JS Institute

JSE-40-01 Exam

JavaScript

Questions & Answers

Question: 1

Examine the following code:

```
let a = 20 + "10";
let b = 20 + +"10";
let c = 20 + -"10" + "10";
let d = "10" - "10" + "100";
let e = "A" - "B" + 0xA;
console.log(`${a}, ${b}, ${c}, ${d}, ${e}`);
```

What will appear in the console as a result?

A. 2010, 2010, 20-1010, 0100, NaN
B. 30, 31, 39, 100, NaN
C. 30, 30, 20, 100, 2
D. 2010, 30, 1010, 0100, NaN

Answer: D

Explanation:

The expressions in the code do not result in the values indicated in this choice.

The expressions in the code do not result in the values indicated in this choice.

The expressions in the code do not result in the values indicated in this choice.

The expression "20" + "10" results in the concatenation of two strings, resulting in "2010". The expression 20 + +"10" converts the second "10" string to a number using the unary plus operator, resulting in 30. The expression 20 + -"10" converts the second "10" string to a negative number using the unary minus operator, resulting in 10, which is then concatenated with the last "10" string to form "1010". The expression "10" - "10" results in the subtraction of two numbers, resulting in 0, which is then concatenated with the last "100" string to form "0100". The expression "A" - "B" results in NaN (Not a Number) since the subtraction operator cannot be applied to non-numeric values, and adding 0xA (hexadecimal value of 10) results in NaN.

Question: 2

In the body of the function (inside it) we declare a variable, e.g

`let x = 10`

Is it local or global?

A. The variable x is a local variable.

B. The variable x is a global variable.
C. The variable x can be either local or global,depending on where the function is declared,and depending on how it is declared.
D. We cannot declare variables within a function.

Answer: A

Explanation:

The variable x is a local variable because it is declared inside the function body. Local variables are only accessible within the scope of the function where they are declared and are not visible outside of that function.

The variable x is not a global variable because it is declared inside the function body. Global variables are declared outside of any function and can be accessed from anywhere in the codebase.

The variable x is local in this scenario because it is declared inside the function. The scope of a variable in JavaScript is determined by where it is declared, and in this case, it is declared inside the function, making it a local variable.

This statement is incorrect. Variables can be declared within a function in JavaScript. These variables are considered local variables and have scope limited to the function in which they are declared.

Question: 3

Analyze the following code:

```
try {
    console.log("start");
} catch (error) {
    console.log("error");
} finally {
    console.log("end");
}
```

What will happen as a result of its execution?

A. The word "error" will appear in the console.
B. The words "start", "error", "end" will appear in the consoleon successive lines.
C. The following words will appear in the console: "error", "end"
D. The following words will appear in the console: "start", "end"

Answer: D

Explanation:

Since there are no errors thrown in the try block, the catch block will not be executed. As a result, the word "error" will not appear in the console.

The catch block will not be executed in this code snippet because there are no errors thrown in the try block. Therefore, the console will log "start" from the try block, followed by the execution of the finally block, which will log "end". The word "error" will not appear in the console.

The catch block will not be executed in this code snippet because there are no errors thrown in the try block. Therefore, the console will log "start" from the try block, followed by the execution of the finally block, which will log "end".

In this code snippet, the try block will execute successfully without any errors. Therefore, the console will log "start" from the try block, followed by the execution of the finally block, which will log "end".

Question: 4

Look at the following declaration

```
let x = 2e4;
```

Which declarations can successfully replace it?

(Select **two** correct answers)

A. let x = 20000;
B. let x = 0.0002;
C. let x = 24000;
D. let x = 200e2;

Answer: A, D

Explanation:

This choice is correct because 2e4 is equivalent to 2 * 10^4, which equals 20000. Therefore, let x = 20000 is a valid replacement for the given declaration.

This choice is incorrect because 0.0002 is not the correct equivalent of 2e4. The exponential notation 2e4 represents 2 * 10^4, which is 20000, not 0.0002.

This choice is incorrect because 24000 is not the correct equivalent of 2e4. The exponential notation 2e4 specifically represents 2 * 10^4, not 24000.

This choice is correct because 200e2 is equivalent to 200 * 10^2, which also equals 20000. Therefore, let x = 200e2 is a valid replacement for the given declaration.

Question: 5

We define a function using the following function expression:

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

What could the definition of the corresponding arrow function look like?

A. let sum = function (a, b)=>{ return (a + b);};
B. let sum = (a, b)-- > a + b;
C. let sum = (a, b) => { a + b };
D. let sum = (a, b) => a + b;

Answer: D

Explanation:

Choice D is incorrect as it does not follow the correct arrow function syntax. Arrow functions do not use the function keyword or curly braces for single-line expressions. The arrow function in choice D is written in a way that resembles a traditional function expression, which is not the correct format for an arrow function.

Choice C is incorrect as it contains a syntax error. The arrow function syntax in JavaScript uses the => symbol to separate the parameters from the function body. The use of -- > is not valid in JavaScript arrow functions, resulting in a syntax error.

The arrow function syntax in choice B is incorrect. When using curly braces in an arrow function, you need to explicitly use the return keyword to return a value. In this case, the return statement is missing, and the function body is not correctly defined.

The correct arrow function syntax for the given function expression is shown in choice A. Arrow functions are a concise way to write function expressions in JavaScript, and in this case, the arrow function directly returns the sum of the two input parameters without the need for explicit return keyword or curly braces.

Question: 6

Analyze the following below:

```
let steps = [3, 2, 1];
for (let n of steps) console.log(n);
```

What will appear in the console as a result?

A. 1 2 3
B. 3 2 1
C. "[3, 2, 1]"
D. 0 1 2

Answer: B

Explanation:

Since the `for...of` loop iterates over the elements in the array in the order they appear, the output will not be "1 2 3" but rather "3 2 1".

The code snippet iterates over the elements in the `steps` array using a `for...of` loop, which sequentially logs each element to the console. Therefore, the output will be "3 2 1" as each element in the array is logged in the order they appear.

The code does not output the array itself, but rather the individual elements of the array. Therefore, the output will not be "[3, 2, 1]".

The elements in the `steps` array are not indices but actual values, so the output will not be "0 1 2". The loop logs the elements themselves, which are 3, 2, and 1, in that order.

Question: 7

Where can we find in the debugger the information

about the currently called functions in our program?

A. We do not have access to such information.
B. In the call stack window.
C. In the watch window.
D. In the console.

Answer: B

Explanation:

Contrary to this choice, we do have access to information about the currently called functions in our program through the call stack window of the debugger. This feature is essential for debugging and understanding the flow of execution in our code.

In the call stack window of the debugger, we can see the hierarchy of functions that have been called in our program. This allows us to track the flow of execution and identify where an error occurred by seeing the sequence of function calls leading up to it.

The watch window in the debugger is used for monitoring the values of specific variables or expressions during program execution. While it can be helpful for tracking the values of variables, it does not display information about the currently called functions in our program. This information is typically shown in the call stack window.

The console in the debugger is primarily used for displaying output messages and logging information. It does not provide information about the currently called functions in our program. The call stack window is the specific area where this information can be found.

Question: 8

Some errors in JavaScript code means that the interpreter

cannot even start executing the program.

What type of errors are these?

A. TypeError
B. RangeError
C. ReferenceError
D. SyntaxError

Answer: D

Explanation:

TypeError occurs when a value is not of the expected type, causing an error during the execution of the program. While TypeError can occur during program execution, it does not prevent the interpreter from starting the program.

RangeError occurs when a number is not within a specific range of allowed values. This error can happen during program execution but does not necessarily prevent the interpreter from starting the program.

ReferenceError occurs when trying to access a variable that is not declared or is out of scope. While ReferenceError can occur during program execution, it does not necessarily prevent the interpreter from starting the program.

SyntaxError occurs when the JavaScript code has syntax errors that prevent the interpreter from even starting the execution of the program. These errors are related to the structure of the code and how it is written.

Question: 9

Analyze the code snippet:

```
let winter = ["December", "January", "February"];
let index = winter.indexOf("February");
```

The index variable will have the value:

A. 1
B. 2
C. 3
D. "February"

Answer: B

Explanation:

The indexOf() method returns the index of the first occurrence of the specified element in the array. In this case, "February" is the third element in the array "winter", so the index variable will not have the value of 1.

The indexOf() method in JavaScript returns the index of the first occurrence of a specified element in an array. Since "February" is the third element in the array "winter", the index variable will have the value of 2.

The indexOf() method returns the index of the first occurrence of the specified element in the array. Since "February" is the third element in the array "winter", the index variable will not have the value of 3.

The indexOf() method returns the index of the first occurrence of the specified element in the array. In this case, the index variable will not have the value of "February" itself, but rather the index of "February" in the array "winter", which is 2.

Question: 10

What is the basic difference between compiled and interpreted languages?

A. In compiled languages, unlike in interpreted languages,all the program code must be transferred into the target formbefore it can be run.
B. In compiled languages, the program is always transformedinto the executable code of a specific machine (processor).
C. In interpreted languages, it is not possible to use variables.
D. In interpreted languages, the step of checking the correctnessof the formal writing of the source code (i.e. syntax) is always omitted.

Answer: A

Explanation:

In compiled languages, the entire program code is translated into machine code or an intermediate representation before execution, which allows for faster execution as the translation only needs to occur once. This differs from interpreted languages where the code is translated and executed line by line during runtime.

In compiled languages, the program is translated into machine code that is specific to the target machine's processor architecture. This allows the compiled program to run directly on the target machine without the need for an interpreter. Interpreted languages, on the other hand, do not produce machine-specific code.

This statement is incorrect as variables are essential components of both compiled and interpreted languages. Variables are used to store and manipulate data within a program, regardless of the language being compiled or interpreted.

The process of checking the correctness of the source code's syntax is essential in both compiled and interpreted languages. Syntax errors can prevent a program from running correctly, so it is crucial to ensure that the code is written correctly in both types of languages.

Question: 11

Which of the following loop instructions is intended **only** to loop

through all elements of the indicated **array**?

A. for ... of
B. do ... while
C. for ... in
D. for

Answer: A

Explanation:

The 'for ... of' loop instruction in JavaScript is specifically designed to iterate over the values of an iterable object, such as an array. It simplifies the process of looping through all elements of an array by directly accessing the values without the need for index manipulation.

The 'do ... while' loop in JavaScript is a post-test loop that executes a block of code at least once before checking the loop condition. It is not specifically designed for looping through all elements of an array and may not be the most efficient choice for this purpose.

The 'for ... in' loop in JavaScript is used to iterate over the properties of an object, rather than the elements of an array. It is not recommended for looping through all elements of an array as it may not guarantee the order of iteration and can include inherited properties.

The 'for' loop in JavaScript is a general-purpose loop that can be used to iterate over a sequence of elements, including arrays. However, it requires manual index manipulation and is not specifically tailored for looping through all elements of an array.

Question: 12

What keyword does the function declaration start with?

A. function
B. let
C. procedure
D. fun

Answer: A

Explanation:

The keyword "function" is used to declare a function in JavaScript. It is the standard keyword for defining a function and is essential for creating reusable blocks of code that can be called and executed later in the program.

The keyword "let" is used for declaring variables in JavaScript, not for defining functions. While "let" is important for variable declaration and scoping, it is not the correct keyword for starting a function declaration.

The keyword "procedure" is not a valid keyword in JavaScript for declaring functions. JavaScript does not have a built-in "procedure" keyword for defining functions; instead, the standard keyword is "function" for function declarations.

The keyword "fun" is not a valid keyword in JavaScript for declaring functions. In JavaScript, functions are declared using the "function" keyword, so using "fun" instead would result in a syntax error.

Question: 13

Point out the correct declaration of the temperature variable:

A. temperature is variable;
B. declare temperature;
C. let temperature;
D. variable temperature;

Answer: C

Explanation:

The syntax "temperature is variable;" is not a valid way to declare a variable in JavaScript. In JavaScript, variables are declared using keywords like `let`, `const`, or `var`, followed by the variable name.

The syntax "declare temperature;" is not valid in JavaScript for declaring a variable. The correct way to declare a variable is by using keywords like `let`, `const`, or `var`, followed by the variable name.

The correct way to declare a variable in JavaScript is by using the `let` keyword followed by the variable name. This syntax is the standard way to declare a variable in JavaScript and is the appropriate way to declare the `temperature` variable.

The syntax "variable temperature;" is not valid in JavaScript for declaring a variable. In JavaScript, variables are declared using keywords like `let`, `const`, or `var`, followed by the variable name.

Question: 14

Analyze the following code:

```
const x = 10;
onsole.log(x);
x += 10;
```

What exception will be thrown as a result of its execution attempt?

A. SyntaxError
B. ReferenceError and TypeError
C. TypeError
D. ReferenceError

Answer: D

Explanation:

The code will not throw a SyntaxError. SyntaxError occurs when there is a mistake in the syntax of the code, such as a missing parenthesis or a misspelled keyword. In this case, the code is syntactically correct, but it will result in a ReferenceError due to the attempt to modify a constant variable.

The code will only throw a ReferenceError, not a TypeError. While the code does involve an attempt to modify a constant variable, which could potentially lead to a TypeError in other scenarios, in this specific case, the error thrown will be a ReferenceError due to the reassignment of a constant variable.

The code will not throw a TypeError. TypeError typically occurs when a value is not of the expected type, such as trying to perform an operation on a non-function or non-object. In this case, the issue is related to reassigning a value to a constant variable, not a type mismatch.

The code will throw a ReferenceError because the variable `x` is declared using `const`, which means its value cannot be reassigned. When the code tries to increment `x` by 10, it will result in a ReferenceError as it is attempting to modify a constant variable.

Question: 15

You have declared an array of flower names and displayed them one by one on the console using a `for` statement:

```
let flowers = ["rose", "lily", "tulip"];
let len = flowers.length;
for (let i=0; i<len; i++)
    console.log(flowers[i]);
```

What should the code look like that will display the same information, but this time written using the `for ... of` statement?

A. for f of flowers console.log(f);
B. for (let f of flowers) console.log(flowers);
C. for (let i of flowers) console.log(flowers[i]);
D. for (let f of flowers) console.log(f);

Answer: D

Explanation:

This syntax is incorrect for the for ... of statement. The correct format should include parentheses and curly braces, as shown in the correct choice A.

While the for ... of statement is used correctly, logging the entire 'flowers' array instead of individual elements will display the array itself multiple times, not each element separately.

Although the for ... of statement is used, the variable 'i' is not necessary and should be 'f' to represent each element of the 'flowers' array. Additionally, accessing elements using the index 'i' is not needed with the for ... of statement.

The correct syntax for using the for ... of statement is to iterate over the elements of the array directly. In this case, the variable 'f' is used to represent each element of the 'flowers' array, and it is logged to the console.

Question: 16

Analyze the following code:

```
let a = 100;
function test() { let a = 20; console.log(a); }
test();
```

What will appear on the console as a result of its execution?

A. 100
B. 20
C. undefined
D. a

Answer: B

Explanation:
The code defines a variable `a` with a value of 100 outside the `test` function, but the `test` function defines its own variable `a` with a value of 20. When the `test` function is called, it will log the value of the inner `a` variable, which is 20, not the outer `a` variable.

The code defines a variable `a` with a value of 100 outside the `test` function and another variable `a` with a value of 20 inside the `test` function. When the `test` function is called, it will log the value of the inner `a` variable, which is 20, to the console.

Since the `test` function does not return any value explicitly, the function call `test()` will not return anything, resulting in `undefined` being displayed on the console. The value of 20 assigned to the inner `a` variable is only logged to the console, not returned by the function.

The code does not log the variable `a` itself to the console, but rather the value of the inner `a` variable defined within the `test` function. Therefore, the output will be the value of the inner `a` variable, which is 20, not the variable name `a`.

Question: 17

You perform the following operation:

```
let x = "flowers".charAt(2);
```

As a result, what value will be written into the variable x?

A. ["fl", "owers"]
B. 2
C. "owers"
D. "o"

Answer: D

Explanation:
The charAt() method in JavaScript does not return an array of substrings. It returns a single character from a specified index in a string. In this case, the character at index 2 in the string "flowers" is "o", so the value written into the variable x will be "o".

The charAt() method in JavaScript does not return the index of the character in the string, but rather the character itself. Therefore, the value written into the variable x will be the character "o", not the index 2.

The charAt() method in JavaScript returns a single character from a specified index in a string. In this case, the character at index 2 in the string "flowers" is "o", not the substring "owers". Therefore, the value written into the variable x will be "o".

The charAt() method in JavaScript returns the character at a specified index in a string. In this case, the index 2 in the string "flowers" corresponds to the character "o", so the value written into the variable x will be "o".

Question: 18

Select a set of data types, containing only primitive types:

A. Array, Object
B. Array, Boolean, Number, BigInt, String
C. Boolean, Number, BigInt, String
D. Array, Boolean, Number, String

Answer: C

Explanation:

Array and Object are not primitive data types in JavaScript. Arrays are considered objects in JavaScript, and objects are reference types that can hold key-value pairs.

Array is not a primitive data type in JavaScript. Arrays are considered objects in JavaScript, and objects are reference types that can hold key-value pairs. Boolean, Number, BigInt, and String are primitive data types.

Boolean, Number, BigInt, and String are all primitive data types in JavaScript. Primitive data types are immutable and directly hold a specific value, making them different from objects which are mutable and can hold multiple values.

Array is not a primitive data type in JavaScript. Arrays are considered objects in JavaScript, and objects are reference types that can hold key-value pairs. Boolean, Number, and String are primitive data types.

Question: 19

The conditional operator is:

A. a ternary operator.
B. a binary operator.
C. a unary operator.
D. a quaternary operator.

Answer: A

Explanation:
The conditional operator, also known as the ternary operator, is a unique operator in JavaScript that takes three operands. It is often used as a shorthand for an if-else statement, making it a ternary operator due to its three operands.

A binary operator is an operator that operates on two operands. While many operators in JavaScript fall under this category, the conditional operator stands out as a ternary operator due to its requirement of three operands for proper functionality.

A unary operator is an operator that operates on only one operand. The conditional operator, despite its unique nature, does not fall under the category of unary operators as it requires three operands to function.

There is no such thing as a quaternary operator in JavaScript. The conditional operator, also known as the ternary operator, is specifically designed to take three operands, making it distinct from unary, binary, or quaternary operators.

Question: 20

In the `daysOfWeek` variable, we place an array with names of the days of the week.

To reverse the order of the array elements, we should call:

A. daysOfWeek.reverse();
B. daysOfWeek.invert();
C. daysOfWeek = reverse(daysOfWeek);
D. daysOfWeek.order(-1);

Answer: A

Explanation:

Calling the `reverse()` method on an array in JavaScript will reverse the order of its elements. This is the correct method to use when you want to reverse the order of the elements in an array like `daysOfWeek`.

There is no `invert()` method in JavaScript for arrays. This choice is incorrect as it does not provide a valid method to reverse the order of the elements in the `daysOfWeek` array.

Assigning the result of calling a non-existent `reverse()` function on the `daysOfWeek` array will not reverse the order of the elements. This choice is incorrect as it does not provide a valid method to reverse the order of the elements in the `daysOfWeek` array.

There is no `order()` method in JavaScript arrays that takes a parameter to reverse the order of elements. This choice is incorrect as it does not provide a valid method to reverse the order of the elements in the `daysOfWeek` array.