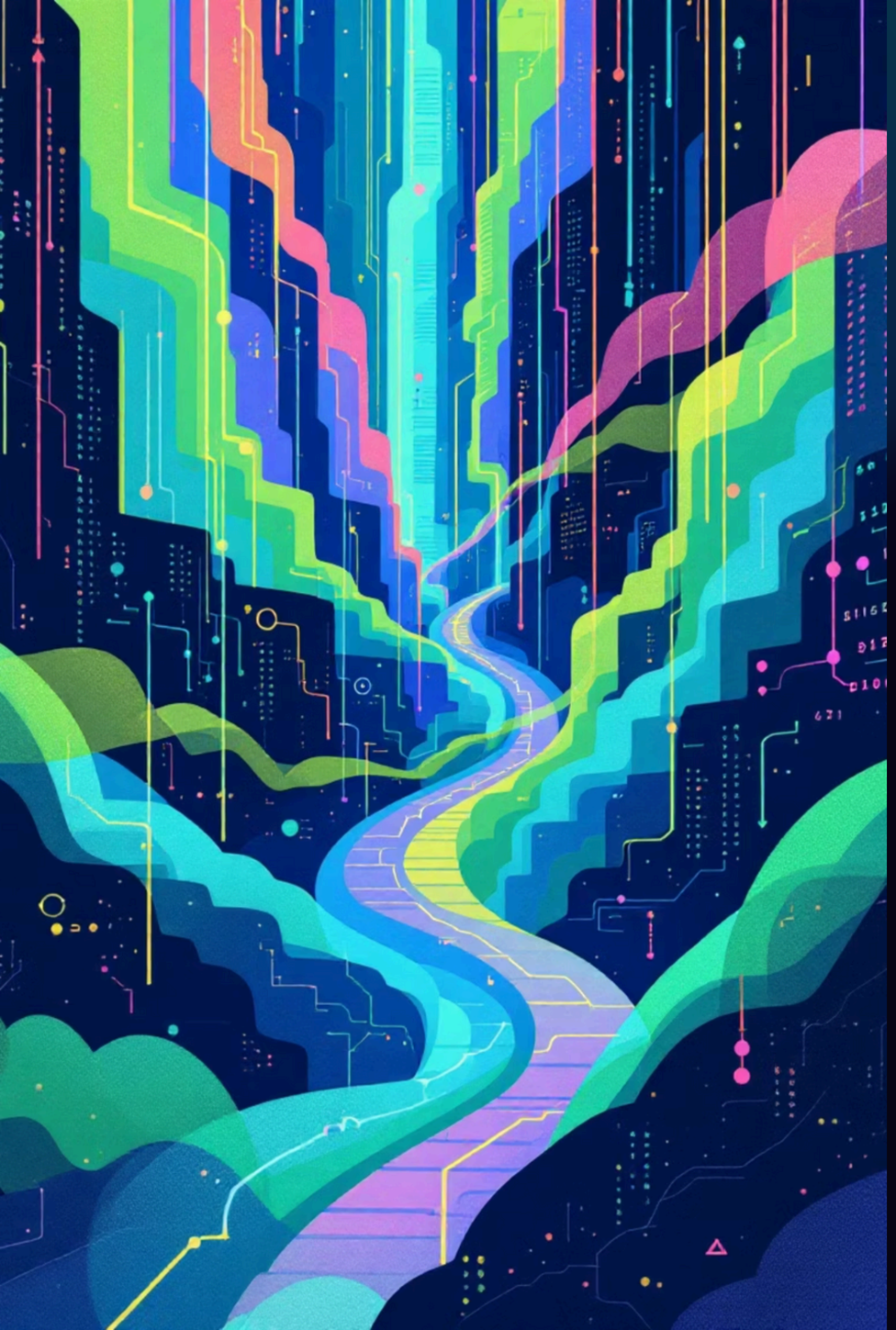
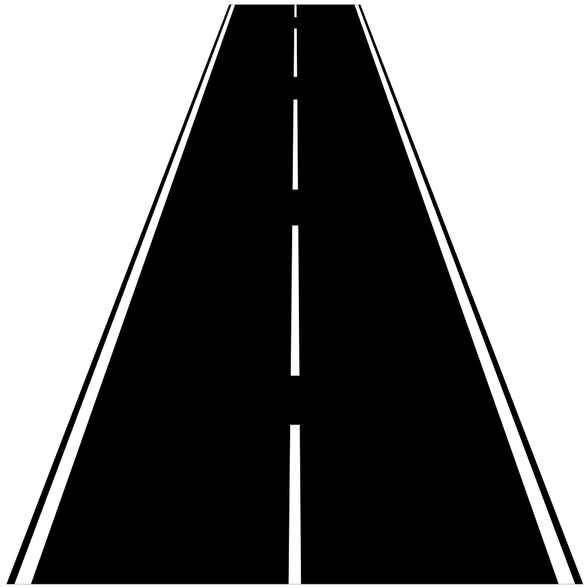


Session 3: Control Statements

Understanding if, if-else, and switch-case statements to make your code intelligent and responsive.

Instructor
Vipin Kumar Soni

Before Control Statment



After Control Statment

Why Control Statements Matter

Direct Program Flow

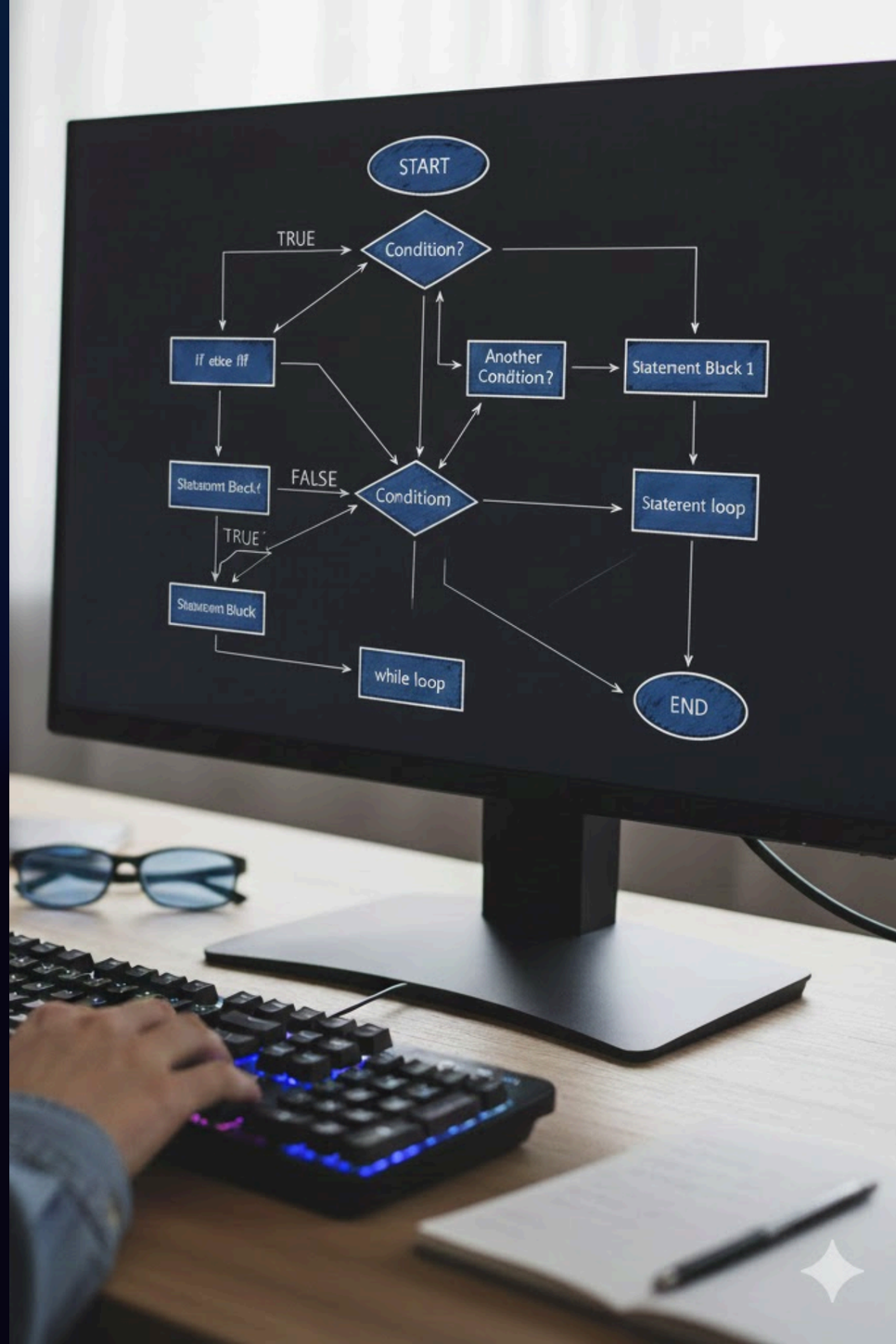
Control statements guide your programme's execution path based on specific conditions, creating logical branches in your code.

Enable Decision-Making

They allow your code to make intelligent choices: "If this condition is true, then execute that action."

Foundation for Dynamics

These structures form the backbone of dynamic, responsive software that adapts to different inputs and scenarios.



If-Else Condition

if

else

The if Statement: Basic Conditional Execution

How It Works

The if statement executes a block of code only when a Boolean condition evaluates to **true**. If the condition is false, the code block is simply skipped.

Syntax Structure

```
if (condition) {
    // code executes if condition is non-zero
}
```

This is the simplest form of conditional logic, perfect for situations requiring a single yes-or-no decision.

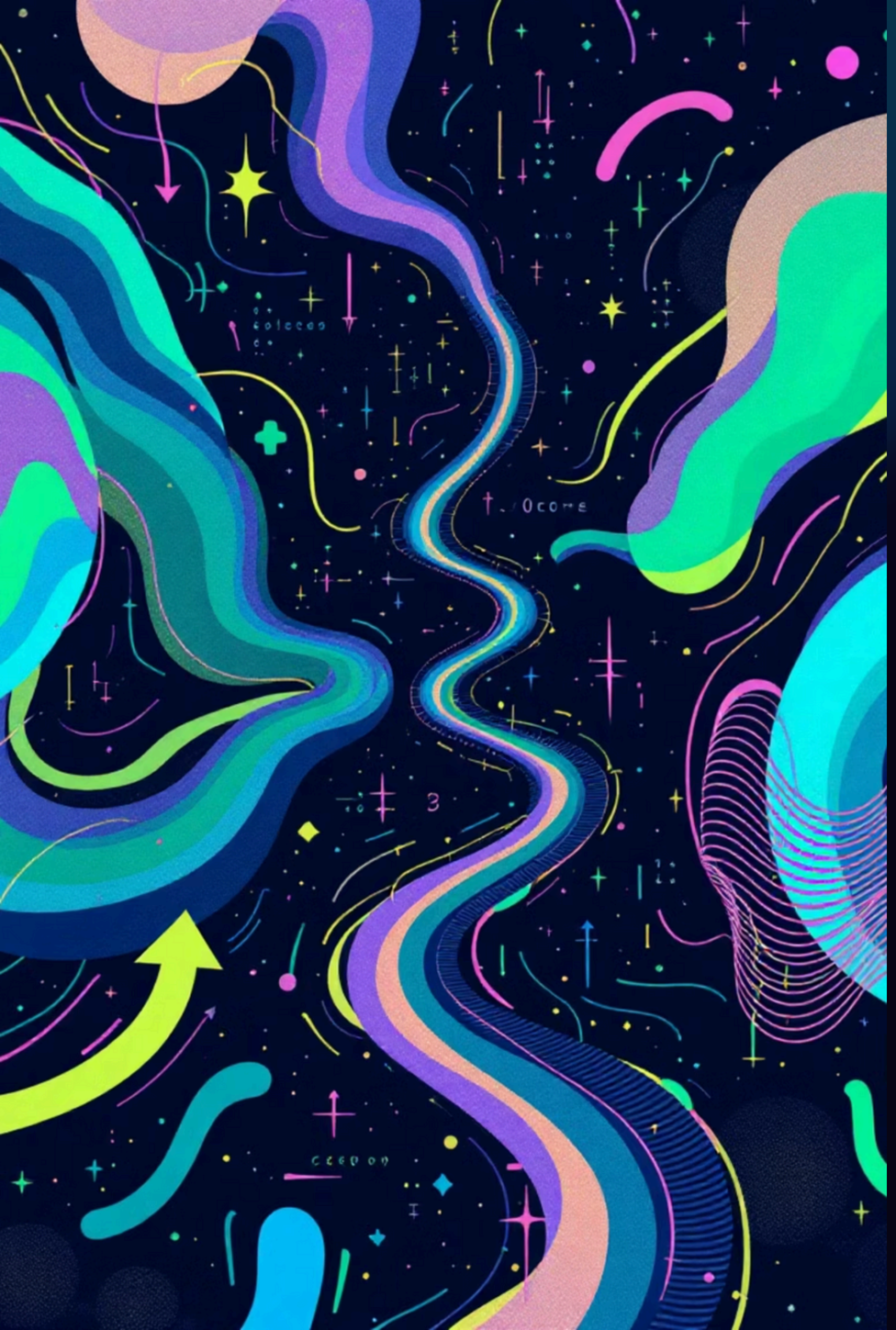
Practical Example

```
#include <stdio.h>
int main() {
    int temperature = 30;

    if (temperature > 25) {
        printf("It's hot!\n");
    }

    return 0;
}
```

In this example, the message prints only when the temperature exceeds 25 degrees. Otherwise, nothing happens.



The if-else Statement: Expanding Conditional Logic

What It Does

The if-else statement extends the basic if, allowing your program to choose between two distinct blocks of code. One block executes if the condition is true, and the other executes if it's false.

Syntax Structure

```
if (condition) {
    // code for true
} else {
    // code for false
}
```

This provides a clear alternative path, ensuring your program always has a specific action to take, regardless of the condition.

Practical Example

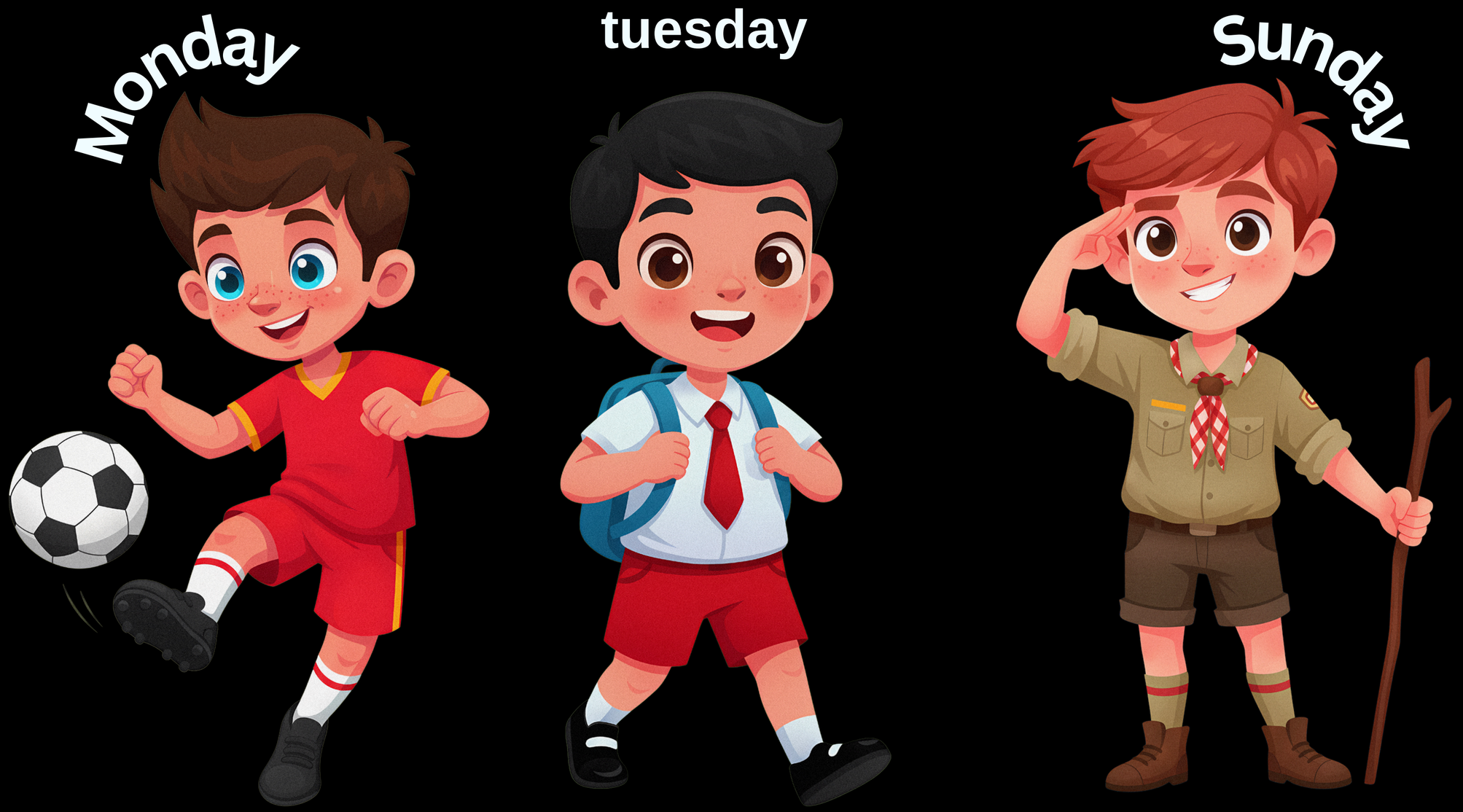
```
#include <stdio.h>
int main() {
    int score = 75;

    if (score >= 60) {
        printf("Pass\n");
    } else {
        printf("Fail\n");
    }

    return 0;
}
```

Here, the program checks if the score is 60 or above. If it is, "Pass" is displayed; otherwise, "Fail" is the result, offering a complete decision outcome.

Switch Case



Switch Case

Less then 30

50%

90% or above

The switch-case Statement: Multiple Choices

How It Works

The switch-case statement provides a more streamlined way to handle multiple possible execution paths based on the value of a single variable or expression. Instead of a long chain of if-else if statements, you can define specific actions for different predefined values.

This structure is highly efficient and improves code readability when dealing with a discrete set of options.

Syntax Structure

```
#include <stdio.h>
int main() {
    int grade = 2;

    switch (grade) {
        case 1:
            printf("Excellent\n");
            break;
        case 2:
            printf("Good\n");
            break;
        case 3:
            printf("Average\n");
            break;
        default:
            printf("Invalid grade\n");
            break;
    }
    return 0;
```

In this example, the code checks the value of the day variable. If it matches "Tuesday" or "Wednesday", "Midweek hustle." is printed. The break statement ensures that only the relevant block of code is executed, preventing "fall-through" to subsequent cases.

Practical Example

```
#include <stdio.h>
#include <string.h> // Required for strcmp()

int main() {
    char day[] = "Tuesday";

    if (strcmp(day, "Monday") == 0) {
        printf("Start of the week.\n");
    }
    // Logic for Tuesday or Wednesday (grouped)
    else if (strcmp(day, "Tuesday") == 0 || strcmp(day, "Wednesday")
== 0) {
        printf("Midweek hustle.\n");
    }
    else if (strcmp(day, "Thursday") == 0) {
        printf("Almost Friday!\n");
    }
    else if (strcmp(day, "Friday") == 0) {
        printf("Weekend loading...\n");
    }
    else {
        printf("Relax, it's the weekend!\n");
    }

    return 0;
}
```

LeetCode : 231. Power of Two

```
0:  0000
1:  0001
2:  0010
3:  0011
4:  0100
5:  0101
6:  0110
7:  0111
8:  1000
9:  1001
10: 1010
```

& : bitwise AND

operator compares bits. If both bits are 1, the result is 1. Otherwise, it is 0.

n & (n - 1)

Scenario A: n is a power of two (e.g., 8)
n = 8 (1000)
n - 1 = 7 (0111)
n & (n - 1) performs 1000 & 0111.
Since there is no position where both numbers have a 1, the result is 0000 (0).
Result: The expression == 0 evaluates to true.

LeetCode : 231. Power of Two

n (Decimal)	n (Binary)	n - 1 (Binary)	n & (n - 1)	Result
4	`0100`	`0011`	`0000`	**true**
8	`1000`	`0111`	`0000`	**true**
6	`0110`	`0101`	`0100`	**false**
7	`0111`	`0110`	`0110`	**false**