1 Problem Statement

Using Python’s turtle graphics module, we would like to design a program that draws a rectangular spiral oriented inward. This spiral should have twenty segments. The leftmost segment should be twenty units long, and each segment should be one unit shorter than the one before it.

We will do the following:

- Write code to solve the problem.
- Discuss how to test the solution.
- Draw an execution diagram of the code.
- Discuss the special property of our code: it is tail-recursive.
2 Analysis and Solution Design

We will use the method of combining to tease out the spiral algorithm. While the problem asks for a spiral with twenty segments, we will generalize, and allow the number of segments to be provided as a parameter. We will understand the length of the leftmost segment to be the same as the number of segments in the spiral.

- Consider the case when the number of segments is zero. In that case there is nothing to draw!

The code to draw this figure is the following.

```python
def draw_spiral0():
    pass
```

- Consider the case when the number of segments is one. In that case, all we need to draw is a single straight line.

The code to draw this figure is the following.

```python
def draw_spiral1():
    forward(1)
```

- Consider the case when the number of segments is two. In that case, in addition to going forward two units, we must turn and go forward one unit. Observe that going forward one is the same as doing `draw_spiral1`. 
The code to draw this figure is the following.

```python
def draw_spiral2():
    forward(2)
    right(90)
    draw_spiral1()  # which is the same as Move the turtle forward 1
```

- Consider the case when the number of segments is three. Here, after going forward three units, we need the upside-down ‘L’-shape that we got from `draw_spiral2`.

The code to draw this figure is the following.

```python
def draw_spiral3():
    forward(3)
    right(90)
    draw_spiral2()
```

**Algorithm**

Cases zero and one look different from cases two and three. In fact, we can make case one look the same as cases two and three by adding a turn and a call to `draw_spiral10`. Such a change does not alter what the line looks like since there is no post-condition for the turtle orientation, and `draw_spiral10` does nothing. Thus we have merely two distinct cases: when the length is zero, and when the length is positive.

The code for a spiral with the number of segments given as a parameter is the following.
```python
def draw_spiral(segments):
    if segments == 0:
        pass
    else:
        forward(segments)
        right(90)
        draw_spiral(segments - 1)
```

**Implementation**

The Python code can be found in the file `hypnotized.py`.

3 Testing (Test Cases, Procedures, etc.)

Testing involves verifying the following cases.

- The program draws the image as specified.
- The function `draw_spiral` yields the following results on the following inputs:
  - There should be a blank canvas when `segments` is zero.
  - There should be a single line when `segments` is one.
  - There should be the specified spiral minus the last segment when `segments` is 19.

4 Execution Diagram and the Concept of Tail-Recursion

4.1 Execution Diagram

We can visualize the execution of the `draw_spiral` function with the following execution diagram.
4.2 Tail-Recursion

Examine the execution diagram. In contrast to the recursion we’ve seen before, there is nothing more to execute after the function makes the recursive call. There are no statements to execute after the recursive call box. Rather, we are simply executing the code over and over again with different values for the parameter segments.

In general, within a function, if there is nothing more to do after a call, that call is referred to as a tail-call\(^1\). If such a call is recursive, the call and the function that contains the call is referred to as tail-recursive.

---

1. Python does not distinguish tail-calls from non-tail-calls.