## 6. A Profusion of Queens

To loop is human, to recurse divine. – Author Unknown.

Programming constructs and algorithmic paradigms covered in this puzzle: Recursive procedures. Exhaustive search through recursion.

Having solved the 8-queens problem, we turn our attention to solving the N-queens problem for arbitrary N. That is, we need to place N queens on an  $N \times N$  board such that no pair of queens attack each other.

Let's now assume that we are not allowed to write nested **for** loops (or other types of loops) that have a degree of nesting more than 2. You might say that this is an artificial constraint, but not only is the deeply nested 8-queens code aesthetically displeasing, but the code is also not general. If you wanted to write a program to solve the N-queens problem for N up to say 20, you would have to write functions to solve 4-queens (with 4 nested loops), 5-queens (with 5 nested loops), all the way to 20-queens (with 20 nested loops!), and invoke the appropriate function depending on the actual value of N when the code is run. What happens if you then want a solution to the 21-queens problem?

We will need to use recursion to solve the general N-queens problem. Recursion occurs when something is defined in terms of itself. The most common application of recursion in programming is where a function being defined is applied within its own definition.

In Python a function can call itself. If a function calls itself, it is called a recursive function. Recursion may also correspond to a function A calling a function B, which in turn calls function A. We'll focus on the simple case of recursion here, namely a function f calling f again.

## **Recursive Greatest Common Divisor**

What exactly happens when a function f calls itself? Surprisingly, this is not very different from the function f calling a different function g from an execution standpoint. Let's look at a simple case of recursion where we are computing the greatest common divisor (GCD) of a number. We can easily do this iteratively using the Euclidean Algorithm as shown below.

1.	<pre>def iGcd(m, n):</pre>
2.	while n > 0:
3.	m, n = n, m % n
4.	return m

Here's recursive code for GCD with equivalent functionality:

```
1. def rGcd(m, n):
2. if m % n == 0:
3. return n
4. else:
5. gcd = rGcd(n, m % n)
6. return gcd
```

We make two key observations:

- 1. rGcd does not call itself in every case there is a base case where if m % n == 0, then rGcd returns n and does not call itself. This corresponds to Lines 2 and 3 above.
- 2. The other observation is that the arguments to the rGcd invocation inside of rGcd the callee rGcd on Line 5 are different from the arguments of the caller rGcd. Over two recursive calls, i.e., rGcd calling rGcd calling rGcd, the arguments of the third call will be smaller than the arguments of the first.

Together these two observations ensure that rGcd terminates. If a function calls itself with exactly the same argument(s) then assuming there is no global state being modified and tested, we will end up with an infinite loop, i.e., a non-terminating program. We will also create a non-terminating program if we do not have a base case with no recursive call.

The execution of rGcd(2002, 1344) is shown below illustrating the called procedures.

```
rGcd(2002, 1344) (Line 5 call)

→ rGcd(1344, 658) (Line 5 call)

→ rGcd(658, 28) (Line 5 call)

→ rGcd (28, 14) (returns on Line 3)

rGcd(658, 28) (returns on Line 6)

rGcd(1344, 658) (returns on Line 6)

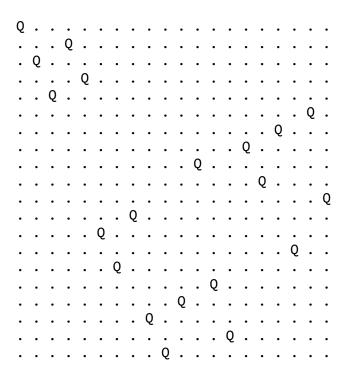
rGcd(2002, 1344) (returns on Line 6)
```

The indentation reflects the recursive calls.

Can you write code for a recursive algorithm that solves N-queens?

## Exercises

**Exercise 1**: Modify the nQueens code to pretty print an actual two-dimensional board as shown below with the solution obtained by nQueens(20). A . signifies an empty square on the board, and a Q signifies a queen. There is a space between each pair of .'s.



**Puzzle Exercise 2**: Modify the nQueens code so it looks for solutions with a queen already placed in a list of locations and prints one if it exists. You can use a 1-D list location that has non-negative entries for certain columns that correspond to fixed queen positions. For example, location = [-1, -1, 4, -1, -1, -1, 0, -1, 5] has three queens placed in the 3<sup>rd</sup>, 8<sup>th</sup> and 10<sup>th</sup> columns for a 10 × 10 board. Your code should produce the solution shown below that is consistent with the specified locations:

•	•	•	•	•	•	•	Q	•	•
•	•	•	•	Q	•	•	•	•	•
•	-					•			
•	•	•	•	•	•	•	•	Q	•
•	•	Q	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	Q
•	•	•	•	•	•	Q	•	•	•
•	•	•	Q	•	•	•	•	•	•
•	•	•	•	•	Q	•	•	•	•
Q	•	•	•	•	•	•	•	•	•

**Exercise 3**: A palindrome is a string that reads the same front to back and back to front. For example, kayak and racecar are palindromes. Write a recursive function using list splicing that determines whether an argument string is a palindrome or not. Your procedure should ignore the case of letters, i.e., it should report that 'kayaK' is a palindrome.

MIT OpenCourseWare https://ocw.mit.edu

6.S095 Programming for the Puzzled January IAP 2018

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.