Day 7: JavaScript Notes.

Comparison Operators

JavaScript has these comparison operators:

- > (Greater than): 5 > 3 is true.
- < (Less than): 5 < 3 is false.
- >= (Greater than or equal to): 5 >= 5 is true.
- <= (Less than or equal to): 5 <= 4 is false.
- == (Equal to, loose comparison): '5' == 5 is true because type coercion happens.
- === (Strict equality): '5' === 5 is false because types differ.
- != (Not equal, loose comparison): 5 != 3 is true.

String Comparisons

Strings are compared character by character based on their Unicode values.

Example:

```
console.log("apple" > "apricot");  // Output: false (because 'p' < 'r')
console.log("hello" < "Hello");    // Output: false (because lowercase 'h' > uppercase 'H')
```

Comparisons Between Different Types

JavaScript automatically converts values to a common type when comparing different types:

```
console.log('5' > 2);        // true  ('5' converted to number)
console.log(true == 1);      // true  (true converts to 1)
console.log(false == 0);     // true  (false converts to 0)
```

Strict Equality (===)

- Does **not** perform type conversion.

- More reliable and predictable than loose equality ==.

Example:

```
console.log(0 == false);     // true  (loose equality converts types)
console.log(0 === false);    // false (strict equality, different types)

console.log('5' == 5);       // true  (string converted to number)
console.log('5' === 5);      // false (different types)
```

Conditional Branching in JavaScript

The if Statement

The if statement allows your program to execute a block of code **only if a specified condition is true**. For example, if you want to check whether a person is an adult, you can write:

```
let age = 18;
if (age >= 18) {
  console.log("You are an adult.");
}
```

If the condition age >= 18 evaluates to true, the message "You are an adult." is printed.

The else Clause

The else clause provides an alternative block of code that runs **if the if condition is false**. This lets your program handle both cases—true and false:

```
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

So, if the age is less than 18, the program will print "You are a minor."

Multiple Conditions with else if

When you want to check **multiple conditions**, use the else if ladder. It evaluates each condition in order and executes the block for the first true condition it finds:

```
if (age < 12) {
  console.log("Kid");
} else if (age < 18) {
  console.log("Teen");
} else {
  console.log("Adult");
}
```

This way, depending on the value of age, the program prints the appropriate category: "Kid", "Teen", or "Adult".

The Ternary (Conditional) Operator ?

The ternary operator is a **compact alternative** to if-else. It follows this syntax:

```
let result = condition ? valueIfTrue : valueIfFalse;
```

For example, checking if a person is a minor or adult can be written as:

```
let person = (age < 18) ? 'Minor' : 'Adult';
```

You can also chain ternary operators to handle multiple conditions:

```
let person = (age < 12) ? 'Kid' :
             (age < 18) ? 'Teen' :
             (age < 60) ? 'Adult' : 'Senior';
```

Although concise, overly long ternary chains can reduce readability, so use them carefully.

The switch Statement

The switch statement is another way to perform **multiple conditional checks** based on the value of an expression. It’s often clearer than many else if statements when checking one variable against different values.

The general syntax is:

```
switch (expression) {
  case value1:
    // code to execute if expression === value1
    break;
  case value2:
    // code to execute if expression === value2
    break;
  default:
    // code to execute if no cases match
}
```

Example:

```
let a = 2 + 2;

switch (a) {
  case 3:
    console.log('Too small');
    break;
  case 4:
    console.log('Exactly!');
    break;
  case 5:
    console.log('Too big');
    break;
  default:
    console.log("I don't know such values");
}
```

Here, since a equals 4, the output will be "Exactly!". The break statements prevent fall-through to subsequent cases.