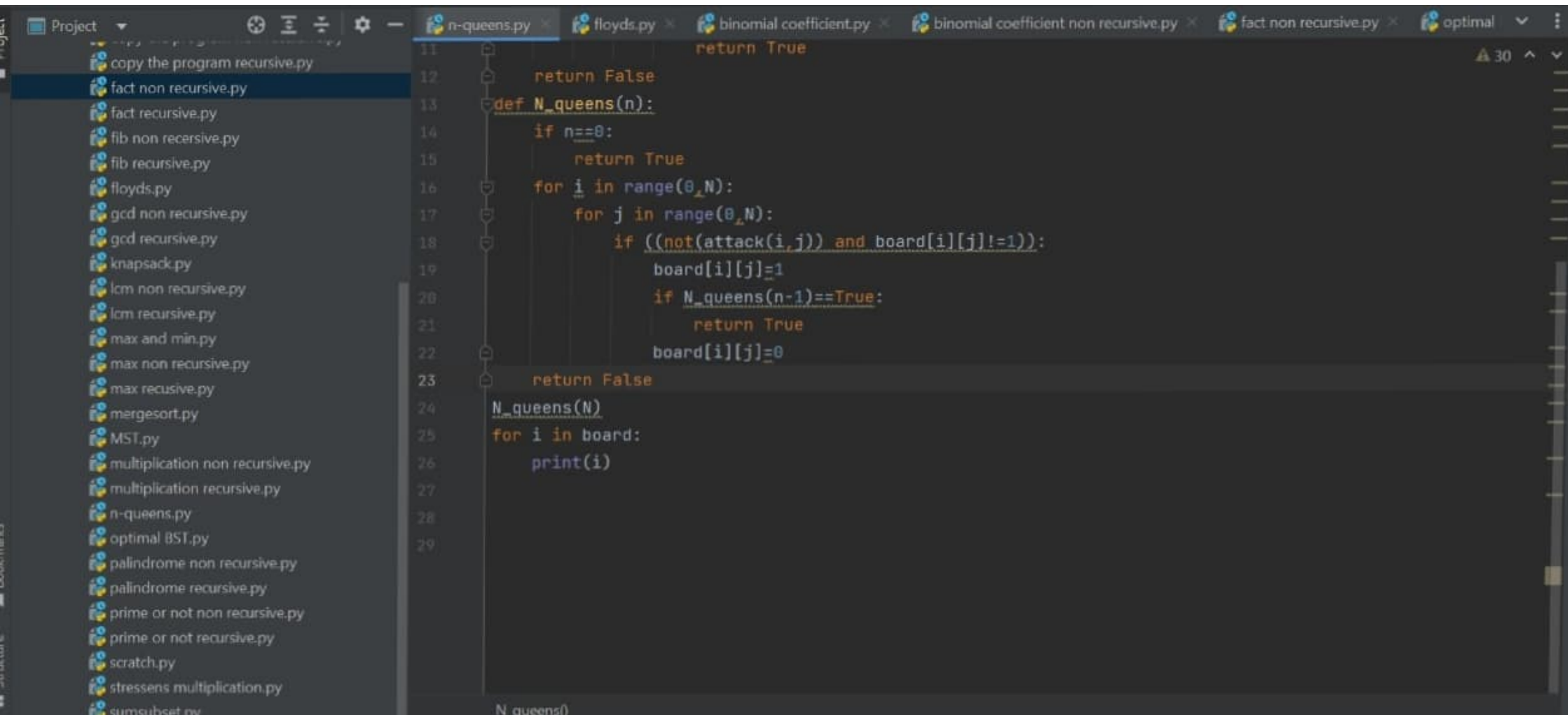


The screenshot shows a code editor with a project sidebar on the left and a main code area on the right. The sidebar lists various Python files, including 'n-queens.py', 'floyds.py', 'binomial coefficient.py', 'binomial coefficient non recursive.py', 'fact non recursive.py', and 'optimal'. The main code area displays the implementation of the N-Queens problem in Python. The code is as follows:

```
1 N=int(input("enter the number: "))
2 board=[[0]*N for f in range(N)]
3 def attack(i,j):
4     for k in range(0,N):
5         if board[i][k]==1 or board[k][j]==1:
6             return True
7     for k in range(0,N):
8         for l in range(0,N):
9             if (k+l==i+j) or (k-l==i-j):
10                if board[k][l]==1:
11                    return True
12     return False
13 def N_queens(n):
14     if n==0:
15         return True
16     for i in range(0,N):
17         for j in range(0,N):
18             if ((not(attack(i,j)) and board[i][j]!=1)):
19                 board[i][j]=1
20                 if N_queens(n-1)==True:
21                     return True
22                 board[i][j]=0
23     return False
24 N=N_queens(N)
25 N_queens()
```

The code implements the N-Queens problem using a recursive approach. It starts by taking an input 'N' and initializing a board of size N x N. The 'attack' function checks if a queen at position (i, j) is under attack by any other queens on the board. The 'N\_queens' function uses a recursive approach to place queens on the board, returning True if a solution is found and False otherwise.





The image shows a code editor with a project sidebar on the left and a main code area on the right. The sidebar lists various Python projects, including 'copy the program recursive.py', 'fact non recursive.py', 'fact recursive.py', 'fib non recursive.py', 'fib recursive.py', 'floyds.py', 'gcd non recursive.py', 'gcd recursive.py', 'knapsack.py', 'lcm non recursive.py', 'lcm recursive.py', 'max and min.py', 'max non recursive.py', 'max recursive.py', 'mergesort.py', 'MST.py', 'multiplication non recursive.py', 'multiplication recursive.py', 'n-queens.py', 'optimal BST.py', 'palindrome non recursive.py', 'palindrome recursive.py', 'prime or not non recursive.py', 'prime or not recursive.py', 'scratch.py', 'stressens multiplication.py', and 'sumsubset.py'. The main code area displays the code for the N-queens problem, which is a recursive function that checks if a queen can be placed at a given position and then recursively checks for the remaining queens.

```
11 return True
12 return False
13 def N_queens(n):
14     if n==0:
15         return True
16     for i in range(0,N):
17         for j in range(0,N):
18             if ((not(attack(i,j)) and board[i][j]!=1)):
19                 board[i][j]=1
20                 if N_queens(n-1)==True:
21                     return True
22                 board[i][j]=0
23     return False
24 N_queens(N)
25 for i in board:
26     print(i)
27
28
29
```



File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject2 - C:\Users\kativ\AppData\Roaming\JetBrains\PyCharmCE2022.1\scratches\n-queens.py

Scratches > n-queens.py

Project

- copy the program recursive.py
- fact non recursive.py
- fact recursive.py
- fib non recursive.py
- fib recursive.py
- floyds.py
- gcd non recursive.py
- gcd recursive.py
- knapsack.py

n-queens.py

```
25 for i in board:
26     print(i)
27
28
29
```

N\_queens()

Run: n-queens

```
C:\Users\kativ\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:/Users/kativ/AppData/Roaming/JetBrains/PyCharmCE2022.1/scratches/n-queens.py
enter the number: 4
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]

Process finished with exit code 0
```

