JS

▼ What does const mean?

it does not mean that the variable is a constant, it just means that you cannot re-assign it

e.g. given an object: the variable is always an object, but its content (properties) can change

▼ With const you can update...

the property value of an object

▼ What is dot notation?

To read or update the value of a property, you can use the dot notation.

e.g. user.age

▼ What is an object?

a data type that allows you to group several variables together into one variable that contains keys and values.

▼ What is a parameter?

a variable in a function definition

▼ What are arguments?

When a function is called, the arguments are the data you pass into the method's parameters.

▼ What happens when you call a function without providing a value for an expected argument?

it will default to undefined

▼ What are two benefits of arrow functions?

- It uses lexical scope
- It can benefit from implicit return

▼ What is arrow function syntax?

Arrow functions always start with the parameters, followed by the arrow ⇒ and then the function body.

▼ What does arrow function syntax look like?

```
const sum = (a, b) => {
    return a + b;
}

// one line example
const tripleIt = value => value * 3;
```

▼ How to write a function by defining a function and giving it a name?

```
function sum(a, b) {
    return a + b;
}
```

▼ How to write a function by defining a variable and assigning it to an anonymous function?

```
const sum = function(a, b) {
    return a + b;
}
```

▼ How to write the Array forEach() with arrow function?

```
grades.forEach(grade => {
    console.log(grade);
});
```

▼ How to write the Array filter() with arrow function?

```
let numbersAboveTen = numbers.filter((number) => {
    return number >= 10;
});
```

▼ What is a callback function?

a "call-after" function, is any executable code that is passed as an argument to other code

▼ How can you do an implicit return that isn't undefined?

1. Your function should be an arrow function.
2. The function body should be only **one line**. This means you have to remove the curly braces.
3. You have to remove the `return` keyword because the function body is one line.

```
const sum = (a, b) => a + b;

sum(1, 3); // 4
```

▼ How to write Array.filter(callback) as a one-liner?

```
const numbers = [9, 5, 14, 3, 11];

const numbersAboveTen = numbers.filter(function(number) {
    return number >= 10;
});
console.log(numbersAboveTen); // [14, 11]

// instead
const numbersAboveTen = numbers.filter(number => number >= 10);
```

▼ How to write Array.find(callback) as a one-liner?

```
const names = ["Sam", "Alex", "Charlie"];

const result = names.find(function(name) {
  return name === "Alex";
});
console.log(result); // "Alex"

// instead
const result = names.find(name => name === "Alex");
```

▼ How to write Array.map(callback) as a one-liner?

```
const numbers = [4, 2, 5, 8];

const doubled = numbers.map(function(number) {
    return number * 2;
});
console.log(doubled); // [8, 4, 10, 16]

// instead
const doubled = numbers.map(number => number * 2);
```

▼ How to write Array.map(callback) with a conditional?

```
export const getRaisedGrades = grades => {
    // return all the grades raised by 1 (grades should not exceed 20)
    return grades.map(grade => {
        if (grade < 20) {
            grade = grade + 1;
        }
        return grade;
    });
}
```

▼ Reduce sum example

```
export const getSumGrades = grades => {
    // return the sum of all the grades
    return grades.reduce((a, b) => a + b, 0);
}
```

▼ What does String.trim() do?

String .trim() removes all leading and trailing space characters.

▼ What does String.startsWith() do?

String .startsWith(substring) returns true when the substring is found at the beginning of the string, and false otherwise.

▼ What does String.endsWith() do?

String .endsWith(substring) returns true when the substring is found at the end of the string, and false otherwise.

▼ What does String.includes() do?

String .includes(substring) returns true when the substring can be found anywhere in the string, and false otherwise

▼ What does String .split(separator) do?

The .split(separator) method divides the string into an array by splitting it with the separator you provide.

```
let apps = "Calculator,Phone,Contacts";
let appsArray = apps.split(",");
console.log(appsArray); // ["Calculator", "Phone", "Contacts"]
```

▼ What does Array.join(glue) do?

It joins an array into a string on a glue character e.g. a space

▼ What does String.replace(search, replace) do?

The .replace(search, replace) method returns a new string where the **first** occurrence of the search parameter you provide is replaced by the replace parameter.

```
const message = "You are welcome.";
message.replace(" ", "_"); // "You_are welcome";
console.log(message); // "You are welcome" (original string is not changed)
```

▼ What does String.replaceAll(search, replace) do?

It replaces **all** occurrences of the search parameter.

```
const message = "You are welcome.";
message.replaceAll(" ", "_"); // "You_are_welcome";
console.log(message); // "You are welcome" (original string is not changed)
```

▼ What is a slug?

In computer science, a *slug* is a string used to identify a certain item.

Oftentimes, this *slug* is used in the URL for Search Engine Optimization and better user experience.

Let's say you've got a product called: `"Easy assembly dining table"`. We cannot use this name in the URL bar because it contains spaces (it won't look nice, for example `https://example.com/item/Easy assembly dining table`).

This is why we use a slug that looks like this:

`easy-assembly-dining-table` so the URL becomes: `https://example.com/item/easy-assembly-dining-table`.

▼ How to generate an HTML string from an array?

```
const html = `<ul>
    ${users.map(user => `<li>${user.name}</li>`).join("")}
    </ul>`;
console.log(html); // <ul> <li>Sam Doe</li><li>Alex Blue</li> </ul>
```

What's very important here is the `.join("")`. If you forget this, you will get the following HTML:

```
<ul><li>Sam Doe</li>,<li>Alex Blue</li></ul>
```

That's because the array returned from .map() will automatically be converted to a string by the browser.

▼ What does Array.every(callback) do?

The Array .every(callback) method returns true when **every item in the array satisfies the condition** provided in the callback.

```
const numbers = [15, 10, 20];

const allAbove10 = numbers.every(number => number >= 10); // true
const allAbove15 = numbers.every(number => number >= 15); // false
```

▼ What does Array.some(callback) do?

The Array .some(callback) method returns true when **at least one item in the array satisfies the condition** provided in the callback.

```
const numbers = [15, 10, 20];

const someOver18 = numbers.some(number => number >= 18); // true
const someUnder10 = numbers.some(number => number < 10); // false
```

▼ How can you empty an array by changing its length?

```
const items = ["Pen", "Paper"];
items.length = 0;

console.log(items); // []
```

▼ What does Array.splice(start[, deleteCount]) do?

The .splice(start[, deleteCount]) method removes items from the array starting from the start index that you specify.

If no deleteCount is provided, it will remove all the remaining items as of the start index.

the deleteCount parameter is optional

When you specify a deleteCount, then it will remove as many items as you provided in the deleteCount from the start index.

▼ What does the reduce() method do?

The goal of the reduce() method is to calculate a single value from an array.

In other terms, you reduce an array into a single value.

e.g. summing an array: We can reduce the array [5, 10, 5] to the number 20.

▼ What is the reducer?

a callback which allows you to configure the logic of how the array will be reduced into a single number

▼ What arguments does reduce() take?

The reducer (callback that reduces array to a single number) and the initial value

▼ What does reduce() look like?

```
const grades = [10, 15, 5];

const sum = grades.reduce((total, current) => {
    return total + current;
}, 0);

// The total is always referring to the last computed value by the reduce function.
// total aka accumulator
```

▼ How to use reduce() to multiply?

```
const numbers = [5, 2, 10];

const result = numbers.reduce((total, current) => {
    return total * current;
}, 1);
```

▼ What is NaN?

Not a Number

▼ What does array destructuring syntax look like?

```
const dimensions = [20, 5]

// create variables
const [width, height] = dimensions;

// log them
console.log(width); //20
console.log(height); //5

// destructuring from an object:

const getLatLng = userLocation => {
    //destructure into 2 variables: lat & lng
    const {lat, lng} = userLocation;
    return `The latitude is ${lat} and the longitude is ${lng}`;
}

const userLocation = {
    lat: 24.235235,
    lng: 2.5734,
};

// The latitude is 24.235235 and the longitude is 2.5734
```

▼ What does array destructuring syntax look like in React hooks?

```
const [counter, setCounter] = useState(0);
```

▼ How can you concatenate/merge several arrays into a new array using the ... spread syntax?

```
const lat = [5.234];
const lng = [1.412];
const point = [...lat, ...lng];
console.log(point); // [5.234, 1.412];

const items = ["Tissues", "Oranges"];
const otherItems = [...items, "Tomatoes"];
console.log(otherItems); // ["Tissues", "Oranges", "Tomatoes"]
```

▼ Is JavaScript a dynamic language?

Even statically typed languages can have a dynamic or variant data type that can contain different data types.

JavaScript is called a dynamic language because it doesn't just have a few dynamic aspects, pretty much everything is dynamic.

▼ What does the Object global variable contain?

methods that are relevant to objects

e.g. Object.keys(user);

▼ What does Object.keys(obj) method do?

```
const user = {
    id: 1,
    name: "Sam Green",
    age: 20
};

const keys = Object.keys(user);
console.log(keys); // ["id", "name", "age"]
```

▼ What does Object.values(obj) method do?

```
Object.values(someObject).forEach(val => console.log(val));
// this will console.log() every value in the given object.
```

Object.values() returns an array whose elements are the enumerable property values found on the object.

▼ Why do you get this error?: Uncaught TypeError: Cannot read property 'toUpperCase' of undefined

This is one of the **most common errors** that you will see in JavaScript. **TypeScript** does a great job at preventing this kind of errors, though it comes with its own overhead.

It's important to be able to read this error message and understand that the issue is not `.toUpperCase()` but instead, the expression that came before it `person.age`.That's because you end up calling `undefined.toUpperCase()` which does not exist.

```
person.age.toUpperCase();
```

▼ What has happened when you see [object Object]?

If you see [object Object], it means you tried to use an object in a context that expects a string

```
const person = {
    id: 1,
    firstName: "Sam"
};

console.log(`Hello ${person}`); // "Hello [object Object]"
```

▼ What does Object.entries(obj) do?

returns an array of arrays representing every key/value pair.

```
const user = {
    id: 1,
    name: "Sam Green",
    age: 20
};

const entries = Object.entries(user);

[
    ["id", 1],
    ["name", "Sam Green"],
    ["age", 20]
]
```

▼ When you access a property that does not exist on an object, you will get...?

undefined

▼ What does Object shorthand look like?

```
const age = 18;
const person = {
    name: "John",
    age
}
```

```
const isAdmin = false;
const darkMode = true;

const settings = {
    isAdmin,
    darkMode
};

console.log(settings); //{isAdmin: false, darkMode: true}
```

Because the property name is the same as the name of the variable used as its value, then you can drop the : age so you're left only with age.

▼ Awesome console.log and object shorthand debugging tip

```
const getProduct = (a, b) => {
    console.log({a, b});

    let product = a * b;
    console.log({product});

    return product;
}

getProduct(2, 3);
/*
{a: 2, b: 3}
{product: 6}
*/
```

▼ How to do object destructuring?

```
const config = {
    id: 1,
    isAdmin: false,
    theme: {
        dark: false,
        accessibility: true
    }
};

// instead of this
const id = config.id;
const isAdmin = config.isAdmin;
const theme = config.theme;

// do this
const {id, isAdmin, theme} = config;

// if only need some variables
const {isAdmin, theme} = config;
```

▼ How to destructure with a default value?

```
const user = {
    id: 1,
    name: "Sam"
};

const {name, isAdmin = false} = user;
console.log(isAdmin); // false
```

▼ How to concatenate objects with ... spread operator?

```
const firstPerson = {
    name: "Sam",
    age: 18
}

const secondPerson = {
    age: 25,
    type: "admin"
}

const mergedObjects = {...firstPerson, ...secondPerson};
console.log(mergedObjects); // {name: "Sam", age: 25, type: "admin"}
```

the order of the objects matters here. the second age persisted

▼ What is optional chaining?

```
// assuming object `user`

const name = user.details?.name?.firstName;
```

allows you to access a property deep within an object **without risking an error** if one of the properties is null or undefined

In case one of the properties is null or undefined, then the ?. will short-circuit to undefined and code won't break.

e.g. instead of something like: Cannot read property 'toUpperCase' of undefined, the code will return undefined.

Optional chaining is only used to access a **property** that may or may not exist. The object must exist.

Optional chaining is only used for reading a property. It cannot be used for assignment.

▼ What is optional chaining usage with arrays?

```
// Given:
const data = {
    temperatures: [-3, 14, 4]
}

let firstValue = undefined;
if (data.temperatures) {
    firstValue = data.temperatures[0];
}

// We can refactor to:
const firstValue = data.temperatures?.[0];
```

If the key temperatures is undefined: we use ?. in front of the [0] to access the first item of the array.

▼ What is optional chaining usage with functions?

```
// Given:
const person = {
    age: 43,
    name: "Sam"
};

let upperCasedName = person.name; // might be undefined
if (person.name) {
    upperCasedName = person.name.toUpperCase();
}
```

```
// We can refactor to:
const upperCasedName = person.name?.toUpperCase();
```

▼ What is nullish coalescing?

The nullish coalescing **?? operator** is a new operator introduced in JavaScript that allows you to **default to a certain value when the left-hand side is a nullish value (null or undefined)**.

This operator is useful to avoid showing undefined or null to the User Interface, which are often signs of bugs. e.g. can just show an empty string.

The nullish coalescing operator will **short-circuit** if the left-hand side returns a non-nullish value. This means that it will not execute the right-hand side.

```
const getName = name => {
    return name ?? "N/A";
}

console.log(getName("Sam")); // "Sam"
console.log(getName(undefined)); // "N/A"
console.log(getName(null)); // "N/A"

const sayHello = name => {
    return `Hello ${name ?? there}!`;
}
```

▼ What is a nullish value?

a value that is either null or undefined

▼ null vs. undefined?

undefined means that the value has not been defined yet.

null means that the **value has been defined but is empty**.

```
const user = {
    id: 1,
    name: "Sam",
    age: null
}

console.log(user.age); // null
console.log(user.birthday); // undefined
```

▼ How to refactor multiple if conditions/statements?

```
// original
const getPushMessage = status => {
    if (status === "received") {
        return "Restaurant started working on your order.";
    } else if (status === "prepared") {
        return "Driver is picking up your food."
    } else if (status === "en_route") {
        return "Driver is cycling your way!";
    } else if (status === "arrived") {
        return "Enjoy your food!";
    } else {
        return "Unknown status";
    }
}

// refactor
const getPushMessage = status => {
    const messages = {
```

```
        "received": "Restaurant started working on your order.",
        "prepared": "Driver is picking up your food.",
        "en_route": "Driver is cycling your way!",
        "arrived": "Enjoy your food!"
    };

    return messages[status] ?? "Unknown status";
}
```

▼ What is implicit conversion?

when type conversion occurs automatically

▼ What are falsy values?

Values that will be converted to false.

Full list of falsy values:

`false`	The keyword `false`.
`0`	The `Number` zero (so, also `0.0`, etc., and `0x0`).
`-0`	The `Number` negative zero (so, also `-0.0`, etc., and `-0x0`).
`0n`	The `BigInt` zero (so, also `0x0n`). Note that there is no `BigInt` negative zero — the negation of `0n` is `0n`.
`""`, `''`, ``` `` ```	Empty string value.
null	null — the absence of any value.
undefined	undefined — the primitive value.
NaN	NaN — not a number.

▼ What is the Logical NOT operator (!) ?

to convert a boolean value to its opposite, you can use the ! operator (Logical NOT operator)

```
!true; // false
!false; // true
```

▼ How to extract properties of an object into an array?

```
const tweets = [
    {
        id: 1080777336298049537,
        message: "Hello Twitter 👋",
        created_at: "2020-01-03 11:46:00"
    },
    {
        id: 1080777336298195435,
        message: "How do you keep track of your notes?",
        created_at: "2021-02-19 15:32:00"
    }
];

const messages = tweets.map(tweet => tweet.message);
console.log(messages);// ["Hello Twitter 👋", "How do you keep track of your notes?"]
```

```
const above30 = tweets.filter(tweet => tweet.stats.likes > 30);
```

▼ How to use array methods on an object?

```
const tweet = tweets.find(tweet => tweet.id === searchId);

tweets.some(tweet => tweet.stats.likes > 30); // true (at least one has more than 30 likes)

tweets.every(tweet => tweet.status.likes > 10); // true (all the tweets have more than 10 likes)
```

▼ How to convert an array of objects into a CSV string?

```
const csv = tweets.map(tweet => tweet.message).join(", ");

console.log(csv); // "Hello Twitter 👋, How do you keep track of your notes?"
```

▼ How to reduce() an array of objects?

```
const grades = [{grade: 10}, {grade: 15}, {grade: 5}];

const sum = grades.reduce((total, current) => {
    return total + current.grade;
}, 0);
```

▼ What does a try catch block do?

it allows us to recover from the error **without stopping the execution**

```
console.log("Step 1");

try {
    nonExistentFunction();
} catch (error) {
    console.error(error); // Uncaught ReferenceError: nonExistentFunction is not defined
}
```

▼ What is a code style?

Code style is the set of rules and guidelines that a certain team/person/company uses while developing a project.

▼ Are arrays are objects in JavaScript?

Yes

▼ What does [] === [] evaluate to and why?

false because it's comparing 2 difference instances of arrays

▼ What does the === operator compare given two objects?

Triple equal === is comparing the references rather than the values.

If you'd like to compare by values, then what you're looking for is called deep equal.

▼ What is Immutability?

An immutable object is an object that cannot be changed.

Every update creates a new value, leaving the old one untouched.

▼ What will this return? "25" === 25

it will return false. no type coercion here

▼ Why does assigning an array to a new variable not make a new copy of it?

Because its a reference

▼ What does using a shallow copy of that array using the spread syntax ... do?

it creates a new array. not a reference.

This technique covers most scenarios for array immutability as with this new array, you will be able to manipulate it without affecting the original array.

```
const grades = [10, 20];
const gradesCopy = [...grades];
console.log(gradesCopy); // [10, 20] (new array, not linked to 'grades')
```

▼ How to create a new array from an old one?

You can immutably update an array and immutably remove an item using the .map() and .filter() methods. The .filter() method returns a new array (so it does not affect the original one).

```
const grades = [10, 20];
const updated = [...grades, 15];
console.log(updated); // [10, 20, 15] (new array, not related to 'grades')


const grades = [10, 20, 15];
const updated = grades.filter(grade => grade !== 20);
console.log(updated); // [10, 15]
```

▼ How to clone an object?

```
const user = {
    id: 1,
    age: 23
};
const cloned = {...user};
console.log(cloned); // {id: 1, age: 23} (new object not related to 'user')
```

▼ How to immutably update an object?

```
const user = {
    id: 1,
    age: 23
};
const clonedUser = {
    ...user,
    age: user.age + 1
};
console.log(clonedUser); // {id: 1, age: 24} (new object not related to 'user')
```

▼ How to immutably delete part of an object?

The reason why this works is because {year, ...rest} = book is destructuring the value of the key year from the book object.

So we end up with rest which is an immutable copy of book excluding the year property.

```
const book = {
    id: 1,
    title: "Harry Potter",
    year: 2017,
    rating: 4.5
}

// GOOD: immutable
const {year, ...rest} = book;
console.log(rest); // { id: 1, title: "Harry Potter", rating: 4.5}
```

▼ How many levels deep is a shallow copy?

1 level deep. This means that for an array of objects if you make an update to the objects in the new array, the objects in the old array will still be updated.

i.e. the internal references will be changed/mutated.

▼ What is a class?

To better organize your code, you can group functions that perform similar functionalities into a single class.

A class is a factory that is able to create instances.

```
// create a new instance of Translation with the word "Table"
const firstTranslation = new Translation("Table");
firstTranslation.isEnglishWord(); //true

// create another instance of Translation with the word "España"
const secondTranslation = new Translation("España");
secondTranslation.isEnglishWord(); //false


class Translation {
       constructor(word) {
            // capture constructor param into instance variable
            // this is explained in the next lesson
            this.word = word;
       }

       isEnglishWord() {
            // returns true when word is English, false otherwise
       }

       isSpanishWord() {
            // returns true when word is Spanish, false otherwise
       }
}
```

▼ How to create an instance of a class?

use new keyword. const bob = new Person('bob');

▼ How to define a class?

use class keyword with name in UpperCamelCase

```
class Person {
  // I am creating an instance of the class
  constructor(firstName, lastName) {
        // capture firstName param into this.firstName instance variable
        this.firstName = firstName;
        // capture lastName param into this.lastName instance variable
        this.lastName = lastName;
    }

  // instance method
```

```
   getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }

    canVote() {
        return this.age >= 18;
    }

     getGreeting() {
        const fullName = this.getFullName(); // call an instance method
        return `Hello ${fullName}`
    }
}

// class usage
const person = new Person("Sam", "Green");
console.log(person.getFullName()); // "Sam Green"
```

▼ What is an instance variable?

An instance variable is a variable that belongs to a specific instance of a class.

▼ What is Object-Oriented Programming (OOP)?

a programming paradigm based on the concept of "objects"

▼ What is a static method?

A method called on the class not the class instance.

They cannot access instance variable or instance methods. Thus, you cannot use this. inside of them.

```
class Config {
    static getYear() {
        // code to get the current year (for example, 2021)
        const date = new Date();
        return date.getFullYear();
    }
}
```

▼ When to use a static method?

- Is the result of this method the same across all instances of the class? If yes, then it should be `static`.
- Is the method not accessing any instance variable of this class? If yes, then most likely it should be `static`.

▼ What is method chaining?

a method called on the result of another method

▼ Method chaining with classes must return what?

They must return this. in order for it to work

```
class Course {
    constructor(name, isCompleted) {
        this.name = name;
        this.isCompleted = isCompleted;
    }

    markAsCompleted() {
        this.isCompleted = true;
        return this; // allows method chaining
    }

    setGrade(grade) {
        this.grade = grade;
        return this; // allows method chaining
    }
```

```
    requestCertificate() {
        this.askedForCertificate = true;
        return this; // allows method chaining
    }
}

course.markAsCompleted().setGrade(18).requestCertificate();
```

▼ What is an example of class inheritance?

The extends Employee will allow Manager to inherit all the methods defined on Employee.

```
class Employee {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }

    getInitials() {
        return this.firstName[0] + this.lastName[0];
    }
}

class Manager extends Employee {
    sendPerformanceReview() {
        console.log(`Sent performance review for current quarter`);
    }
}
```

▼ What is an example of overriding functions with class inheritance?

When a "child" class inherits from a "parent" class, the "child" class will automatically get all the methods defined on the "parent" class. But you can override them.

```
class Manager extends Employee {
    // overrides the employee func
    getFullName() {
        return `${this.firstName} ${this.lastName} (manager)`;
    }

    sendPerformanceReview() {
        console.log(`Sent performance review for current quarter`);
    }
}
```

▼ What is the syntax for inheritance in JavaScript?

class Child extends Parent

▼ What is super()?

the super keyword is used to call functions on the parent class

super() calls the parent class' constructor in this example:

```
class Employee {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

```
    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    }

    getInitials() {
        return this.firstName[0] + this.lastName[0];
    }
}

class Manager extends Employee {
    constructor(firstName, lastName, department) {
        super(firstName, lastName); // super must be called first
        this.department = department;
    }

    sendPerformanceReview() {
        console.log(`Sent performance review for current quarter in ${this.department}`);
    }
}
```

▼ How can you call parent instance methods with super?

the getFullName() instance method calls the Employee's getFullName() with super.getFullName()

```
class Manager extends Employee {
    constructor(firstName, lastName, department) {
        super(firstName, lastName); // super must be called first
        this.department = department;
    }

    sendPerformanceReview() {
        console.log(`Sent performance review for current quarter in ${this.department}`);
    }

    getFullName() {
        return super.getFullName() + " [manager]";
    }
}
```

▼ What is the type of a class in JavaScript?

the type is function

why? class is syntactic sugar

A class is creating a function with the constructor pattern (has to be called with new)

▼ What is a function's prototype?

an object that contains all the instance methods that a certain function can have

By assigning a new function to Rectangle.prototype you are adding a new instance method to the instances of Rectangle

Every time you create a new Rectangle, that instance will have an instance method called isSquare which can access the instance properties via this. (for example, this.width and this.height

```
//This is the constructor
function Rectangle(width, height) {
    this.width = width;
    this.height = height;
}

// this is an instance method (that you can call on new instances of Rectangle)
// Note: this has to be a function (not an arrow function), will be explained later on in Lexical scope
Rectangle.prototype.isSquare = function() {
    return this.width === this.height;
}
```

▼ What is a class?

A class is creating a function with the constructor pattern (has to be called with new)

▼ What can you do with prototypal inheritance?

you can create any new object which combines methods from other objects

▼ Prototypal inheritance example

```
class Gorilla {
    // define methods here
}
class Banana {
    // define methods here
}
class GorillaBanana() {
    // do not use 'extends' because we'd like to choose the methods one by one
}

Gorilla.prototype.eat = function() {
    // ...
}

Banana.prototype.peel = function() {
    // ...
}

// Extend our GorillaBanana with the Gorilla's eat() method
GorillaBanana.prototype.eat = Gorilla.prototype.eat;

// Extend our GorillaBanana with Banana's peel() method
GorillaBanana.prototype.peel = Banana.prototype.peel;
```

▼ What happens when you use extends keyword in JavaScript?

JavaScript is copying behind the scenes all the methods into the prototype of the class aka type function

▼ Classical vs. prototypal inheritance

With classical inheritance, you inherit all of the parent's methods (more popular inheritance using extends keyword)

With prototypal inheritance, you can inherit specific functions by adding them to the .prototype.

▼ What does the extends keyword simulate?

When you use the extends keyword, JavaScript simulates classical inheritance by copying all the methods (by using prototypal inheritance).

▼ What is the prototype chain?

The chain of prototypal inheritance for a given object.

▼ Example of prototype chain

```
class Welcome {
    sayHello() {
        return "Hello World!";
    }
}

const welcome = new Welcome();

// the welcome prototype
Object.getPrototypeOf(welcome); // {constructor: fn(), sayHello: fn()}

// the Object prototype
```

```
Object.getPrototypeOf(Object.getPrototypeOf(welcome)); // {constructor: fn(), hasOwnProperty: fn(), isPrototypeOf: fn(), ...}

// if you walk up one more you get null
Object.getPrototypeOf(Object.getPrototypeOf(Object.getPrototypeOf(welcome))); // null
```

▼ In JavaScript, every object inherits from...

Object

▼ null and undefined do not inherit from...

Object

▼ What is __ proto __ ?

The __ proto __ that you see in the console is not part of the JavaScript language, however, all browsers have implemented it.

▼ Inheritance works in JavaScript because of...

the prototype chain.

▼ What are public class fields in JavaScript?

Public class fields allows you to define instance variables on a class without having to override the constructor.

has some browser support for this feature

```
class User {
    votingAge = 18; // public class field

    constructor() {
        console.log("Do something else here...");
    }
}
```

▼ How to create a private instance variable and private method?

Private class fields have to be defined outside of the constructor first

has some browser support for this feature

```
class User {
    #votingAge;

    constructor() {
        this.#votingAge = 18;
        this.#logAge();
    }

    // get a private instance variable
    get votingAge() {
        return this.#votingAge;
    }

    // The #logAge private method cannot be called from outside of the class
    #logAge() {
        console.log(this.age);
    }
}
```

▼ What is an asynchronous callback?

a callback that runs somewhere in the future

▼ What is execution order?

JavaScript code always runs top to bottom.

However, some parts of the code might be queued for the future aka asynchronous code.

▼ What is the purpose of a callback?

to schedule work for future performance

they allow the browser to continue responding to user input instead of blocking and waiting until a function has finished executing

▼ What is the callback pattern?

a programming pattern where you pass a function definition as a parameter to a function.

This function will then be automatically called once the function has been completed successfully.

▼ What is callback hell?

when the code becomes too complicated to read because it's too nested.

▼ Callback v. Promise

This is clearer because we don't have to rely on the order of the callbacks, instead, they are scheduled in the future with .then() and .catch() and one of them will run depending on the outcome of the promise.

```
// callback
sumTemperatures(temperatures, value => {
    console.log(value); // 18 (the sum of temperatures)
}, reason => {
    // this callback will run when there's an error
    console.error(reason);
});

// promise
sumTemperatures(temperatures)
    .then(value => {
        console.log(value); // 18 (the sum of temperatures)
    })
    .catch(reason => {
        // this callback will run when there's an error
        console.error(reason);
    });
```

▼ What is a promise?

a JavaScript feature that allows us to schedule work in the future and then runs callbacks based on the outcome of the promise (whether it was successful or not)

▼ What states can a promise have?

- `pending`
- `fulfilled`
- `rejected`

▼ When you create a promise, it will start in what state?

pending

▼ When a promise has been completed successfully, it will end in what state?

fulfilled

▼ Internal state of a promise example

```
const result = wait(1000);
console.log(result); // Promise {<pending>}
result.then(() => {
    console.log(result); // Promise {<fulfilled: undefined}
});
console.log(result); // Promise {<pending>} (because your code runs top to bottom.
//However, the promise callback gets scheduled into the future)
```

▼ What is a resolved promise?

When a promise completes successfully

▼ What is a promise resolving data?

This means that the promise is giving us an answer after it has been completed.

▼ How to extract a value out of a promise?

You have to add a .then() callback and you will only be able to access the data inside the .then callback.

This is because the promise callback will only run in the future (once the promise has been completed).

```
const data = getWeatherIn("Amsterdam").then(() => {
  console.log(data);
});
```

▼ How to use a promise?

```
// this is a promise
export const getWeatherDescription = (city) => {
    return new Promise((resolve, reject) => {
        if (!city || typeof city !== "string") {
            reject("City must be a string");
        }
        if (!["amsterdam", "tokyo"].includes(city.toLowerCase())) {
            reject("City must be Amsterdam or Tokyo");
        }
        // simulate network request
        setTimeout(() => {
            if (city.toLowerCase() === "amsterdam") {
                resolve("Cloudy");
            }
            if (city.toLowerCase() === "tokyo") {
                resolve("Sunny");
            }
        }, 1000);
    });
};


import {getWeatherDescription} from "./weather.js";

// this is using a promise
const logWeatherDescription = cityName => {
    getWeatherDescription(cityName).then(data => {  // then() accesses resolved data
        console.log(data);
    });
}

// Sample usage
logWeatherDescription("Amsterdam"); // will eventually log "Cloudy"
logWeatherDescription("Tokyo"); // will eventually log "Sunny"
```

▼ What happens when a promise rejected?

it allows us to handle the error by showing an error message

▼ What does .catch(callback) allow you to do?

The .catch(callback) allows you to handle the rejected state of a promise.

```
getWeatherIn("Amsterdam")
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error(error); // {error: "Connection issue"}
    });
```

▼ Can a promise never resolve?

Yes, when the condition that resolves the promise is never achieved (this is okay)

▼ What is the Promise.finally() callback?

The .finally() callback will execute whenever the promise's state changes from pending to either fulfilled or rejected.

```
startLoader();
getWeatherIn("Amsterdam")
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error(error);
    })
    .finally(() => {
        console.log("Done fetching weather");
        stopLoader();
    });
```

▼ What is consuming functions?

Most of the time you will be consuming (using) functions that return a promise (like the fetch API)

▼ The Promise class receives one argument which is called the...

executor (() ⇒ {})

The executor receives as the first argument a function that you can call whenever the promise has completed its work.

▼ What is resolve?

a function that you can call whenever the promise has completed its work.

Even though you can call this argument whatever you want, it is very common to call it resolve.

```
const waitOneSecond = () => {
    return new Promise((resolve) => {
        // do some work
        // when it's done, call resolve()
        resolve();
    });
}
```

▼ When to call the resolve() function?

call the resolve() function when the promise needs to move from pending to fulfilled

▼ Calling resolve() with a certain value will make that value available in the...

.then(callback)

▼
▼
▼
▼
▼
▼
▼
▼
▼
▼