**HTML/CSS**

1. Which tags are **mandatory** on the page?

Mandatory tags are (doctype), html, head, meta charset, title, body

2. What is **HTML5**? What is the difference with HTML4? How to create an HTML5 page?

HTML5 is a markup language used for structuring and presenting content on the world wide web, 5 stands for fifth and final version.

Differences are, HTML5 is an extended version of HTML4 and it supports audit and video playback without the flash player, it also has many advanced features, has better error handling, and supports canvas.

Put doctype on top , and then put html tag , inside html we put head tag, and there also a body tag, main page content, title inside header and meta tag with charset.

3. What is a tag **attribute**? What are they used for?

The tag attribute is special words inside the opening tag that controls the element’s behaviour. For example, tags can have some styling attributes or action attributes, such as color, heigh, font-weight or action like link tag can have a reference to a link that is redirected when pressed, href …

4. What is the difference between **display: block;** **display: inline; display: inline-block;**

**дисплей: inline**

Коли елемент має стиль *display:inline*, він не починатиметься з нового рядка, займе стільки ширини, скільки вміст, який він містить, і не спричинить розрив рядка після нього.

**дисплей: inline-block**

Різниця між *inline* елементом і *inline-block* елементом полягає в тому, що *inline-block* елемент може займати задану ширину та висоту. Але він також не розпочнеться з нового рядка в своєму батьківському або спричинить розрив рядка після нього.

**дисплей: блок**

Будь-який елемент зі стилем *display: block* є полярною протилежністю display:inline. Блоковий елемент починається з нового рядка і займає доступну ширину свого батьківського елемента або його вказану ширину.

5. Which tags are inline and which are block tags?

Block

Inline

6. How to set width and height for span elements?

To set width and height to span an element which is an inline element, I will have to change it to either inline-block or block and only then be able to change its height and width.

7. What is the space between inline elements? How to remove it?

The space between inline elements is default padding. There are 2 ways to remove it :

Add font-size:0 to the parent element and then declare a sensible font-size on the children.

Use flex box, put display flex on the parent element and in the child put display inline-block.

8. Do all tags have a closing tag?

Nope, not all the tags have closing tags in HTML, for example, img, br and many more…

9. What is semantic markup? Why is it important?

Semantic HTML refers to syntax that makes the HTML more comprehensible by better defining the different sections and layout of web pages. It makes web pages more informative and adaptable, allowing browsers and search engines to better interpret content.

10. How to create a common website layout? Which tags will you use?

I would definitely use, header that has some kind of website name , logo company …

Then I would have navigation bar, that has navigation bar of the website

Then some kind of main content that would be on the main tag, maybe an aside tag that has some kind of pictures or additional information.

Article would be a good tag to have , to have some kind of more information about the website.

And of course footer with some copyright, contact information, or social media links.

11. What are default browser styles? What is the difference between styles reset and styles normalization?

Reset will strip all styles from all elements, when normalized will leave useful styles, it renders all elements more consistently and in line with modern standards.

12. How to display html code on the page? (<> - should be displayed as it is);

You can use &at for < or &#60

You can use &gt for > or &#62

The result would be &at&gt or &#60&#62

13. What is the usage of <**figure**> tag?

The figure tag represents self-container content such as illustrations, diagrams, photos, code listings etc.

14. What is specific about **id** attributes? What is it used for?

The id attribute in html is unique and its used to style specific element on the page, or in javascript document.getElementById( “id name”) to work with

DOM elements.

15. How to create vertical space between elements on the page?

By using display block on parent and give child margin-top

By using flex box display flex, column-gap and row -gap

16. What are CSS selectors? What are they used for?

CSS selectors used to target HTML elements on your webpage that we want to style.

17. How to find an element on the page?

To find element on the page u can use :

• Based on name, id, class (simple)

• Based on a specific relationship between them (combinator)

• Based on an attribute or attribute value (attribute)

18. What kinds of CSS selectors do you know? Provide examples please?

• CSS Universal Selector.

\* {

color: blue;

font-size: 21px;

}

• CSS Element Selector.

ul {

border: solid 1px #ccc;

}

• CSS Id Selector.

#box {

width: 90px;

margin: 10px;

}

• CSS Class Selector.

.square {

margin: 20px;

width: 20px;

}

• CSS Attribute Selector.

input[type="text"] {

background-color: #fff;

width: 100px;

}

19. What is the difference between pseudo classes and pseudo elements?

*We use pseudo-class* when we need to apply css based on the state of an element. Such as:

Apply css on hover of anchor element (:hover)

Apply css when you focus on an html element (:focus).

*We use pseudo-element* when we need to apply css to the specific parts of an element or a newly inserted content. Such as:

Apply the css to first letter or first line of an element (::first-letter)

Insert content before, or after, the content of an element (::before, ::after)

Below is the example of both:

<html>

<head>

<style>

p::first-letter {

/\* pseudo-element \*/

color: #ff0000;

}

/======

a:hover {

/\* pseudo-class \*/

color: red;

}

</style>

</head>

<body>

<a href="#" >This is a link</a>

<p>This is a paragraph.</p>

</body>

</html>

20. What is a box-model? How to define element size?

The box model represents the size of an element and its border, padding, and margins. You can style each of these elements individually. Every element is surrounded by a box. Being able to understand how these boxes work is an essential part of positioning items in the way you want on a web page.

To define element size, you need to give that element height and width.

21. If different selectors define styles for the same element then how to know which style will be applied?

In a Chrome dev tool if I select that element I will be able to see what is a sector of it.

22. How to select the 3rd element in a list of 3 elements? Give multiple ways please

I can use :last-of-type selector or last-child selector.

23. What is style inheritance?

Inheritance in CSS occurs when an inheritable property is not set on an element. It goes up in its parent chain to set the property value to its parent value. Css properties such height, width, border, margin, padding, etc. are not inherited.

24.What are absolute and relative units in CSS? List all you know please?

The absolute units in CSS are

The relative units in CSS are

25. What is **absolute positioning?** **What is it used for?**

An absolute positioned element is an element whose computed position value is absolute or fixed. The top, right, bottom, and left properties specify offsets from the edges of the element’s containing block. ( The containing block is the ancestor relative to which the element is positioned ).

This is a very powerful type of positioning that allows you to literally place any page element exactly where you want it. You use the positioning attributes top , left , bottom , and right to set the location.

26. What is the float layout? What is clear fix?

The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it. The element is removed from the normal flow of the page, though still remaining a part of the flow (in contrast to absolute positioning).

A clear fix is a way for an element to automatically clear or fix its elements so that it does not need to add additional markup. It is generally used in float layouts where elements are floated to be stacked horizontally.

27. What is the flex box layout? What are the main and secondary axes in flex box layout?

Flex box is a one-dimensional layout method for arranging items in rows or columns. Items flex (expand) to fill additional space or shrink to fit into smaller spaces.

When working with flex box you need to think in terms of two axes — the main axis and the cross axis. The main axis is defined by the flex-direction property, and the cross axis runs perpendicular to it.

28. You have a div (width and height 10 px) inside another div (width and height 100 px)? How to center vertically and horizontally a small div inside a big one? List several ways please.

• Using position property.

*.parent\_div {*

*background:black;*

*height: 100 vh;*

*width: 100 vw;*

*position: relative;*

*}*

*.child\_div {*

*background:grey;*

*height: 50 vh;*

*width: 50 vw;*

*border: 2 px solid white;*

*position: absolute;*

*top: 50 % ;*

*left: 50 % ;*

*transform: translate(-50 % , -50 % );*

*}*

• Using flexbox property

.parent\_div {

background:black;

height: 100 vh;

width: 100 vw;

display: flex;

justify - content: center;

align - items: center;

}

.child\_div {

background:gray;

height: 50 vh;

width: 50 vw;

border: 2 px solidwhite;

}

• Using display grid property

.parent\_div {

background:black;

height: 100 vh;

width: 100 vw;

display: grid;

place - items: center;

}

.child\_div {

background:gray;

height: 50 vh;

width: 50 vw;

border: 2 px solid white;

}

29. What is margin overlap? How does it work for elements inside flexbox containers?

In CSS, adjacent margins can sometimes overlap. This is known as “margin collapse”, and it has a reputation for being quite dastardly. This is an interactive element! Instead of sitting 48px apart, their 24px margins merge together, occupying the same space!

To use margin overlapping with a flexbox container I need to use position relative and give the item negative values such as top -150px; that is the way I always do it.

30. How to style a checkbox?

A checkbox is an HTML element that takes input from the user. It is possible to style a checkbox using Pseudo Elements like :before, :after, hover and :checked.

31. How to define a different border color for each side of the element?

Each side can be set individually using

*border-top-color, border-right-color, border-bottom-color, and border-left-color;*

or using the writing mode-aware

*border-block-start-color, border-block-end-color, border-inline-start-color, and border-inline-end-color.*

32. What are mandatory elements for an HTML form?

The HTML required Attribute is a Boolean attribute which is used to specify that the input element must be filled out before submitting the Form. This attribute works with other types of input like radio, checkbox, number, text, etc.

33. How to create a multiline text input in a form?

To create a multi-line text input, use the HTML <textarea> tag. You can set the size of a text area using the cols and rows attributes. It is used within a form, to allow users to input text over multiple rows.

34. What validation attributes do you know?

I know validation attributes such as:

• Required

• Minlength and maxlength

• Pattern (Regular Expression)

• Min and max (used for numbers in between)

35. What is an iframe? What is it used for?

An inline frame (iframe) is a HTML element that loads another HTML page within the document. It essentially puts another webpage within the parent page. They are commonly used for advertisements, embedded videos, web analytics and interactive content.

36.How to create a list in HTML? What kind of lists do you know?

To create a list in HTML I use

<ul>

<li>Coffee</li>

<li>Tea</li>

<li>Milk</li>

</ul>

And …

<ol>

<li>Coffee</li>

<li>Tea</li>

<li>Milk</li>

</ol>

37. How to create CSS animation?

To make a CSS animation, you need three things: an HTML element to animate, a CSS rule which binds the animation to this element, and a group of keyframes that defines the styles at the start and end of the animation. You can also add declarations to further customize your animation, like speed and delay.

@keyframes example {

0% {background-color:red; left:0px; top:0px;}

25% {background-color:yellow; left:200px; top:0px;}

50% {background-color:blue; left:200px; top:200px;}

75% {background-color:green; left:0px; top:200px;}

100% {background-color:red; left:0px; top:0px;}

}div {

width: 100px;

height: 100px;

position: relative;

background-color: red;

animation-duration: 4s:}

38. What is responsive markup? How to achieve this?

What is Responsive Web Design? Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones)

To make an HTML page to be responsive, the viewport meta tag has to be included. This sets the page width to device-width and initial zoom to 1. If the meta tag is not included the mobile or tablet will try to fit the desktop layout but it might not fit properly. It’s important and the right approach to do mobile first site and then from there do tablet and desktop.

39. What is the difference between adaptive and responsive markup?

The quickest and simplest way to differentiate the two is to understand that a responsive site uses a single fluid layout that changes to fit any screen size. Adaptive sites use multiple static designs with breakpoints that signal the appropriate layout for various screen sizes.

40. What is CSS breakpoint? How to choose breakpoints?

CSS breakpoints are points where the website content responds according to the device width, allowing you to show the best possible layout to the user. CSS breakpoints are also called media query breakpoints, as they are used with media query.

Design the content to fit on a small screen size first, then expand the screen until a breakpoint becomes necessary. This allows you to optimize breakpoints based on content and maintain the least number of breakpoints possible.

What are common breakpoints? Common breakpoints are 320px — 480px for mobile devices, 481px — 768px for iPads & tablets, 769px — 1024px for small screens like laptop, 1025px — 1200px for large screens like Desktops, and 1201px and above for extra large screens like TV.

41. How do mediaqueries work? Which properties we can handle with mediaqueries?

The media queries are a special syntax for CSS that allows us to define some styles that will only be applied in the case that defined conditions are met. We can assimilate them to optional lines of code, which will only be displayed for some users or devices.

42. What is a preprocessor? What features does it have? What problems does it solve?

A CSS preprocessor is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on.

CSS preprocessors make it easy to automate repetitive tasks, reduce the number of errors and code bloat, create reusable code snippets, and ensure backward compatibility.

With a preprocessor, you can make use of variables, operators, functions and other useful features that are not available when using just CSS on its own.

43. What is the BEM methodology? Which problems does it solve?

BEM is a front-end naming method for organizing and naming CSS classes. The Block, Element, Modifier methodology is a popular naming convention for class names in HTML and CSS. It helps to write clean CSS by following some simple rules.

The BEM methodology solves the collision issue using naming conventions for CSS classes, providing unique names for all components and their parts.

***Javascript***

Здесь собраны основные вопросы по ЈS. Процесс подготовки к собеседованию - двигайся по каждому вопросу и пробуй отвечать на него. Если есть трудности - возвращаешься к нужной теме по программе, Гуглишь и разбираешься вопросе. Так же пиши в чат если будут трудности

Ответы на основные вопросы пишем в своем файле ;)

Overall - 160 questions

**JS basics to warm up**

1. What is **EcmaScript** (ES)?

**ECMAScript** is a Standard for scripting languages.

The way that I understand it is that EcmaScript is like an engine that is the same in 3 different cars, but the exterior and interior is different in each car that is JavaScript, JScript and some other script type programming languages.

Наскільки я розумію, EcmaScript схожий на двигун, який є однаковим у 3 різних автомобілях, але зовнішній вигляд і інтер’єр у кожному автомобілі різні, тобто це JavaScript, JScript та деякі інші мови програмування типу script..

• What's the difference between **JavaScript** and **ECMAScript**?

The differences between JavaScript and EcmaScript is that JavaScript is a programming language and EcmaScript is a scardard for other script languages as well as creating one alike to JavaScript.

2. What **data types** do you know in JavaScript?

I do know that JavaScript has primitive values and objects.

The primitive data types in JavaScript are:

1.**Boolean**  
2.**Null**  
3.**Undefined**  
4.**Number**  
5.**String**  
6.**Symbol** (new in ECMAScript 6)

7.And the last one would be an **Object** which is a complex data type in JavaScript since it can contain all kinds of different data types at once

3. What is the difference between **primitives** and **objects**?

**Primitive** values are not immutable - meaning you can not change them after you created them when **objects** are immutable, you can change them however you like, because objects have references to them.

4. How does **typeof** work?

It will take any type as a parameter and simply return which data type it is in a simply readable string.

Він прийматиме будь-який тип як параметр і просто повертатиме, який це тип даних, у легко читаному рядку.

5. What does **typeof return for a function**? Why this result?

It will return a **function**, because when defining typeof(); many years ago, Brendan Eich decided that function should be different from all other objects and because the EcmaScript specification says so, it's a rule. Guessing to make a developer easier to find and work with functions in JavaScript.

6. How to **define variables** in JavaScript?

To define variables in JavaScript, I use the word **‘let’** then the **name** of the variable , that makes sense to me and easy to understand for others, followed by the equal sign **‘=’** and finally put my data that I want to be stored in that variable.

7. What is the difference between not defined and undefined in JS?

Is **variable not defined** the same as **variable is undefined**?

The difference is quite simple, both of them are related to memory space.

• Variable **not defined** = variable doesn’t exists in memory or its not exists in scope (not declared)

• Variable **undefined** = variable exists but it has no value or is used without its value.

8.What is the difference between **undefined** and **null**?

**Undefined** means that variable has been declared but has no value or not yet been assigned a value.

**Null** on the other hand is an assignment value. It can be assigned to a variable as a representation of *NO VALUE.*

9.What is **NaN**? How can we get this value?

In JavaScript, **NaN** is short for **Not A Number** which means that it’s not a legal number. The Number.isNaN() method returns true if the value is NaN, and the type is a Number. To get this value I usually use the method **Number.isNaN()** or **isNaN()**.

10.What is the difference between **var** and **let**? Tell me all the differences you know.

The difference between **var** and **let** is that scope of variable defined with **let** is limited to the block in which it is declared in, meanwhile variable declared with **var** has the global scope.

11.What is the difference between **const** and **let**?

**let**

**const**

It can be updated but cannot be re-declared into the scope.

It cannot be updated or re-declared into the scope.

It can be declared without initialization.

It cannot be declared without initialization.

It can be accessed without initialization as its default value is “undefined”.

It cannot be accessed without initialization, as it cannot be declared without initialization.

12.What is a strict **comparison** **===.** What is the difference with **==** ?

**===** will check if two operands are the same data type and equal between each other , it will return boolean(true/false)

**===** will also compare data types , if data types are different between each other, it will return false.

**==** will not compare for the data types, will not do the conversion, if operands are the same but data types not, it will return true.

13. What is the **reference** **data type**?

**Reference data types** in JavaScript are dynamic, can change in size and also they have references to them, like the home address of a house, reference data types could be an object, an array function and collections of all other types of objects.

14. What is **type–conversion** in JavaScript? Give me some examples.

In JavaScript type conversion means to convert data from one type to another.

There are two types of conversion. Automatic and Manual.

**Automatic type conversion** works as it should, like converting Alert to a string to show the message.

**Manual type conversion** is when we need to convert it manually:

*Most common conversion are:*

**String Conversion** - Occurs when we output something. Can be performed with **String(value)**

**Numeric Conversion** - Occurs in math operations. Can be performed with **Number(value)**

15.How to convert value to **boolean**? Which values are converted to **false**?

• There are **two ways to convert to Boolean:**

**• !!(value);**

**• Boolean(value);**

I prefer the first one, it looks better and makes sense to me.

• If you convert these to Boolean, will always get **false**:

**• False**

**• Undefined**

**• Null**

**• NaN**

**• 0**

**• “”(empty string)**

16. What is a **function** in JavaScript? Why do we need them?

It’s a piece of code that can be reusable to minimize code duplications and do some tasks, calculate and return or use in another function. In JavaScript function is a first class object , because it can have properties and methods just like any other objects.

With the help of functions we can create a piece of code and reuse it multiple times whenever we want.

17. What is an **anonymous function**? Give me examples of its usage.

An **anonymous function** in JavaScript is a function without a name. Can be used as an argument to another function, executing the function right after declaring it.

**Anonymous functions are useful** because they help you control which functions are exposed. More Detail: If there is no name, you can't reassign it or tamper with it anywhere but the exact place it was created.

Examples:

*function () {*

*console.log('Immediately invoked function execution');*

*})*

*setTimeout(function() {*

*console.log('Execute later after 1 second')*

*}, 1000);*

18. What is **pure function**?

**Pure function** in JavaScript is a function that passes two points in checklist:

**• Same Input => Same Output**

**• No side-Effects**

Meaning, what goes in goes out and it doesn’t affect outside code of a function scope.

Чиста функція — це функція (блок коду), яка завжди повертає той самий результат, якщо передано однакові аргументи . Це не залежить від будь-якого стану чи зміни даних під час виконання програми. Швидше, це залежить лише від його вхідних аргументів.

19. What is **function expression** and **function declaration**?

Which is better and why? Where to use one and another?

**Function expression** is an anonymous function assigned to a variable.

*It won’t hoist*

**Function declaration** has function name, return type and parameters.

*It can hoist*

If you ask me which one is better, I would say, I prefer using function declaration, because I can name it and pass function inside it as a parameters and return whatever i want and use it after I’ve declared it and reused it, also using a callback functions makes it easier and code becomes cleaner.

20. What is a **callback function**?

A **callback function** in JavaScript is a function passed into another function as argument, which is then invoked inside the auteur function to complete some kind of routine or action. *Very useful in asynchronous code.*

21. What is the **arrow function**? Why do we need them? What is the difference with function declaration? Tell me as much differences as you know

Arrow function in JavaScript is a compact alternative to regular functions.

**Main reason for arrow functions is for callbacks.**

**Limitation are :**

**• It has no bindings to *this, arguments,super.***

**• Don’t have access to *new.target***

**• Aren’t suitable for *call,apply,bind.***

**• Can not be used as a constructor*.***

**• Can not use *yield,* within its body.**

Arrow function introduces concise body syntax, or implicit return. This allows omission of the curly brackets and return keyword. Implicit return is useful for creating succinct one-line operations in map, filter and other common array methods.

22. When I should **not use arrow functions**? Name three or more cases.

Avoid using arrow functions for **event handlers, object methods, prototype methods, callback functions with dynamic context, invoking constructors, too short syntax**, make sure it's not too complicated.

23. What is **IIFE (Immediately Invoked Function Expression)**?

**IIFE** in JavaScript, it’s kinda self explanatory, it’s a function that runs as soon as it’s defined, it’s also known as **Self-Executing Anonymous Function**.

IIFE є хорошим способом захисту області . Ви можете використовувати IIFE, щоб запобігти проблемам із визначенням глобальних змінних, псевдонімам змінних, захистити особисті дані та уникнути конфліктів використання багатьох бібліотек, які експортують однакові назви об’єктів.

24. What is the **argument objec**t inside the function? What data type does it have?

JavaScript functions have a built-in object called the arguments object. The argument object contains an array of the arguments used when the function was called.

For the whole argument object type of will return object, but if you want to get data type of single item inside argument object, you can access it by idex.

25.What is the **rest** operator? Give me an example of it.

The **rest operator(...)** allows us to call a function with any number of arguments and then access those excess arguments as an array.

The rest operator also allows us to **destruct** an array or an object.

26. How to **round the number**? How to round the number with some precision?

To **round the number** i can use methods from Math library , such as :

**• Math.round()**

**• Math.floor()**

**• Math.ceil()**

But if you want to round number with some precision, then i would advise to use :

**• toFixed(how many numbers after decimal point)**

27. What is **strict mode**? Do you use it? Why?

**Strict mode is a modern mode**, it makes several changes in JavaScript semantics. It **eliminates silent errors** and instead throws them so that the code won’t run with errors in the code. It will also point out mistakes that prevent JavaScript engines from doing optimization.

If I’m using modules and classes in JavaScript files, there is no need to say that the script will run in strict mode, it’s implemented automatically.

28. What is the **difference between** **`**,**'** and **“** ?

Between **single quotes and double quotes there is no difference**, they both indicate string, just that if the string contains any of them then I use the combination of them in order to form a correct format of a string that contains a string.

**Backticks** are used for **template string** in javascript, meaning you can put variables in the string and it will display the result or the value of a variable;

For Example:

let firstName = “John”;

let lastName = “Doe”;

let text = `Welcome ${firsName}, ${lastName}!`;

console.log(text); // Welcome John, Doe!

29. We have the **P** symbol in **cyrillic** alphabet and the same **P** symbol in the **Latin** alphabet. What will be the result of comparing these two symbols with **==** and **===** ? And why?

The result of these two comparisons will always give **false**, doesn't matter if I compare with == or deep compare with ===, because the **unicode for them will be different.**

30. How to find a **substring** inside another string? Tell me **3 ways** to do that.

To find a **substring** in JavaScript use this:

• includes();

• indexOf();

• contains();

• lastIndexOf();

31. Is Javascript **static or dynamic**? Is it a strong or weak type? And is it good or bad?

**JavaScript is a dynamic-typed language**, that means that you can create anything at any time, even when a program is running, and you don't have to declare the type of a variable or data type.

I think J**avaScript is a weak type** language because it doesn’t have compilation steps, where strongly typed languages would validate types.

32. What is the difference between **var** and **let** in terms of performance? Why?

In terms of performance comparison,**let** is **faster** than **var**, but **only** when inside a **different scope** than the main scope of a function. In the **main scope**, **var** and **let** are **roughly identical in performance**.

33. What is **HOF (high order function)**?

**Higher order functions** are functions that operate on other functions, either by taking them as arguments or by returning them. In simple words, a HOF, is a function that receives a function as an argument or returns a function as output.

Map is a higher order function and addEventListener as well, because they both take function as an argument. *This is very useful because we can dynamically reuse them.*

34. What is **Memoization**? Describe how it works?

In programming **memoization** refers to a technique to optimize the application , make it more efficient and make it faster.

It works a certain way, it computes results in cache and retrieves that same information from cache the next time it’s needed instead of computing it again.

**Arrays. Data structures**

1. What is an **array** in JavaScript? What **data can be stored in an array**?

**Array** in JavaScript is a special variable that can store inside **different types of elements**. It can store multiple elements of different data types, it doesn’t require the value of the same data types.

*It's very useful, to organize data and use it to access it via a single variable.*

2. Can I have **string** and **numbers** in the same array? And what about **string** with **boolean**?

In an array we can store all types of data, so it means that strings with numbers can exist in one array the same as string with boolean.

3. How to **define an array** in JS? Tell me all the options you know. What is the best way? Why?

**There are ways to create and array as follows :**

**• Basic** => *const animals = ['*🐼*', '*🦁*', '*🐷*', '*🦊*'];*

**• Constructor** => *const Animals = new Array('*🐼*', '*🦁*', '*🐷*', '*🦊*');*

**• Array.of()** => *const Animals = Array.of('*🐼*', null, '*🦊*', undefined);*

**• Spread Operator** => *const moreAnimals = ['*🐵*', ...animals ];*

**• From another array** =>

*const Animals = new Array('*🐼*', '*🦁*', '*🐷*', '*🦊*');*

*const copyOfAnimals = Array.from(Animals);*

**• From array-like objects** =>

*const divs = document.querySelectorAll('div');*

*const divsArray = Array.from(divs);*

**• Map and Reduce** =>

*const animals = ['*🐼*', '*🦁*', '*🐷*', '*🦊*'];*

*const animalsCopy = animals.map(a => `${a}'s kid`);*

*console.log(animalsCopy);*

*//* 👆 *(4) ["*🐼*'s kid", "*🦁*'s kid", "*🐷*'s kid", "*🦊*'s kid"]*

4. How to **access values in the array**?

To access values in an array in JavaScript, we need to refer to it’s index number of the item in square brackets. Elements start from index 0(first element) and goes to array.lenght-1(last element)

5. How can we **iterate an array**? Give me all the options you've used.

There are many ways to iterate array in JavaScript :

• Using for loop

• Using for-of loop

• Using while loop

• Using forEach method

• Using Every method

• Using Map

• Using Filter

• Using Reduce

• Using Some

6. What is a **pseudo-array**? Tell me about a situation where you work with it?

**Pseudo array** in JavaScript, they like array, but do not have array methods like normal array does, it only has index and length, because prototype of pseudo array is not the same as prototype of regular array.

7. What **array methods** do you know? Which ones mutate arrays, which ones are not?

8. What is the difference between .**map** and .**forEach** ?

The difference between these two is that **forEach** method returns undefined, it’s good for reading arrays, when **map** method returns a new array with changed elements, great if you need to do the same operation to every single element of an array and return new array with changed all elements when forEach is good to iterate through array.

9. How does the **sort** method work?

In JavaScript sort method, goes through each value inside the array, compares first value “a” with the second value “b” if the difference between a and b is negative number or 0, it does not change the sorting and moves onto the next set. It will mutate the array, and will return a mutated array that is sorted out.

10. How does the array **reduce** method work?

The reduce method executes a user-supplied “reducer” call back function on each element of the array, in order, passing the return value from calculation of the array is a single value. Great for calculating something out of the whole array. For example, like total price.

11. **What** value will the **accumulator** in reduce have on the **1st iteration** if it is not set by **default**?

If no initial accumulator is provided , the **first element is taken as an accumulator,** and the callback will be called from the second element onwards.

12. Take a look on the code below:

*let arr1 = [7, 14];*

*let arr2 = arr1;*

Will **arr1** be **changed** if **arr2** **changed**? Why?

Yes the arr1 will be changed because objects are reference data types, so arr1 is reference to arr2 if we change arr2 then arr1 will also be changed and its reference will be to arr2.

13. ​ What is the **spread operator**? How does it work for arrays? What is it used for?

The **spread operator unpacks elements of iterable objects** such as arrays, sets and maps into a list. The rest parameter is also denoted by three dots( … ) , and the remaining arguments it will pack as an array. It's great to clone an iterable object or merge an iterable object into one.

14. How does the **rest operator** work for arrays? What is it used for?

In JavaScript functions, **rest gets used as a prefix of the function’s last parameter.** The rest operator ( … ) instructs the computer to add whatever other info (arguments) supplied by the user into an array.

15. What is **destructuring**? How does it work with arrays? Give me examples

Destructuring in JavaScript, expression that makes it possible to **unpack values from arrays, or properties from objects, into distinct variables**.

In array destructuring used to open up an array with a list of elements, it can be used to put two arrays together , merge them, or use the list afterward as whatever I might like to, and at the same time I can give it a variable so that I can use it.

16. How to **join two arrays**? Tell me all the options you know.

There are 5 ways to join arrays in JavaScript:

• Using traditional **for loop**. Looping through and **pushing** each element.

• Using the **merge function**, *merge ( [1,2,3] , [1,2,3] );*

• Using **spread operator**, *const merge = [... arr1, …arr2];*

• Using **concat method**, *const merge = arr1.concat(arr2);*

• Using **reduce method**,

*cont merge = arr2.reduce((arr, item) => {*

*arr.push(item);*

*return arr;*

*}, arr1);*

17. How do you **reverse an array**? Show me all the options you know.

• Iterate array backwards with *for loop* and and *push* elements to a new array

• Use while loop while array.length use push with pop like this ,

*while (array.length) {*

*output.push(array.pop());*

• Or simply use method *reverse()*on the array that I want to reverse .

18. Tell me about **some** and **every** method in Array?

Method **some** will only check if any of array elements pass a test ( provided in callback function ). It will stop and return true if at least one element matches the provided check.

Method **every** will test where all elements pass a test ( provided in a callback function ). It will stop only at the end of the array, and return true if all elements pass a test, if at least one does not match then it will stop and return false.

19. Tell me about **slice** and **splice** methods?

Method **slice** will slice the array one argument is the start and second argument is the end of the cutted slice from array. If there are no arguments, it will make a copy of an array which is very convenient. It will not modify the original array.

Method **splice** will change the content of your array by removing or replacing existing elements with new ones.. It modifies the original array and returns the removed elements as a new array.

20. How to **empty** an **array** in JavaScript?

There are 4 ways to empty an array:

*let a = [ 1,2,3 ];*

• Assigning to a new empty array =>  *a = [ ];*

• Setting its length to zero => *a.length = 0;*

• Using splice method => *a.splice( 0, a.length );*

• Using pop method => *while ( a.length > 0 ) { a.pop(); }*

**Objects**

1. What is an **object**?

In JavaScript, an object is an **unordered collection** of **key-value pairs**.

2. How to **access properties** of an object? Tell me all the options you know. What is the best way? Why?

To **access object properties** you can do it in three different ways:

**• dot property** => *object.property*

**• square brackets** => *object [ ‘property’ ]*

**• object destructuring** => *{ property } = object*

It’s hard to pick one, because it all depends on the situation. If I know the variable ahead, I use dot accessor, if it's dynamic or I don't know I use square brackets and if I need to store it right away in a variable to use it afterwards I use destructuring.

3. How to **define objects** in JS? Tell me all the options you know. What is the best way? Why?

There are 4 ways to define an object in JavaScript:

• Using **object literals** =>

*const person = { fName : ”Dima”,lName : ”Bond” };*

• Using keyword **new Object()** =>

*const person = new Object ();*

*person.fname = “Dima”;*

*person.lName = “Bond”;*

• Using **Object.create()** method =>

*const person = { };*

*const student = Object.create (person );*

• And finally, using **class** as a **constructor** =>

*class person {*

*constructor(name, age) {*

*this.name = name;*

*this.age = age }*

*}*

*const student1 = new person ( “Dima Bond”, 34 );*

I think the best and easiest way is to use object literals, well that is the one i use most of the time, but if i need to create an object from something already created such as another object or class constructor i will of course use class constructor or object.create method. So it all depends on the circumstances.

4. How can we **iterate objects**? Give me all the options you've used.

There are **5 methods** that i personally used when iterating over object in JavaScript :

*• Object.entries();*

*• Object.keys();*

*• Object.values();*

*• Object.getOwnPropertyNames*

*• And a for in loop*

5. How to **add property** **to the object**? Tell me all the options you know. What is the best way? Why?

There are also **5 ways to add property** to the object:

• Using dot syntax => great for static objects.

• Using square brackets => ideal for dynamic values, such as API user input

• Using Object.definProperty() method => good for getters and setters.

• Using the Object.assign() method => it is more suitable when dealing with other objects and using a spread operator is also a way to go.

It depends on the situation and the task I’m working on.

6. How can we **remove property** from an object?

To remove property from an object in JavaScript:

I use **delete operato**r =>

*delete person.age;*

7. How to **copy objects**? Tell me all the options you know. What is the best way? Why?

There are **3 ways to copy** object:

• Using spread *( … )*

• Using *Object.assign( );*

• Using *JSON.stringify( ); and JSON.parse( );* methods

8. Take a look on the code below:

*let obj = { price: 100};*

*let anotherObj = obj;*

Will **obj be changed if anotherObj changed**? Why?

**Both objects will change**, because **objects are reference types** in JavaScript.

9. What is the **spread operator**? How does it work for objects? How does it work for arrays? What is it used for?

**Spread operator** gives access to the iterable objects. => (...)

**Spread operator** can be used to clone an object or merge them. But it’s shallow, not deep. Reason being that the objects most of the time are not iterable.

10. How does the r**est operator work for objects**? What is it used for?

**Rest operator is used as a prefix**, and tells the computer to put other info whatever the user wants after that prefix. So it also merges two objects together.

11. What is **destructuring**? How does it work **with objects**? Give me examples.

**Destructuring** is destructuring and assigning an object to a variable.

We should have an existing object on the right side, that we want to split into variables. The left side contains an object-like “pattern” for corresponding properties. In the simplest case, that’s a list of variable names in {...}.

12. How can we make a **deep copy of the objects**? Objects have any keys-values ?

I usually use the **JSON.parse(JSON.stringify(object))** because the **Object.assign()** method will make a shallow copy and objects inside the object will only copy references to it.

13. What **Set data structure** is used for? Give me at least two examples of using it?

Set data structure is used to store unique values of any type, primitive value or object references.

Examples :

*const fruits = new Set(['*🍇*', '*🍉*', '*🍊*', '*🍈*' ]);*

*fruits.add('*🍋*');*

*fruits; // Set(5) {'*🍇*', '*🍉*', '*🍊*', '*🍈*', '*🍋*'};*

*var set1 = new Set([1, 2, 1, 3, 4, 5]);*

*set1; // Set(5) {1, 2, 3, 4, 5} ; duplicate values are removed.*

*Creating an empty set*

*var set2 = new Set();*

14. What **Map data structure** is used for? Give me at least two examples of using it?

**Map data structure** in JavaScript holds key-value pairs and remembers the original insertion order of the keys.

15. What is the difference between **Set and WeakSet**? **Map and WeakMap**?

**WeakMap is Map - like collection that allows only objects as keys and removes them together with associated value once they become inaccessible by other means**. **WeakSet is a Set -like collection that stores only objects and removes them once they become inaccessible by other means.**

**Sets can store any value. WeakSets are collections of objects only. WeakSet does not have size property. WeakSet does not have clear, keys, values, entries, forEach methods.**

**A WeakMap accepts only objects as keys whereas a Map, in addition to objects, accepts primitive data types such as strings, numbers etc. 2) WeakMap objects don't avert garbage collection if there are no references to the object which is acting like a key.**

**Lexical environment, hoisting, closure**

1. What is the **Lexical Environment**? What is stored in LexicalEnvironment? Tell me about its structure

A **lexical environment** is a data structure that holds identifier-variable mapping. (here identifier refers to the name of variables/functions, and the variable is the reference to the actual object [including function object or primitive value.]

Lexical Environment in JavaScript is an object that is stored deep inside, we don’t have access to it and it stores all kinds of variables inside. We need a lexical environment to store variables so that we later on can use them. It’s simply JavaScript working with memory. There are **3 ways when a lexical environment is created**, first of all when we **run the script**, run the program, second of all **when function is called** , and last but not least **when JavaScript sees a block of code** , such as if , loop.

Lexical environment has two components :

• The environment record.

• Reference to the outer environment.

2. What is an **Environment Record**?

The environment record is the actual place where the variable and function declarations are stored. It also records the identifier bindings that are created within the scope of its associated lexical environment. This basically means that an ER keeps track of variable and function names and their associated values.

3. What is **closure**? Give at least 2 examples of using closure.

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function’s scope from an inner function.

Variables in closures can help you maintain a state that you can use later. They provide data encapsulation. They help remove redundant code. They help maintain modular code.

Two Examples of using closures that I personally used, was:

• To make an independent counter function that counts separately how many times I call a function1 and function2.

• To make a calculator with memory.

4. What is **hoisting**? How does it work for **var**, **let** and **const**? How does it work for **functions**?

In JavaScript **hoisting** refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of code. Hoisting allows functions to be safely used in code before they are declared.

**Var** will hoist to the top of the script, unless it's declared inside the function , functional scope will stop it from hoisting.

**Let** and **Const** will also hoist, but there is a difference that it will throw an error that it can not read before initialization.

And for the **functions** , **function declaration** will **hoist** and will be available before initialization , even though it’s a **bad practice** to do so, if you look from **code quality** perspective, when **function expression** will **not hoist** and throw the same error as let and const can not access before initialization.

5. What is **scope**?

**Scope** in JavaScript refers to the current context of code , which determines the accessibility of variables to JavaScript. In other words, scopes is an area of code that can see and access variables.

There are **3 types of scopes** in JavaScript:

**Globack Scope, Block Scope and Function Scope**

6. What is the **difference between scope and lexical environment**? Or is there no difference?

**Lexical environment** in JavaScript is an object, a complex mechanism inside a programming language that works with memory, that stores all kinds of data structures for further use. That is called Environment record and also lexical environment has references to outer environment, on one level higher as a parent.

**Scope** is simply a technical term that all developers use, to talk about the area of accessibility of stored data structures in code. It's snot the same as a lexical environment, it just gives the idea of where code can access and what it can access in the script. That’s all.

7. How does **scope** work for variables defined with **var** ?

Give at least 2 code examples

Only **function scope** that is var is defined inside of, will be functional scope or **local scope** all other ways of defining var as a variable always be in global scope.

*Global Scope Var Function Scope Var*

*var a = 17; function f() {*

*if (a) { var a = 1;*

*var a = 1;*

*}*

*console.log(a); // 1*

*} console.log(typeof a);*

*// Undefined*

*console.log(a); // 1*

8. Which function will be **stored in the lexical environment first**

**function test() {}** or **const test1 = () = {};**? And why?

This is kinda tricky question, but to be honest I believe that **function test(){}** will be hoisted first, simply because **const test1 = () => {}**

Will not hoist.

9. When will **Lexical Environment be removed from memory**?

When a **function is executed** and **its lexical environment is not linked to anything** or t**he end of the script**.

**Context**

1. What is **context**? What does it **reference** for?

Контекст – це завжди значення ключового слова this, яке є посиланням на об’єкт, який «володіє» поточним кодом, що виконується, або функцією, де він переглядається. Він посилається на об'єкт у функції, що виконується.

It’s an object that helps functions to share data between each other and at the same time, this data will be protected and isolated from outside manipulation. Functions become more functional, it has more possibilities.

2. What does it mean to **'lose context**'? Give **3 examples** how context can be lost. How to avoid it?

There **3 ways to lose context** :

1. function is called separately without the object.(function expression)

2. callback function is called. (most used cases in real projects)

3. nested function is called. (sometimes will see in real projects)

There are **3 ways to fix context.**..

1. bind the function to the context.

2. use the arrow function.

3. use call or apply.

In other words in practice use arrow functions when working with context and will have no issues with losing context.

3. What approaches do you know to **control or set the context**?

We can control or set context in JavaScript by using such methods as:

• bind();

• call();

• apply();

4. What is the difference between **.bind**, .**apply** and **.call** ? Which one is better to use?

**Call** викликає функцію та дозволяє передавати аргументи по одному.

**Apply** викликає функцію та дозволяє передавати аргументи у вигляді масиву.

**Bind** повертає нову функцію, що дозволяє передати цей масив і будь-яку кількість аргументів.

It all depends on the situation, as they all add this context to the function in slightly different ways, if I need to work with the function right away then I use either call or apply, but if I need to use this context later or using asynchronous code, then I use bind.

5. What **arguments** does **.bind** can take?

**const bindedFunc = Func.bind(thisObj,optionsArg1,optionalArg2,optionalArg3,...);**

The first argument will be the **object** that will be made as a copy of a function and other arguments can be whatever you want to the new object that will be created with this operation.

6. How to **set context without call, bind, apply** methods?

There are a few ways to do that:

1. In the **external** **function** we write **this** to any **variable** and **this variable** will be available in the **internal function**, like all variables (usually **this** variable is **called self**). Thus, we will **pass this** **from** the **outer function** **to** the **inner** one:

*let elem = document.querySelector('#elem');*

*elem.addEventListener('blur', parent);*

*function parent() {*

*console.log(this.value); // выведет 'text'*

*let self = this; // запишем this в любую переменную, например, в self*

*function child() {*

*console.log(self.value); // выведет 'text'*

*}*

*child();*

*}*

2. There is another solution to the problem. Let's make the **function child take a paramete**r:

*let elem = document.querySelector('#elem');*

*elem.addEventListener('blur', parent);*

*function parent() {*

*child(this); // передаем параметром this*

*function child(param) {*

*console.log(param.value); // выводим value инпута*

*}*

*}*

3. The third solution to the problem is to use **arrow functions** :

*let elem = document.querySelector('#elem');*

*elem.addEventListener('blur', parent);*

*function parent() {*

*console.log(this.value); // выведет 'text'*

*let child = () => {*

*console.log(this.value); // выведет 'text'*

*}*

*child();*

*}*

7. What is a **global object** ?

**Global object** is the object that has access to the global scope and can share data with other objects. In the browser it is a window object and in node.js it is a global object. In the **browser** we can access the global object by using the **window** keyword and in **node.js** we can access the global object by using the **global** keyword. There is also a **globalThis** object, this is the global property allows one to access the global object regardless of the current environment.

8. How do **arrow functions** work with **context**?

Arrow functions are more or less equivalent to function statements, except that they bind this argument to that of the parent scope.

In other words, if an arrow function appears in the top scope, this argument will always refer to the global scope(e.g., window in browsers or global in node.js), while an arrow function inside a regular function will have its argument the same as its outer function.

**Classes & Prototypes**

1. What is **class**?

Клас JavaScript — це схема для створення об'єктів . Клас інкапсулює дані та функції, які маніпулюють даними.

They are like special functions that use strict mode, encapsulate everything inside of the class and very nice syntax.

2. What is a **class instance**?

Екземпляр - це об’єкт, що містить дані та поведінку, описану класом . Оператор new створює екземпляр класу в JavaScript: instance = new Class() .

3. What is the **constructor function**? How does it work?

**Конструктор** — це спеціальна функція, яка створює та ініціалізує екземпляр об'єкта класу . У JavaScript конструктор викликається, коли об’єкт створюється за допомогою ключового слова new. Метою конструктора є створення нового об’єкта та встановлення значень для будь-яких наявних властивостей об’єкта.

4. What is a **prototype** in JavaScript?

Кожна функція в JavaScript (які самі по собі є об’єктами) фактично має член під назвою «прототип», який відповідає за надання значень, коли їх запитує об’єкт . Наявність цього елемента дозволяє працювати механізму конструктора (за допомогою якого об’єкти створюються з функцій).

Властивість прототипу дозволяє додавати властивості та методи до будь-якого об’єкта.

**Прототип** — це об’єкт, який за замовчуванням пов’язано з усіма функціями та об’єктами в JavaScript, де властивість прототипу функції доступна та змінна, а властивість прототипу об’єкта (він же атрибут) невидима . Кожна функція включає прототип об'єкта за замовчуванням.

5. What is **prototype inheritance**?

**Prototypal Inheritance** — це функція в JavaScript, яка використовується для додавання методів і властивостей в об'єкти . Це метод, за допомогою якого об’єкт може успадкувати властивості та методи іншого об’єкта. Традиційно, щоб отримати та встановити [[Prototype]] об’єкта, ми використовуємо Object. getPrototypeOf і Object.

У програмуванні ми часто хочемо взяти щось і розширити це. Наприклад, у нас є об’єкт користувача з його властивостями та методами, і ми хочемо зробити адміністратора та гостя як його дещо змінені варіанти.

6. Can you explain the thesis '**Everything is an Object in JavaScript'**?

**Майже все в JavaScript є об’єктами** , **окрім** шести речей, які не є об’єктами: **null , undefined , strings, numbers, boolean, and symbols**. Вони називаються примітивними значеннями або примітивними типами.

*У JavaScript все є об’єктом, навіть коли це щось інше. Функції - це об'єкти. Рядки є об'єктами. Числа - це об'єкти. Масиви - це об'єкти. Предмети є предмети.*

7. Each **object** in JavaScript has a **.toString()** method. **How is it achieved?**

**• Метод toString()автоматично викликається**, коли очікується рядкове значення об’єкта.

**• Метод успадковується нащадками об'єкта**.

**• Об’єкти замінюють метод, щоб повернути конкретне значення рядка.**

Якщо toString()метод не перевизначено, [object type]повертається.

8. Which **methods of Object class** do you know? Tell me what they do.

In object class will be the methods that we will write inside that class, and will be available to use, when creating an instance of a class using new and name of the class.

9. What is **OOP**? **Is it used in JS**?

OOP - is a n Object Oriented Programming.

It’s not the same as in other programming languages such as C++, Java, because it's not a class -based language, instead it's a prototype-based language.

10. What is **polymorphism**? Give me at least 2 examples.

**Поліморфізм** означає «**багато форм**», і це відбувається, коли ми маємо **багато класів, пов’язаних один з одним успадкуванням.**

For Example:

Class Vehicles, vehicles can be different , like , bike, tricycle, car, boat, and so forth , so that vehicles will be main and super class and all other classes will be inherited from class vehicles class.

Class User, will have username and password and basic information about the user and will be super class, there also can be admin that has same information as user does plus one more if its admin or not, and from there you can give more rights to that user, such as deleting something or updating something, and there also will be class visitor, this is like a user but doesn’t have to have fully filled out profile information on the site, it needs only login pass. This is an example of polymorphism.

11. What is **encapsulation**? Give me at least 2 examples

**Інкапсуляція** — **це упаковка даних і функцій в один компонент** (наприклад, клас ), а потім контроль доступу до цього компонента для створення «чорної скриньки» з об’єкта . Через це користувачеві цього класу потрібно знати лише його інтерфейс (тобто дані та функції, доступні за межами класу), а не приховану реалізацію.

12. What is **inheritance**? Give me at least 2 examples

**Спадкування** дає змогу визначити клас, який бере всі функції від батьківського класу та дозволяє додавати більше . Використовуючи успадкування класу, клас може успадкувати всі методи та властивості іншого класу. Спадкування — це корисна функція, яка дозволяє повторно використовувати код.

13. What is functional programming? What is OOP programming? Which type does JS use?

**Функціональне програмування** — це парадигма створення комп'ютерних програм за допомогою виразів і функцій без зміни стану та даних . Дотримуючись цих обмежень, функціональне програмування прагне написати код, який є зрозумілим і більш стійким до помилок.

**Object-oriented programming** (OOP) це стиль програмування, що характеризується ідентифікацією класів об’єктів, тісно пов’язаних із методами (функціями), з якими вони пов’язані. Він також включає ідеї успадкування атрибутів і методів.

**JavaScript is a functional oriented programming language**, but it can also be used as object oriented programming language with classes and prototype with javascript pattern.

**DOM & Events**

1. What is **DOM**?

**DOM (об'єктна модель документа)** являє собою HTML-розмітку в JavaScript. Кожен тег представлений у вигляді об’єкта JavaScript і дозволяє керувати (додавати / змінювати / видаляти) елементи на сторінці з наших скриптів

*DOM розглядає документ HTML як дерево вузлів. Вузол представляє елемент HTML.*

2. What is **BOM**?

Об’єктна модель веб-переглядача (BOM) — це угода, що стосується конкретного веб-переглядача й стосується всіх об’єктів, доступних веб-браузеру . BOM дозволяє JavaScript «взаємодіяти» з браузером. Об'єкт window представляє вікно браузера та всі його відповідні функції.

3. How to **find DOM elements** on the page? Count all methods you know.

Щоб знайти елементи на сторінці, є багато способів.

У більшості випадків потрібно використовувати 2 методи: **element.querySelector();і element.querySelectorAll();**

Є й інші ( *document.getElementById, document.getElementByTagName, document.getElementByClassName* і т.д.), але вони застаріли і краще не користуватися ними.

4. What is a **browser event?**

**Подія – це сигнал від браузера** у тому, що щось сталося. Всі DOM-вузли подають такі сигнали (хоча події бувають не тільки в DOM).

5. How can you **navigate through DOM objects** on the page?

The nodes in the node tree have hierarchical relationships to each other. The terms parent, child, and sibling are used to describe the relationships.

• The top node called top node or the root or the root node

• Every node has exactly one parent, except the root node

• A node can have a number of children

• Siblings (brother and sister) are nodes with the same parent.

To **navigate** through **DOM** we can use properties such as :

• parentNode

• childNodes[nodenumber]

• firstChild

• lastChild

• nextSibling

• previousSibiling

6. How to **define an event handler**? What methods do you know?

💩назначити обробник *прямо в HTML-розмітці*.

💩Вкажіть обробіток, як *властивість елемента DOM*

😎Назначити ф-ції **element.addEventListener();**

7. How to define **multiple handlers for the same event** on the element?

You can add as many as you want event handles to the same element, they will not overwrite each other. Use **addEventListener();**

For example:

*element.addEventListener("click", myFunction);*

*element.addEventListener("click", mySecondFunction);*

8.What is the difference between

**someElement.onclick = function () { ... }** *and* **someElement.addEventListener('click', ....)**?

9. What is the **difference between event.stopPropagation() and event.preventDefault()?**

The *event.preventDefault()* will not allow the user to leave the page and open the URL.

The *event.stopPropagation()* method stops the propagation of an event from occurring in the bubbling or capturing phase.

*event.preventDefault()* - Метод event.preventDefault дозволяє скасувати дії браузера за замовчуванням. Наприклад, зробити так, щоб при натисканні на кнопку переходу на іншу сторінку не було або після натискання на кнопку форма не відправлялася на сервер.

*event.stopPropagation()* - припиняє подальшу передачу поточної події. Зупиняє всплиття. Корисно для делегування подій, можна написати один обробник подій повісити його на батьківський елемент та використовувати його для пошуку потрібного елемента event.target

10. How does **removeEventListener** work?

The *removeEventListener()* method removes an event handler from an element.

For Example:

If a button is disabled after one click you can use removeEventListener() to remove a click event listener.

*target.removeEventListener(type, listener[, options]);*

• Type описує тип події яку потрібно видалити.

• Listener функція яку потрібно видалити

11. What is **event bubbling**? How can we use it in practice?

Коли подія відбувається на елементі, спочатку запускаються обробники на ньому, потім на його батьківському елементі, а потім на інших предках.

For Example:

When I had a list of books, in an ordered list and had a delete button next to each book, before bubbling to its potential, I was adding event listeners to each button. There was a problem: if I needed to add another book , I had to add another event listener to the new delete button.

So, instead I added an event listener to ul and from there i was checking if the user clicked on a button and deleting the book that needed to be deleted, at the end i had only one event listener and it would work if i add unlimited amount of book to the list.

12. What is event **delegation**? How does it work?

**Delegation** is a way that you can add an event listener once for multiple elements with a support adding extra children later on.

In the handler we get event.target to see where the event actually happened and handle it.

Algorithm:

• Add one handler per container.

• In the handler - check the event.target output element.

• If the event occurred inside the element we're interested in, process the event.

Benefits:

• Simplifies initialization and saves memory: no need to add many handlers.

• Less Code: No need to add/remove handlers when adding or removing elements.

• DOM Modifications: We can bulk add/remove elements using innerHTML etc.

I personally used to be on the list of fruits, when clicked made the background color red. When clicked on the list , that li became a red background, even if I added a fruit.

**Async code, promises, fetch, error handling**

1. What is **EventLoop**? How does it work?

JavaScript is a single tretted

EventLoop in JavaScript is an endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks. The event loop executes tasks from the event queue only when the call stack is empty, there are no ongoing tasks at the moment.

The Event Loop has one simple job - to monitor the Call Stack and Callback Queue.

If the Call Stack is empty, the Event Loop will take the first event from the queue and will push it to the Call Stack, which effectively runs it. Such an iteration is called tick in the Event Loop.

In other words, the website or application won’t freeze up when there is an asynchronous action happening at the moment. Like a http request to the server, the app will still run at the because that request will go to WEB API and while we wait for the http response from the server, other code that needs to be executed will still run, response will go to queue and as soon as the stack will get empty that response will be pushed to the stack and will be executed there.

*Stack last goes in , first goes out.*

*Queue first goes in first goes out.*

2. Can I **run JS code in 2 threads**? (NEW)

**JavaScript is a single threaded programming language,** so it can only handle one operation at the time.

Запуск **JavaScript на GraalVM** підтримує багатопотоковість . Залежно від сценарію використання потоки можна використовувати для виконання паралельного коду JavaScript з використанням кількох об’єктів контексту або кількох робочих потоків.

3. What is **Promise**? How is it used in front-end applications?

**Promise is a special object** *that is used to handle asynchronous operations* in javascript. Promise is a container for a future value.

**Promise has 3 states**

*Pending* - initial state, neither fulfilled nor rejected.

*Fulfilled* - meaning that the operation was completed successfully.

*Rejected* - meaning that the operation failed.

**Promise has 2 properties**

State - initial state is pending, after fulfilled or rejected it is immutable.

Result - initial value is undefined, after fulfilled it is a value, after rejected it is an error.

4. What is **promisification**? Give me at least 2 examples

**Promisification** is a fancy word for converting a callback function into a promise function. It does help to avoid callback hell, since code doesn’t grow horizontally and it easier to read and understand and maintain this way, all you have to do is use function .then() to get the result of successful promise and work with it or catch the error if that happened and handle that error in correct manner.

5. Which **statuses promise** can have? Is it possible to change it? If yes, how? (NEW)

A **promise status**  is in one of these states: pending: initial state, neither fulfilled nor rejected. Fulfill meaning that the operation was completed successfully and rejected meaning that the operation failed. I can not change the status of a promise.

6. What **static method of Promise** do you know?

7. How to run **multiple promises simultaneously** and wait till all of them will be

resolved?

I can use **promise.all()**  to achieve this result, it will run until all the promises will be resolved. But there is a catch if one will be rejected, another after that one will be ignored, but if I use **promise.allSettled()**  then even rejected ones will go inside the array of the results of each and single promise.

8. When **should I use Promise.all()**? Give me some examples when you've used it in projects?

I should use the promise.all() method when I have multiple asynchronous code running, and I need to do something to all of the results from resolved promises. Like waiting for several API requests to finish so you can combine their results.

9. What is **asynchronous code?** Why is it important for web applications? Give examples of async and sync code in JavaScript

Asynchronous code in JavaScript is the code that can run parallel to another code, without blocking the process of the whole program. For example, image processing and server requests take some time, if we had no asynchronous code in javascript , while we wait for the server response or until the image will finish loading, everything else on the application will freeze and won't be responding. This is happening because JavaScript is a single threatening programming language, meaning only one thing at the time can be processed, that’s where asynchronous code comes into place and makes our lives easier and application more responsive and intuitive for users.

*Examples of synchronous code is :*

console.log('1')

console.log('2')

console.log('3')

*Examples of asynchronous code is :*

console.log('1')

setTimeout(function afterTwoSeconds() {

console.log('2')

}, 2000)

console.log('3')

10. What is **async, await**? How to **handle errors** when using **async await**?

Async/Await makes it easier to write promises. The keyword ‘async’ before a function makes the function return a promise, always. And the keyword await is used inside the async function, which makes the program wait until the promise resolves.

To handle error when working async await we can do one of the following:

• We can use try...catch for synchronous code.

• We can use try...catch (in combination with async functions) and the . .catch() approaches to handle errors for asynchronous code.

• When returning a promise within a try block, make sure to await it if you want the try...catch block to catch the error.

• Be aware when wrapping errors and rethrowing, that you lose the stack trace with the origin of the error.

11. What is a JavaScript **error**? What kinds of errors do you know? How to handle errors?

**Errors** are statements that don't let the program run properly.

I know these types of errors:

• Reference Error

• Range Error

• Syntax Error

• Type Error

To handle Errors is the best practice to use try catch.

For example:

try

{

// code that may throw an error

}

catch(ex)

{

// code to be executed if an error occurs

}

finally{

// code to be executed regardless of an error occurs or not

}

12. What is the **throw** keyword in JS?

The throw statement throws a user-defined exception. Execution of the current function will stop (the statements after throw won't be executed), and control will be passed to the first catch block in the call stack.

Use the JavaScript throw statement to throw a user-defined exception. Users can create user-defined exceptions and throw the error message according to their needs.

General JS

1. What is a **callback**? How does it work and why do we need that?

A callback function is a function passed into another function as an argument, which is then invoked inside the pouter function to complete some kind of a routine or action.

We need callback functions because many JavaScript actions are asynchronous, which means they don’t really stop the program (or a function) from running until they’re completed, as you’re probably used to. Instead, it will execute in the background while the rest of the code runs.

2. What are **ES Modules?** How does it work? What is the difference between named and default export?

A JavaScript module is a file that allows you to export its code. This allows other JavaScript files to import and use exported code as their dependencies. It's very useful when we need to use a function or a variable from one file into another file, helps us to have a structured file system and organized code.

The difference between named export and default export is that you can only have one default export per file, when named exports can be as many as you like per file, and you put them in a curly brackets.

3. What is the difference between **localStorage** and **sessionStorage**? When is one better?

The main difference is that **sessionStorage**  will be lost if the page is reloaded when in **localStorage**  that won’t happen.

4. What are **cookies**?

Cookies are small pieces of data that are stored locally on your computer. It helps websites to understand more users and users get more interactive and useful and helpful to get personal.

5. What is usage of **JSON.stringify** and **JSON.parse** methods?

• JSON.stringify() takes a JavaScript object and then transforms it into a JSON string.

• JSON.parse() takes a JSON string and then transforms it into a JavaScript object.

6. What is the difference between **cookies/localStorage/sessionStorage**?

7. What is **chaining**? How does it work?

Chaining in JS is nothing but grouping in one single line using dot notation. This type of chaining makes the code very concise and also improves the performance. For example, I can use filter map and reduce one array like this:

*let ages = data*

*.filter(isDog)*

*.map(dogYears)*

*.reduce(sum);*

*// ages = 84*

**WEB related questions**

1. What is **HTTP**? What does the HTTP request consist of?

It's a protocol that is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. It follows the classic client-server model , with a client open connection to make a request, then waiting for the server to make a response. HTTP is a stateless protocol meaning that the server does not keep any data (state) between two requests, each request and response is independent.

An HTTP request is made out of three components: **request line, headers and message body.**

**Request**

The request line is sent by the client in order to start the action on the server. It includes following:

• An HTTP method

• The request-target which can be a URI or URL to either a path or protocol. URI is an identifier for a specific page, while URL is a special type of identifier that can also contain information about how to access resources.

• The HTTP version that defines the structure of the remaining message.

**Header**

The HTTP header allows for addiction information to be passed between server and client such as cookies, information about the authorization token, or user agent using a special string that helps server identify client browser and OS version… Similar to the same basic structure of HTTP request, the HTTP headers are case-sensitive and are followed by colon (:) and value.

**Message Body**

The server uses the message body to deliver the information back to the client.   
The message body contains the information, the request line, header, an empty line, the message body that is optional. While not all requests have body, the ones that do, often use POST to deliver payload.

2. What is the **request body**?

**Request Body** is the part of the HTTP Request where additional content can be sent to the server. For example, a file type of JSON or XML. Request body tries to send additional information required by the server to process the current request properly.

3. What is the **header in an HTTP** request? Give me some examples you've used?

Header in an http request is additional information for the server.

I used it for key tokens, for content types such as json and some other meta information.

4. How does the **fetch** method work?

The **fetch() method** starts the process of fetching a resource from the network, returning a promise which is fulfilled once the response is available. It takes arguments such as URLs.

5. What is the **status of an HTTP request**? What statuses do you know?

This is a code of http request to show the status of that request.

There are a total of 63 status code of a http request, I usually check what it stats :

*For Example:*

100’s are informational status ( continue, processing, early hints )

200’s are successful status ( ok, created, accepted and alike )

300’s are redirections status ( temporary redirect, permanent redirect or moved )

400’s client error status ( bad requests, unauthorized, forbidden )

500’s server error ( internal server error, bad gateway, service unavailable )

6. What is **REST API**? Why is it good?

A REST API is a standardized architecture style for creating a Web Service API. One of the requirements to be a REST API is the utilization of HTTP methods to make a request over a network. You need to have a URL Path , HTTP Method, Header, Parameters and Body.

7. Which **methods of the REST API** do you know?

I know 5 REST API methods :

• GET

• POST

• PUT

• PATCH

• DELETE

8. What is the difference between **PUT &** **PATCH** methods?

Both of these methods update the data , PUT updates the data and modifies the whole source when PATCH methods only updates and modifies resources where the client sends partial data that is to be updated without modifying the entire data.

9. What is the **OPTIONS** method in the REST API? Why can I make HTTP requests with this method automatically when the page is loaded?

OPTION method returns info about API (methods/content type)

It will shoot the OPTION method at first to check if it's safe to do any other requests from that REST API , and it will give all the information needed to work with that REST API from that point on.

10. What is **CORS**?

Cross Origin Resource Sharing is an HTTP-header based mechanism that allows a server to indicate any origins ( domain, scheme or port) other than its own from which a browser should permit loading resources.

11. Tell me about **APIs you've used** on your projects?

I used crudcrud.com and api-football-data.org for practice and mockapi.com for my projects.

12. What is **JSON**?

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

13. How to make **comments in JSON**?

JSON is data-only, that’s for , you can not use regular commenting as you use in programming languages, you can include data as a comment like this “\_comment” but that will be a data inside that JSON as well, even though it might be ignored by the apps that use JSON format for data.

14. What is **AJAX**?

AJAX is Asynchronous JavaScript And XML.

It’s a technology that lets you update pieces of a website without updating the whole page, it uses format such as text file, html or json object to send to the server and server only updates that piece of website that needs to be updated.

15. When I open any web site, let's say some new website, what is happening? Describe the flow step by step.

The browser goes to the DNS server and finds the real address of the server that the website lives on, then it sends a http request asking for the copy of that server , then if everything is alright, it sends the context to the browser and the browser displays the server.

16. What is **Websockets**? What is it used for?

It's an advanced technology that makes it possible to open a two-way interactive communication session between the user’s browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

17. What is **axios** lib used for? Have you used it?

Axios is used to get data from the server http requests, similar to fetch but fetch is two-step process when handling JSON data-first, to make an accrual request, second to call the .json() method to response, when axios is automatic transforms of JSON data.

**Programming principles**

1.How to **debug** JavaScript code in the browser?

To debug JavaScript code in the browser , I need to open up a project that I want to debug with live server and then in the browser right click on the windows and inspect that will open up DevTools, from there I need to go to Source , select the file that I want to debug and set some breakpoints , in the code , that will pause the execution of the code at that time I can see what each variable is equal to, and see if everything is working the way I want it to work. Fix code in VsCode and the updated code will be in the debugger, repeat this process until I am satisfied with the results.

2.Describe your last case where you've **used a debugger** in the project?

When I was building toDoList in JavaScript, I had to debug that app in order to see if the correct values are stored in the todo and if i check the todo as done it won’t be sorted and stayed at the same spot, that’s when i used the debugger.

3. What is a **good code**? **And bad**? Give me an example that you've had on your projects?

For me the bad code is the code that needs refactoring or code that need to be redone from the scratch, like if there is unnecessary variables and for loops that can be solved by methods and simplified solutions, it also a code that is hard to read , bad naming of variables, or just too complex and one function or method is doing to much when return has some kind of operation , calculations in it, when there are unnecessary iterations through arrays or objects

For me a good code is that it is simplified, refactored, easy to read, understand, easy to add things to and maintain, the code that has straightforward naming and is formated.

I personally try to make the application work, maybe with the bad coding, and then go back to it and try to refactor it and make it better as well as follow coding styles and best practices.

4. How to **avoid code duplications** in your app?

In order to avoid code duplication in my application, I try to put something in a variable that is used more than once, create a function or use destructuring to get values that I need instead of getting it everytime.

5. What are **SOLID** principles? Which of them have you used?

• Single Responsibility - class must do one thing

• Open/Close - open for extension closed for modification

• Liskov’s Substitution - layers of usage of classes

• Interface -

• Dependency

I used the single responsibility principle, every function should do one thing and make another function that can take that function inside to do more things. Same thing with classes.

**React**

REACT- intro

1. **What is React?** Which problems does it solve?

**React is an open-source JavaScript library** developed by the team at Facebook. It is used to build rich graphical user interfaces (UIs). It is based on the concept of reusable components, which allows you to create complex UIs out of small isolated pieces of code called components.

• Don’t touch the DOM, I'll do it. (Virtual Dom)

• Build websites like lego blocks (Components)

• Unidirectional data flow(easy to find bugs)

• UI, The rest is up to you(cross platform with other libraries of react)

**At its core, React is a solution to a problem** that developers faced when building user interfaces. It allows developers to build complex user interfaces which have components that will change frequently over time, without having to write a lot of very tricky JavaScript code.

2. **What is JSX?** How does it work?

**JSX is a syntax extension to JavaScript.** It describes how the UI should look like. JSX looks like HTML language but it comes with the full power of JavaScript.

The way it works, that babel will transform JSX to JavaScript for the browser to understand, and it makes developers and regular people understand JSX syntax easier. So we can develop in JSX and babel will transform it to JavaScript.

3. Can I **use JSX? without React?**

**Yes you can use JSX without React,** JSX was invented before React and was used without React.

4. Can I put a **JSX expression to the variable?**

**Yes I can put JSX expressions into a variable**, but when I need to use it I need to put it in curly braces. And btw, when I do render multiple elements I can put them in a variable and render the variable to the document.

5. How to **put a variable value into the JSX?**

To put variable value into JSX all I have to do is to wrap it up in curly braces, then it becomes a simple JavaScript variable.

6. What **data types** can **pass** to the **attributes in JSX?** What is the difference with HTML attributes?

I can pass all data types to the attributes in JSX just like I do in HTML, but for naming I need to use camelcase , for example class, becomes className , for becomes htmlFor.

7. What is **create-react-app?** Is it better than custom config?

Create-react-app is a npm command.Used to create a react application, It will create a react application with all dependencies and everything ready to use, one downfall from that is that I need to delete all other unnecessary files from it that comes straight out of the box. Custom configuration takes time, and I will only create what I need in the project and use. If I can custom config my react app, to the latest configuration, and have that folder somewhere backed up in order for making another react app in ease , then custom configuration is better.

8. What is the **React component?** How to **create a React component?**

**React Components** are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

To create a react component is the best idea to separate each component into a separate file and name file starting with a capital letter and extension .js or .jsx. But the best practice is to use extension .jsx

When making a class component, it needs to extend from React.Component class to have all the future from the main React.Component library.

9. What is the **difference between class component and functional component?** Which is better and why?

For me it makes more sense to use class components, simply because I came from C# and .NET, but there are some disadvantages in using class components, they are slower because of live cycles methods and state hooks, and plus you can lose context with this in class components. So I have to be careful! Compared to functional components, they are easier to use, because they are simple pure functions from JavaScript, they are easier to test and modify and if wont lose context.

10. What does the **React component return?**

React components return JSX, basically it’s a mixture of HTML and JavaScript.

11. What are **component properties?** How to change properties from inside the component?

Component properties are arguments passed into react components. They are like functions, arguments in JavaScript and attributes in HTML. The component receives the arguments as a props object.It is used in React to pass data from one component to another, from parent component to child.

You can not change props from inside the component.

12. What **data types can be passed through the properties?**

We can pass any JavaScript data types through props, from integer to object or array, pretty much any data types can be passed through props.

13. What is the **component state?** How to **create it**? How to **update it**?

Component state is an instance of a react component class, it can change over the lifetime of a component and whenever it changes component re renders.

To create a state inside a react component, I need either use class and add this.state to a constructor of a class or use hooks inside functional components.

To update state we can use the setState method to change the state of the component directly using a JavaScript object where keys are the name of the state and values are the updated value of that state. Often we update the state of the component based on its previous state.

14. How does **setState work**? Is it **synchronous or asynchronous**?

**setState(updater[, callback])** setState() ensures that the component has been updated and calls for re-rendering of the component. It’s an asynchronous action, since it's a request rather than an immediate comment to update the component. For better performance, react may delay the update and later on update multiple components at once.

15. What is the **trigger for components to be re-rendered?**

To trigger component to be rerendered:

• State changes, every time state changes react runs the render method.

• When props change

• Re-render with key prop

• Force a re-render

16. How to **optimize component rendering?** Tell me all options you've used in your projects? (NEW) (ADVANCED)

17. What is the **difference between state and props?**

18. What is **Virtual DOM**? How is it different from **JS DOM**?

Dom is an interface that allows the script to update the content, style, and structure of the document. Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.

19. How to **handle an event in React?**

To handle events in React is similar to handling events on DOM elements, there are some syntax differences; naming of an event is rather camelcase with a prefix on, than lowercase and with JSX you pass a function as the event handler, rather than a string. You can not return false to prevent default behavior in React, you must call preventDefault explicitly.

20. **What is usage of addEventListener in React apps?**

When using React, you generally don't need to call addEventListener to add listeners to a DOM element after it is created. Instead, just provide a listener when the element is initially rendered. You have to be careful about the meaning of this in JSX callbacks. In JavaScript, class methods are not bound by default.

REACT

1. What is **conditional rendering?** How to achieve it?

It’s the same as conditions work in JavaScript. I can use if statement, if else statement, switch case, && || logic operators, ternary operators, even state can be taken into consideration and conditional rendering happening when state is like boolean.

2. How to **render lists in React?** What is specific?

We can use map to put elements in ordered or unordered lists, we put in the callback of map JSX as a list item , that’s all. But specific about lists in react that list items need to have unique id.

3. What is the reason to use **key property for list items?** Why should it be **unique?** What can happen if keys are **not unique?**

The reason is for the list items to have keys, it's for reacting to know which item was updated , added or deleted, and it has to be unique in order for performance optimizations and work properly in big lists and don’t have interference between each other. If the keys are not unique it might break logic of the application but for sure will slow down the performance of the application.

4. Can I use **array indexes for key values?** What to do if I don't have a unique identification in objects I need to render in a list?

I can use array indexes as key values, since they are unique. If I don’t have a unique identifier in the object to use for list items, I use the library to generate strong and unique keys for the react list called UUID or UNIQID. This is the npm package. But the best bet is to use a unique id from the database if I get one.

5. What is the **component mounting process?**

Component mounting process is when a component is created and inserted into DOM.

6. What *lifecycle hooks of React components do you know?*What are they used for?

I know useEffect, and it used to control all lifecycles and states at the same time in react components.

7. What is **componentDidMount** ?What actions should be done there?

This method is invoked immediately after the component is mounted into a page to, into the tree. This is the best place to use network information requests and to add subscriptions to the events.

8. What is **componentWillUnmount** ? What actions should be done there?

This method is invoked immediately before the component is unmounted and destroyed. This is the best place to clean up unnecessary, such as, invalidating timers, canceling network requests, or cleaning up any subscriptions to any events that were created in componentDidMount method.

9. What is **componentDidUpdate** ? What actions should be done there?

This method is immediately invoked after updating occurs. It takes previous props and previous state and takes a snapshot of that and compares it to the current props and current state to see if it was updated or not.

10. What is a **constructor**? What actions should be done there?

Constructors are used to react only in class components , to initialize the state of instance of a component or add handlers there.

11. What is a **Pure Component** ? (NEW)

Pure components in react are the components which do not re-render when the value of state and props has been updated with the same value. If the value of the previous state or props and the new state or props is the same, the component will not re-render.

12. What is **shouldComponentUpdate**? How does it work? What do you know about PureComponent

The shouldComponentUpdate is a lifecycle method in React. This method makes the component re-render only when there is a change in state or props of a component and that change will affect the output.

PureComponent will not re-render if state nor props stay the same.

13. What is a **controlled component** and what is **uncontrolled**?

Controlled components is the component that is controlled within the component, meaning by state or props.

Uncontrolled components are the component that is controlled by the DOM.

14. How to work with **forms in React**? Tell me all the options you know? Maybe you've *used some libraries*?

In React, form data is usually handled by the components. When the data is handled by the components, all the data is stored in the component state. I can control changes by adding handlers in the onChange attribute. Also I can use a special library for react forms such as Formik or Redux-Form and as well as React Hook For.

15. What is **ref**? When is it reasonable to use refs?

Refs are a function provided by React to access the DOM element and React element that you might have created on your own. They are used in cases where we want to change the val;ue of a child component, without making use of props and all.

16. What is the **stateful** and **stateless** **component**?

In React, a stateful component is a component that holds some state.

Stateless components are those components which don't have any state at all, which means you can't use this. setState inside these components. It is like a normal function with no render method. It has no lifecycle, so it is not possible to use lifecycle methods such as componentDidMount and other hooks.

17. What is **unidirectional data flow**? Which other types of data flow do you know in modern UI frameworks?

Unidirectional data flow describes a one-way data flow where the data can move in only one pathway when being transferred between different parts of the program. React, a Javascript library, uses unidirectional data flow. The data from the parent is known as props.

I also know that there is a bidirectional data flow in some frameworks, such as Angular.

18. How do **different components share data**?

If I want to pass data from parent component to child component I use props to pass data.

There are no direct ways to pass data from child component to a parent component, there are workarounds such as :

*Following are the steps to pass data from child component to parent component:*

• In the parent component, create a callback function. ...

• Pass the callback function to the child as a props from the parent component.

• The child component calls the parent callback function using props and passes the data to the parent component.

19. What is **children's property**?

Children is a special prop, automatically passed to every component, that can be used to render the content included between the opening and closing tags when invoking a component. These kinds of components are identified by the official documentation as “boxes”.

20. How not to miss **context in components methods**?

Either bind context, use arrow function or use functional components. Those are three easiest ways that I have practiced.

21. What is **State Lifting**? Give me examples where you've used it in your projects.

State lifting means to move the state up to its ancestor.

When I made a simple app that has two inputs, one in celsius and another one in fahrenheit, and as I type in numbers in other of them the second one would convert it and under it would say if the water will boil or not.

22. What tools did you use to t**est your React apps**?

I know there are frameworks to test components in react, enzyme if good. Also I can write tests in Jest, Mocha, Ava but I personally use the chrome extension and run the app in development mode and open, run chrome dev tools and use Profiler to debug it.

23. Write a component with conditional rendering and write tests for it.

24. What is a **Single Page Application** (SPA)?

Single Page Application, is an application that loads a document once and after that it only updates specific parts on a page that needs to be updated for a user to display.

25. What is **Progressive Web Application** (PWA)?

Progressive Web Application (PWA) – це додаток, створений за допомогою веб-технологій і є гібридом звичайного веб-сайту, доступ до якого здійснюється через браузер (в тому числі і браузер мобільного додатка). Такі гібридні веб-додатки імітують досвід використання активних додатків і мають максимально наближений до них зовнішній вигляд і юзабіліті.

26. What is **Server Side Rendering** (SSR)? Which benefits does it have? Do you have experience using it?

SSR - це здатність програми перетворювати файли HTML на сервері на повністю відрендерену сторінку HTML для клієнта . Веб-браузер надсилає запит на інформацію від сервера, який миттєво відповідає, надсилаючи повністю відтворену сторінку клієнту.

Візуалізація на стороні сервера покращує швидкість сайту та покращує показники Core Web Vitals . Однак іноді це може бути важко реалізувати, а також може збільшити затримку першого введення.

27. What is **routing**? How to **implement routing** in React app?

React Router is a fully-featured client and server-side library for React, a JavaScript library for building user interfaces. React Router runs anywhere React runs; on the web, on the server with node.js and on React Native. React Router is used to define multiple routes in the application.

*To implement React Router into aur project we need to*

• Run npm install react-router-dom@6

• import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';

• And finally implement Router, Switch, Route,Link into code.

28. What are **route params?** How to get information **from the URL in your React component?**

Route params are parameters whose values are set dynamically in the page's URL. This allows a route to render the same component while passing that component dynamic portion of the URL, so that it can change its data based on parameter.

To get information from URL into my React component;

*import {useParams} from “react-router-dom”; (In component)*

*<Route path= “/client/:id” exact> (in app)*

*<ClientDetail />*

*</Route>*

*const ClientDetail = () => {*

*const { id }: { id: string } = useParams(): (in component)*

29. How to build **nested routing with react-routing-dom**?

To build nested routes, we need to append a

/\*

to the end of our / messages path, we’ve essentially telling React Router that messages has a nested Routes component and our path should math for / messages as well as any other location that the / messages/\* pattern.

Then also we need to:

*import { Routes, Route, Link, Outlet } from ‘react-router-dom’;*

30. What do you know about **React hooks?** Tell me about **at least 2 of them and how they work?**

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

I know and use most of a time two hooks or regular basis;

*usestate();* Hook to control and change state inside a functional component.

useState accepts an initial state and returns two values:

• The current state.

• A function that updates the state.

*Example:*

*import { useState } from "react";*

*function FavoriteColor() {*

*const [color, setColor] = useState("");*

*}*

*useEffect()*; Hook that uses the same as lifecycle hooks, in functional components.

The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

*Example:*

*useEffect(() => {*

*//Runs on the first render*

*//And any time any dependency value changes*

*}, [prop, state]);*

31. What is **HOC** (**Higher Order Components**)? What are cases of using HOC?

High Order Component is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are patterns that emerge from React’s compositional nature.

Concretely, a higher-order component is a function that takes a component and returns a new component.

For Example :

If I need to toggle functionality in two different components such as toggle between paragraph and an input, this will allow me to toggle between editing and viewing the title of a post.

If I need to toggle between collapsing and expanding the list.

Another Example:

If I need to create, let say a component that is showing stocks and prices of them. I make a table component with stocks and also a row component that shows the prices of stocks.

If I need to make a component that shows a list of users and rows of students that are users as well.

*I can use the same High Order Component to make the same functionality as I did in an app before with stocks.*

**Redux**

• What is Redux?

Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great development experience, such as live coding editing combined with a time traveling debugger.

You can use Redux together with React, or with any other view library.

In a simple words, Redux helps us to store state and control it from it’s core, basically we don't need to pass state to parent component and then use it with props in another component, when app gets big it becomes confusing and bad practice, so it's easier and makes sense to pass state to redux and use action via reducer to change the state and app becomes more efficient and easier to edit, fix, test and grow.

• What are 3 main principles that Redux follows?

3 principles of Redux are:

1. Single source of truth - the state of your whole application is stored in an object tree within a single store.

2. State is read-only - the only way to change the state is to emit an action, an object describing what happened.

3. Changes are made with pure functions - to specify how the state tree is transformed by actions, you write pure reducers.

• What are the advantages and disadvantages of Redux? (NEW)

Redux allows you to manage your app's state in a single place and keep changes in your app more predictable and traceable. It makes it easier to reason about changes occurring in your app. But all of these benefits come with tradeoffs and constraints.

Pros:

• Increase the Predictability of s State

• It is Highly Maintainable

• It Prevents -Rerendering

• Redux Optimizes Performance

• Makes Debugger Easier

• Useful in Server-Side Rendering

• Provides Ease of Testing

Cons:

• Lack of Encapsulation

• Restricted Design

• Excessive Memory Use

• Increased Complexity

• Time-Consuming

• What is the difference between React context and React redux? (NEW)

The main difference between these two libraries is that Redux works in a centralized manner, in other ways in redux you can set the data in one component while other components have the right to access the data.

In React Context you can share information between components using props but that is only one direction and that is a problem.

So to sum it up in simple words, if you have a big project that contains a lot of components you should use redux because context api can't wrappe a lot of components.

• How do you understand the 'Single source of truth' principle?

In React-Redux applications, when your Redux is a single source of truth, it means that the only way to change your data in UI is to dispatch redux action which will change state within redux reducer. And your React components will watch this reducer and if that reducer changes, then UI will change itself too.

• How do you understand the 'State is read-only' principle?

State is read-only: The only way to change the state is to emit an action, an object describing what happened. That means that every time that you need to change a property data in the store, you need to dispatch an action.

• What is pure function? (NEW)

A function is called pure if it abides by the following rules − A function returns the same result for the same arguments. Its evaluation has no side effects, i.e., it does not alter input data. No mutation of local & global variables. It does not depend on the external state like a global variable.

• How do you understand the 'Changes are made with pure functions' principle?

Reducers are just pure functions that take the previous state and an action, and return the next state. Remember to return new state objects, instead of mutating the previous state. You can start with a single reducer, and as your app grows, split it off into smaller reducers that manage specific parts of the state tree. Because reducers are just functions, you can control the order in which they are called, pass additional data, or even make reusable reducers for common tasks such as pagination.

• What are components of Redux?

Redux has 4 components namely, Store Reducer Action and Middleware.

• Store holds the state tree of our app.

• Reducer is a pure function, which returns the state of the application based on the action dispatched by the store.

• Action is a payload information that is transmitted to a store.

• Middleware is the suggested way to extend Redux with custom functionality. Middlewares are used to dispatch async functions. We configure Middleware while creating a store.

• What is Redux Store?

A store holds the whole state tree of your application. The only way to change the state inside it is to dispatch an action on it.

A store is not a class. It's just an object with a few methods on it. To create it, pass your root reducing function to createStore.

• How to set initial state in Redux? (NEW)

You can set it at the reducers . Reducers can also set initialState by looking at the incoming state argument (which would be undefined if createStore is not called with initialState ) and returning the values they would like to use as default.

• How to make changes to the Redux store?

The only way to update a state inside a store is to dispatch an action and define a reducer function to perform tasks based on the given actions. Once dispatched, the action goes inside the reducer functions which performs the tasks and return the updated state to the store.

• How to get data from Store in React components? Do you know about selectors? What are the benefits of using selectors?

It's simple to get access to the store inside a React component – no need to pass the store as a prop or import it, just use the connect function from React Redux, and supply a mapStateToProps function that pulls out the data you need. Then, inside the component, you can pass that data to a function that needs it.

A "selector function" is any function that accepts the Redux store state (or part of the state) as an argument, and returns data that is based on that state.

• What are actions in Redux? (NEW)

Actions are a plain JavaScript object that contains information. Actions are the only source of information for the store. Actions have a type field that tells what kind of action to perform and all other fields contain information or data.

• How to make Ajax requests in Redux?

Redux already has an official version of that "async function middleware", called the Redux "Thunk" middleware. The thunk middleware allows us to write functions that get dispatch and getState as arguments. The thunk functions can have any async logic we want inside, and that logic can dispatch actions and read the store state as needed.

Writing async logic as thunk functions allows us to reuse that logic without knowing what Redux store we're using ahead of time.

• What is asynchronous redux action? How do you work with asynchronous code in your react - redux applications?

Just like with a normal action, we first need to handle a user event in the application, such as a click on a button. Then, we call dispatch() , and pass in something, whether it be a plain action object, a function, or some other value that a middleware can look for.

• What is Redux middleware?

Redux Middleware allows you to intercept every action sent to the reducer so you can make changes to the action or cancel the action. Middleware helps you with logging, error reporting, making asynchronous requests, and a whole lot more.

• How to add multiple middlewares to Redux? (NEW)

You can use applyMiddleware() .

It lets you dispatch a Promise async action, and dispatches a normal action when the Promise resolves. Middleware is not baked into createStore and is not a fundamental part of the Redux architecture, but we consider it useful enough to be supported right in the core.

• What are reducers in Redux? (NEW)

In redux, the reducers are the pure functions that contain the logic and calculation that needed to be performed on the state. These functions accept the initial state of the state being used and the action type. It updates the state and responds with the new state.

• What are Redux selectors and Why to use them? (NEW)

A selector is a function that accepts Redux state as an argument and returns data that is derived from that state. Selectors can provide performance optimizations to your application and can also help you encapsulate your global state tree.

In a typical Redux application, the logic for deriving data is usually written as functions we call selectors. Selectors are primarily used to encapsulate logic for looking up specific values from state, logic for actually deriving values, and improving performance by avoiding unnecessary recalculations.

• What is thunk? Which benefits does it have? (NEW) (ADVANCED)

Redux Thunk is a middleware that lets you call action creators that return a function instead of an action object. That function receives the store's dispatch method, which is then used to dispatch regular synchronous actions inside the function's body once the asynchronous operations have been completed.

Thunk allows us to dispatch actions manually, which gives us the power to incorporate some logic or run some asynchronous code before dispatching an action. The function returned from action is called a thunk function which is called with two arguments :

1. dispatch: It is a method used to dispatch actions, that can be received by reducers.

2. getState: It gives access to store inside the thunk function.

• What is redux-saga? Which benefits does it have? (NEW) (ADVANCED)

Redux-saga is a library that aims to make application side effects (i.e. asynchronous things like data fetching and impure things like accessing the browser cache) easier to manage, more efficient to execute, easy to test, and better at handling failures.

By working with effects, Redux Saga makes sagas declarative, rather than imperative, which adds the benefit of a function that returns a simple object, which is easier to test than a function that directly makes an asynchronous call.

• What are the differences between redux-saga and redux-thunk?

Both Redux Thunk and Redux Saga take care of dealing with side effects. In very simple terms, applied to the most common scenario (async functions, specifically AJAX calls) Thunk allows Promises to deal with them, Saga uses Generators. Thunk is simple to use and Promises are familiar to many developers, Saga/Generators are more powerful but you will need to learn them. When Promises are just good enough, so is Thunk, when you deal with more complex cases on a regular basis, Saga gives you better tools.

As an example, what happens when you start an AJAX call in a route/view and then the user moves to a different one? Can you safely let the reducer change the state anyway? Saga makes it trivial to cancel the effect, Thunk requires you to take care of it, with solutions that don't scale as nicely.

In practical terms choosing one or the other one really depends (tautologically) on the project.

One thing to keep in mind is that the two middlewares can coexist, so you can start with Thunks and introduce Sagas when/if you need them (and then choose how/what to refactor with hands-on experience... A solution that especially fits "learning projects", MVPs, et similia) In general terms, Sagas are more powerful and easier to test, but they introduce many new concepts that can be a bit overwhelming if you're also learning other technologies (Redux especially).

Specifically, while dealing with the simple and effective Redux philosophy (actions (literal objects) fed into reducers (pure functions)), you can deal with side effects with Thunk that is more limited but easy to grasp (Promise.then().error()), or with Saga that requires you to face the (powerful) notion that you can do more complex things with those actions.

It's also worth mentioning (redux-)observable has an even more complex (and even more powerful) paradigm to deal with side effects, just in case you are unfamiliar with it (if you already are, it might be easier to use than learning Saga).