FSD - Assignment – 2

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

- **Undefined :** It is the default value assigned to a variable that has been declared but not initialized. It represents the absence of a value indicating that a value is missing or not been assigned yet.

  Example : let x;

  console.log(x); // undefined

- **null :** It is assigned explicitly to a variable to indicate that it is empty or has no value.

  Example : let y = null;

  console.log(y); //null

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)**

- 10 is a string, so the + operator performs string concatenation

  Result : 105

2. **console.log('10' - 5);**

- When using the - operator, javascript attempts to convert the string to a number.

  Result : 5

3. **console.log(true + 2);**

- True is coerced to 1

Result : 1+2=3

4. **console.log(false + undefined);**

- False is 0, but undefined cannot be converted to a number
- Any arithmetic operation with undefined results in NaN

Result : NaN

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

- **"=="** performs type coercion, meaning it converts the values to the **same type** before making the comparison.

  Example :

  console.log(5 == '5'); // true, because '5' is converted to number 5 before comparison.

- **"==="** does not perform type coercion. It checks for equality without converting values. It requires both the value and type to be the same for a comparison to return true.

  Example:

  console.log(5 === '5'); // false, because the types are different (number vs string).

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. **console.log(0 == false);**

- This uses loose equality(==), which **performs type coercion**. 0 is coerced to false, so 0==false evaluates to true.

Result : true

2. **console.log(0 === false);**

- This uses strict equality(===),which **does not perform type conversion**. 0 is a number and false is a boolean; they are not the same type. So it results in false.

  Result : false

3. **console.log('' == 0);**

- Using loose equality, an empty string is coerced to 0

  Result : true

4. **console.log('' === 0);**

- Strictly equality compares both value and type without coercion. An empty string is not the same type as the number 0. So it evaluates to false.

  Result : 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. **console.log(0 || 1 && 2 || 3);**

- && has higher precedence than ||, so 1 && 2 is evaluated first, resulting in 2
- Then it becomes 0 || 2 || 3
- || returns the first truth value, which is 2
- So the final result is 3 (as 2 || 3 evaluates to 2)

  Result : 2

2. **console.log(false || (true && false) || true);**

- (true && false) evaluates to false

- Then it becomes false || false || true
- The last true is the first truth value, so it returns true

  Result : true

3. **console.log(0 && 1 || 2 && 3);**

- 0 && 1 evaluates to 0 (as && returns the first falsy value)
- 2 && 3 evaluates to 3
- It becomes 0 || 3
- || returns the first truth value, which is 3

  Result : 3

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. **console.log(a + b + c);**

- Simple addition 10+20+30=60

  Result : 60

2. **console.log((a + b) * c);**

- Parentheses first: (10 + 20) = 30
- Then multiplication: 30 * 30 = 900

  Result : 900

3. **console.log(a+b>c?a:b)**

- a + b = 30, which is equal to c (30)
- So the condition is false, and it returns b, which is 20

  Result : 20

4. **console.log((a > b) && (b > c) || (a > c));**

- (a > b) is false (10 is not greater than 20)
- (b > c) is false (20 is not greater than 30)
- (a > c) is false (10 is not greater than 30)
- false && false || false evaluates to false

  Result : false

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

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

1. **console.log([ ] + {}) // output: [object Object]**
   - [] is converted to "".
   - {} is converted to "[object Object]".
   - "" + "[object Object]" gives "[object Object]".

   Result : [object Object]

2. **console.log({} + []);**
   - {} + [] is interpreted as An empty block {} (which is ignored)
   - The addition operation + [] converts [] to an empty string ""
   - {} as an object is converted to "[object Object]"

   Result: [object Object]

3. **console.log([]== ![]) // true**
   - ![] evaluates to false.
   - false is converted to 0.

- [] is converted to 0 when compared with a number.
- Therefore, 0 == 0 evaluates to true, so console.log([] == ![]) outputs true

Result : true

4. **console.log('' == []) //true**
   - Both sides are coerced to 0 for comparison
   - 0 == 0 is true

   Result : 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. **console.log(+"");**
- The unary + converts an empty string to 0

  Result : 0
2. **console.log(+true);**
- true is converted to 1

  Result : 1
3. **console.log(+false);**
- false is converted to 0

  Result : 0
4. **console.log(+null);**
- null is converted to 0

  Result : 0

5. **console.log(+undefined);**

- undefined cannot be converted to a number, resulting in NaN

  Result : NaN