Short Answers

1. What is ECMAScript?

ECMAScript is a standardized scripting language specification developed by ECMA International. It serves as the foundation for many scripting languages, one of which being JavaSCript. The purpose of ECMAScript is to define the core language features, including syntax, types, objects, and behaviors, to ensure compatiblity across different implementations of JavaScript.

2. What is the JavaScript engine?

The JavaScript engine is a program (or interpreter) that executes JavaScript code. It is a high-level programming language that conforms to ECMAScript. It reads the JavaScript code, compiles it, and runs it on the host environment, such as a web browser. Key features of the JavaScript engine include parsing, just-in-time compilation, first-class functions, and object-oriented programming. V8 is a popular JavaScript engine used in Google Chrome and Node.js.

3. Explain just-in-time (JIT) compilation. What's the difference between JIT compilation and interpretation?

Just-in-time compilation compiles code during runtime. The entire code is converted into machine code, then executed immediately, which allows for runtime optimizations. JIT compilation improves performance by generating and reusing optimized machine code. This is different from interpretation, which compiles during runtime. The interpreter parses the sorcde code and directly executes instructions line by line. Interpretation is slower, often requires repeating work for frequently executed code, and has limited optimization capabilities.

4. What is REPL?

REPL stands for Read-Eval-Print-Loop. It is a programming language environment that accepts user inputs (statements, expressions, etc.) and outputs the result to the console after execution. We can use REPLs to test basic syntax and operations. Some popular REPLs include Replit and CodeSandbox.

5. What are primitive data types in JS?

Primitive data types are simple, immutable values (meaning, they cannot be changed) that are not objects, have no methods, and are stored on the stack in JavaScript.

6. What are reference data types in JS?

Reference data types in JavaScript are types that store references (memory addresses) to the actual data rather than the data itself. In JavaScript, these are considered objects, and they include arrays, objects, maps, sets, and even functions. Reference data types are mutable, which means you can change them. For example, you cannot change the number 5, but you can change an array with the number 5 in it to a different number, for example you can change [5] to [6].

7. What is type coercion, and how does it differ from type conversion?

In JavaScript, primitive data types can be coerced, which is when one data type is implicitly changed to another, for example, `true + 9` works, and becomes the number 10. In JavaScript, there is also type conversion, where we manually convert one type to another. For example, we can use `String(10)` to turn the number 10 into the string `"10"`, as well as other type conversion functions such as `Boolean()`, `Number()`, `parseInt()`, and the unary `+` operator.

8. What is dynamic typing?

In JavaScript, dynamic typing means you do not need to declare the type when you create a variable. In other words, the type of a variable is determined at runtime based on the value assigned to it, rather than being explicitly declared by the person writing the code (as in some languages such as C or Rust). Generally in "higher-level" languages such as JavaScript or Python, you do not need to declare this, as opposed to "lower-level" programming laugnages.

9. What is immutability? What data types are immutable?

Immutability means you cannot change the data after it has been created. Primitive data types include: string, number, boolean, undefined, null, symbol, and BigInt. For example, you cannot change the number `5` to the number `6`; once the number `5` is assigned to a variable, it will always be `5` unless the variable is explicitly reassigned to a different value.

10. What is the difference between == and ===?

The `==` operator means loose equality. The `===` means strict equality. When we use `==`, it will convert the operands to the same type before the comparison. Cate followed up with some more specifics on the loose equality type coercions that I'm pasting here:

> When you use == in JavaScript, it performs type coercion to compare values of different types. Here's how it generally works:
>
> - If one value is a string and the other is a number, JavaScript converts the string to a number and then compares the numbers.
> - If one value is a boolean, JavaScript converts the boolean to a number (true -> 1, false -> 0) and then continues comparing.
> - If one value is null or undefined, they are only equal to each other (not to anything else).
> - If one value is an object (like an array) and the other is a primitive, JavaScript tries to convert the object to a primitive using the object's toString or valueOf methods.

When we use `===`, JavaScript compares both the value and the type of the operands. It's best practice not to use the loose equality too often because it might have unintended results.

11. What are some examples of falsy values in JS?

Examples of falsy values in JavaScript include null, undefined, the boolean `false`, the number 0, NaN, and an empty string `""`. However, an empty array `[]` or empty object `{}` is considered truthy.

12. Explain short-circuit evaluation.

Short-circuit evaluation is when we evaluate a boolean expression from left to right, and the right operand is evaluated only if the left operand is not enough to determine the final outcome. We use this concept a lot in React when rendering data on a page. For example, the following code will check if articles is true, if it is true, it will render <h2>s with the article titles and <p> tags with the article texts:

```
{articles && articles.map(article => (
    <>
        <h2>{article.title}</h2>
        <p>{article.description}</p>
    </>
))}
```

13. How do primitive and reference data types differ in where they're stored in memory? How does this affect them when they are passed as arguments to a function?

In JavaScript, primitive data types are stored on the stack, while reference data types are stored in the heap, which is memory used for larger, dynamic data. When JavaScript compares reference data types like arrays or other objects, no matter if you use the == or === operators, it will always compare their reference to memory, and not their actual contents. So even if their contents are the same, if two arrays point to different memory locations, they are not equal. Conversely, if they point to the same memory location, they will always be equal to each other. This is slightly different in Python. If we use Python's `Is` operator, it will compare the reference to memory, but when we use Python's == operator, it will compare its contents.

When a reference data type is passed to a function, the reference (address) to the original object is passed. This means that the function can modify the original object because both the original variable and the function parameter refer to the same memory location. But when a primitive data type is passed to a function, the actual value is copied and passed. This means that the function works with a copy of the value, and any changes made inside the function will not affect the original value outside the function.

14. What are 3 ways to declare functions? What is their syntax?

The three ways to declare a function are: function declaration, function expression, and arrow function. The syntax for each is as follows:

```
function print() {}  // function declaration
const print = function() {}  // function expression
const print = () => {}  // arrow function (also function expression)
```

15. How do first-class functions differ from higher-order functions?

First-class functions are treated like other variables. They can be assigned to variables, passed as arguments to a function, and returned from a function. Higher-order functions take in functions as arguments or returns functions.

16. What are pure functions?

Pure functions depend only on their inputs and do not change anything outside of their scope. Given the same input, they will always produce the same result. Reducers in the Redux store are an example of pure functions.

17. What are 3 ways to iterate an array? What is their syntax?

The three ways to iterate an array are: the standard `for loop`, the `forEach()` method, and the `for...of` loop. There's also a `while loop`. The syntax for each is as follows:

```
for (let i = 0; i < array.length; i += 1);  // standard for loop
array.forEach(ele => console.log(ele));  // forEach()
for (let ele of array) {}  // for...of loop

i = 0
while (i < array.length) {  // while loop
    i += 1
}
```

18. What are the major differences between a set and array?

The main difference is that a set only stores unique values (in curly brackets `{}`) while an array can store duplicate values (in square brackets `[]`). They also have different methods. For example, you would use `push()`, `pop()`, `reduce()` on arrays, while you use `add()`, `delete()`, and `has()` on sets. You would use the property `.length` on an array, while you use the property `.size` on a set. You can use a `forEach()` or `for...of` to iterate through both, but you cannot you a standard `for loop` to iterate through a set because a set has no indices.

19. What are the major differences between a map and object?

A map is a special collection in JavaScript that stores key-value pairs where the keys can be of any type (such as a function, object, number, or boolean), and the order of insertion is preserved. In regular objects, keys are typically strings. Maps have different methods such as using `set()` for adding key-value pairs or `get()` for retrieving. For maps, instead of `Object.values()`, `Object.keys()`, and `Object.entries()`, it uses `.values()`, `.keys()`, and `.entries()` (very similar to objects in Python). Like a set, you can use `forEach()` or `for...of` to iterate through both, but you cannot use a standard `for loop` for the same reason.

20. What is the DOM?

DOM stands for Document Object Model. It is a programming interface for web documents that has a tree structure, where HTML elements are represented as nodes containing objects.

21. How can you select an HTML element using JS?

There are many ways to select HTML elements using JavaScript as shown in the following code:

```
document.getElementById();
document.querSelector();
document.querySelectorAll();
document.getElementsByTagName();
document.getElementsByClassName();
```

22. What is a DOM event?

DOM events are actions or occurences on the webpage that your code listens for and responds to using event handlers (`.addEventListener()` and `.removeEventListener()`). Event types include: click, focus, blur, keypress, submit, etc.

23. What is the Event interface?

The Event interface in JavaScript represents an event that takes place in the browser, typically in response to user interactions such as clicks, key presses, mouse movements, etc. The Event object contains properties and methods that describe the event and provide details about what occurred, such as which element was involved or the type of event, and is automatically passed to event handlers.

24. How do we register event handlers for a selected element?

You can also register event handlers by setting the `onclick` property of the element, as in the following code:

```
const button = document.getElementById("custom-button");
button.onclick = function (event) {
    console.log(event.target);
}
```

But it's better to use the `addEventListener()` method, as in the following code:

```
const button = document.getElementById("custom-button");
button.addEventListener("click", event => {
        console.log(event.target);
    });
```

25. What is event propagation? How many phases are there? In what order does it occur?

Event propagation refers to the way events are handled in the DOM when an event is triggered on an element. There are three phases: capturing, target, and bubbling. In the capturing phase, events travel from the root down (the window) into the target element. In the target phase, the event reaches the target element and is handled there. In the bubbling phase, the event then bubbles back up from the target element to the root (the window), allowing parent elements to handle the event.

26. Explain event delegation. Why is it important?

Event delegation is an optimization technique for when you have multiple event handlers that do the same thing. For example, if many sibling elements have the same event handler logic, instead of binding the handler to each individual element, we can bind it once to their parent element. The event handler is triggered when it bubbles up from the target element and out to the parent.