COMP10001 Foundations of Computing
Semester 2, 2016

Tutorial Questions: Week 9

Recursion: (definition) noun. See recursion.

Recursion: formal definition: an algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

Recursive solutions involve two major parts:

1. Base case(s), in which the problem is simple enough to be solved directly
2. Recursive case(s). A recursive case has three components:
   (a) Divide the problem into one or more simpler or smaller parts of the problems,
   (b) Invoke the function (recursively) on each part, and
   (c) Combine the solutions of the parts into a solution for the problem
3. Depending on the problem, any of these may be trivial or complex.

(excerpt from MIT OCW 6.189 IAP 2011: Recursion Notes)

Questions

1. What is the output of the following code:

```
def extremum(numlist, comp=max):
    return comp(numlist)

numlist = [1, 4, 0, -1, 6]
print(extremum(numlist))
print(extremum(numlist, comp=max))
print(extremum(numlist, comp=min))
```

2. Rewrite the following code so that the function is contained in a second file named `header.py`, which is imported into the original file, where the function `tokenise` is then called:

```
def tokenise(text):
    wordlist = []
    for word in text.split():
        wordlist.append(word.strip(".,:;?!'\""))
    return wordlist

text_file = "war-peace.txt"
wordlist = tokenise(open(text_file).read())
```

3. Study the following mysterious functions. For each one, answer the following questions:

   - Which part is the base case?
   - Which part is the recursive case?
   - What does the function do?

   (a)
```
def mystery(x):
    if len(x) == 1:
        return x[0]
    else:
        y = mystery(x[1:])
```

```
        if x[0] > y:
            return x[0]
        else:
            return y
```

(b)
```
def mistero(x):
    a = len(x)
    if a == 1:
        return x[0]
    else:
        y = mistero(x[a/2:])
        z = mistero(x[:a/2])
        if z > y:
            return z
        else:
            return y
```

4. Translate the following recursive function into an iterative function:

```
def permute(lst):
    if lst == []:
        return [[]]
    permuted_lst = []
    for i in range(len(lst)):
        for permuted_rest in permute(lst[0:i]+lst[i+1:]):
            permuted_lst.append([lst[i]] + permuted_rest)
    return permuted_lst
```

5. Translate the following iterative function into a recursive function:

```
def base_change(n,base):
    retval = 0
    n = str(n)[::-1]
    for i in range(len(n)):
        retval += base**i * int(n[i])
    return retval
```