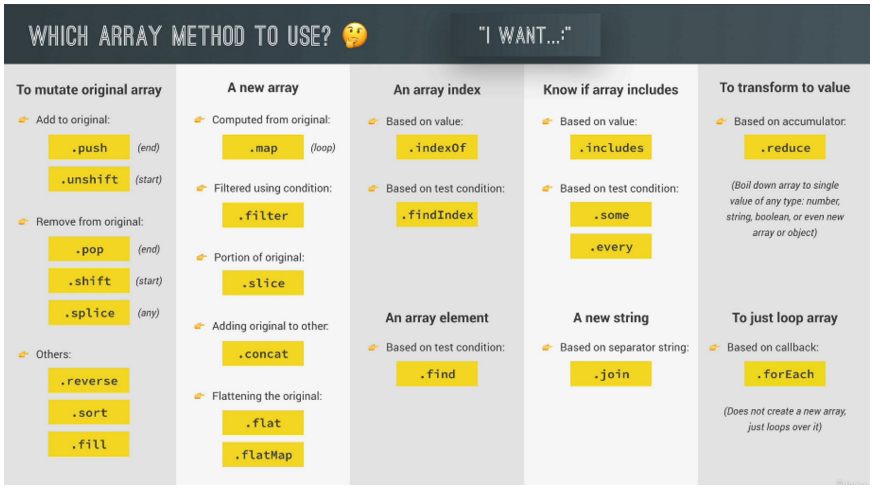
The string to be parsed as HTML or XML and inserted into the tree.

Visualization of position names

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  foo
  <!-- beforeend -->
</p>
<!-- afterend -->
```

Note: The `beforebegin` and `afterend` positions work only if the node is in the DOM tree and has a parent element.

18 - JULY - 2022



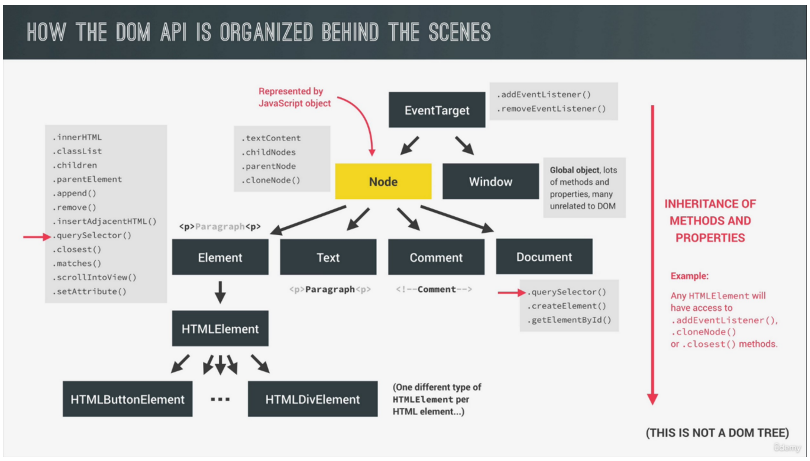

```
// Conversion
console.log(Number('23'));
console.log(+'23' + 1);
// Parsing
console.log(Number.parseInt('30px', 10));
console.log(Number.parseInt('e23', 10));
console.log('21');
console.log(Number.isNaN('20'));
// Checking that the number is a number or not
console.log(Number.isFinite(20)); // True
console.log(Number.isFinite('20')); // False
```

http://www.lingoes.net/en/translator/langcode.htm

```
const nnow = new Date();
const options = {
  hour: 'numeric',
  minute: 'numeric',
  day: 'numeric',
  month: 'numeric',
  year: 'numeric',
  weekday: 'long',
};

labelDate.textContent = new Intl.DateTimeFormat('ur-PK', options).format(now);
```

```
const startLogOutTimer = function () {
  // Set time to 5 minutes
  let time = 10;

  // Call the timer every second
  setInterval(function () {
    const min = String(Math.trunc(time / 60)).padStart
    (2, 0);
    const sec = String(time % 60).padStart(2, 0);

    // In each call, print the remaining time to UI
    labelTimer.textContent = `${min}:${sec}`;

    // Decrese 1s
    time--;

    // When 0 seconds, stop timer and log out user
  }, 1000);
};
```

19 - JULY - 2022



DATA ATTRIBUTES

```
</head>
<body>
  <header class="header">
    <nav class="nav">
      <img
        src="img/logo.png"
        alt="Bankist logo"
        class="nav__logo"
        id="logo"
        designer="Jonas"
        data-version-number="3.0"
      />
      <ul class="nav__links">
```

```
// Data Attributes
console.log(logo.dataset.versionNumber);
```

```
section1.scrollIntoView({
    behavior: 'smooth',
  });
```

About 15,300,000 results (0.54 seconds)

Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element or document object (Provided the handler is initialized).

https://en.wikipedia.org › wiki › Event_bubbling

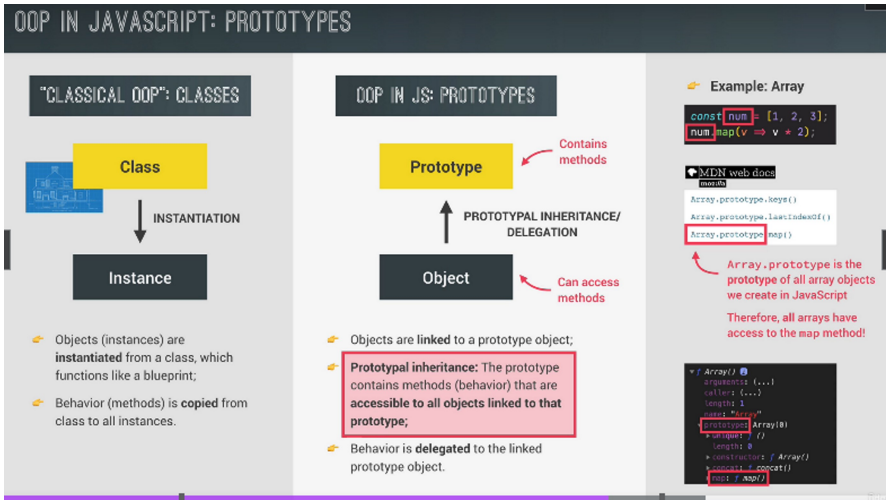
Event bubbling - Wikipedia

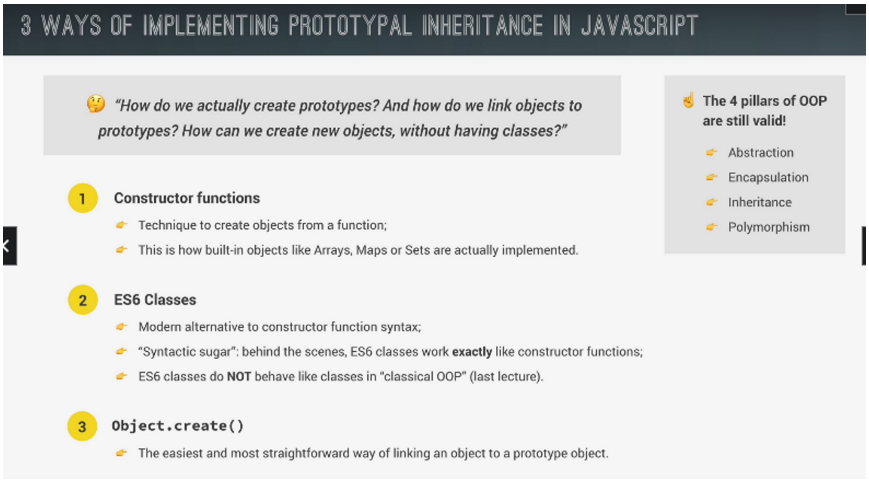
INTERSECTION OBSERVER

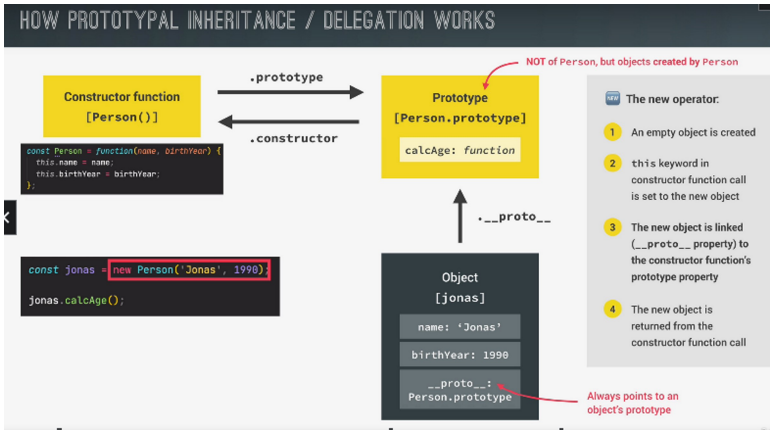
```
const stickyNav = function (entries) {
  const [entry] = entries;
  console.log(entry);
  if (!entry.isIntersecting) nav.classList.add('sticky');
  else nav.classList.remove('sticky');
  // nav.classList.add('sticky');
};
// const header = document.querySelector('.header');
const headerObserver = new IntersectionObserver(stickyNav, {
  root: null,
  threshold: 0,
  rootMargin: `-${navHeight}px`,
});
headerObserver.observe(header);
```

```
// Menu fade animation
const handleHover = function (e, opacity) {
  const siblings = e.target.closest('.nav').querySelectorAll('.nav__link');
  siblings.forEach(el => {
    if (el !== e.target) el.style.opacity = opacity;
    logo.style.opacity = opacity;
  });
};
nav.addEventListener('mouseover', function (e) {
  handleHover(e, 0.5);
});
nav.addEventListener('mouseout', function (e) {
  handleHover(e, 1);
});
```

20 - JULY - 2022







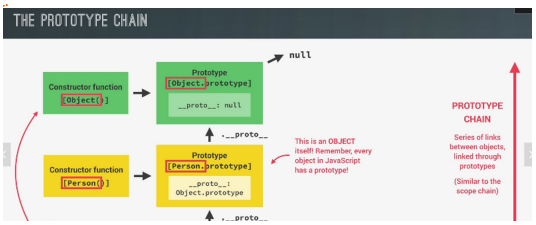
PROTOTYPE INHERITANCE

Functions inside constructor vs prototype

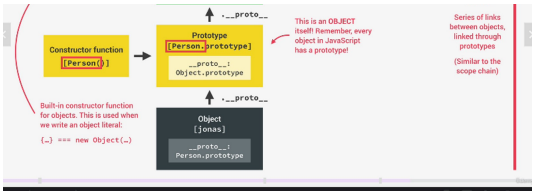
From <https://stackoverflow.com/questions/30487565/functions-inside-constructor-vs-prototype>

I perform a quick test. If you declare function in the constructor, two object instances have different function instances even after optimizations. However with prototype, you have only one instance of the function which explains the performance difference.

```
var Person = function () {
    var self = this;
    self.firstName = null;
    self.lastName = null;
    self.fullName = function () {
        return self.firstName + self.lastName;
    };
};

Person.prototype.fullName2 = function () {
    return this.firstName + this.lastName;
```



```
        self.fullName = function () {
            return self.firstName + self.lastName;
        };
    };

    Person.prototype.fullName2 = function () {
        return this.firstName + this.lastName;
    };

    var a = new Person();
    var b = new Person();

    console.log(a.fullName == b.fullName); // returns false
    console.log(a.fullName2 == b.fullName2); // returns true
```



IF WE HAVE THE FUNCTION INSIDE THE CONSFTRUCTOR FUNCTION THEN, EACH OBJECT WILL HAVE ITS OWN INSTANCE OF THAT FUNCTIONS …….. BUT WE MAKE THE FUNCTION IN THE PROTOTYPE THEN EACH OBJECT WILL NOT HAVE DIFFERENT INSTANCES OF THAT FUNCTION …. RESULTING IN THE PERFORMANCE IMPROVEMENT…

*** Prototype property of the constructor is available for all the objects made from that constructor.

>CONSTRUCTOR FUNCTIONS

1. Making a constructor function … uses this keyword to initialize the values.
2. Then make objects using …. New keyword.
3. Then use .prototype keyword to append any function in the constructor

>ES6 Classes

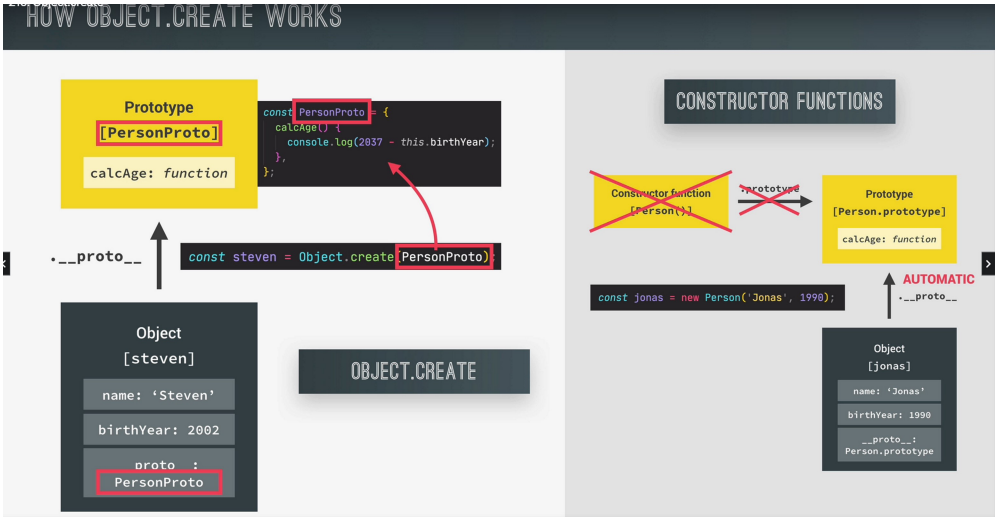
```
class PersonCl {
  constructor(firstName, birthYear) {
    this.firstName = firstName;
    this.birthYear = birthYear;
  }
  calcAge() {
    console.log('Khud nikalo boss 🤷');
  }
}
const Umair = new PersonCl('Umair', 1999);
console.log(Umair);
Umair.calcAge();
```

```
// Class declaration
class PersonCl {
  constructor(fullName, birthYear) {
    this.fullName = fullName;
    this.birthYear = birthYear;
  }
  // Instance methods
  // Methods will be added to .prototype property
  calcAge() {
    console.log(2037 - this.birthYear);
  }
  greet() {
    console.log(`Hey ${this.fullName}`);
  }
  get age() {
    return 2037 - this.birthYear;
  }
  // Set a property that already exists
  set fullName(name) {
    if (name.includes(' ')) this._fullName = name;
    else alert(`${name} is not a full name!`);
  }
  get fullName() {
    return this._fullName;
  }
  // Static method
  static hey() {
    console.log('Hey there 👋');
    console.log(this);
  }
}
```

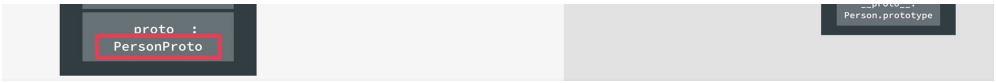
INSTANCE METHODS

STATIC METHODS

- They can be called directly .. Like PersonCL.hey() …. i.e without making objects unlike instance methods.



Steven is the object of PersonProto …
& PersonProto is prototype



Steven is the object of PersonProto …
& PersonProto is prototype

21 - JULY - 2022

OOP In JS

Inheritance

- To make the data member/member function protected, we use '_' sign before that variable/function name.

PUBLIC FIELDS: Created in the class, before/other than Constructor().
PRIVITE FIELDS: Used '#' symbol to make it private.
PRIVITE METHODS : // Same as above!.
Static Methods : Use 'static' keyword before.

- `public` scope to make that property/method available from anywhere, other classes and instances of the object.
- `private` scope when you want your property/method to be visible in its own class only.
- `protected` scope when you want to make your property/method visible in all classes that extend current class including the parent class.

you don't use any visibility modifier, the property / method will be public.

/ore: (For comprehensive information)

- PHP Manual - Visibility

hare Improve this answer Follow

edited Jul 22, 2020 at 5:52 Mathias Bader 3,264 ● 6 ● 37 ● 58

answered Dec 5, 2010 at 22:17 Sarfraz 369k ● 73 ● 529 ● 573

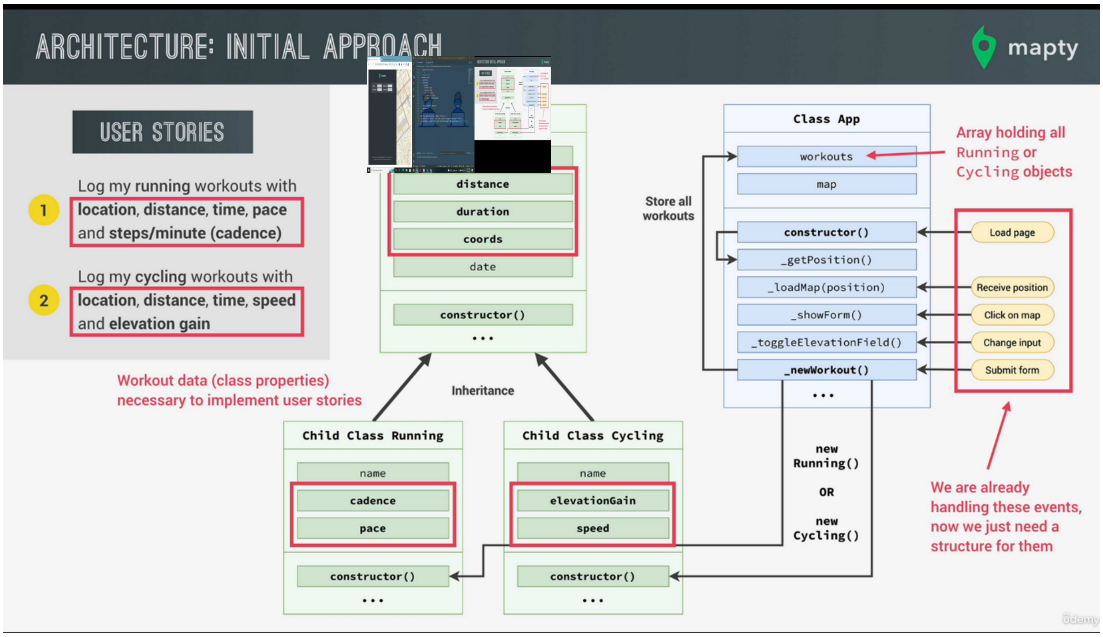
8 `protected` scope when you want to make your variable/function visible in all classes that extend current class AND its **parent classes**. – Shahid May 2, 2012 at 8:24

CLASSES

```
class StudentCl extends PersonCl {
  constructor(fullName, birthYear, course) {
    super(fullName, birthYear);
    this.course = course;
  }
  introduce() {
    console.log('Introduce');
  }
}
const hamza = new StudentCl('M. Hamza', 2018, 'CE');
hamza.introduce();
hamza.calcAge();
```

MAPTY PROJECT

22 - JULY - 2022



LOCAL STORAGE

```
_setLocalStorage() {
    localStorage.setItem('workouts', JSON.stringify(this.#workouts));
  }
  _getLocalStorage() {
    const data = JSON.parse(localStorage.getItem('workouts'));
    if (!data) return;
    this.#workouts = data;
    console.log(this.#workouts);
    this.#workouts.forEach(work => {
      this._renderWorkout(work);
    });
  }
```

```
reset() {
    localStorage.removeItem('workouts');
    location.reload();
  }
```

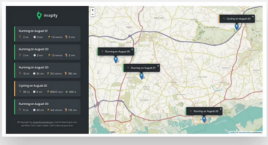
Asynchronous JavaScript:

When we call the event which is using the event or something that isn't yet loaded.

10 ADDITIONAL FEATURE IDEAS: CHALLENGES

mapty

- Ability to **edit** a workout;
- Ability to **delete** a workout;
- Ability to **delete all** workouts;
- Ability to **sort** workouts by a certain field (e.g. distance);
- **Re-build** `Running` and `Cycling` objects coming from Local Storage;
- More realistic error and confirmation **messages**;
- Ability to position the map to **show all workouts** [very hard];
- Ability to **draw lines and shapes** instead of just points [very hard];
- **Geocode location** from coordinates ("Run in Faro, Portugal") [only after asynchronous JavaScript section];
- **Display weather** data for workout time and place [only after asynchronous JavaScript section].

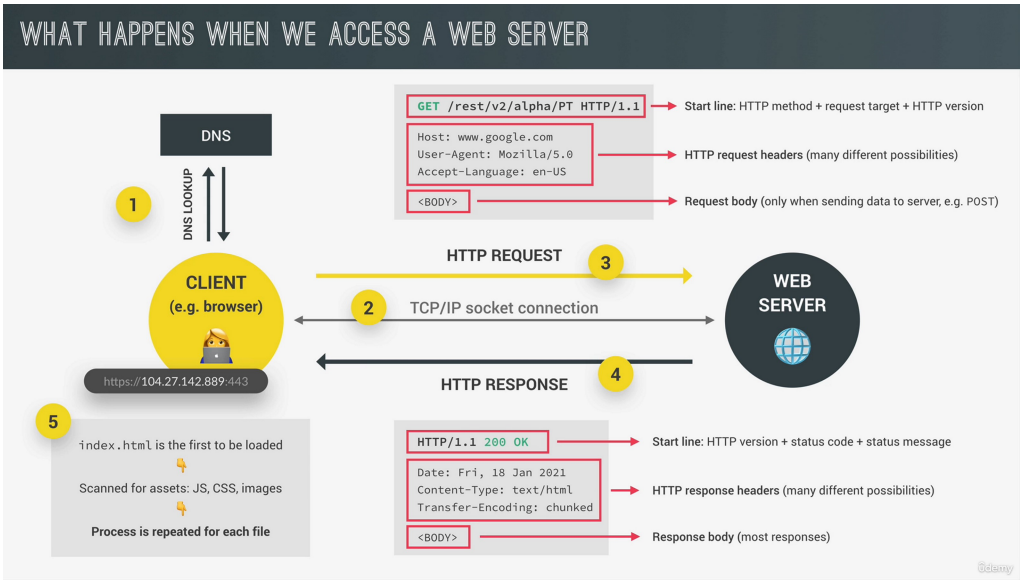


FINDING CHILD NODES

```
let errorMsg = '';
    e.target.childNodes.forEach(cn =>
      cn.className == 'errorMsg' ? (errorMsg = cn) : ''
    );
```

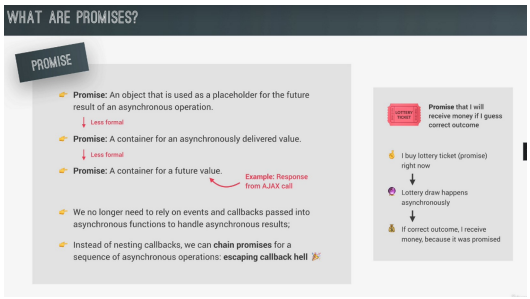
https://restcountries.com/v2/

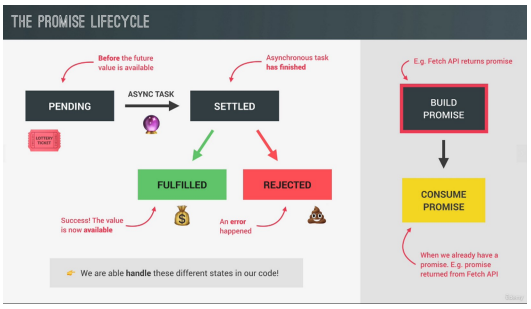
From <https://www.udemy.com/course/the-complete-javascript-course/learn/lecture/28841478#announcements>



CALL BACK HELL

When there are lot of Callbacks inside callback... for asynchronous js






```
const request = fetch(`https://restcountries.com/v2/name/pakistan`);
console.log(request);
```

```
▼Promise {<pending>}
  ▶[[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
  ▶[[PromiseResult]]: Response
```

```
// Handling FULFILLED promise
// .json is the async function ( it will return promise )
// prettier-ignore
fetch(`https://restcountries.com/v2/name/${country}`)
.then(response => response.json()
.then(data => renderCountry(data[0]))
);
};
```

```
// CHAINING PROMISES
const getCountryData = function (country) {
  // Handling FULFILLED promise
  // .json is the async function ( it will return promise )
  fetch(`https://restcountries.com/v2/name/${country}`)
    .then(response => response.json())
    .then(data => {
      renderCountry(data[0]);
      const neighbour = data[0].borders[0];
      if (!neighbour) return;
      //   Country 2 ( NEIGHBOUR )
      return fetch(`https://restcountries.com/v2/alpha/${neighbour}`);
    })
    .then(response => response.json())
    .then(data => renderCountry(data, 'neighbour'));
};
getCountryData('usa');
```

28 - JULY - 2022

This will catch all types of errors in THIS CHAIN!!!

```
.catch(err =>
      renderError(`Something went wrong 🤔 ${err.message}. Try AGAIN!`)
    );
```

```
.finally(() => {
    countriesContainer.style.opacity = 1;
  });
```

Applying finally after the catch..... (It will work when, catch Will return promise).

Throwing the custom error message

```
.then(response => {
      console.log(response);
      if (!response.ok)
        throw new Error(`Country not found!(${response.status})`);
      return response.json();
    })
```

1 - AUGUST - 2022

PROMISES vs Try_CATCH

You should **use Promises only for asynchronous functions and nothing else**. Do not abuse them as an error monad, that would be a waste of resources and their inherent asynchrony will make everything more cumbersome. When you have synchronous code, use try / catch for exception handling.

From <https://www.google.com/search?q=Difference+between+Try+catch+and+promises&oq=Difference+between+Try+catch+and+promises&aqs=chrome..69i57j0i5129.10734j0j7&sourceid=chrome&ie=UTF-8>

```
// Building a simple promise
const lotteryPromise = new Promise(function (resolve, reject) {
  setTimeout(function () {
    if (Math.random() >= 0.5) {
      resolve('Win');
    } else {
      reject(new Error('You lost'));
    }
  }, 1000);
});
lotteryPromise.then(res => console.log(res)).catch(err => console.error(err));
```

3 - AUGUST - 2022

PROMISES

CODING CHALLENGE

```
const wait = function (seconds) {
  return new Promise(function (resolve) {
    setTimeout(resolve, seconds * 1000);
  });
};

const images = document.querySelector('.images');

const createImage = function (imgPath) {
  return new Promise(function (resolve, reject) {
    const img = document.createElement('img');
    img.src = imgPath;
    img.addEventListener('load', function () {
      images.append(img);
      resolve(img);
    });
    img.addEventListener('error', function () {
      reject(new Error('Image not found 🐱'));
    });
  });
};

document.querySelector('.btn-country').style.display = 'none';
const imgPath = 'img/img-1.jpg';
let currentImg;

createImage(imgPath)
  .then(img => {
    currentImg = img;
    console.log(`${img.src.replace(/^.*[\\/]/, '')} Loaded !!
`);
```

Wait function to make the delay

This is the createImage function which is creating the function and it will return the Promise.

Promise will contain ..... RESOLVE and REJECT .....
RESOLVE -> promised to return the image. (as we send the img as an argument)
REJECT -> it will return an error.

```
Live reload enabled.                    (index):55
img-1.jpg Loaded !!                     challenge2.js:31
img-2.jpg Loaded !!                     challenge2.js:40
>
```

```
let currentImg;

createImage(imgPath)
  .then(img => {
    currentImg = img;
    console.log(`${img.src.replace(/^.*[\\\/]/, '')} Loaded !!
`);
    return wait(2);
  })

  .then(() => {
    currentImg.style.display = 'none';
    return createImage('img/img-2.jpg');
  })
  .then(img => {
    currentImg = img;
    console.log(`${img.src.replace(/^.*[\\\/]/, '')} Loaded !!
`);
    return wait(2);
  })
  .then(() => {
    currentImg.style.display = 'none';
  })
  .catch(err => console.log(`ERROR FOUND : ${err}`));
```

.then and .catch are used to get the resolved and rejected calls from the promise.

TO GET THE PROMISE RESPONSE, we use the .THEN ....
TO GET THE PROMISE REJECTION, we use the .CATCH .....

await waits for async to resolve or reject.

The try...catch...finally statements combo handles errors without stopping JavaScript.
The try statement defines the code block to run (to try).
The catch statement defines a code block to handle any error.
The finally statement defines a code block to run regardless of the result.
The throw statement defines a custom error.
Both catch and finally are optional, but you must use one of them.

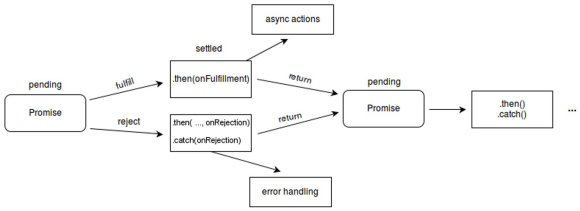
From <https://www.w3schools.com/jsref/jsref_try_catch.asp>

```
(async function () {
  try {
    const city = await whereAmI('Pakistan');
    console.log(`1: ${city}`);
  } catch (err) {
    console.error(`2: ${err.message}`);
  } finally {
    console.log(`3: DONE!`);
  }
})();
```

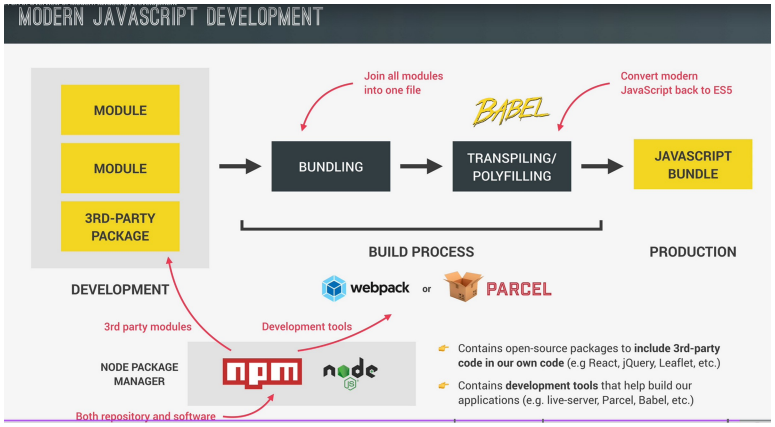

```
(async function () {
  try {
    const city = await whereAmI('Pakistan');
    console.log(`1: ${city}`);
  } catch (err) {
    console.error(`2: ${err.message}`);
  } finally {
    console.log(`3: DONE!`);
  }
})();
```

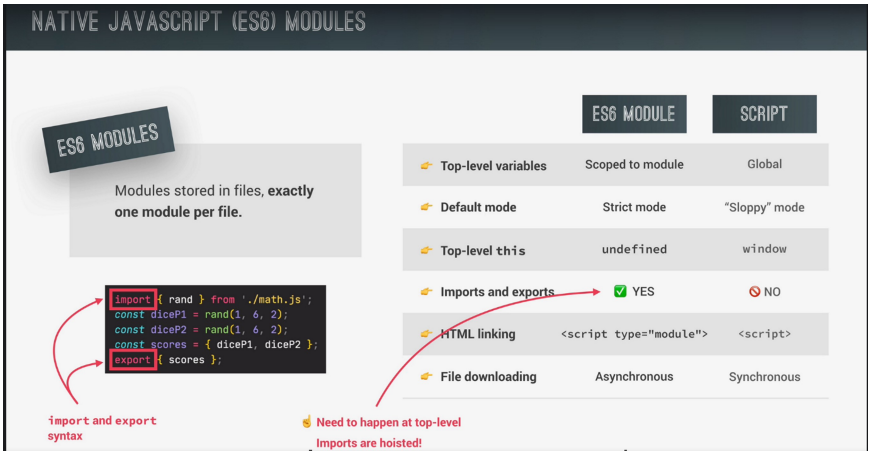

PROMISE.all()

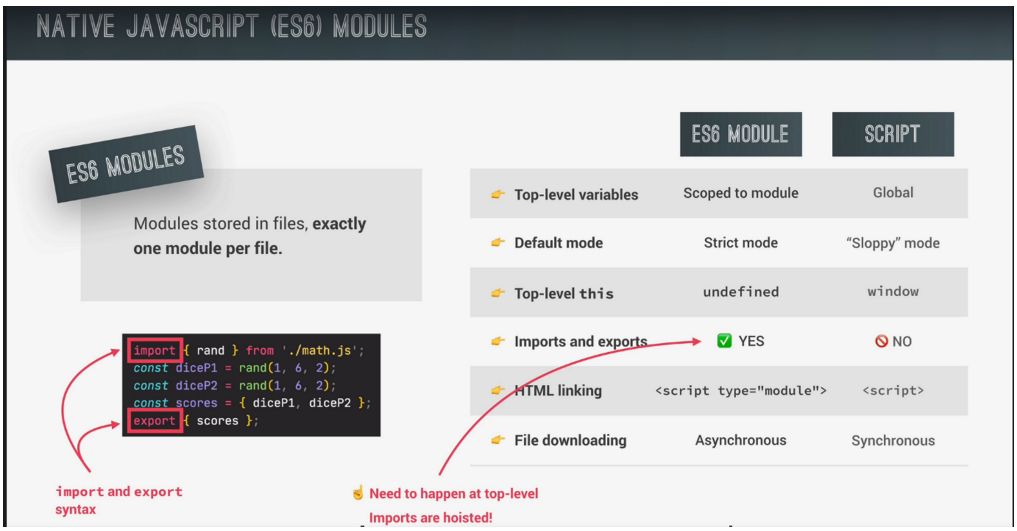
```
const data = await Promise.all([
    getJSON(`https://restcountries.eu/rest/v2/name/${c1}`),
    getJSON(`https://restcountries.eu/rest/v2/name/${c2}`),
    getJSON(`https://restcountries.eu/rest/v2/name/${c3}`),
  ]);
  console.log(data.map(d => d[0].capital));
} catch (err) {
  console.error(err);
}
};
get3Countries('portugal', 'canada', 'tanzania');
```



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise


MODERN JAVASCRIPT DEVELOPMENT
MODULE
MODULE
3RD-PARTY PACKAGE
DEVELOPMENT
Join all modules into one file
BUNDLING
BABEL
Convert modern JavaScript back to ES5
TRANSPILING/ POLYFILLING
JAVASCRIPT BUNDLE
BUILD PROCESS
PRODUCTION
webpack or PARCEL
3rd party modules
Development tools
NODE PACKAGE MANAGER
npm
node
Contains open-source packages to include 3rd-party code in our own code (e.g React, jQuery, Leaflet, etc.)
Contains development tools that help build our applications (e.g. live-server, Parcel, Babel, etc.)
Both repository and software


NATIVE JAVASCRIPT (ES6) MODULES
ES6 MODULES
Modules stored in files, exactly one module per file.
ES6 MODULE
SCRIPT
Top-level variables
Scoped to module
Global
Default mode
Strict mode
"Sloppy" mode
Top-level this
undefined
window
Imports and exports
YES
NO
HTML linking
<script type="module">
<script>
File downloading
Asynchronous
Synchronous
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
import and export syntax
Need to happen at top-level
Imports are hoisted!


NATIVE JAVASCRIPT (ES6) MODULES
ES6 MODULES
Modules stored in files, exactly one module per file.
ES6 MODULE
SCRIPT
Top-level variables
Scoped to module
Global
Default mode
Strict mode
"Sloppy" mode
Top-level this
undefined
window
Imports and exports
YES
NO
HTML linking
<script type="module">
<script>
File downloading
Asynchronous
Synchronous
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
import and export syntax
Need to happen at top-level
Imports are hoisted!

AVOID NESTED CODE

- Use early `return` (guard clauses)
- Use ternary (conditional) or logical operators instead of `if`
- Use multiple `if` instead of `if/else-if`
- Avoid `for` loops, use array methods instead
- Avoid callback-based asynchronous APIs

ASYNCHRONOUS CODE

- Consume promises with async/await for best readability
- Whenever possible, run promises in **parallel** (`Promise.all`)
- Handle errors and promise rejections

Two fundamentally different ways of writing code (paradigms)

IMPERATIVE

- Programmer explains **"HOW to do things"**
- We explain the computer *every single step* it has to follow to achieve a result
- **Example:** Step-by-step recipe of a cake

```
const arr = [2, 4, 6, 8];
const doubled = [];
for (let i = 0; i < arr.length; i++)
  doubled[i] = arr[i] * 2;
```

DECLARATIVE

- Programmer tells **"WHAT do do"**
- We simply *describe* the way the computer should achieve the result
- The **HOW** (step-by-step instructions) gets abstracted away
- **Example:** Description of a cake

```
const arr = [2, 4, 6, 8];
const doubled = arr.map(n => n * 2);
```

FUNCTIONAL PROGRAMMING

- **Declarative** programming paradigm
- Based on the idea of writing software by combining many **pure functions**, avoiding **side effects** and **mutating** data
- **Side effect:** Modification (mutation) of any data **outside** of the function (mutating external variables, logging to console, writing to DOM, etc.)
- **Pure function:** Function without side effects. Does not depend on external variables. **Given the same inputs, always returns the same outputs**.
- **Immutability:** State (data) is **never** modified! Instead, state is **copied** and the copy is mutated and returned.
- **Examples:** React, Redux

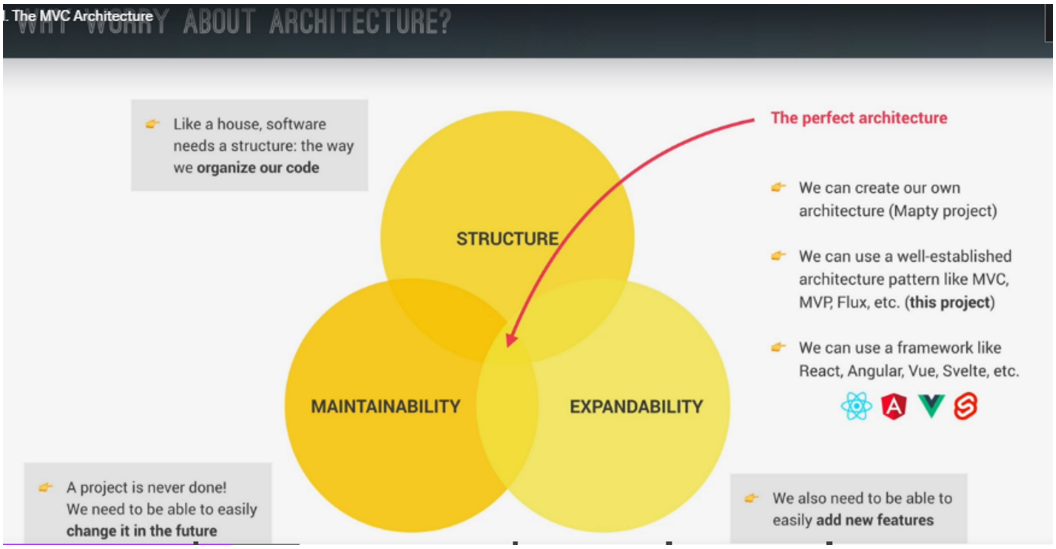
FUNCTIONAL PROGRAMMING TECHNIQUES

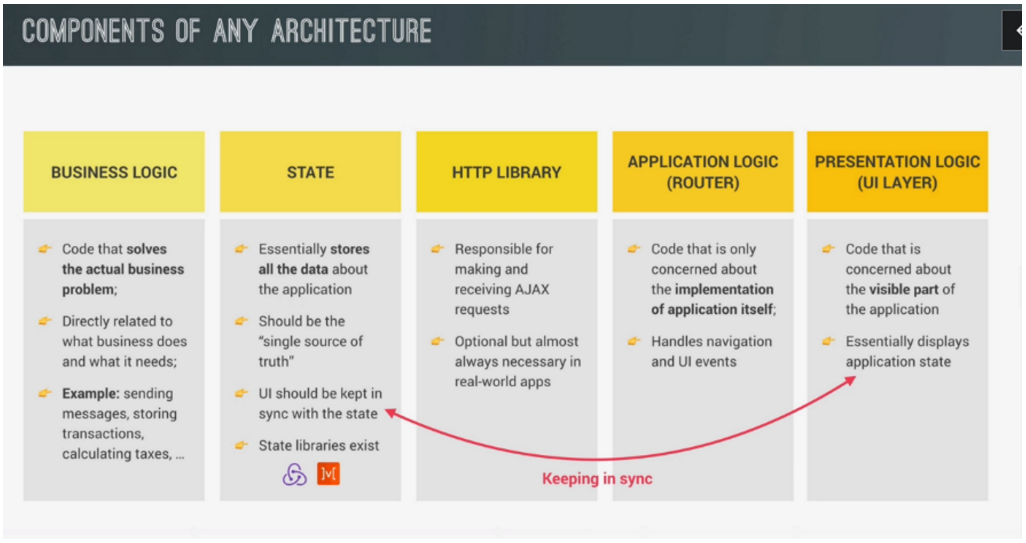
- Try to avoid data mutations
- Use built-in methods that don't produce side effects
- Do data transformations with methods such as `.map()`, `.filter()` and `.reduce()`
- Try to avoid side effects in functions: this is of course not always possible!

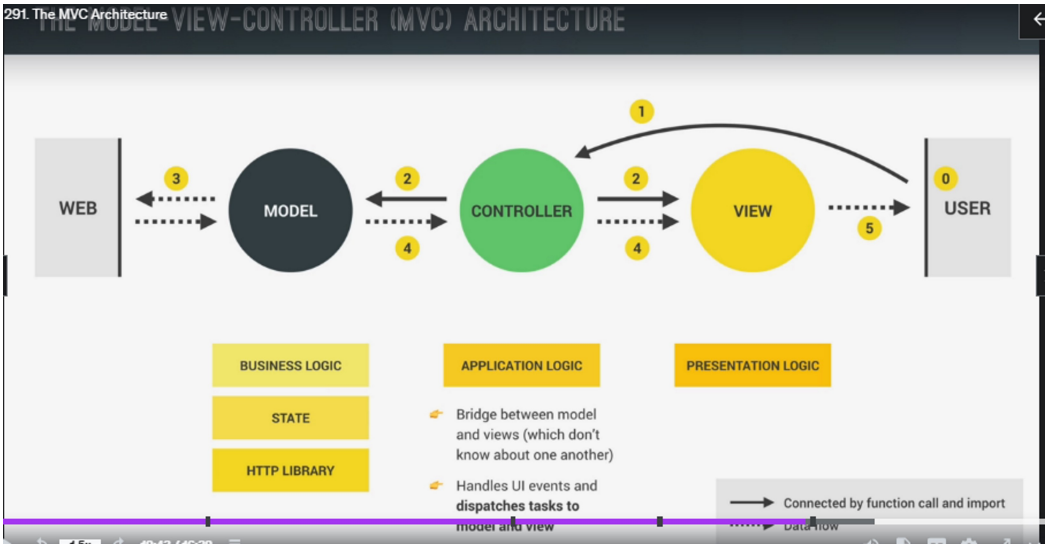
DECLARATIVE SYNTAX

- Use array and object destructuring
- Use the spread operator (`...`)
- Use the ternary (conditional) operator
- Use template literals

5 - AUGUST - 2022


1. The MVC Architecture
WHY WORRY ABOUT ARCHITECTURE?
Like a house, software needs a structure: the way we organize our code
STRUCTURE
MAINTAINABILITY
EXPANDABILITY
The perfect architecture
We can create our own architecture (Mapty project)
We can use a well-established architecture pattern like MVC, MVP, Flux, etc. (this project)
We can use a framework like React, Angular, Vue, Svelte, etc.
A project is never done! We need to be able to easily change it in the future
We also need to be able to easily add new features


COMPONENTS OF ANY ARCHITECTURE
BUSINESS LOGIC
Code that solves the actual business problem;
Directly related to what business does and what it needs;
Example: sending messages, storing transactions, calculating taxes, ...
STATE
Essentially stores all the data about the application
Should be the "single source of truth"
UI should be kept in sync with the state
State libraries exist
HTTP LIBRARY
Responsible for making and receiving AJAX requests
Optional but almost always necessary in real-world apps
APPLICATION LOGIC (ROUTER)
Code that is only concerned about the implementation of application itself;
Handles navigation and UI events
PRESENTATION LOGIC (UI LAYER)
Code that is concerned about the visible part of the application
Essentially displays application state
Keeping in sync


291. The MVC Architecture
THE MODEL-VIEW-CONTROLLER (MVC) ARCHITECTURE
WEB
MODEL
CONTROLLER
VIEW
USER
BUSINESS LOGIC
STATE
HTTP LIBRARY
APPLICATION LOGIC
Bridge between model and views (which don't know about one another)
Handles UI events and dispatches tasks to model and view
PRESENTATION LOGIC
Connected by function call and import
Data flow

15 - AUGUST - 2022

Rule - AWAIT KEYWORD

- Don't forget to use the 'await' keyword while fetching the async functions like Fetch etc.

17 - AUGUST - 2022

```
  newElements.forEach((newEl, i) => {
    const curEl = curElements[i];
    console.log(curEl, newEl.isEqualNode(curEl));

    if (!newEl.isEqualNode(curEl) &&) {
      curEl.textContent = newEl.textContent;
    }
  });
}
```

.EqualNode() is used for checking the equality of 2 DOM Elements.

PUBLISHER SUBSCRIBER PATTERN

```
const init = function () {
  BookmarksView.addHandlerRender(controlBookmarks);
  RecipeView.addHandlerRender(controlRecipes);
  SearchView.addHandlerSearch(controlSearchResults);
  RecipeView.addHandlerAddBookmark(controlAddBookmark);
  PaginationView.addHandlerClick(controlPagination);
};
```

18 - AUGUST - 2022

API KEYS
Get from: https://forkify-api.herokuapp.com/v2

98cc99e0-6bf6-4b4b-a19c-a39d4f8c5a6b

From <https://forkify-api.herokuapp.com/v2>

THE END