Day - 01

What is if else ?

The if-else statement is used to **make decisions** in a program.

It works like this:

- "**If**" a condition is true, do something.
- "**Else**" (if it's not true), do something else.

🔁 General Syntax

Python:

```
if condition:
    # do this
      else:
    # do that
```

C++ / Java:

```
if (condition) {
    // do this
} else {
    // do that
}
```


Question

Write a program to check whether a given number is even or odd using **four different approaches**:

1. **Modulus Operator (%)**
2. **Bitwise AND (&)**
3. **Subtract 2 Again and Again**
4. **Divide and Multiply by 2**

The program should demonstrate all four methods for the same input number.

Input: An integer `n` (can be positive, negative, or zero)
Output: For each method, print `"Even"` or `"Odd"`.
Also, mention the **time complexity**

of each method in comments.

C++ Solution (Beginner-friendly with complexity table)

```
#include <iostream>
using namespace std;

/*
-----------------------------------------------------------
Approach                  | Time Complexity | Space Complexity
-----------------------------------------------------------
1. Modulus Operator (%)   | O(1)            | O(1)
2. Bitwise AND (&)        | O(1)            | O(1)
3. Subtract 2 Loop        | O(n)            | O(1)
4. Divide & Multiply      | O(1)            | O(1)
-----------------------------------------------------------
*/

// 1. Modulus Operator
bool isEven_Modulus(int n) {
    return (n % 2 == 0);
}

// 2. Bitwise AND
bool isEven_Bitwise(int n) {
    return (n & 1) == 0;
}

// 3. Subtract 2 Again & Again
bool isEven_Subtract2(int n) {
    if (n < 0) n = -n; // handle negatives
    while (n > 1) {
        n -= 2;
    }
    return (n == 0);
}

// 4. Divide & Multiply
bool isEven_DivMul(int n) {
    int half = n / 2;
    return (half * 2 == n);
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
```

```
    if (isEven_Modulus(n))
        cout << "Using Modulus Operator: Even" << endl;
    else
        cout << "Using Modulus Operator: Odd" << endl;

    if (isEven_Bitwise(n))
        cout << "Using Bitwise AND: Even" << endl;
    else
        cout << "Using Bitwise AND: Odd" << endl;

    if (isEven_Subtract2(n))
        cout << "Using Subtract 2 Loop: Even" << endl;
    else
        cout << "Using Subtract 2 Loop: Odd" << endl;

    if (isEven_DivMul(n))
        cout << "Using Divide & Multiply: Even" << endl;
    else
        cout << "Using Divide & Multiply: Odd" << endl;

    return 0;
}
```

Java Solution (Beginner-friendly with complexity table)

```
import java.util.Scanner;

/*
-------------------------------------------------------
Approach                  | Time Complexity | Space Complexity
-------------------------------------------------------
1. Modulus Operator (%)   | O(1)            | O(1)
2. Bitwise AND (&)        | O(1)            | O(1)
3. Subtract 2 Loop        | O(n)            | O(1)
4. Divide & Multiply      | O(1)            | O(1)
-------------------------------------------------------
*/

public class EvenOddCheck {
    static boolean isEven_Modulus(int n) {
        return (n % 2 == 0);
    }

    static boolean isEven_Bitwise(int n) {
        return (n & 1) == 0;
    }

    static boolean isEven_Subtract2(int n) {
        if (n < 0) n = -n;
        while (n > 1) {
            n -= 2;
        }
        return (n == 0);
    }

    static boolean isEven_DivMul(int n) {
        int half = n / 2;
        return (half * 2 == n);
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        if (isEven_Modulus(n))
            System.out.println("Using Modulus Operator: Even");
        else
            System.out.println("Using Modulus Operator: Odd");

        if (isEven_Bitwise(n))
            System.out.println("Using Bitwise AND: Even");
        else
            System.out.println("Using Bitwise AND: Odd");

        if (isEven_Subtract2(n))
            System.out.println("Using Subtract 2 Loop: Even");
        else
            System.out.println("Using Subtract 2 Loop: Odd");

        if (isEven_DivMul(n))
            System.out.println("Using Divide & Multiply: Even");
        else
            System.out.println("Using Divide & Multiply: Odd");

        sc.close();
    }
}
```

Python Solution (Beginner-friendly with complexity table)

```
"""
-------------------------------------------------------
Approach                | Time Complexity | Space Complexity
-------------------------------------------------------
1. Modulus Operator (%)   | O(1)            | O(1)
2. Bitwise AND (&)        | O(1)            | O(1)
3. Subtract 2 Loop        | O(n)            | O(1)
4. Divide & Multiply      | O(1)            | O(1)
-------------------------------------------------------
"""

def isEven_Modulus(n):
    return n % 2 == 0

def isEven_Bitwise(n):
    return (n & 1) == 0

def isEven_Subtract2(n):
    n = abs(n)
    while n > 1:
        n -= 2
    return n == 0

def isEven_DivMul(n):
    half = n // 2
    return half * 2 == n
```

```
n = int(input("Enter a number: "))

if isEven_Modulus(n):
    print("Using Modulus Operator: Even")
else:
    print("Using Modulus Operator: Odd")

if isEven_Bitwise(n):
    print("Using Bitwise AND: Even")
else:
    print("Using Bitwise AND: Odd")

if isEven_Subtract2(n):
    print("Using Subtract 2 Loop: Even")
else:
    print("Using Subtract 2 Loop: Odd")

if isEven_DivMul(n):
    print("Using Divide & Multiply: Even")
else:
    print("Using Divide & Multiply: Odd")
```

What is a Loop?

A **loop** is used to **repeat a block of code** multiple times.

Instead of writing the same code again and again, you **use a loop** to do it automatically.

☑ Types of Loops:

1. **For Loop** – repeats a block a specific number of times
2. **While Loop** – repeats while a condition is true
3. **Do-While Loop** (C++/Java only) – like while, but runs **at least once**

General Syntax

Python - for and while:

```
# For loopfor i in range(1, 6):
    print(i)
# While loop
i = 1while i <= 5:
```

```
    print(i)
    i += 1
```

C++ / Java – for, while, do-while:

```
// For loopfor (int i = 1; i <= 5; i++) {
    cout << i;
}
// While loopint i = 1;while (i <= 5) {
    cout << i;
    i++;
}
// Do-while loopint i = 1;do {
    cout << i;
    i++;
} while (i <= 5);
```

Question 1

Write a program to check whether a given number is a **prime number** using the **brute force method**.
A prime number is a number greater than 1 that has no divisors other than 1 and itself.

Input: An integer `n`
Output: `"Prime"` if the number is prime, otherwise `"Not Prime"`.
Approach: Check divisibility by all numbers from `2` to `n-1`.

Approach 2 // Logic (Brute Force)

1. If `n <= 1` → Not Prime (because prime numbers start from 2)
2. Loop from `i = 2` to `n-1`
3. If `n % i == 0` → Not Prime (found a divisor)
4. If no divisor found → Prime

Time Complexity: O(n)
Space Complexity: O(1)

C++ Code (Beginner-friendly)

```cpp
#include <iostream>
using namespace std;

/*
Brute Force Prime Check
Time Complexity: O(n)
Space Complexity: O(1)
*/

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    bool isPrime = true;

    if (n <= 1) {
        isPrime = false; // Numbers <= 1 are not prime
    } else {
        for (int i = 2; i < n; i++) {
            if (n % i == 0) { // Found a divisor
                isPrime = false;
                break;
            }
        }
    }

    if (isPrime)
        cout << "Prime" << endl;
    else
        cout << "Not Prime" << endl;

    return 0;
}
```

Java Code (Beginner-friendly)

```java
import java.util.Scanner;

/*
Brute Force Prime Check
Time Complexity: O(n)
Space Complexity: O(1)
*/

public class PrimeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        boolean isPrime = true;

        if (n <= 1) {
            isPrime = false; // Numbers <= 1 are not prime
        } else {
            for (int i = 2; i < n; i++) {
                if (n % i == 0) { // Found a divisor
                    isPrime = false;
```

```
                    break;
                }
            }
        }

        if (isPrime)
            System.out.println("Prime");
        else
            System.out.println("Not Prime");

        sc.close();
    }
}
```

Python Code (Beginner-friendly)

```
"""
Brute Force Prime Check
Time Complexity: O(n)
Space Complexity: O(1)
"""

n = int(input("Enter a number: "))
is_prime = True

if n <= 1:
    is_prime = False  # Numbers <= 1 are not prime
else:
    for i in range(2, n):
        if n % i == 0:  # Found a divisor
            is_prime = False
            break

if is_prime:
    print("Prime")
else:
    print("Not Prime")
```


Approach 2// Logic (Optimized)

1. If `n <= 1` → Not Prime
2. Check divisibility only from **2** to **√n** instead of `n-1`.
    - Because if `n` has a factor greater than √n, the corresponding smaller factor would have already been found earlier.
3. If divisible by any number in that range → Not Prime, else Prime.

Time Complexity: O(√n)
Space Complexity: O(1)

C++ (Optimized Prime Check)

```cpp
#include <iostream>
#include <cmath>
using namespace std;

/*
Optimized Prime Check
Time Complexity: O(sqrt(n))
Space Complexity: O(1)
*/

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    bool isPrime = true;

    if (n <= 1) {
        isPrime = false; // Numbers <= 1 are not prime
    } else {
        for (int i = 2; i <= sqrt(n); i++) {
            if (n % i == 0) { // Found a divisor
                isPrime = false;
                break;
            }
        }
    }

    if (isPrime)
        cout << "Prime" << endl;
    else
        cout << "Not Prime" << endl;

    return 0;
}
```

Java (Optimized Prime Check)

```java
import java.util.Scanner;

/*
Optimized Prime Check
Time Complexity: O(sqrt(n))
Space Complexity: O(1)
*/

public class PrimeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        boolean isPrime = true;

        if (n <= 1) {
            isPrime = false;
```

```
        } else {
            for (int i = 2; i <= Math.sqrt(n); i++) {
                if (n % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime)
            System.out.println("Prime");
        else
            System.out.println("Not Prime");

        sc.close();
    }
}
```

Python (Optimized Prime Check)

```
"""
Optimized Prime Check
Time Complexity: O(sqrt(n))
Space Complexity: O(1)
"""

import math

n = int(input("Enter a number: "))
is_prime = True

if n <= 1:
    is_prime = False
else:
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            is_prime = False
            break

if is_prime:
    print("Prime")
else:
    print("Not Prime")
```

☑ Now you have:

- **Brute Force:** O(n) (checks all numbers up to n-1)
- **Optimized:** O(√n) (checks only up to square root)