MISE Summer Programming Camp

Class 6 Worksheet - Recursion

This worksheet has 3 sections, which should take you around 1 hour to complete. Section 1 contains a series of short answer questions about what we did in class. You'll have to run some code in your IDE to see the result and write it down. Section 2 is optional and it reviews backtracking, which we saw briefly in class. You'll be guided through it since it doesn't assume knowledge of the topic. Finally on Section 3 you'll have to write some programs and submit them to CodeForces.

1. Review questions

Question 1. The following are all recursive functions. Some of them have cycles, meaning they never terminate, they just keep calling themselves again and again until they crash and the compiler shows an error. Look at each of them and try to predict if they will terminate or not. Then type them in your IDE and run them. Did they match what you expected? If not, try to think about why they don't.

a.

```python
def f(n):
    print(n)
    return f(n-1)
```

b.

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

c.

```python
def rangeList(n):
    if n == 0:
        return []
    else:
        return [n] + rangeList(n - 1)
```

d.

```python
def sum(L):
    if (len(L) == 0):
        return 0
    else:
        Lcopy = L.copy()
        Lcopy.pop(0)
        return L[0] + sum(Lcopy)
```

e.

```python
def reverse(s):
    if (len(s) < 2):
        return s
    else:
        mid = len(s)//2
        return reverse(s[mid:]) + reverse(s[:mid])
```

f. (This example is more advanced so feel free to skip it)

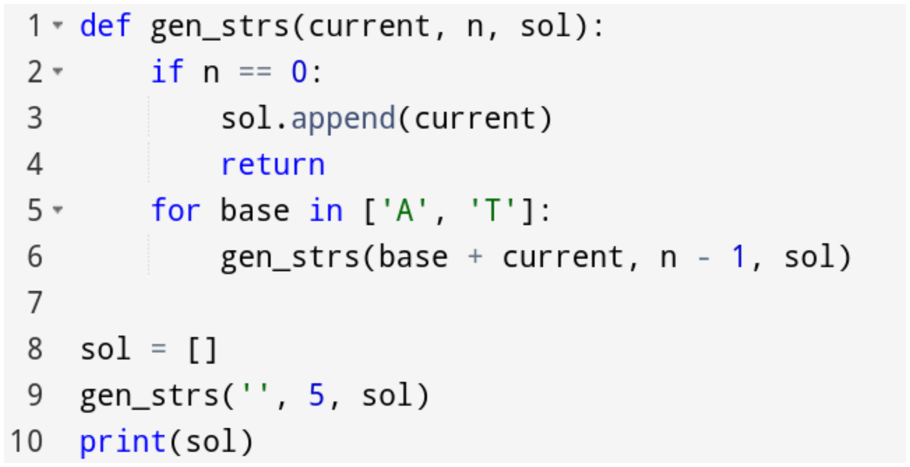
```python
def print2D(n, m):
    vis = []
    for i in range(n + 1):
        vis2 = []
        for j in range(m + 1):
            vis2.append(False)
        vis.append(vis2)
    print2DHelper(n, m, vis)

def print2DHelper(n, m, vis):
    if n == 0 or m == 0:
        return
    if vis[n][m]:
        return
    vis[n][m] = True
    print(n, m)
    print2DHelper(n, m - 1, vis)
    print2DHelper(n - 1, m, vis)
```

(This example has a lot of nuances. One of the reasons it works is because lists in functions are referenced, meaning that if we change the list in the function we change it outside the function. Refer to Class 4 and list references for more information about this).

2. Backtracking (Optional)

We learned about backtracking in class and looked at the example of generating DNA strings incrementally. Let's look at the same example but consider a "simplified DNA" that only contains two bases: A and T. So we want to generate all words of length **n** that contain the letters A or T. Here is the same code from class to do so:



```python
def gen_strs(current, n, sol):
    if n == 0:
        sol.append(current)
        return
    for base in ['A', 'T']:
        gen_strs(base + current, n - 1, sol)

sol = []
gen_strs('', 5, sol)
print(sol)
```

To help you really understand what is going on here, let's use the simulation tool that we used in class 3 and 4. Click here to see the simulation of the previous code. Go through the simulation by pressing 'Next' or 'Prev'. Pay special attention to a couple of things: You'll see several frames on the right all for the *gen_strs()* function, these represent the different recursive calls. Look at how the value of *current* changes through the execution of the program. Finally, look at what happens to the *sol* list as we append values to it.

3. Coding Questions

You should see a new problem set called "Class 6 Problems" (https://codeforces.com/group/K1Fxw6skwV/contest/384756). You should solve at least the first problem (Sum of digits - Recursion). The second problem (Generating Permutations) is also fairly easy, but it's optional. The last 3 problems are much harder and are intended to be challenging problems for the more experienced students (Maximum Increase, Vacations and LIS).