Full Stack AI Software Development

Conditional and loop statements

Job Connector Program

Outline

Conditional Statement
Learn about conditionals, which let programs make decisions by checking if something is true or false.

Loop Statement
Learn about loops, which let programs repeat actions without writing the same code again.

Conditional and Loop Statements

When writing programs, we often need to:

- **Make decisions** → Run certain code only if a condition is true.
- **Repeat tasks** → Run the same block of code multiple times efficiently.

That’s where conditional statements and loops come in.

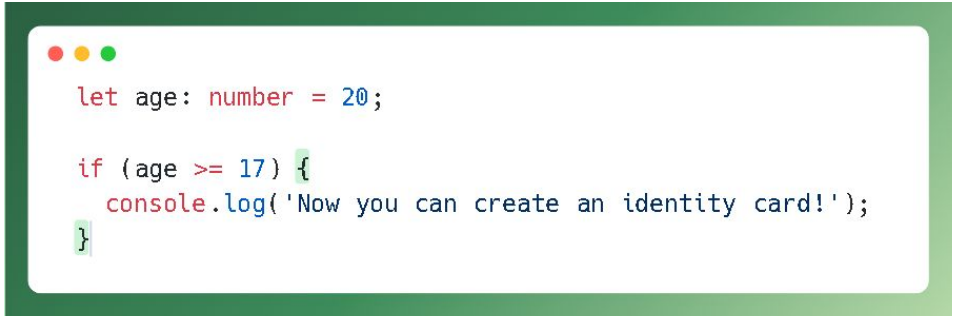
What are Conditional Statements?

- **Conditional statements** let your program **choose different paths depending on whether conditions are true or false**.
- We often use:
  - if
  - if...else
  - if...else if...else
  - switch

If statement

The **if statement** is the simplest way to make decisions in code. It checks a condition, and if the condition is true, the code inside will run.

In the example, the message will only show **if the condition (age >= 17) is true**.

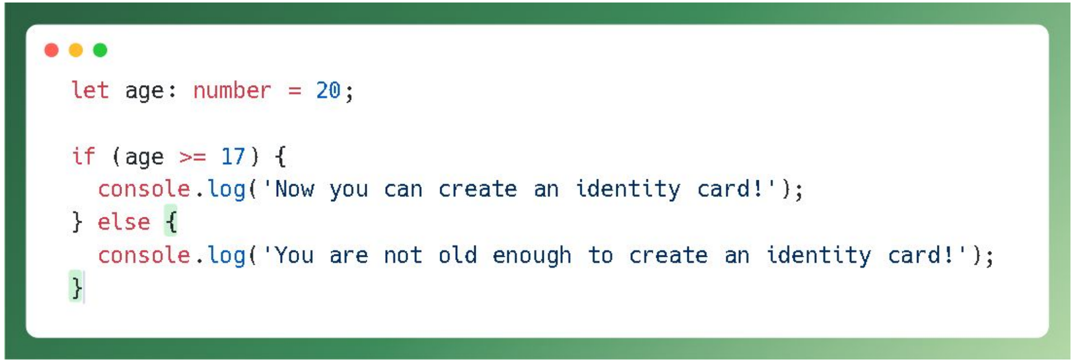


```
let age: number = 20;

if (age >= 17) {
  console.log('Now you can create an identity card!');
}
```

Else statement

The **else** statement is like a backup for an **if**. It doesn’t need a condition, it just **runs when the if condition is false**.

In this example, we added an else statement. If the condition is not true, the else block will run instead. So **if age is less than 17**, the output will be: “You are not old enough to create an identity card!”



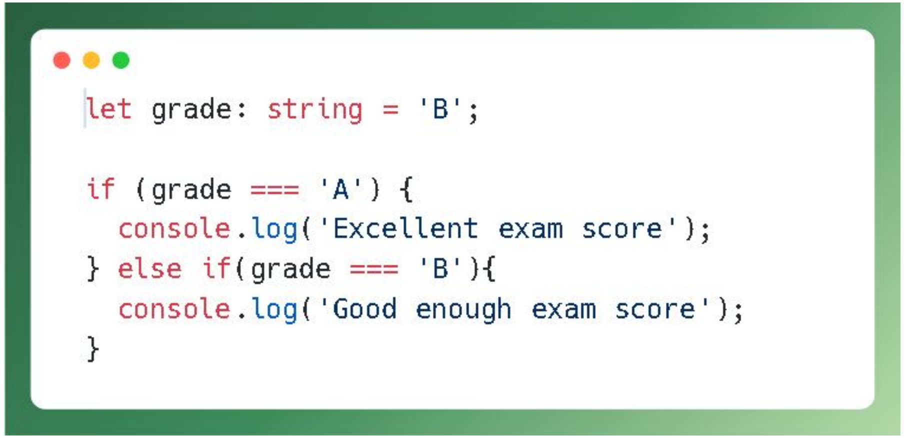
```
let age: number = 20;

if (age >= 17) {
  console.log('Now you can create an identity card!');
} else {
  console.log('You are not old enough to create an identity card!');
}
```

Else if statement

Sometimes, we need more than two possible outcomes. This is where the else if statement helps.

An **else if** is like **combining if and else**. It acts as a backup, but it also has its own condition to check.

In the example, grade has the value “B”. The first if condition (grade === ‘A’) is false, so the code checks the else if condition. Since grade === ‘B’ is true, the code inside runs and the output will be: “Good enough exam score”.

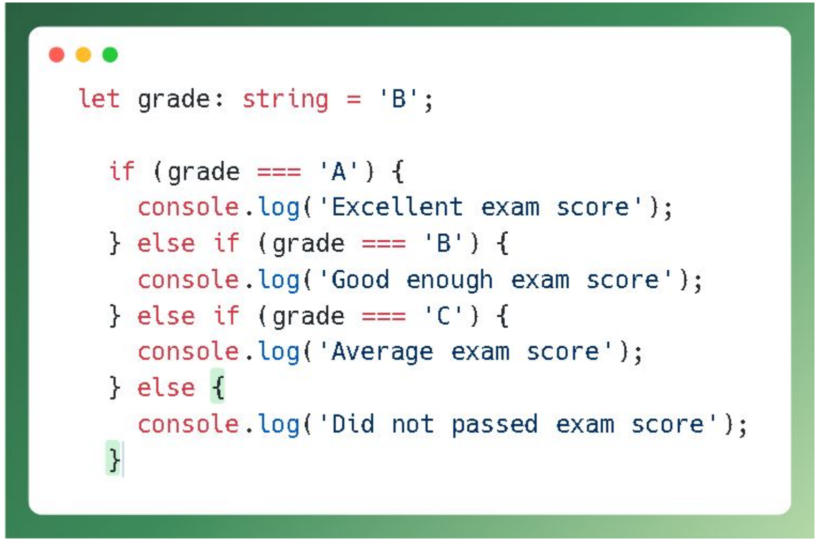


```
let grade: string = 'B';

if (grade === 'A') {
  console.log('Excellent exam score');
} else if(grade === 'B'){
  console.log('Good enough exam score');
}
```

Chaining conditions

We can chain multiple else if statements to handle more possible outcomes.

We can also add a final else statement to cover any other cases that don't match.



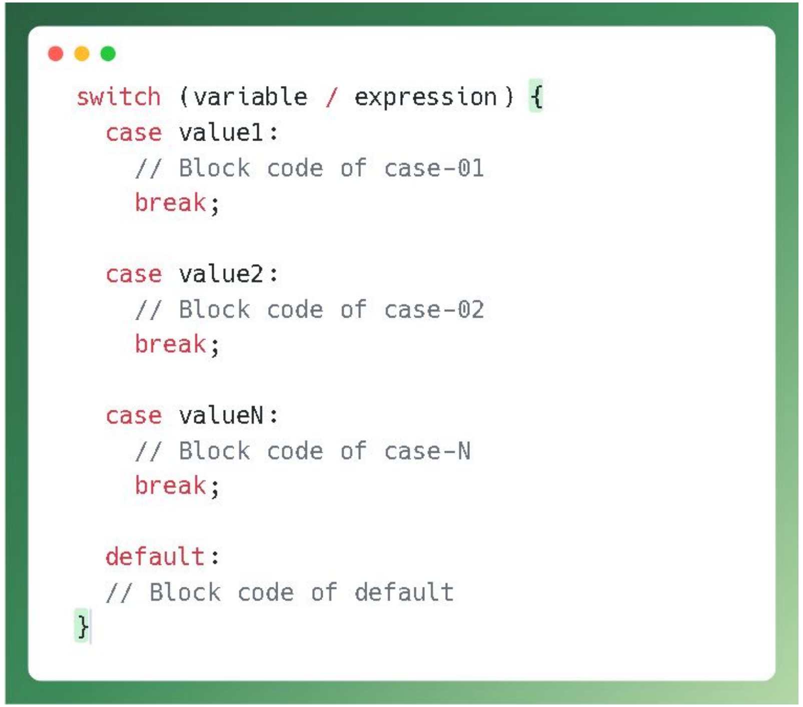
```
let grade: string = 'B';

  if (grade === 'A') {
    console.log('Excellent exam score');
  } else if (grade === 'B') {
    console.log('Good enough exam score');
  } else if (grade === 'C') {
    console.log('Average exam score');
  } else {
    console.log('Did not passed exam score');
  }
```

Switch Case

The **switch statement is another way to make decisions**. It checks the value of an expression and runs the code block that matches that value.

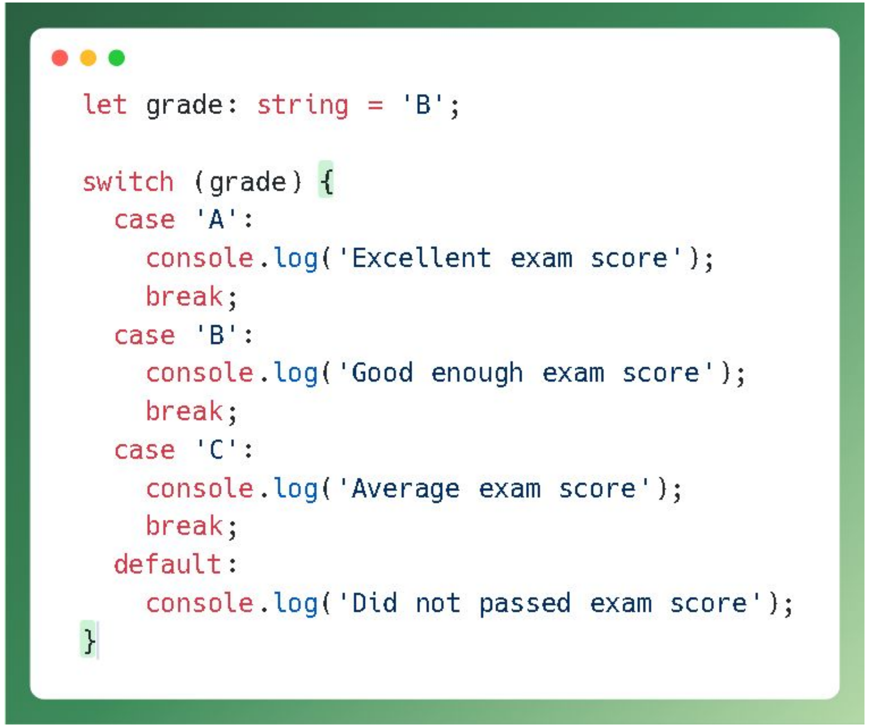
The **switch statement** checks a value inside parentheses ().

- If it matches case 1, the code in that block runs.
- If it matches case 2, the code in that block runs.
- This continues for all cases.
- If no case matches, the default block runs.



```
switch (variable / expression) {
  case value1:
    // Block code of case-01
    break;

  case value2:
    // Block code of case-02
    break;

  case valueN:
    // Block code of case-N
    break;

  default:
  // Block code of default
}
```

Switch Case - Example



```
let grade: string = 'B';

switch (grade) {
  case 'A':
    console.log('Excellent exam score');
    break;
  case 'B':
    console.log('Good enough exam score');
    break;
  case 'C':
    console.log('Average exam score');
    break;
  default:
    console.log('Did not passed exam score');
}
```

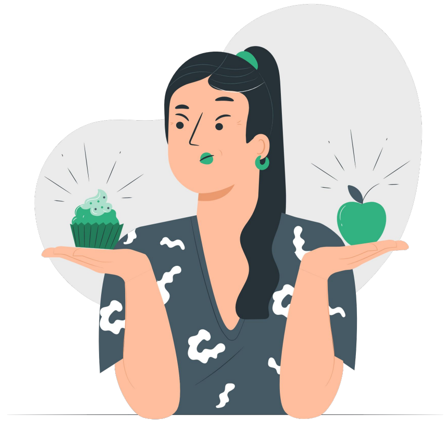
Difference Between *if else* and *switch* Statement

	if-else	**switch**
Definition	Runs code blocks based on conditions (**true** or **false**).	Runs the code block that matches a case.
Evaluation	Works with numbers, strings, booleans, and logical conditions.	Works best with fixed values like numbers or strings.
Testing	Can test equality and logical expressions (>, <, &&, ||)	
Expression	Needs multiple if...else if statements for many conditions.	One switch can handle many possible values clearly.

Truthy and *falsy* values

Truthy and Falsy are terms used to describe how values behave in a boolean context (like inside an if statement).

- A truthy value is treated as true. For example, 1 or "hello" are truthy.
- A falsy value is treated as false. For example, 0, "" (empty string), null, undefined, and NaN are falsy.



Falsy

- "" (empty string)
- 0
- null
- undefined
- NaN

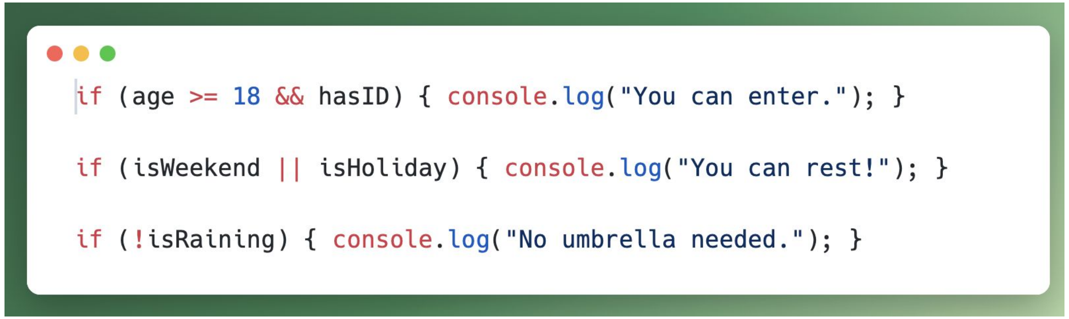
Truthy

- " " (blank character string)
- [] (empty array)
- {} (empty object)
- 1
- "1" (string)
- "0" (string)
- "false" (string)
- "true" (string)

Logical Operators

Logical operators are used to combine or invert conditions in decision making. They return a boolean value (true or false).

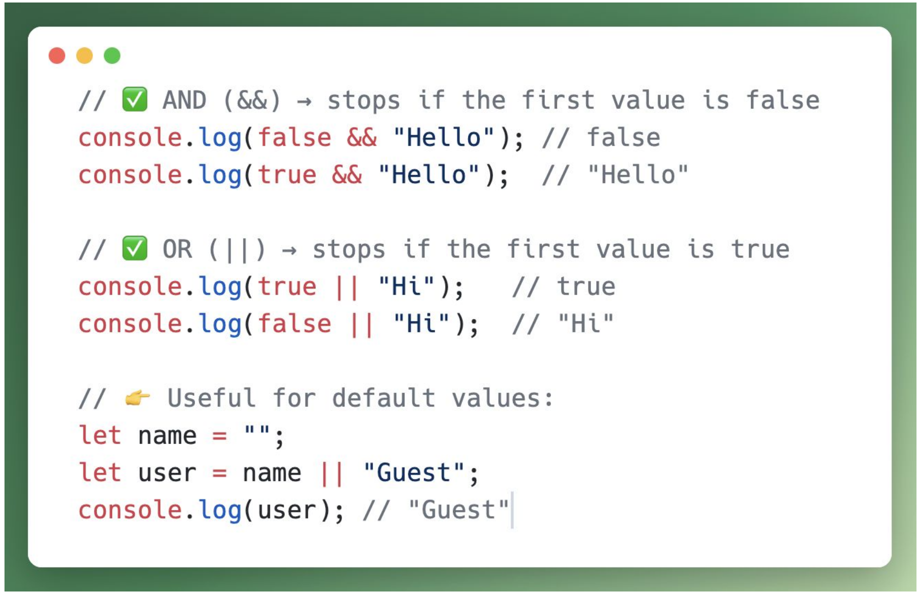
- **AND (&&)** → Returns true only if both conditions are true.
- **OR (||)** → Returns true if at least one condition is true.
- **NOT (!)** → Reverses a condition. !true becomes false, and !false becomes true.



```
if (age >= 18 && hasID) { console.log("You can enter."); }

if (isWeekend || isHoliday) { console.log("You can rest!"); }

if (!isRaining) { console.log("No umbrella needed."); }
```

Short-Circuiting

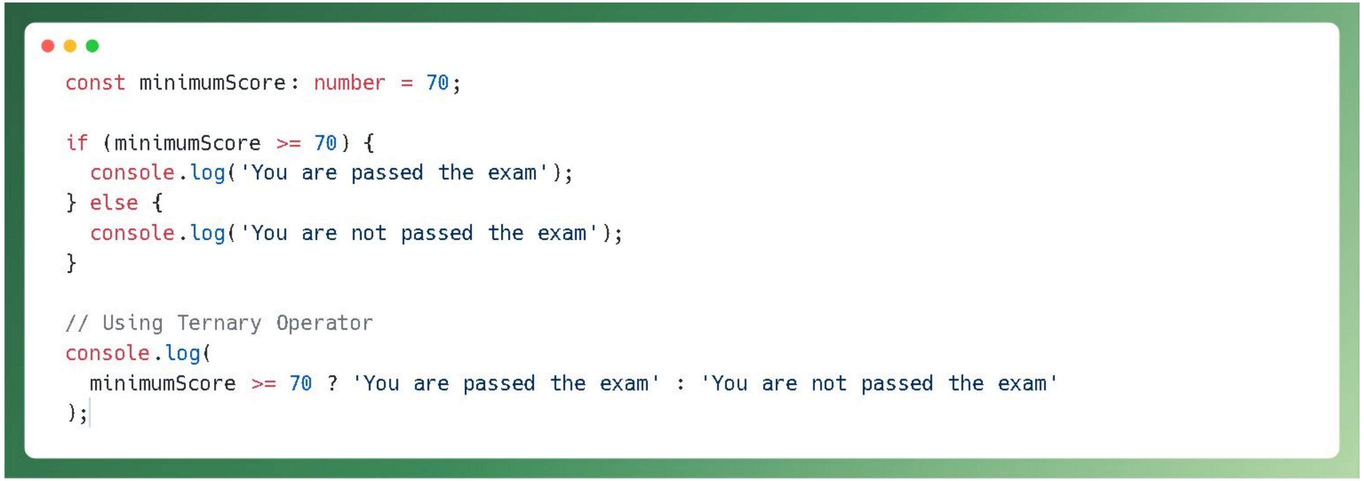
Logical operators can stop early once the result is known.



```javascript
// ✅ AND (&&) → stops if the first value is false
console.log(false && "Hello"); // false
console.log(true && "Hello");  // "Hello"

// ✅ OR (||) → stops if the first value is true
console.log(true || "Hi");    // true
console.log(false || "Hi");  // "Hi"

// 👉 Useful for default values:
let name = "";
let user = name || "Guest";
console.log(user); // "Guest"
```

Ternary Operator

- A shortcut for if...else in one line.
- Syntax: **condition ? valueIfTrue : valueIfFalse**



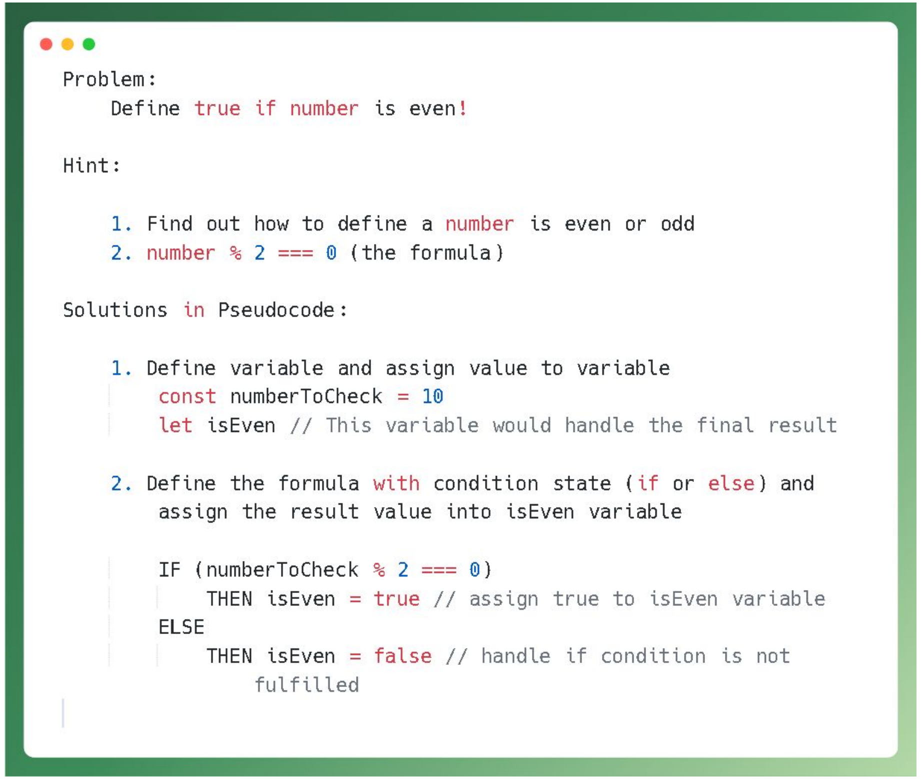
```
const minimumScore: number = 70;

if (minimumScore >= 70) {
  console.log('You are passed the exam');
} else {
  console.log('You are not passed the exam');
}

// Using Ternary Operator
console.log(
  minimumScore >= 70 ? 'You are passed the exam' : 'You are not passed the exam'
);
```

Pseudocode in Conditional Statement

Since you have learn about pseudocode in the last session. Here is example of pseudocode implemented in conditional statement.

Remember, pseudocode will help you to solve a problem with easier approach!

Don’t forget to convert this pseudocode into a programming code!



```
Problem:
    Define true if number is even!

Hint:

    1. Find out how to define a number is even or odd
    2. number % 2 === 0 (the formula)

Solutions in Pseudocode:

    1. Define variable and assign value to variable
        const numberToCheck = 10
        let isEven // This variable would handle the final result

    2. Define the formula with condition state (if or else) and
        assign the result value into isEven variable

        IF (numberToCheck % 2 === 0)
            THEN isEven = true // assign true to isEven variable
        ELSE
            THEN isEven = false // handle if condition is not
                fulfilled
```

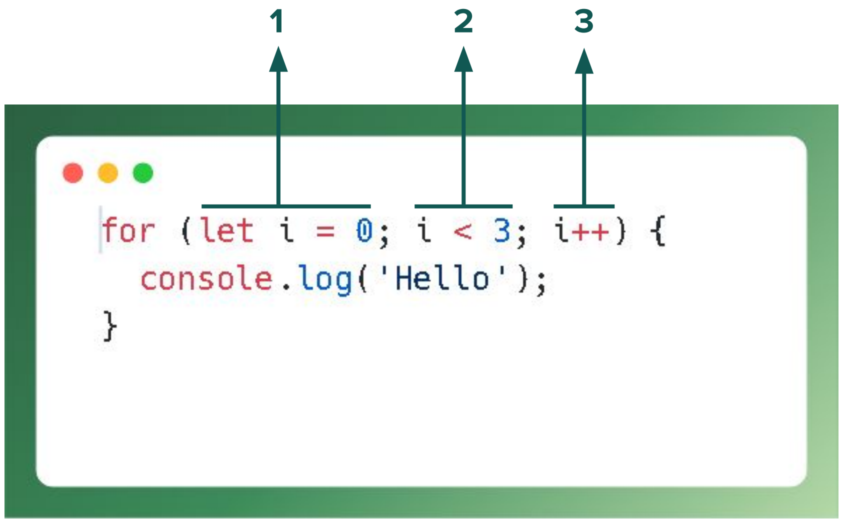
What are Loop Statements?

- **Loops** allow you to repeat code until a condition is met.
- Instead of writing the same line many times, we use **loops**.
- We often use:
  - for
  - while
  - do...while
  - for...of
  - for...in

For Loop

A ***for loop*** consists of 3 statements in its conditions.

- **The first** statement is executed once before the execution of the code block, to initialize the iteration variable.
- **The second** statement defines the condition for executing the code block.
- **The third** statement is executed every time after the execution of the code block.

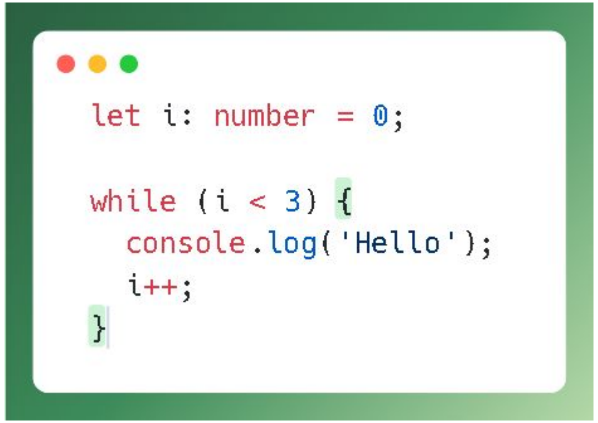


```
for (let i = 0; i < 3; i++) {
  console.log('Hello');
}
```

While Loop

- While loops are basically ***if conditions* that are repeated.**
- As long as the condition is **true**, the loop will continue.

This is how you should make a while loop statement. In every iteration, the ***i*** variable will be incremented, therefore the condition will eventually result in a false boolean

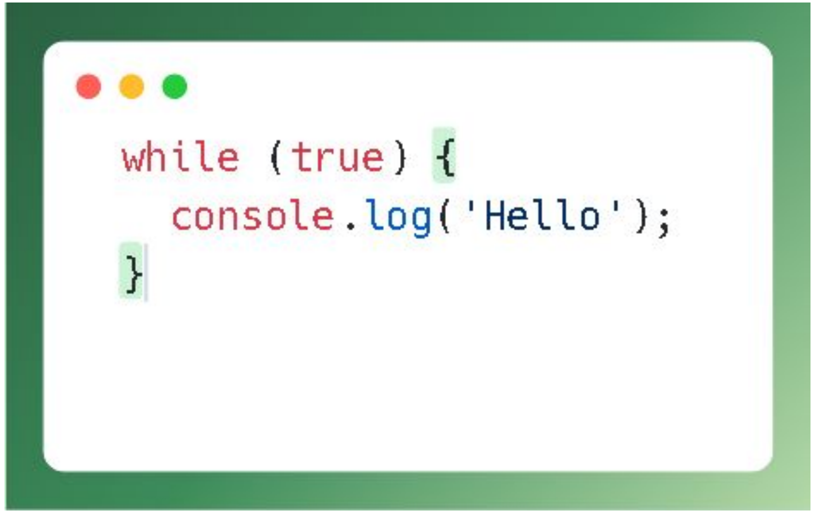


```
let i: number = 0;

while (i < 3) {
  console.log('Hello');
  i++;
}
```

While Loop

This loop will result in an infinite loop. Which means the loop will never stop.

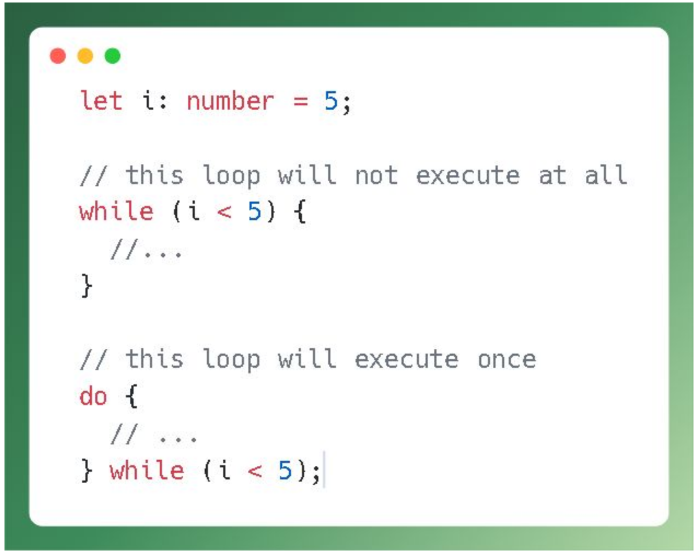
Keep in mind that when using loops, **we should always set a condition** so that the loop will eventually break/stop.



```
while (true) {
  console.log('Hello');
}
```

Do ... While loop

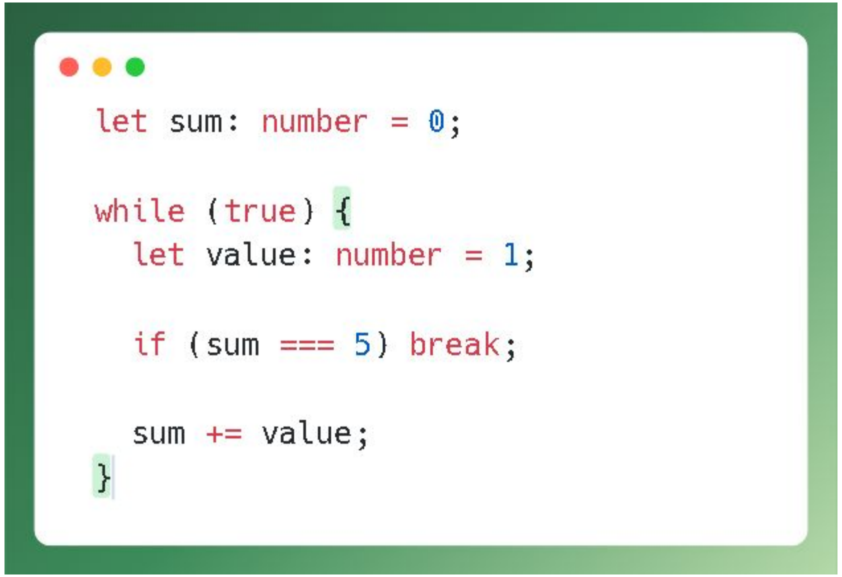
- Do while loops are very similar to while loops. The only difference it has is that it only starts **checking the condition after the first code block execution.**
- In this example, the **i** variable already has a value of **5**.
- The while loop will not execute since the condition is checked before the code block execution, and the condition itself results in a false value.
- However the do while loop will execute at least once, because the condition is checked only after the first code block execution



```
let i: number = 5;

// this loop will not execute at all
while (i < 5) {
  //...
}

// this loop will execute once
do {
  // ...
} while (i < 5);
```

Break

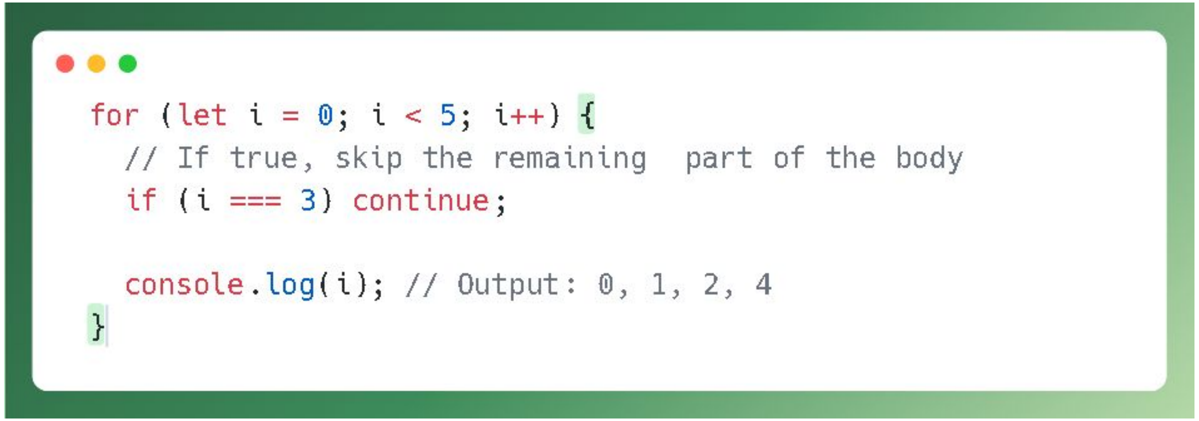
- Normally, a loop exits when its condition becomes **falsy**. But we can force the exit at any time using the special **break** directive.
- In this code, the loop will stop when the value of **sum** is 5.



```
let sum: number = 0;

while (true) {
  let value: number = 1;

  if (sum === 5) break;

  sum += value;
}
```

Continue

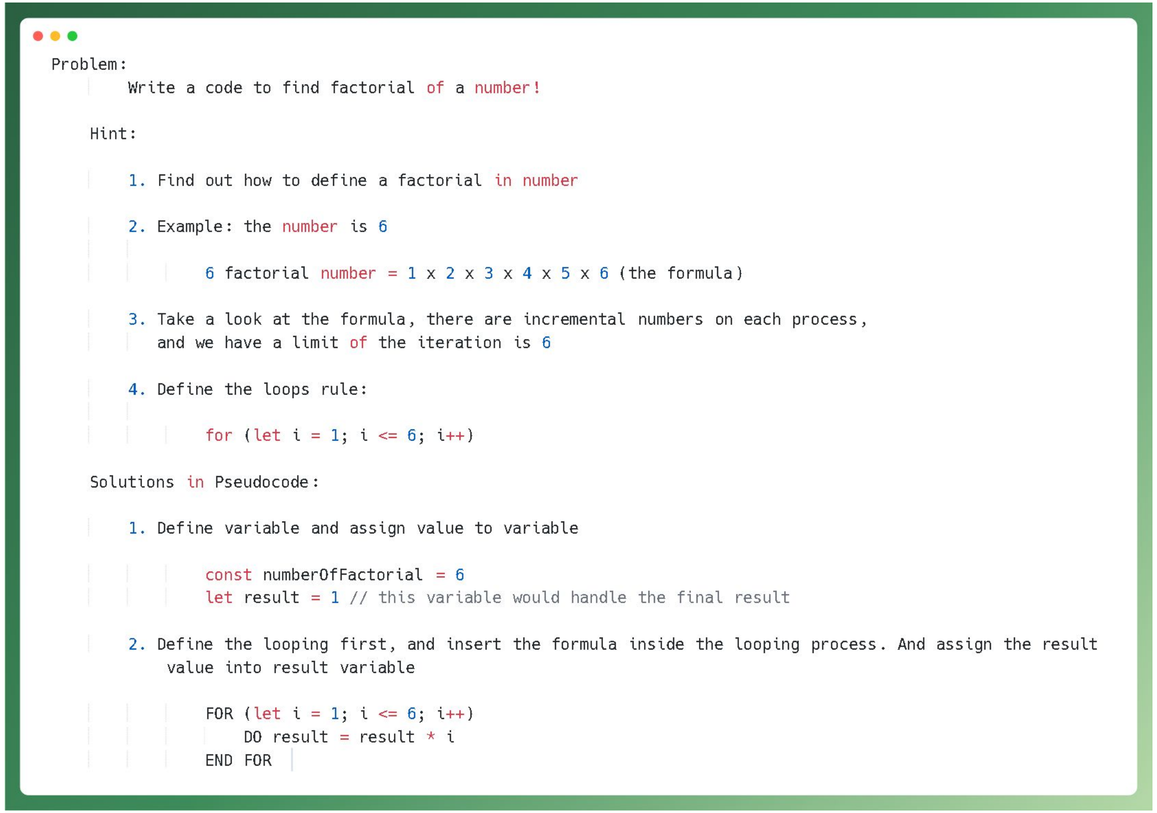
- The continue directive is a “**lighter version**” of break. It doesn’t stop the whole loop. Instead, it **stops the current iteration and forces the loop to start a new one** (if the condition allows).
- We can use it if we’re done with the current iteration and would like to move on to the next one.



```
for (let i = 0; i < 5; i++) {
  // If true, skip the remaining  part of the body
  if (i === 3) continue;

  console.log(i); // Output: 0, 1, 2, 4
}
```

Pseudocode in Looping Statement

Check out this pseudocode in order to solve a problem that needs to implement looping statement. Try to solve this problem with another looping such as WHILE!



```
Problem:
    Write a code to find factorial of a number!

    Hint:

        1. Find out how to define a factorial in number

        2. Example: the number is 6

                6 factorial number = 1 x 2 x 3 x 4 x 5 x 6 (the formula)

        3. Take a look at the formula, there are incremental numbers on each process,
           and we have a limit of the iteration is 6

        4. Define the loops rule:

                for (let i = 1; i <= 6; i++)

    Solutions in Pseudocode:

        1. Define variable and assign value to variable

                const numberOfFactorial = 6
                let result = 1 // this variable would handle the final result

        2. Define the looping first, and insert the formula inside the looping process. And assign the result
            value into result variable

                FOR (let i = 1; i <= 6; i++)
                    DO result = result * i
                END FOR
```

Exercise

- Write a code to check whether the number is odd or even
    - Example: 25 ➔ odd number, 2 ➔ even number
- Write a code to check whether the number is prime number or not
    - Example: 7 ➔ 7 is a prime number
    - Example: 6 ➔ 6 is not a prime number
- Write a code to find the sum of the numbers 1 to N
    - Example: 5 ➔ 1 + 2 + 3 + 4 + 5 = 15
    - Example: 3 ➔ 1 + 2 + 3 = 6
- Write a code to find factorial of a number
    - Example: 4! ➔ 4 x 3 x 2 x 1 = 24
    - Example: 6! ➔ 6 x 5 x 4 x 3 x 2 x 1 = 720
- Write a code to print the first N fibonacci numbers
    - Example: 15 ➔ 610

Thank you