HASH AGILE

PROBLEM STATEMENT:

Write a program to generate and print the first n rows of Pascal's triangle without using built-in math or array functions. For n = 5, the output should be:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

PROGRAM CODE:

```
function generatePascalsTriangle(n: number): void {
  let previousRow: number[] = []; // To store the previous row values

  for (let i = 0; i < n; i++) {
    let currentRow: number[] = new Array(i + 1).fill(1); // Initialize the current row with 1s

    // Compute intermediate values based on the previous row
    for (let j = 1; j < i; j++) {
      currentRow[j] = previousRow[j - 1] + previousRow[j];
    }

    // Convert row to a string with spaces between elements
    const rowString = currentRow.join(" ");

    // Calculate leading spaces to center the row
    const padding = " ".repeat(n - i - 1);

    // Print the formatted row
    console.log(padding + rowString);

    // Update the previous row for the next iteration
    previousRow = currentRow;
  }
}

// Call the function with n = 5
generatePascalsTriangle(5);
```

EXPLANATION OF THE CODE:

1. Function Definition

function generatePascalsTriangle(n: number): void {

This function generates and prints the first n rows of Pascal's Triangle.

2. Initialization of previousRow

let previousRow: number[] = [];

previousRow is used to store the values of the previous row in the triangle. It helps compute the current row iteratively.

3. Outer Loop (Rows)

for (let i = 0; i < n; i++) {

Iterates through each row from 0 to n−1n-1n−1. i represents the row index.

4. Creating the Current Row

let currentRow: number[] = new Array(i + 1).fill(1);

Initializes a new row with i + 1 elements, all set to 1.
The first and last elements of each row in Pascal's Triangle are always 1.

5. Filling Intermediate Values

```
for (let j = 1; j < i; j++) {
   currentRow[j] = previousRow[j - 1] + previousRow[j];
}
```

For rows beyond the first two, intermediate values are calculated based on the sum of two elements from the previous row.

6. Formatting the Output

```
const rowString = currentRow.join(" ");
const padding = " ".repeat(n - i - 1);
console.log(padding + rowString);
currentRow.join(" ")
```

converts the row into a string with numbers separated by spaces.
padding adds spaces before the row to center-align it. The number of spaces decreases with each row. The row is printed with the padding added to the beginning.

7. Updating previousRow

previousRow = currentRow;

The currentRow becomes the previousRow for the next iteration.

COMPLEXITY ANALYSIS

Time Complexity:

The outer loop runs nnn times, and the inner loop computes each row's values with a total of **O(n^2)2n×(n−1)=O(n2),** including string formatting.

Space Complexity:

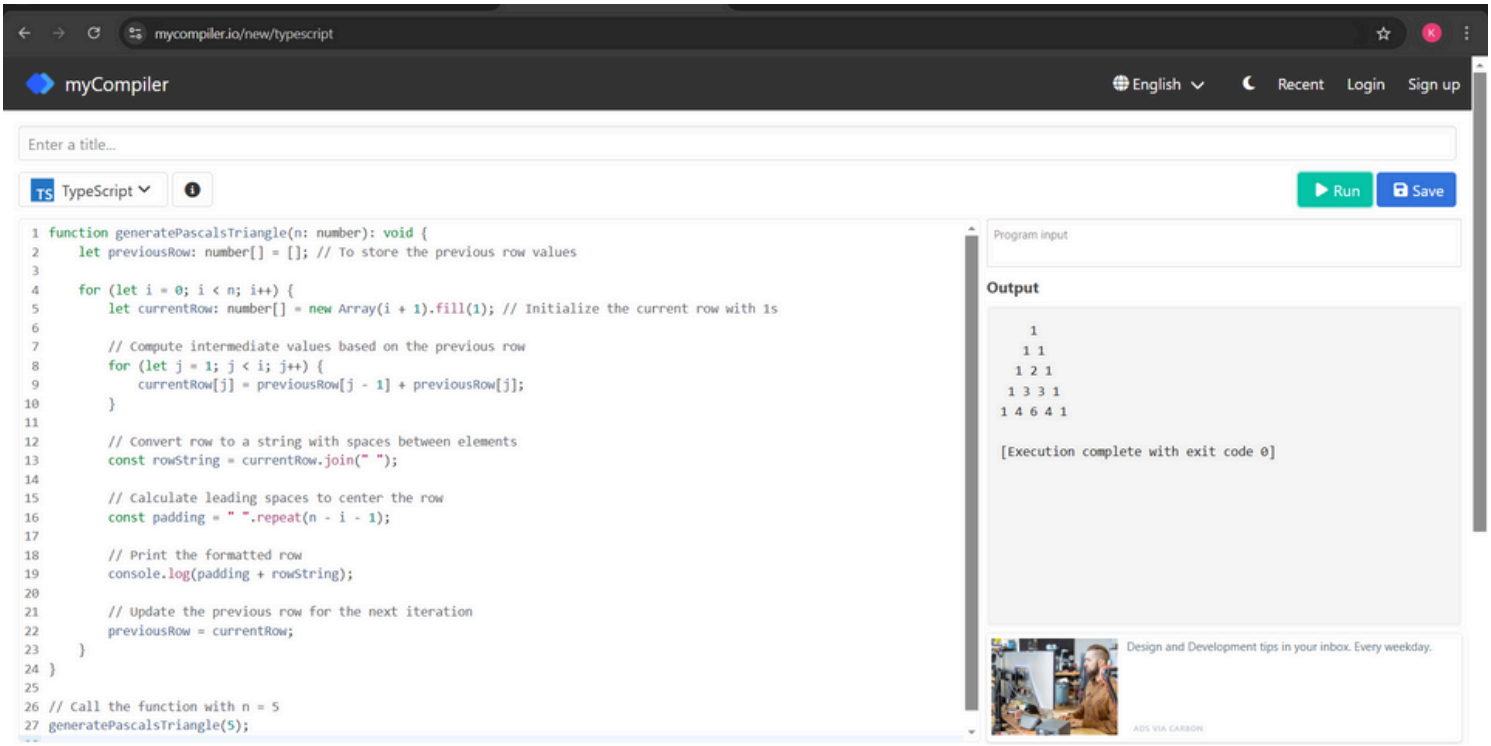
The algorithm uses O(n) space for two arrays (current and previous rows) and padding.

OUTPUT

INPUT : n = 5

OUTPUT :

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```



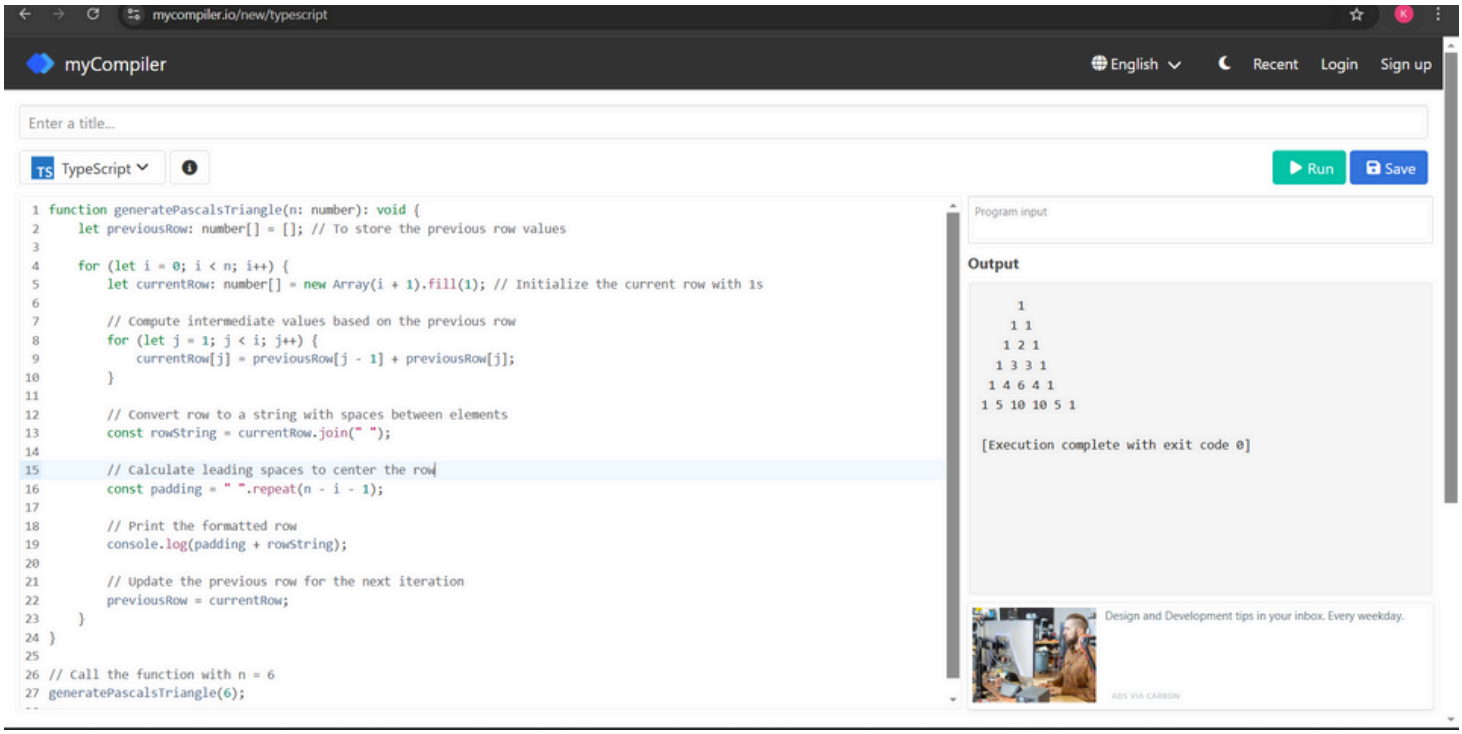
SAMPLE OUTPUT 1 :

INPUT : n = 6

OUTPUT :

```
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
1 5 10 10 5 1
```

SCREENSHOT :



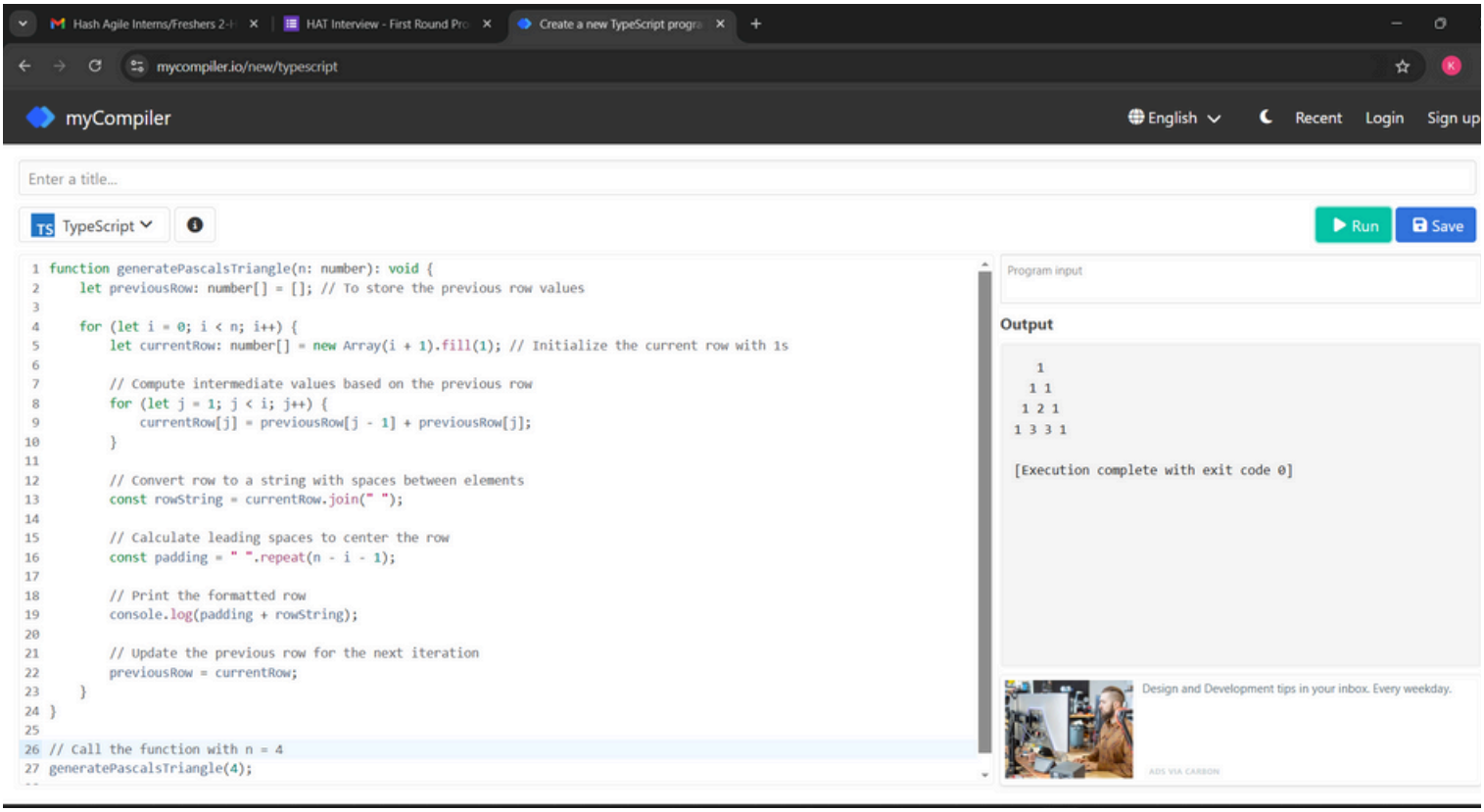
SAMPLE OUTPUT 2 :

INPUT : n = 4

OUTPUT :

```
   1
  1 1
 1 2 1
1 3 3 1
```

SCREENSHOT :



SAMPLE OUTPUT 3 :

INPUT : n = 7

OUTPUT :

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
```

SCREENSHOT :