**How many primitive types there are in JS?**

In JavaScript, there are six primitive types:

1. **Boolean**:

Represents a logical value of either true or false.

2. **Null**:

Represents the intentional absence of any object value.

3. **Undefined**:

Represents a variable that has been declared but has not been assigned a value.

4. **Number**:

Represents numeric values, including integers and floating-point numbers.

5. **String:**

Represents a sequence of characters enclosed in single or double quotes.

6. **Symbol:**

Represents a unique and immutable value that may be used as the key of an object property.

These primitive types are distinct from objects, which are more complex data structures in JavaScript.

**Which is false?**

A) Quotes cannot be used inside a String

**B)** **There is no distinction between single-quoted strings and double-quoted strings in JavaScript**

C) The length of string is found in the built- in length

The false statement is B) There is no distinction between single-quoted strings and double-quoted strings in JS.

In JavaScript, both single quotes ('') and double quotes ("") can be used to define strings. They are interchangeable, and there is no functional difference between them. You can use either single or double quotes to create a string based on your preference or specific requirements.

So the correct statement is: There is no distinction between single-quoted strings and double-quoted strings in JS.

**Aggregation should be used when**

A) you do not want a single item to share the same interface as many items

B) creating updates to immutable state

**C)** **collections of objects which need to share common operations**

Aggregation refers to the process of combining multiple objects or entities into a single unit. It is commonly used when you have a collection of objects that need to share common operations or behaviors. By aggregating these objects together, you can apply the same set of operations to the entire collection.

In the context of programming, aggregation is often used to manage collections of objects or to create higher-level abstractions. It allows you to treat a group of objects as a single entity, providing a way to interact with them collectively.

Option A is incorrect because aggregation is typically used to group objects together, rather than separating them into individual items.

Option B is incorrect because creating updates to immutable state is usually associated with immutability and functional programming principles, rather than aggregation.

Therefore, the correct answer is C) collections of objects which need to share common operations.

**Which if the following is a way to register for an event?**

A) Using DOM element properties

B) Using DOM event handler

**C) Both are true**

Both using DOM element properties and using DOM event handlers are valid ways to register for an event in JavaScript.

A) Using DOM element properties: In this approach, you can register an event handler function by assigning it to a specific property of the DOM element. For example, you can assign a function to the `**onclick**` property of a button element to handle the click event.

B) Using DOM event handlers: In this approach, you can use the `**addEventListener`** method to register an event listener function for a specific event type on a DOM element. This method allows you to specify the event type (e.g., "click") and the event handler function to be called when the event occurs.

Both methods allow you to respond to user interactions and handle events triggered by the DOM elements in your web page. They provide different ways to associate event handlers with DOM elements and achieve similar results.

**In event handlers, "this" is set to ...**

A) **the element the event fired from**

B) the window object

C) the global object

The correct answer is A) the element the event fired from.

In JavaScript event handlers, the `**this**` keyword refers to the DOM element on which the event occurred. When an event is triggered, such as a button click or a keypress, the event handler function is executed with `**this**` set to the element that fired the event.

For example, if you have a button element with an event handler function assigned to its onclick property, when the button is clicked, the event handler function will be invoked with this referring to the button element itself.

Here's an example to illustrate this:

**const button = document.querySelector('button');**

**button.onclick = function() {**

**console.log(this); // 'this' refers to the button element**

**};**

**// When the button is clicked, the event handler function is called,**

**// and 'this' will refer to the button element.**

In this case, `**this**` inside the event handler function will be bound to the button element that triggered the event.

Therefore, the correct answer is A) the element the event fired from.

**Which of the following is a use case for currying?**

A) Code reuse **B)** **Both are true** C) Retain Scope

Currying is a technique in functional programming where a function with multiple arguments is transformed into a sequence of functions, each taking a single argument. The process of currying allows you to create specialized functions by partially applying a function with a subset of its arguments. This can lead to code reuse and also enables you to retain scope.

Here's a brief explanation of the two use cases mentioned:

1. Code reuse: Currying allows you to create reusable functions by partially applying arguments. You can create specialized versions of a function by fixing some of its arguments, and then reuse those specialized functions in different contexts. This promotes code reuse and modularity, as you can create smaller, more focused functions that can be easily composed together.
2. Retain Scope: Currying can also be useful for retaining the scope of variables and functions. When you curry a function, you can capture variables from the outer scope and use them in the returned specialized functions. This can be particularly helpful in scenarios where you want to create closures or encapsulate certain data or behavior within a function.

In summary, currying provides benefits such as code reuse and the ability to retain scope, making it a valuable technique in functional programming. Therefore, the correct answer is B) Both are true.

**Which of the following is NOT a mutator method?**

**A)** **slice()** B) sort() C) splice()

In JavaScript, mutator methods are array methods that modify the original array. They typically change the array in place rather than creating a new array. The mutator methods include methods like `**sort()**`, `**splice()**`, and others.

However, the `**slice()**` method is not a mutator method. It returns a new array containing a shallow copy of a portion of the original array. It does not modify the original array but instead creates a new array with selected elements.

So, the method that is NOT a mutator method among the options given is A) slice().

**The constructor is a special method for creating and initializing an object created with a class**

**A)** **yes** B) no

A) Yes, the constructor is a special method for creating and initializing an object created with a class in JavaScript.

When you define a class in JavaScript, you can include a constructor method within the class definition. The constructor is a special method that is automatically called when you create a new object using the new keyword with the class name. It allows you to initialize the object's properties and perform any setup tasks.

**Event registration is done by providing a callback function** **...**

**A) yes B) no**

A) Yes, event registration is typically done by providing a callback function.

In JavaScript, when registering an event listener for a specific event on a DOM element, you typically provide a callback function that will be executed when the event occurs. The callback function defines the code that should run in response to the event.

For example, to register a click event handler on a button element, you would typically provide a callback function that specifies the actions to be taken when the button is clicked:

**const button = document.querySelector('button');**

**button.addEventListener('click', function(event) {**

**// Callback function executed when the button is clicked**

**console.log('Button clicked!');**

**});**

**Which of the following is NOT a type of error in JS?**

A) Syntax Error

**B)** **IndexOutOfBound Error**

C) Runtime Error

In JavaScript, the term "IndexOutOfBound Error" is not a standard error type. The correct term for an error related to accessing an array or string with an index that is out of bounds is "RangeError" or "TypeError" depending on the specific situation.

SyntaxError and TypeError are common error types in JavaScript, representing errors in the syntax of the code or errors when performing operations on incompatible types, respectively.

Runtime Error is a more general term that encompasses a wide range of errors that can occur during the execution of a program, such as referencing undefined variables, calling non-existent functions, or encountering unexpected conditions.

**Which of the following is false about closures?**

A) You can encapsulate logic inside a closure

B) The context of an inner function includes the scope of the outer function

**C)** **Closures work only if you use them with IIFE**

IIFE (Immediately Invoked Function Expressions) is false.

Closures in JavaScript are not limited to being used with IIFE. While IIFE is one way to create and immediately invoke a closure, closures can also be created and used in other ways. A closure is formed whenever an inner function references variables from its outer lexical scope, regardless of whether it is within an IIFE or not.

**Which of the following is incorrect? First-class functions...?**

A) Can be assigned as a value to a variable

**B)** **Can't be returned by another function**

C) Can be passed as an argument to another function

In JavaScript, functions are considered first-class citizens, which means they can be treated like any other value or data type. This includes being assigned to variables, passed as arguments to other functions, and returned as values from other functions.

So, the correct answer is B) Can't be returned by another function is incorrect.

**Which of the following is true?**

A) FOM deals with browser components aside the document

**B)** **BOM deals with browser components aside the document**

C) BOM and DOM are the same thing

The Browser Object Model (BOM) represents the components of a web browser aside from the document itself. It provides objects and methods to interact with the browser window, history, location, and other browser-specific functionalities.

The Document Object Model (DOM), on the other hand, represents the structure of an HTML or XML document and provides a way to interact with the elements and content within the document.

So, the correct answer is B) BOM deals with browser components aside the document.

**Array' s length and types are fixed in JS ...**

**A)** **no** B) yes

In JavaScript, the length of an array is not fixed. Arrays are dynamic data structures, meaning their length can change during runtime. You can add or remove elements from an array, which automatically adjusts its length accordingly.

Furthermore, JavaScript arrays can hold values of different types. Unlike some statically typed languages, JavaScript arrays can contain elements of different data types in the same array.

**Once you have defined the property as non-configurable, there is only one behaviour you can change. Which is it?**

A) Enumerable

B) Configurable

**C)** **Writable**

Once a property is defined as non-configurable in JavaScript, you can still change its writable attribute to control whether the value can be modified. The writable attribute determines if the value of a property can be changed by assignment.

So, if you define a property as non-configurable, you can still modify its writability. However, you cannot change its configurability or enumerability.

**Which of the following is true?**

A) Methods are actions that cannot be performed on objects

B) Methods are stored in properties as values

**C)** Methods are stored in properties as function definitions

In JavaScript, methods are functions that are associated with objects. They are typically stored in properties of objects, and the values of those properties are the function definitions. Methods allow objects to perform actions and have behavior associated with them. When a method is invoked on an object, the function code is executed in the context of that object.

**Are variables in JS directly associated with a particular value type?**

**A)** no B) yes

In JavaScript, variables are dynamically typed, which means they are not directly associated with a particular value type. Unlike some statically typed languages, such as Java or C++, where variables have a specific type that cannot be changed, JavaScript variables can hold values of different types at different times. The value assigned to a variable can be of any type, and the type of the variable is determined by the type of the value currently assigned to it. This flexibility allows variables to be used with different types of values throughout the execution of a JavaScript program.

**JavaScript is a static programming language...**

**A)** no B) yes

JavaScript is not a static programming language. It is a dynamically typed language, which means that variable types are determined at runtime based on the value assigned to them. In JavaScript, you can assign values of different types to variables without explicitly declaring their types. The type of a variable can change during the execution of a program, allowing for greater flexibility but also requiring careful handling of type-related issues.

**Which of the following is true?**

A) Only statements are hoisted B) Only expressions are hoisted

**C)** **Only declarations are hoisted**

In JavaScript, only variable and function declarations are hoisted. Hoisting is a behavior where declarations are moved to the top of their respective scope during the compilation phase, allowing them to be accessed before they are actually declared in the code. This means that you can use variables and functions before they are declared in your code, but it's important to note that only the declarations are hoisted, not the initializations or assignments. Expressions and statements are not hoisted in JavaScript.

**Which of the following is false?**

A) Child class inherits data + methods from it's parent

B) The extends keyword is used to create a class which is a child of another class

**C)** **Child class constructors do NOT need to invoke the parent constructor**

This statement is false. In JavaScript, when defining a child class that extends a parent class, the child class constructor must invoke the parent class constructor using the super() method. The super() method is used to call the constructor of the parent class and initialize the inherited properties and behavior.

By invoking the parent constructor in the child class constructor, the child class can inherit and properly initialize the properties and behavior defined in the parent class. If the child class does not invoke the parent constructor using super(), the parent class properties and behavior may not be correctly inherited and initialized in the child class.