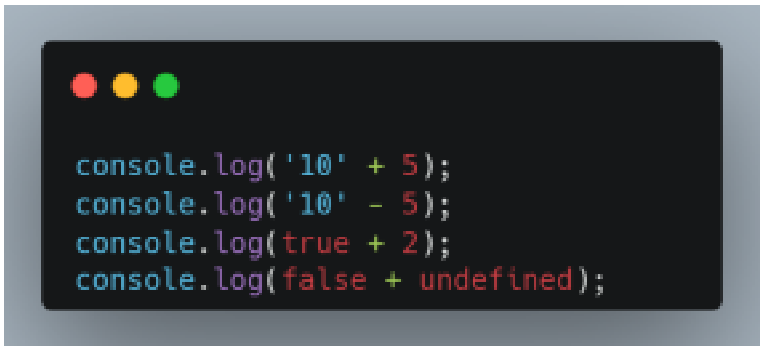
FSD - Assignment – 2

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

- undefined means a variable has been declared but not assigned a value, while null is an explicit value assigned to indicate "no value."

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);
```

- 105 - When we concatenate a string with a number, it converts the number to a string and two strings are concatenated. So the output is 105.
- 5 - when we use the '-' operator with a string and number, it converts the string to a number and performs subtraction. So, the output is 5.
- 3 - when you use the `+` operator with a boolean and a number, the boolean is converted to a number (`true` becomes `1` and `false` becomes `0`), and then the addition is performed.
- NaN - when you use the `+` operator with `false` and `undefined`, JavaScript tries to convert both values to numbers: `false` is converted to `0`. `undefined` is converted to `NaN` (Not-a-Number). Adding any number to `NaN` results in `NaN`. So, `false + undefined` results in `NaN`.

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

`==` checks for equality with type conversion

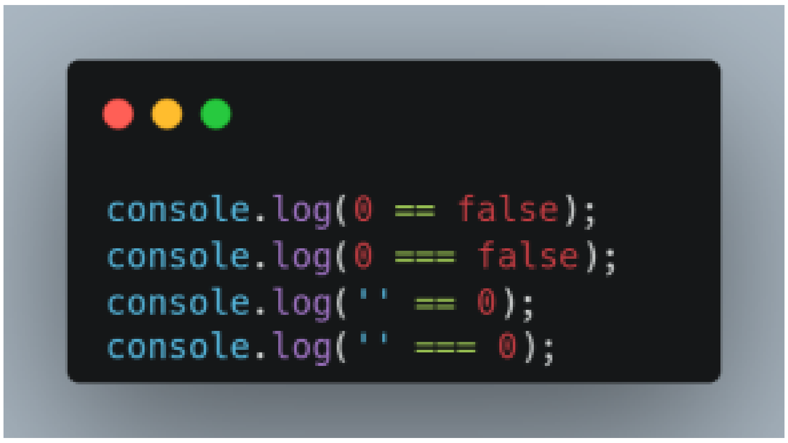
=== checks for equality without type conversion,

Examples:

console.log(5 == '5'); // Output: true (converts '5' to number 5)

console.log(5 === '5'); // Output: false (number 5 is not the same as string '5')

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

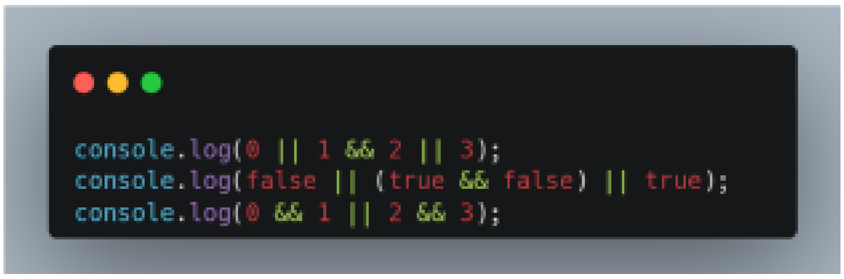


```
console.log(0 == false);
console.log(0 === false);
console.log('' == 0);
console.log('' === 0);
```

- true - '==' performs the type conversion. It converts the false to 0 so comparison becomes 0==0. So, this returns true.
- false- '===' compares the type as 0 is a number and false is a boolean value. As both are different types it gives false.
- true- '==' performs the type conversion. It converts the ' ' to 0 so comparison becomes 0==0. So, this returns true.
- false - '===' does not perform the type conversion as it is not the same type

returns false.

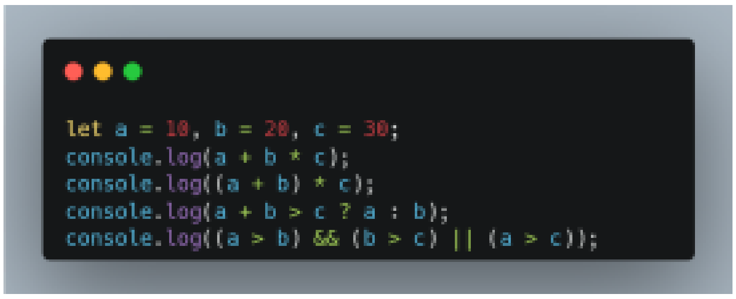
Q5. **[1 Point]** Given the following code, what will be the output and why?



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

- 2 - In `0 || 1 && 2 || 3`, the `&&` operator has higher precedence, so `1 && 2` evaluates to `2`. Then, `0 || 2` results in `2`, and `2 || 3` is also `2`.
- true - In `false || (true && false) || true`, `(true && false)` results in `false`. Then, `false || false || true` evaluates to `true`.
- 3 - In `0 && 1 || 2 && 3`, the `&&` operators are evaluated first: `0 && 1` results in `0`, and `2 && 3` results in `3`. Then, `0 || 3` results in `3`.

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

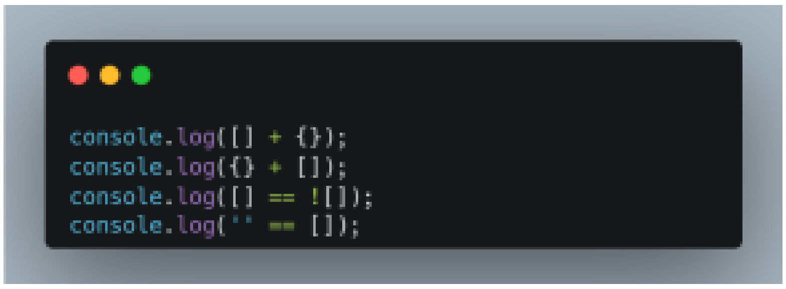


```
let a = 10, b = 20, c = 30;
console.log(a + b * c);
console.log((a + b) * c);
console.log(a + b > c ? a : b);
console.log((a > b) && (b > c) || (a > c));
```

- 610 - Among ‘+’ and ‘’ , ‘*’ has the highest precedence . So, first b and c are multiplied then the result is added to a.
- 900- Here, parentheses has the highest precedence than ‘*’ , so first and b are added then the result is multiplied by c.
- 20- This is a ternary operator as the first condition a+b >c is false, it executes the third condition i.e. b.

- false - This is a logical operator. First it checks the condition (a>b) this is false. Then it checks if the condition (b>c) is false then it will perform logical operation (a>b) && (b>c) which gives false because if one of the conditions is false then it gives false. Next it checks with (a>c) this is also false. The final result is false as false || false is false.

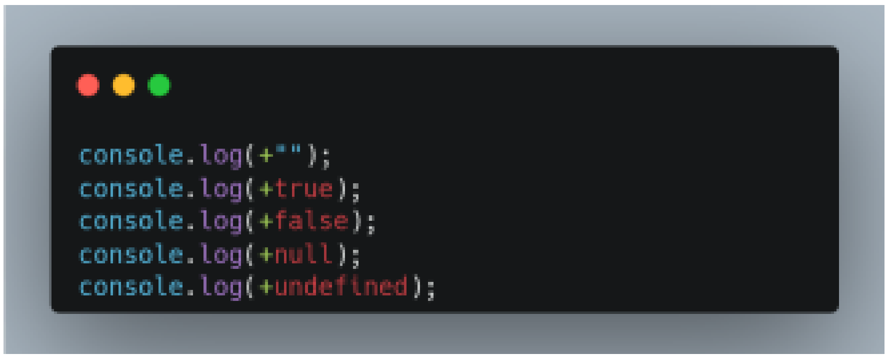
Q7. **[2 Points]** Analyze and explain the output of the following code snippets:



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

- In `[] + {}`, the array `[]` is converted to an empty string `""`, and the object `{}` is converted to `"[object Object]"`. The result is the string `"[object Object]"`.
- In `{} + []`, the `{}` is treated as a block statement, so the expression is effectively `+[]`. This converts the empty array `[]` to `0`.
- In `[] != []`, each array is a distinct object in memory. Therefore, they are not equal, so the result is `true`.
- In `'' == []`, the empty array `[]` is converted to an empty string `''`, so the comparison is effectively `'' == ''`. This results in `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);
```

- 0- When you use the unary `+` operator with a string that contains only a space (`" ")`, treat it as an empty string after trimming. An empty string is converted to `0`, so `+" "` actually results in `0`.
- `1- +true` converts the boolean `true` to a number. The `+` operator converts `true` to `1` and `false` to `0`. So, `console.log(+true)` outputs `1`.
- 0- `+false` converts the boolean `false` to a number. The `+` operator converts `false` to `0`. So, `console.log(+false)` outputs `0`.
- 0 - `+null` converts the value `null` to a number. The `+` operator converts `null` to `0`. So, `console.log(+null)` outputs `0`.
- NaN - `+undefined` converts the value `undefined` to a number. Since `undefined` cannot be converted to a meaningful number, it results in `NaN`
  So, `console.log(+undefined)` outputs `NaN`.