Assignments For JavaScript Fundamentals

1) What is the difference between unary, binary, and ternary operators in JavaScript? Provide examples of each.

Ans:-
In JavaScript, unary, binary, and ternary operators are used to perform different types of operations on operands (values or variables). The distinction lies in the number of operands they work with:

Unary Operators:
Unary operators work with a single operand. They can be used to perform various operations like negation, type conversion, increment, and decrement.
Example:

```
let x = 5;
let y = -x;       // Unary negation
let a = "10";
let b = +a;       // Unary plus (type conversion to number)
let count = 0;
count++;          // Unary increment
```

Binary Operators:
Binary operators work with two operands, performing operations like addition, subtraction, multiplication, division, comparisons, logical operations, and more.
Example:

```
let num1 = 10;
let num2 = 5;
let sum = num1 + num2;       // Addition
let difference = num1 - num2; // Subtraction
let product = num1 * num2;    // Multiplication
let quotient = num1 / num2;   // Division

let isGreater = num1 > num2;  // Greater than comparison
let isEqual = num1 === num2;  // Strict equality comparison
let logicalAnd = true && false; // Logical AND
```

Ternary Operator:
The ternary operator is a shorthand for an if-else statement. It takes three operands and returns a value based on a condition. It's the only JavaScript operator that takes three operands.
Syntax: condition ? expressionIfTrue : expressionIfFalse

Example:

```
let age = 18;
let isAdult = age >= 18 ? "Adult" : "Minor";
console.log(isAdult); // Outputs: "Adult" (if age is 18 or greater)
```

In this example, if the age is greater than or equal to 18, the value of isAdult becomes "Adult"; otherwise, it becomes "Minor".

These operators are fundamental to programming in JavaScript (and many other programming languages), as they allow you to manipulate values and perform various types of operations in your code.

2) Explain the concept of short-circuit evaluation in the context of logical operators. Provide an example.

Ans:-

Short-circuit evaluation is a concept in programming, specifically with logical operators, where the second operand of a logical expression is only evaluated if the outcome of the expression can be determined by the value of the first operand. In other words, if the first operand of a logical operation already determines the result, the second operand is not evaluated, saving processing time and potentially preventing errors.

In JavaScript, the two main logical operators that exhibit short-circuit behavior are the logical AND (&&) and logical OR (||) operators.

1.) Logical AND (&&) Short-Circuit Evaluation:

In a logical AND operation, if the first operand is false, the overall result will be false, regardless of the value of the second operand. Therefore, if the first operand is false, the second operand is not evaluated.

Example:

```
let result = false && someFunction();  // someFunction() is not called
console.log(result);  // Outputs: false
```

In this example, `someFunction()` is not called because the first operand is `false`, and the overall result will be `false` regardless of the value of `someFunction()`.

2.) Logical OR (||) Short-Circuit Evaluation:

In a logical OR operation, if the first operand is true, the overall result will be true, regardless of the value of the second operand. Therefore, if the first operand is true, the second operand is not evaluated.

Example:

```
let result = true || someFunction();  // someFunction() is not called
console.log(result);  // Outputs: true
```

Here, `someFunction()` is not called because the first operand is `true`, and the overall result will be `true` regardless of the value of `someFunction()`.

Short-circuit evaluation is not only a performance optimization but also a mechanism to avoid unnecessary computations or side effects that might occur in the evaluation of the second operand. It's a behavior that can be used strategically to handle conditional expressions efficiently.

3) Given the following code snippet, what will be the output?

```
let x = 5;
let y = "10";
let z = x + y;
console.log(z);
```

Ans:-

510

4) Write a JavaScript function named calculateArea that takes the radius of a circle as a parameter and returns the area of the circle. Use the appropriate arithmetic operator.

Ans:-

```
function calculateArea(radius) {
   // Calculate the area using the formula: π * radius^2
   const area = Math.PI * Math.pow(radius, 2);
   return area;
}

// Example usage
let radius = 5;
let circleArea = calculateArea(radius);
console.log(`The area of the circle with radius ${radius} is ${circleArea}`);
```

5) Explain the difference between the equality operator (==) and strict equality operator (===) in JavaScript. Provide examples to demonstrate their behavior

Ans:-

In JavaScript, the equality operator (`==`) and the strict equality operator (`===`) are used for comparison, but they behave differently in terms of type coercion and value comparison.

Equality Operator (`==`):
The equality operator compares two values for equality, and it performs type coercion if the operands are of different types. This means it attempts to convert the operands to a common type before making the comparison.
Example:

```
console.log(5 == "5"); // Outputs: true
```

In this example, even though the left operand is a number and the right operand is a string, the equality operator performs type coercion and converts the string to a number before making the comparison. As a result, the comparison returns true.

Strict Equality Operator (`===`):
The strict equality operator, also known as the identity operator, compares two values for equality without performing any type coercion. It checks both the value and the type of the operands, ensuring that they are exactly the same.
Example:

```
console.log(5 === "5"); // Outputs: false
```

Here, the strict equality operator does not perform type coercion. Since the operands have different types (number and string), the comparison returns `false`.

Examples with Different Types:

```
console.log(5 == "5");  // Outputs: true
console.log(5 === "5"); // Outputs: false

console.log(true == 1); // Outputs: true
console.log(true === 1);// Outputs: false

console.log(null == undefined); // Outputs: true
console.log(null === undefined); // Outputs: false
```

In summary, the `==` operator performs type coercion and then compares the values, while the `===` operator compares both values and types without coercion. It's generally recommended to use the strict equality operator (`===`) to avoid unexpected results due to type coercion.

6) Write a program to convert Fahrenheit to Celsius.

Ans:-

```
function fahrenheitToCelsius(fahrenheit) {
    // Conversion formula: (F - 32) * 5/9
    const celsius = (fahrenheit - 32) * (5/9);
    return celsius;
}

// Example usage
let fahrenheit = 32; // Change this value to the desired Fahrenheit temperature
let celsius = fahrenheitToCelsius(fahrenheit);
console.log(`${fahrenheit} Fahrenheit is equal to ${celsius.toFixed(2)} Celsius`);
```