THE UNIVERSITY OF SYDNEY

INFO1110 Week 11 Tutorial

Recursion

Recursion

Recursion is a function that calls itself. This obviously introduces the same problem as infinite loops: code that never halts. To resolve this we try to introduce a general formalism for recursive functions that should ensure that it will eventually return an answer.

Recursive functions are broken into one or more base cases and recursive cases. The base case is a condition under which the recursion stops and some value is returned, while the recursive case is where the function calls itself.

Python applies an arbitrary limit on recursive depth, by default the recursive limit is 1000. It is normally suggested to find a way to implement a loop based solution rather than a recursive solution where possible.

Question 1: A new type of endlessness

Observe what the following code will do:

```
def rec(a):
    return rec(a - 1)
```

What happens when you try to run it?

What about this code?

```
def rec(a):
    if a < 2:
        return a
    else:
        return rec(a - 1) + rec(a - 2)
```

Try running this code for a few low values (less than 10) and then try a higher value (such as 30 or 40). What can you say about the performance of this code?

Question 2: Factorial

Write a recursive factorial function that will compute a factorial number of N.

What is your base case for a factorial function?

```
def rec_factorial(n)
```

Execute your code using the following statements:

```
print(rec_factorial(2)) # 2
print(rec_factorial(3)) # 6
print(rec_factorial(4)) # 24
```

What do you observe if you execute this:

```
print(rec_factorial(1001))
```

Question 3: Binomial Coefficients

Using your factorial function from the previous question. Write a binomial coefficient function (commonly referred to as an **n choose k** function).

Implement it with the following function prototype:

```
def binomial_coefficient(n, k)
```

A binomial coefficient is defined as:

$$\frac{n!}{k!(n-k)!}$$

Question 4: Fibonacci

Write a Python function that prints the n^{th} Fibonnaci number with recursion.

```
def rec_fibonacci(n)
```

The fibonacci recurrence relation is defined as:

$$F_n = F_{n-1} + F_{n-2}$$

The fibonacci sequence is as follows: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$

You will need to consider the first two initial values of this sequence for your base case.

Question 5: Hailstones

Implement the Hailstone problem recursively, record all the numbers it computes to reach 1.

```
def rec_hailstone(n)
```

- If the number is even divide it by two
- If the number is odd multiply it by three and add one.

Test your code with the following statements:

```
print(hailstone(5)) # [5, 16, 8, 4, 2, 1]
```

Question 6: Loop Fibonacci

Convert your recursive fibonacci function to use loops. This is known as an iterative method and you may find it more complex than the recursive method.

```
def fibonacci_iterative(n)
```

- What complexities are associated with this method?
- What are the benefits of implementing it recursive over iteratively?
- Why would we still prefer to implement it iteratively than recursively?

Discuss the assignment

Your tutor will discuss the Assignment 2 (Rover256) specification.

Please ask any questions related about the assignment with your tutor.

Question 7: Tower of Hanoi (Extension)

Tower of Hanoi is a puzzle invented by E. Lucas in 1883. It consists of three rods, and a number of disks of different sizes arranged from largest on the bottom to smallest on the top placed on a rod. The objective of the puzzle is to take minimum number of steps to move the entire stack to another rod with following rules:

- Only one disk can be moved at a time.
- Cannot place a larger disk onto a smaller disk.
- A disk can only be moved if it is the top of the stack.

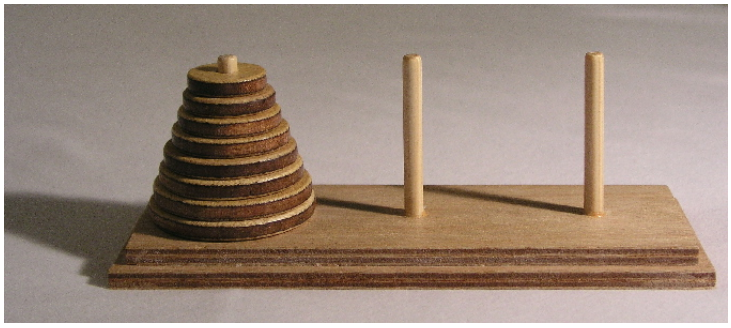

Figure 1: A model set of the Tower of Hanoi with 8 disks (source: Wikipedia)

Write a `hanoi.py` program to move the given number (from command line argument) of disks from one rod to another using recursion.

Question 8: Functional Recursion (Extension)

Try and write a recursive lambda function. What is the problem with the most naıve method? Can you find a way to do this with an unnamed lambda?

Can you define and then immediately execute a lambda function? If so, could we use this directly to create anonymous recursion?