Lesson 02.01

- logical operators (==, <, >, <=, >=, ===, !=, !==)
- conditional logic: if - else if - else
- variable scope let & const vs var in { code block}
- single (=), double (==) and triple (===) equal signs
- = is the Assignment Operator; it assigns a value to a variable
- == equality (1 and '1' are equal -- "close enough")
- === strict equality (1 and '1' are NOT strictly equal)
- = Assignment Operator; it sets a value
- == Comparison Operator -- compares 2 values, but does NOT change any value
  - compares two values to see if they are equal in value
  - the comparison returns a boolean (true or false)
- === Strict Comparison Operator -- returns false if value matches BUT data type does not
- == vs === console.log("1 == '1':", 1 == '1'); - true console.log("1 === '1':", 1 === '1'); - false
- != vs !== console.log("1 != '1':", 1 != '1'); - false console.log("1 !== '1':", 1 !== '1'); - true
- 'Hello' vs 'hello' -- not equal, not even w less-strict ==, cuz JS is case-sensitive console.log("'Hello' == 'hello':", 'Hello' == 'hello'); - false console.log("'Hello' === 'hello':", 'Hello' === 'hello'); - false

2. Using ==, log x == 5 and x == 6:

```
console.log(x == 5); -  true
console.log(x == 6); -  false
```

3. Compare x, which is 5, to the number-like string "5". Even though 5 is not "5", the == sees them as equal in value.

```
console.log(x == "5"); -  true
```

=== is the Strict Comparison Operator. To return true, the comparsion must be equal in both value and data type.

4. Using ===, compare of x to "5". It's false, because 5 and "5", have different data types.

```
console.log(x === "5"); -  false
```

Comparison operators:

```
>   greater than in value
<   less than in value
>=  greater than or equal to in value
<=  less than or equal to in value
!   not (opposite)
!=  not equal in value
!== not equal in both value and data type
== equal in value but not necessarily in data type
=== equal in both value and data type
```

5. Open the console and type these comparisons, hitting enter each time:

```
8 == 8; -  true
8 == "8"; -  true
8 === "8"; -  false
```

6. Try the ! (NOT) operator:

```
7 != 8; -  true
7 != 7; -  false
7 != '7'; -  false
7 !== '7'; -  true
```

7. Try the greater than / less than operators:

```
7 >= 8; -  false
7 >= '7'; -  true
7 <= 8; -  true
7 <= '7'; -  true
```

alphanumeric characters for comparison purposes

The "digits" of number-like strings are characters, so comparisons are alphabetic, not numeric.

The following character sets evaluate in ascending order:

- number-like strings ("0123456789")
- uppercase letters ("ABCDEFGHIJKLMNOPQRESUVWZYZ")
- lowercase letters ("abcdefghijklmnopqurstuvwxyz")

- alphanumeric low-to-high: "0123456789ABCDEFGHIJKLMNOPQRESUVWZYZabcdefghijklmnopqurstuvwxyz"

8. Open the Console, and type these comparisons:

```
'1' < 'A'; –  true
'A' < 'a'; –  true
'A' == 'a'; –  false
'Z' > 'a'; –  false
'z' > 'a'; –  true
```

9. Still in the Console, compare some numbers as well as their number-like string equivalents:

```
1000 > 50; –  true (numeric)
"1000" > "50"; –  false (alphabetic)
10000 < 5; –  false (numeric)
"10000" < "5"; –  true (alphabetic)
```

conditional logic with "if"

An "if statement" compares two values in parentheses and returns a boolean (true / false).

- The comparison is called the condition
- If the condition is true, the code inside the curly braces runs

evaluating a number in an if statement

10. Write an if statement that checks if x equals 5. It runs, because the condition is true:

```
if(x == 5) {
console.log("x equals 5"); //  runs
}
```

If the condition is false, the code inside the curly braces does not run

11. Try another if statement that checks if x equals 3. It doesn't run, because the comparison is false:

```
if(x == 3) {
    console.log("x equals 3"); //  doesn't run
}
```

if else

Often, we will want to run one piece of code if the condition is true
and some other piece of code if it is false. The "other part" is handled
by "else". The "else part" does not evaluate a condition, and therefore has no parentheses.

evaluating a number in an if-else statement

12. Add an else part to run if the condition is false:

```
if(x == 3) {
    console.log("x is 3"); // doesn't run
} else {
console.log("x is not 3"); // runs
}
```

13. "x is not 3" doesn't tell us much about x, so concatenate x into the output:

```
if(x == 3) {
    console.log("x is 3"); // doesn't run
} else {
    console.log(`x is ${x}, not 3`); // runs
}
```

14. Try the **!=** (not equal) operator, where the logic is reversed:

```
if(x != 3) {
    console.log(`x is ${x}, not 3`); // runs
} else {
    console.log("x is 3"); // doesn't run
}
```

15. Try the **>** (greater than) operator:

```
if(x > 3) {
    console.log(`x is ${x}, which is greater than 3`); // runs
} else {
    console.log(`x is ${x}, which is not greater than 3`); // doesn't
run
}
```

16. Try the < (less than) operator:

```
if(x < 3) {
    console.log(`x is ${x}, which is less than 3`); //  doesn't run
} else {
    console.log(`x is ${x}, which is not less than 3`); //  runs
}
```

17. Try the <= (less than or equal to) operator:

```
if(x <= 5) {
    console.log(`x is ${x}, which is less than or equal to 5`); //
runs
} else {
    console.log(`x is ${x}, which is not less than or equal to 5`); //
doesn't run
}
```

evaluating a string with if-else logic

18. Declare that the weather is "sunny", and do an if-else that checks if the weather is sunny:

```
let weather = "sunny";

if(weather == "sunny") {
    console.log("It's sunny! Let's go to the beach!"); //  runs
} else {
    console.log("It's not sunny! No beach today!"); //  doesn't run
}
```

19. Change the weather to "stormy", and run it again. This time, concatenate the variable into the output so we can see what the weather is:

```
weather = "stormy";

if(weather == "sunny") {
    console.log("It's sunny! Let's go to the beach!"); //  doesn't run
} else {
    console.log(`It's ${weather}! No beach today!`); //  runs
}
```

evaluating a boolean with if-else logic

20. Declare a boolean, and evaluate it in an if statment to see if it's true:

```
let raining = true;

if(raining == true) {
    console.log("It's raining!"); //  runs
}
```

21. Get rid of the == comparison, and just evaluate the variable itself:

```
if(raining) {
    console.log("It's raining!"); //  runs
}
```

22. Check if raining is false. Since raining is true, add an else part:

```
if(raining == false) { -  if it is true that raining is false
    console.log("It's not raining!"); //  doesn't run
} else { -  else it is false that raining is false
    console.log("It's raining!"); //  runs
}
```

Checking if a boolean is false involves "reverse logic":

- Is it true that it's true? If so, it's false that it's false.
- Is it true that it's false? If so, it's false that it's true.
- **!** (NOT operator) before a boolean checks for false (NOT true)

23. Get rid of the == comparison, and put ! before the variable:

```
if(!raining) {  -  if raining is false (NOT true)
    console.log("It's not raining!"); //  doesn't run
} else { -  else raining is true (NOT false)
    console.log("It's raining!"); //  runs
}
```

if-else if-else logic

else if adds another condition to evaluate. Think of it as a sandwich, where "if" and "else" are the bread and "if else" is the ingredient(s) between the bread.
Let's try an "if - else if - else", where we compare two numbers and do one of three things, depending on their relative values.

24. Declare two numeric variables:

```
let highScore = 15000;
let myScore = 10000;
```

25. Check if my score is greater than the high score:

```
if(myScore > highScore) {
    console.log("Congrats! You beat the high score!"); //  doesn't run
}
```

26. Add an else part:

```
if(myScore > highScore) {
    console.log("Congrats! You beat the high score!");
} else {
    console.log("Sorry! You didn't beat the high score!"); //  runs
}
```

The if-else checks for higher and lower, but what if you tied the high score?

27. Add an else if between the if and the else. Now we handle all three possible outcomes:

```
if(myScore > highScore) {
    console.log("Congrats! You beat the high score!");
} else if(myScore < highScore) {
    console.log("Sorry! You didn't beat the high score!"); //  runs
} else {
    console.log("Wow! You tied the high score!");
}
```

28. Make myScore greater than highScore and run it. Now the "if part" runs ("Congrats!")

```
myScore = 18000;
```

29. Make myScore equal to highScore and run it. Now the "else part" runs ("Wow!")

```
myScore = 15000;
```

30. Revisiting the weather logic, let's add an "else if part":

```
weather = "cloudy";

if(weather == "sunny") {
    console.log("It's sunny!");
} else if(weather == "raining") {
    console.log("It's raining!");
} else { –  it's neither sunny nor raining
    console.log(`It's ${weather}`); –  It's cloudy
}
```

31. Change weather to "sunny" so that the "if part" runs:

```
weather = "sunny";
```

32. Change weather to "raining" so that the "else if" runs:

```
weather = "raining";
```

multiple "if else" conditions

Just as a sandwich can have multiple items between the bread, so too can there be mulitple "else if" blocks between the "if" and "else". Only ONE code block ever runs, however, regardless of how many "else if" blocks there may be.

Challenge: Using multiple else if blocks, set the grade based on the score. Possible grades are:

- A (90+), B (80+), C (70+), D (60+), F (less than 60)

33. Declare a numeric variable.

```
let score = 77;
let grade = "";
```

34. If the score is at least 90, declare a grade variable with a value to "A":

```
if(score >= 90) {
    grade = "A";
}
```

35. If the score is less than 90, make that grade of "F":

```
if(score >= 90) {
    let grade = "A";
} else {
    let grade = "F";
}
```

36. A failing grade for anything less than 90 is pretty harsh, so add an "else if" part that awards a "B" if the score is at least 80:

```
if(score >= 90) {
    let grade = "A";
} else if(score >= 80) {
    let grade = "B";
} else {
    let grade = "F";
}
```

37. We're still failing any score below 80, so add two more if else blocks for grades of "C" and "D":

```
if(score >= 90) {
    let grade = "A";
} else if(score >= 80) {
    let grade = "B";
} else if(score >= 70) {
    let grade = "C";
} else if(score >= 65) {
    let grade = "D";
} else {
    let grade = "F";
}
```

38. Concatenate the score and grade into a report card and check the Console:

```
let reportCard = `Score: ${score} | Grade: ${grade}`;
```

We get an error: grade is not defined. But why? Clearly, we declared the grade variable and assigned it a string value Comment out the reportCard line to get rid of the error. This brings us to the important topic of variable scope.

variable scope: global vs. block scope

Variable scope refers to where in the code it exists and is available. We need to consider global scope vs. block scope.

- **score** is declared in the **global scope** and is therefore available everywhere in the script. By global scope we mean that the variable is not declared inside curly braces.
- **grade**, however, is declared inside curly braces. The code inside curly braces is known as a code block. Variables declared inside a code block are **block-scoped**, that is, only available inside the code block.

39. Move the declaration of grade into the global scope, right after score. Since we do not have a grade yet, set it equal to an empty string. Let's change the score while we are at it:

```
score = Math.floor(Math.random() * 61) + 40;
let grade = "";
```

40. Rewrite the "if else if - else" logic without the "let". Now, the global grade variable is assigned the "B":

```
if(score >= 90) {
    grade = "A";
} else if(score >= 80) {
    grade = "B";
} else if(score >= 70) {
    grade = "C";
} else if(score >= 65) {
    grade = "D";
} else {
    grade = "F";
}
```

41. Make a new reportCard and log it:

```
let reportCard = `Score: ${score} | Grade: ${grade}`;
console.log(reportCard); -  Score: 87 | Grade: B
```