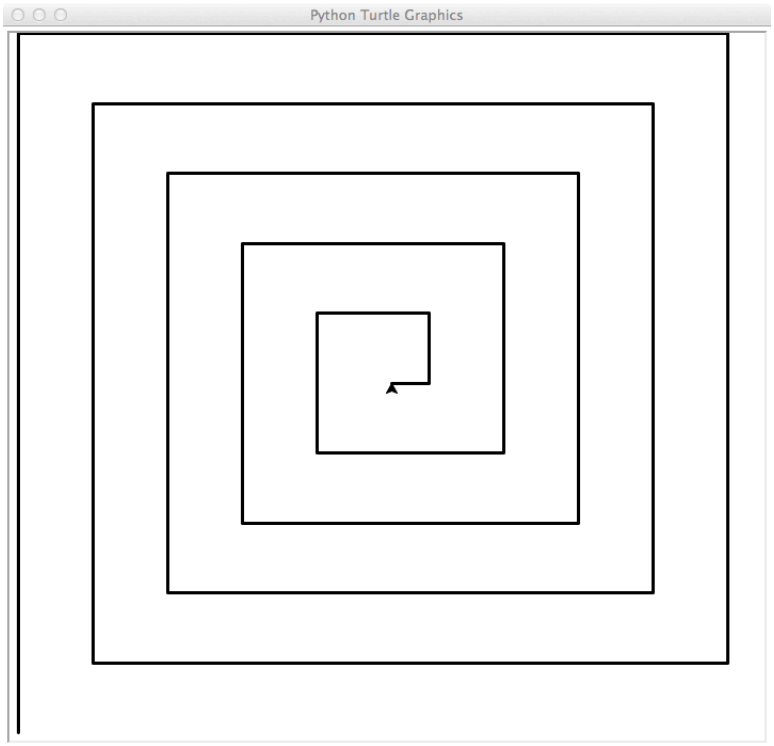
Computer Science I Hypnotized

CSCI141 Lecture (1/2)

09/13/2013

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.



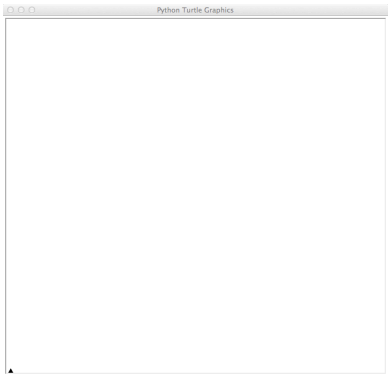
We will do the following.

- Write code to solve the problem.
- Discuss how to test the solution.
- Draw an execution diagram of the code.
- Observe that our code has a special property: it is *tail-recursive*.
- Introduce the techniques of *assignment* and *iteration* as an alternative to tail-recursion.

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.

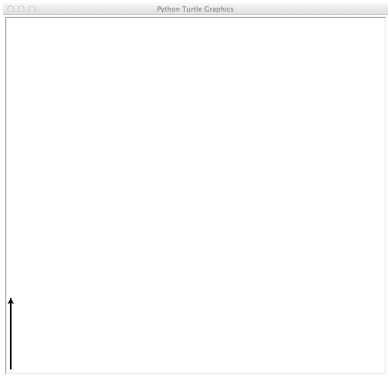
- Consider the case when the number of segments is zero. In that case there is nothing to draw!



The code to draw this figure is the following.

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

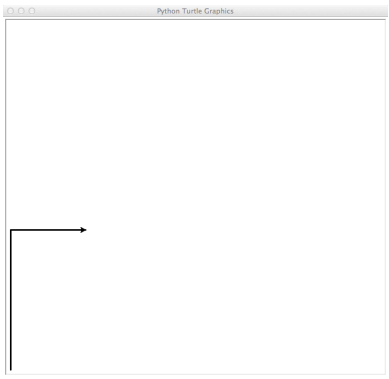
- Consider the case when the number of segments is one. In that case, all we need to draw is a single straight line.



The code to draw this figure is the following.

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

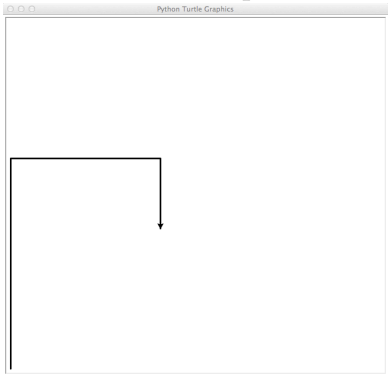
- Consider the case when the number of segments is two. In that case, in addition to going forward two units, we must turn and go forward one unit. Observe that going forward one is the same as doing `draw_spiral1`.



The code to draw this figure is the following.

```
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.

```
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_spiral0`. Such a change does not alter what the line looks like since there is no post-condition for the turtle orientation, and `draw_spiral0` 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.

```
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.)

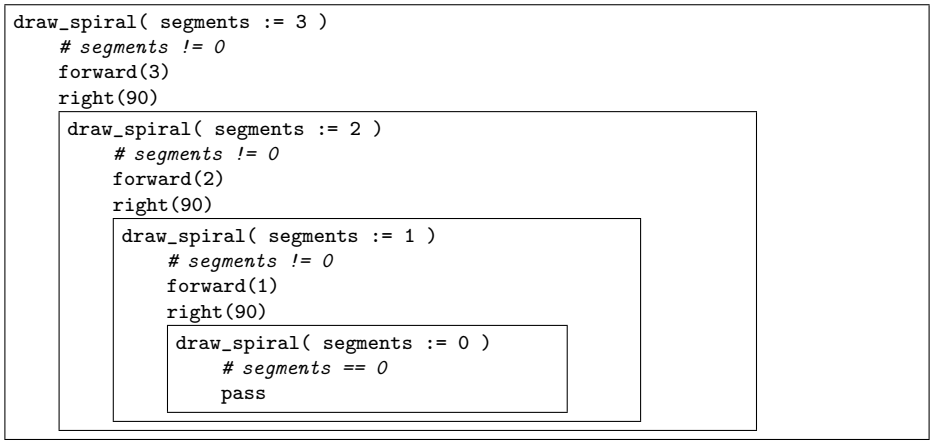
Testing involves verifying the following cases.

- 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

We particularly notice in the execution diagram, in contrast to the recursion we've see before, that there is nothing more to do after 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`. Thus we can legitimately express the code above as follows.

```
define draw_spiral(segments):
    repeatedly execute until we are done:
        if segments == 0:
            we are done
        else:
            move the turtle forward segments
            turn right 90 degrees
            change segments to the value of segments - 1
```

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*. Tail-recursion can always be understood as *iteration* or repetition.

5 Assignment

The algorithm in the previous sections leaves us with a puzzle: how can we change a parameter in Python, and what does it mean to change the parameter?

The statement `change segments to the value of segments - 1` is expressed succinctly in Python as `segments = segments - 1`. However, the Python statement may be even more confusing! It looks like an equation, but it's not. It cannot be an equation, because the equation $s = s - 1$ implies $0 = -1$, which is a contradiction. The variable `segments` cannot and does not refer to a number. But up until now, it has appeared that variables referred to numbers. If variables don't refer to numbers, what do they refer to?

A variable refers to an *address*; an address is a name for a location in computer memory. Python keeps track of which variable is associated with which address in a table. The computer memory itself also resembles a table. Thus the following sequence of assignments leads to the following characterization involving the variable table and memory.

```
x = 15
y = 18
z = 20
```

Variable Table
`x` : a_0
`y` : a_1
`z` : a_2

Memory	
a_0	15
a_1	18
a_2	20

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

Most expressions, including the expression on the right-hand side of an assignment, evaluate variables by looking up the variable in the variable table, finding its associated address, and then using the address to look at the appropriate place in memory to find the value. In contrast to a mathematical equation, the left-hand side of an assignment is treated differently from the right-hand side. When evaluating the variable on the left-hand side, the assignment looks up only the address. Then the contents of that address is replaced with the value the right-hand side evaluates to. Thus, when executing `x = x + z`, the right-hand side evaluates to 35, the left-hand side evaluates to a_0, 35 becomes the contents of a_0, and we have the following characterization involving the variable table and memory.

Variable Table
`x` : a_0
`y` : a_1
`z` : a_2

Memory	
a_0	35
a_1	18
a_2	20

6 An Iteration Construct

The statement from the earlier pseudocode for spiral that says "`repeatedly execute until we are done`" and "`if segments == 0: we are done`" can be translated into actual Python code as `while segments != 0`. Thus the Python function is written as follows.

```
def draw_spiral(segments):
    """ draw_spiral: NatNum -> NoneType
        Draws a line segment of 'segments' units and then turns right 90 degrees
        segments - The number of segments in the spiral
    """
    while segments != 0:
        forward(segments)
        right(90)
        segments = segments - 1
```

This is a looping algorithm instead of a recursive algorithm.

The statement `while` e executes the statements indented below it repeatedly, checking that the expression e evaluates to true before each iteration; the first time e evaluates to false, execution resumes after the loop, i.e., at the same indentation level as the `while`.