Mock1: ChatGBT answers

Question 38 - JS equivileancy

1. **Primitive Types**: For primitive types like numbers, strings, booleans, `null`, and `undefined`, `===` checks if the values are exactly the same and also of the same type. If both conditions are met, it returns `true`. For example, comparing two numbers with the same value (like `5 === 5`) will return `true`.
2. **Object Types**: For object types (objects, arrays, functions, etc.), `===` checks if the two operands refer to the exact same object in memory. This means that even if two different objects have identical keys and values, they will not be considered equal because they do not reference the same instance. For example:

   ```
   javascriptCopy code
   const obj1 = { name: "John" };
   const obj2 = { name: "John" };
   console.log(obj1 === obj2); // false, because they are dif
   ferent instances
   ```

   In JavaScript, `=`, `==`, and `===` are three distinct operators used for assignment and comparison. Here's a detailed explanation of each:

   1. `=` **(Assignment Operator)**:
      - The `=` operator is used to assign the value on its right to the variable on its left.
      - This operator is fundamental for setting and updating the values of variables.
      - Example:

```
javascriptCopy code
let x = 5; // Assigns the value 5 to the variable x
```

2. `==` **(Equality Operator)**:
   - The `==` operator is used to compare two values for equality after performing type coercion if necessary.
   - Type coercion means that if the two values being compared are not of the same type, JavaScript will try to convert them into a common type before making the comparison.
   - This can lead to unexpected results if not properly understood.
   - Example:

```
javascriptCopy code
console.log(5 == '5'); // true, because '5' is coerc
ed to 5 before comparison
console.log(0 == false); // true, because 0 and fals
e are considered equivalent
```

3. `===` **(Strict Equality Operator)**:
   - The `===` operator is also used to compare two values for equality, but unlike `==`, it does not perform type coercion.
   - Both the value and the type must be exactly the same for the comparison to return `true`.
   - This operator is generally recommended over `==` for comparisons because it avoids the quirks of type coercion and leads to more predictable code.
   - Example:

```
javascriptCopy code
console.log(5 === '5'); // false, because the types
```

```
are different (number vs. string)
console.log(5 === 5); // true, because both the valu
e and the type are the same
```

Overall, it's advisable to use `===` (strict equality) in most cases to ensure that comparisons are both type-safe and predictable. The `==` (equality) operator can sometimes be useful, but its behavior can be unexpected due to type coercion, so it should be used with caution.