FSD - Assignment – 2

Q1. [1 Point] Explain the difference between null and undefined in JavaScript.

ANS:-

Null: An assignment value that represents no value or no object. It is explicitly set by the programmer to indicate that a variable intentionally has no value.

Undefined: A variable that has been declared but not assigned a value. It is automatically assigned by JavaScript.

Q2. [1 Point] What will be the output of the following code snippet, and why?

```
console.log('10' + 5);
console.log('10' - 5);
console.log(true + 2);
console.log(false + undefined);
```

1. console.log('10' + 5):

Here, the (+) operator is used with a string (10) and a number (5). In JavaScript, if one of the operands is a string, the (+) operator concatenates the operands.

Output: '105'

2. console.log('10' - 5):

The (-) operator only works with numbers, so JavaScript converts the string '10' to a number.

Output: 5

3. console.log(true + 2):

In JavaScript, the Boolean true is converted to the number 1 when used in arithmetic operations.

Output: 3

4. console.log(false + undefined):

The Boolean false is converted to the number 0. However, undefined cannot be converted to a number and remains NaN (Not a Number).

Output: NaN

Q3. **[1 Point]** What is the difference between == and === in JavaScript? Provide examples

ANS:-

Equality Operator (==): Compares values after converting them to the same type. Example: 5 == '5' returns true because '5' is converted to the number 5.

Strict Equality Operator (===): Compares both value and type without any conversion. Example: 5 === '5' returns false because the types (number and string) are different.

Q4. **[1 Point]** Predict the output of the following expressions and explain your reasoning:

```
console.log(0 == false);
console.log(0 === false);
console.log('' == 0);
console.log('' === 0);
```

1. output: true
   The (==) operator converts both operands to the same type before comparing them. In this case false is converted to 0, so the comparison becomes 0==0 which is true.
2. output: false
   The (===) operator checks both value and type without converting the operands. Here 0 is a number and false is a boolean, so they are of different types, and the comparison returns false.
3. output: true
   The (==) operator converts both operands to the same type before comparing them. In this case, the empty string is converted to 0, so the comparison becomes 0==0 which is true.
4. output: false
   The (===) operator checks both value and type without converting the operands. Here, empty string is a string and 0 is a number, so they are of different types and the comparison returns false.

Q5. **[1 Point]** Given the following code, what will be the output and why?

```
console.log(0 || 1 && 2 || 3);
console.log(false || (true && false) || true);
console.log(0 && 1 || 2 && 3);
```

1. Output: 2
    - The && operator has higher precedence than || so it is evaluated first.
    - 1 && 2 evaluates to 2. The expression becomes 0||2||3.
    - 0||2 evaluates to 2 because || returns the first truthy operand it encounters.
    - Finally, 2||3 remains 2 because || returns the first truthy operand it encounters.
2. Output: true
    - Parentheses alter the default order, so true && false is evaluated first, which results in false
    - The expression simplifies to false || false || true.
    - false || false evaluates to false.
    - false || true evaluates to true.
3. Output: 3
    - The && operator has higher precedence than ||, so it is evaluated first.
    - 0 && 1 evaluates to 0 because && returns the first falsy operand.
    - 2 && 3 evaluates to 3 because both operands are truthy, so && returns the last operand.
    - The expression becomes 0 || 3.
    - 0 || 3 evaluates to 3 because || returns the first truthy operand.

Q6. **[1 Point]** Predict the output of the following expressions and explain your reasoning:

```
let a = 10, b = 20, c = 30;
console.log(a + b * c);
console.log((a + b) * c);
console.log(a + b > c ? a : b);
console.log((a > b) && (b > c) || (a > c));
```

1. Output: 610
    - The * operator has higher precedence than the + operator.
    - First, evaluate b * c: 20 * 30 = 600
    - Then, add a: 10 + 600 = 610.
2. Output: 900
    - Parentheses change the order of evaluation.
    - First, evaluate a + b: 10 + 20 = 30.
    - Then, multiply by c: 30 * 30 = 900.
3. Output: 20

- The ternary operator checks the condition a + b > c.
- Evaluate a + b: 10 + 20 = 30.
- Compare with c: 30 > 30 is false (because 30 is not greater than 30).
- Therefore, the expression evaluates to the second operand of the ternary operator, which is b, so the output is 20.

4. Output: false
    - The && operator has higher precedence than ||.
    - Evaluate (a > b) && (b > c):
    - a > b: 10 > 20 is false.
    - b > c: 20 > 30 is false.
    - false && false evaluates to false.
    - Evaluate (a > c): 10 > 30 is false.
    - false || false evaluates to false.

Q7. **[2 Points]** Analyze and explain the output of the following code snippets:

```
console.log([] + {});
console.log({} + []);
console.log([] == ![]);
console.log('' == []);
```

1. Output: [object object]
   When using the + operator with an array and an object, JavaScript converts both operands to strings.
    - [] (an empty array) is converted to an empty string.
    - {} (an empty object) is converted to "[object Object]".
    - The expression "" + "[object Object]" results in "[object Object]".
2. Output: [object object]
    - {} (an empty object) is converted to "[object Object]".
    - [] (an empty array) is converted to an empty string.
    - The expression "[object Object]"+ "" results in "[object Object]".

3. Output: true
    - ![] involves applying the logical NOT (!) operator to an empty array
    - In JavaScript, an empty array ([]) is considered truthy, so ![] negates this truthy value to false.
    - In JavaScript, [] is converted to an empty string in this comparison.
    - [] is converted to false in the context of equality checks.
    - ![] is false, and the comparison [] == false evaluates to true due to JavaScript's specific type conversion rules.
4. Output: true
    - When comparing an empty string ('') with an empty array ([]), JavaScript performs type conversion.

- The empty array [] is converted to a string. In JavaScript, an empty array is converted to an empty string '' when used in a string context.
- The comparison '' == [] actually becomes '' == ''.
- The result is true.

Q8. **[2 Points]** What will be the output of the following code, and why?

```
console.log(+"");
console.log(+true);
console.log(+false);
console.log(+null);
console.log(+undefined);
```

1. Output: 0
   The + operator is used to convert a value to a number. An empty string "" is converted to 0 when using the unary + operator.
2. Output: 1
   The boolean true is converted to the number 1 by the unary + operator.
3. Output: 0
   The boolean false is converted to the number 0 by the unary + operator.
4. Output:0
   The value null is converted to the number 0 by the unary + operator.
5. Output: NaN
   The value undefined cannot be converted to a number, so the result is NaN (Not a Number).