CS 171 - Lab 8

Professor Mark W. Boady and Professor Adelaida Medlock

Week 8

Detailed instructions to the lab assignment are found in the following pages.

- Complete all the exercises and type your answers in the space provided.

What to submit:

- Lab sheet as a PDF.
- Screnshots with your code for questions 18, 19 and 20.

Submission must be done via Gradescope

- Please make sure you have tagged all questions and your teammates if any.
- We only accept submissions via Gradescope.

Students' Names: Tony Kabilan Okeke

User ID (abc123): tko35

Possible Points: 88

Your score out of 88:

Lab Grade on 100% scale:

Graded By (TA Signature):

Question 1: 7 points

The Towers of Hanoi puzzle was invented in 1883 by Edouard Lucas. Our program is not about allowing humans to solve the puzzle but about generating solutions. To understand the puzzle itself, go to https://www.mathsisfun.com/games/towerofhanoi.html and play. First try to solve the puzzle with three disks, then four.

(a) (1 point) Did you beat the game for 3 and 4 disks? You win if you can move all the disks in as little moves as possible.

- ○ Yes
- ○ No

FYI: You may need to go back and play the game again to answer some of the questions to come, but you should do so deliberately. Play with the goal of answering questions

(b) (2 points) What limitations do the rules place on moving disks?

- Disks can not be placed on smaller disks
- Only move one disk at a time
- For each move, take the top disk in one stack and place it on top of a different stack

If the three towers are labeled A, B, and C, here is a solution for two disks:

$$A \rightarrow B$$
$$A \rightarrow C$$
$$B \rightarrow C$$

(c) (2 points) What is a solution for three disks? (Write in the same notation as the above example.)

$A \rightarrow C$
A → B
C → B
A → C
B → A
B → C
A → C

(d) (2 points) What is a solution for four disks?

A → B
A → C
B → C
A → B
C → A
C → B
A → B
A → C
B → C
B → A
C → A
B → C
A → B
A → C
B → C

Question 2: 2 points

Is there any pattern to the sequence of moves in a solution? Make your best guess.

First, shift all but the bottom disk from A to B (shuffle them between A, B and C to get this done), then shift the bottom disk on A to C. Finally, shuffle the remaining disks from B to C.

Question 3: 1 point

Make a Python program with the following function.

```
def hanoi1HardCoded(start, end) :
  print( start + " -> " + end)
```

Execute it to start at peg *A* and end at peg *C*.
How many disks does it solve the Hanoi problem for? 1

Question 4: 1 point

Create the following function.

```
def hanoi2HardCoded(start, spare, end) :
    print( start + " -> " + spare)
    print( start + " -> " + end)
    print( spare + " -> " + end)
```

Does `hanoi2HardCoded("A","B","C")` produce the correct solution? Yes (it solves the problem for 2 disks)

Question 5: 2 points

Create the following function.

```
def hanoi3HardCoded(start, spare , end) :
   print( start + " -> " + end)
   print( start + " -> " + spare)
   print( end + " -> " + spare)
   print( start + " -> " + end)
   print( spare + " -> " + start)
   print( spare + " -> " + end)
   print( start + " -> " + end)
```

Does `hanoi3HardCoded("A", "B", "C")` produce the correct solution? Yes (it solves the problem for 3 disks)

Question 6: 5 points

Answer the following questions about this code

```
def hanoi3HardCoded(start, spare, end) :
  print( start + " -> " + end)
  print( start + " -> " + spare)
  print( end + " -> " + spare)
  print( start + " -> " + end)
  print( spare + " -> " + start)
  print( spare + " -> " + end)
  print( start + " -> " + end)
```

(a) (1 point) What is accomplished by the first three lines in `hanoi3HardCoded`?
The first three lines move the top 2 disks from A to B

(b) (1 point) What is accomplished by the fourth line in `hanoi3HardCoded`?
It moves the bottom disk from A to C

(c) (1 point) What is accomplished by the last three lines in `hanoi3HardCoded`?
They move the 2 disks on B to C (in the right order)

(d) (2 points) What relevance does the largest disk have while the last three lines are solving this subproblem?

The largest disk was moved to peg C by line 4, and can be ignored when solving this subproblem.

Question 7: 2 points

These methods are said to have solutions **hard coded** into them: the computer doesn't so much solve the puzzle as spit out a prerecorded answer. Would this be a good approach for producing, say, a solution for seven disks? Why or why not?

No, a hard coded solution would not be ideal for finding a solution for 7 disks. There would be 127 moves needed to solve such a problem, and it is impractical to write code that long when it could be simplified by using recursion (since the tower of Hanoi has multiple independent subproblems).

Question 8: 1 point

Confirm that the following code works correctly for 1 disk.

```
def hanoi1CallingSimplerMethods ( start, end) :
   print( start + " -> " + end)
```

Does the code work for 1 disk?

- ○ Yes
- ○ No

Question 9: 1 point

Confirm that the following code works correctly for 2 disks.

```
def hanoi2CallingSimplerMethods (start, spare, end):
   hanoi1CallingSimplerMethods (start, spare)
   print(start + " -> " + end)
   hanoi1CallingSimplerMethods (spare, end)
```

Does the code work for 2 disks?

- ○ Yes
- ○ No

Question 10: 1 point

Why does `hanoi2CallingSimplerMethods` call `hanoi1CallingSimplerMethods(start, spare)`?
This line will move the top (smallest) disk from the start peg ("A") to the spare peg ("B")

Question 11: 1 point

Why does `hanoi2CallingSimplerMethods` call `hanoi1CallingSimplerMethods(spare, end)`?
This line will move the (smallest) disk from the spare peg ("B") to the end peg ("C")

Question 12: 1 point

Review the following function.

```
def hanoi3CallingSimplerMethods (start, spare, end):
    hanoi2CallingSimplerMethods (start, end, spare)
    print (start + " -> " + end)
    hanoi2CallingSimplerMethods (spare, start, end)
```

Do you think this function has better or worse readability then `hanoi3HardCoded`. Explain why or why not.

This has better readability than the hardcoded function since it is shorter, and the calls to hanoi2CallingSinplerMethods make the function easier to understand.

Question 13: 7 points

If we call `hanoi2CallingSimplerMethods('A', 'B' , 'C')` the function calls look like

Method	Arguments
hanoi2CallingSimplerMethods	start = A, spare = B, end = C
hanoi1CallingSimplerMethods	start = A, end = B
hanoi1CallingSimplerMethods	start = B, end = C

Draw a similar table to show how the parameters and functions are called when we execute:

`hanoi3CallingSimplerMethods('A', 'B', 'C').`

Method	Arguments
hanoi3CallingSimplerMethods	start = A, spare = B, end = C
hanoi2CallingSimplerMethods	start = A, spare = C, end = B
hanoi1CallingSimplerMethods	start = A, end = C
hanoi1CallingSimplerMethods	start = C, end = B
hanoi2CallingSimplerMethods	Start = B, spare = A, end = C
hanoi1CallingSimplerMethods	start = B, end = A
hanoi1CallingSimplerMethods	start = A, end = C

Question 14: 10 points

(a) (1 point) Does **`spare`** have the same value throughout the call stack? If so, what is it? If not, why not?
No, spare takes on different values in the call stack. This is because, when solving the subproblems, say moving 2 disks from A to B, C would serve as the spare peg. While, when moving from B to C, a would serve as the spare peg. So, the value of spare depends on what the start and end pegs are

(b) (1 point) Does **`end`** have the same value throughout the call stack? If so, what is it? If not, why not?
No, end is set to different values within the call stack. This is because, the end peg is different for the different subproblems

(c) (1 point) What is the same between `hanoi3CallingSimplerMethods` and `hanoi2CallingSimplerMethods`?
Both functions contain the line print(start + " -> " + end) between calls to other Hanoi functions

(d) (1 point) What is different between `hanoi3CallingSimplerMethods` and `hanoi2CallingSimplerMethods`?

Hanoi3CallingSimpleMethods calls hanoi2CalingsimplerMethods, while hanoi2CallingSimplerMethods calls hanoi1CallingSimplerMethods

(e) (5 points) Write a `hanoi4CallingSimplerMethods`. Confirm that it works correctly.

```
In [17]: def hanoi4CallingSimplerMethods(start, spare, end):
    ...:     hanoi3CallingSimplerMethods(start, end, spare)
    ...:     print(start + " -> " + end)
    ...:     hanoi3CallingSimplerMethods(spare, start, end)

In [18]: hanoi4CallingSimplerMethods("A", "B", "C")
A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
A -> C
B -> C
B -> A
C -> A
B -> C
A -> B
A -> C
B -> C
```

(f) (1 point) Would this be a good approach for producing a solution 50 disks? Why or why not?

This would be a band approach for solving 50 disks because it would require 49 other functions (that produce solutions for 49, 48,…,1 disk cases) to already be explicitly defined.

Question 15: 7 points

Add the following function to your Python file.

```
def hanoi (start, spare, end, n):
    if n == 1:
       print(start + " -> " + end)
    else:
       hanoi(start, end, spare, n - 1)
       print(start + " -> " + end)
       hanoi(spare, start, end, n  -1)
```

Modify your code to run

```
hanoi ("A", "B", "C" ,3)
```

(a) (1 point) What does this version of the program print?

It prints the solution to the tower of Hanoi with 3 disks.

(b) (1 point) What does the fourth argument specify?
The number of disks

(c) (1 point) How can you use `hanoi` to produce a solution for four disks?
hanoi("A", "B", "C", 4)

(d) (1 point) What about seven disks?
hanoi("A", "B", "C", 7)

(e) (1 point) How is `hanoi` similar to `hanoi3CallingSimplerMethods`?
It has the line print(start + " -> " + end) between calls to Hanoi with the start, spare, and end arguments shuffled.

(f) (1 point) How is it different?
hanoi() is a recursive function with a base case of print(start + " -> " + end). While hanoi3CallingSimplerMethods is not recursive, rather it makes calls to other, separate, functions that solve smaller problems

(g) (1 point) `hanoi3CallingSimplerMethods` called `hanoi2CallingSimplerMethods` to solve subproblems. What does `hanoi` call?
It calls itselt

Question 16: 3 points

Add the following function to your Python File

```
def hanoi_debug (start, spare, end, n):
    print("start = ", start, "spare = ", spare, "end = " , end)
    if n == 1:
        print(start + " -> " + end)
    else :
        hanoi_debug (start, end, spare, n - 1)
        print(start + " -> " + end)
        hanoi_debug (spare, start, end, n - 1)
```

When we run

```
hanoi_debug ("A", "B", "C", 2)
```

The output is

```
start = A   spare = B   end = C
start = A   spare = C   end = B
A -> B
A -> C
start = B   spare= A   end = C
B -> C
```

What is the output when you run:

```
hanoi_debug ("A", "B", "C", 3)
```

Write it below.

start = A spare = B end = C
start = A spare = C end = B
start = A spare = B end = C
A -> C
A -> B
start = C spare = A end = B
C -> B
A -> C
start = B spare = A end = C
start = B spare = C end = A
B -> A
B -> C
start = A spare = B end = C
A -> C

Question 17: 7 points

(a) (1 point) `hanoi` calls itself to solve easier problems. In what sense are they easier?
The subproblems are easier because they contain fewer disks.

(b) (1 point) For what argument values does `hanoi` *not* recursively call itself?
When n == 1

(c) (2 points) The following version of the code has no base case.

```
def hanoi (start, spare, end, n):
    hanoi (start, end, spare, n - 1)
    print(start + " -> " + end)
    hanoi (spare, start, end, n - 1)
```

What happens when you try to use it? Why?

The python interpreter raises a RecursionError. This is because the function makes infinite calls to itself (with decreasing values of n) and fails to terminate, ultimately resulting in stack overflow.

(d) (3 points) What happens if this version of the code? Why?

```
def hanoi_debug (start, spare, end, n):
    print ("start = ", start, "spare = ", spare, "end = ", end)
    if n == 1:
        print (start + " -> " + end)
    else:
        hanoi_debug (start, end, spare, n)
        print (start + " -> " + end)
        hanoi_debug (spare, start, end, n)
```

This also produces a RecursionError. In this case, it is because the calls to hanoi_debug within the else block are not smaller problems, since the n value is the same. This results in infinite recursive calls to hanoi_debug resulting in stack overflow since there are too many copies of the function in the call stack.

Recursive Mutant Ninja Turtles

Question 18: 10 points

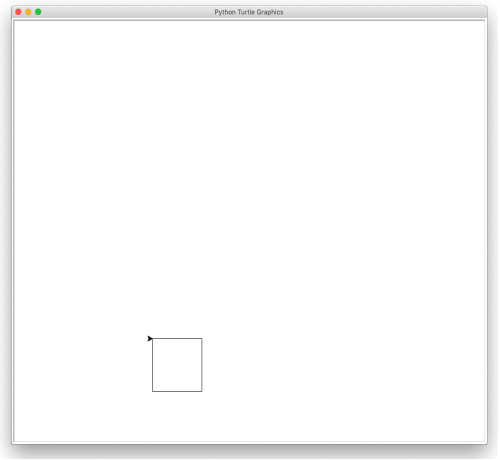
Turtle is a Python library for making simple Drawings. It is included in Thonny.

First, we need to **`import turtle`**

Basics Commands:

- `turtle.penup()` Lift the pen up. Commands will not draw lines, just move the pointer
- `turtle.pendown()` Put the pen down. Commands will draw lines.
- `turtle.forward(x)` Move forward *x* spaces
- `turtle.backward(x)` Move backward *x* spaces
- `turtle.setpos(x, y)` Move to position (*x*, *y*)
- `turtle.right(x)` Turn Right *x* degrees
- `turtle.left (x)` Turn Left *x* degrees

(a) (5 points) Write code to replicate the below image. Submit a screenshot of your code.




```
In [1]: import turtle

In [2]: window = turtle.Screen()
   ...: t = turtle.Turtle()
   ...:
   ...: t.forward(100)
   ...: t.right(90)
   ...: t.forward(100)
   ...: t.right(90)
   ...: t.forward(100)
   ...: t.right(90)
   ...: t.forward(100)
   ...: t.right(90)
```

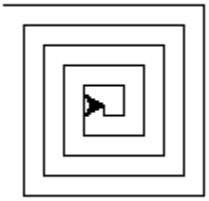
(b) (5 points) Implement the following algorithm pseudocode.

```
function A(lineLen)
   if lineLen > 0 then
      Move Forward lineLen
      Turn Right 90 Degrees
      A(lineLen - 5)
    end if
end function
```

```
In [6]: window = turtle.Screen()
   ...: t = turtle.Turtle()
   ...:
   ...:
   ...: def A(lineLen):
   ...:     if lineLen > 0:
   ...:         t.forward(lineLen)
   ...:         t.right(90)
   ...:         A(lineLen - 5)
```

What does it draw? Submit a screenshot of your code.

It draws a spiral. A(100) is shown here:



Question 19: 5 points

Implement the following algorithm pseudocode. **Hint:** Turn the turtle 90 degrees left before calling this function.

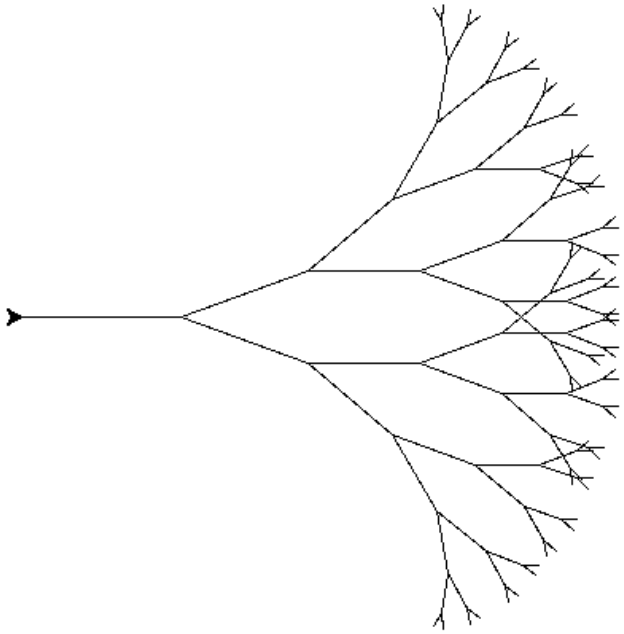
```
function B(line)
  if line > 5 then
     Forward line
     Right 20 degrees
     B(line - 15)
     Left 40 degrees
     B(line - 15)
     Right 20 degrees
     Backward line
  end if
end function
```

```
In [1]: import turtle

In [2]: window = turtle.Screen()
   ...: t = turtle.Turtle()
   ...:
   ...:
   ...: def B(line):
   ...:     if line > 5:
   ...:         t.forward(line)
   ...:         t.right(20)
   ...:         B(line - 15)
   ...:         t.left(40)
   ...:         B(line - 15)
   ...:         t.right(20)
   ...:         t.backward(line)
```

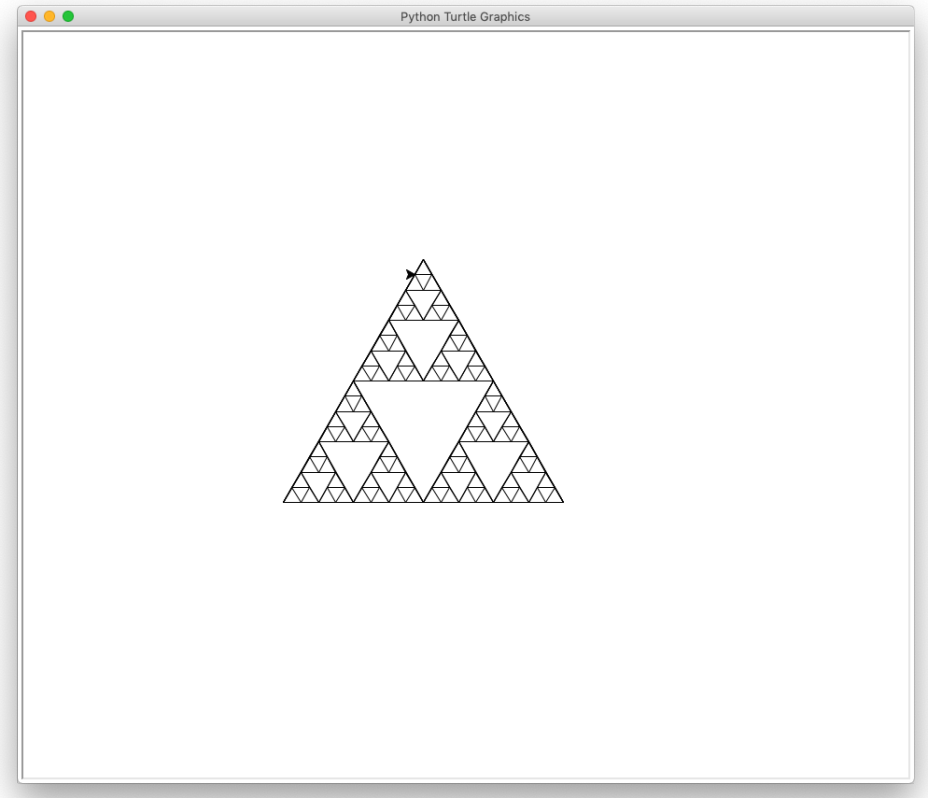
What does it draw? Submit a screenshot of your code.

The function draws a tree branch (with ends lying along an invisible circle).
Here is the output for B(100)



Question 20: 14 points

Create a Sierpinski Triangle.




```
function sierpinski(x,y,s)
   if s > 10 then
      Draw an Equilateral Triangle at (x, y) with length s
      Draw a Sierpinski Triangle at (x, y) with length s/2
      Draw a Sierpinski Triangle at (x + s/2, y) with length s/2
      Draw a Sierpinski Triangle at (x + s/4, y + s√3/4) with length s/2

   end if
end function
```

(a) (10 points) Give your function definition for the Sierpinski Triangle. Submit a screenshot of your code.

```
In [1]: import turtle

In [2]: window = turtle.Screen()
   ...: t = turtle.Turtle()
   ...:
   ...:
   ...: def sierpinski(x, y, s):
   ...:     if s > 10:
   ...:         # Draw equilateral triangle
   ...:         t.penup()
   ...:         t.setpos(x, y)
   ...:         t.pendown()
   ...:         t.forward(s)
   ...:         t.left(120)
   ...:         t.forward(s)
   ...:         t.left(120)
   ...:         t.forward(s)
   ...:         t.left(120)
   ...:         # Draw sierpinski
   ...:         sierpinski(x, y, s / 2)
   ...:         # Draw sierpinski
   ...:         sierpinski(x + s / 2, y, s / 2)
   ...:         # Draw sierpinski
   ...:         sierpinski(x + s / 4, y + (s * 3**0.5) / 4, s / 2)
```

(b) (4 points) How difficult do would it be to draw the Sierpinski Triangle without recursion? Discuss why.

Drawing the Sierpinski triangle without recursion would be a lot more difficult than with recursion. We would have to iterate through a list of all triangle positions and sizes, and draw each triangle with these specifications. This would become increasingly complex as the side length increased, as the number of triangles seems to increase exponentially.