STRUCTURING PROGRAMS: DATA STRUCTURES

ES 112

A Few Preliminaries

- Mid sem will (most probably) be pen and paper
- We will cover all the material up to last class (Functions)

Enjoy your vacation
Merry Xmas and Happy New Year

Brief Recap

- Scope
- Recursion
- Accessing Variables Outside the Current Scope
- Keyword Arguments

Menu for Today!

- Discussion on Lab Problems
- Lists
- Accessing Elements of a List
- Modifying List Elements

Factorial using Recursion

```
def fact(n):
    if n == 0:
        return 1
    else:
        n * fact(n-1)
```

Fibonacci Series

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return(fib(n-1) + fib(n-2))
```

Sum of Digits

```
def sumDigits(num):
    if (num < 10):
        return num
    else:
        return((num % 10) + sumDigits(num // 10))
```

Power using Recursion

```
def power(a,b):
    def power1(a,b):
        if (b == 0):
            return 1
        else:
            return(a * power1(a,b-1))

    if (b < 0):
        print(f'{b} is less than zero')
    else:
        return power1(a,b)
```

GCD with Recursion

```
def gcd(a,b):
    def innerGcd(a,b):
        if b%a == 0:
            return a
        else:
            return innerGcd(b%a, a)
    if a > b:
        a,b = b,a
    return(innerGcd(a,b))
```

Palindrome

```
def palindrome(x):
    def reverse(x, r = 0):
        if ( x < 10):
           return(r*10+x)
        else:
           return(reverse(x//10, r*10+x%10))

    return(x == reverse(x))
```

Pascal’s Triangle

```
def pascal(n):
    st=''
    for i in range(n+1):
        term=fact(n)//(fact(i)*fact(n-i))
        st+= str(term) + " "
    return st
num=int(input())
length=len(pascal(num-1))
for i in range(num):
    print(pascal(i).center(length))
```

Structuring Programs

- So far, we have seen how to structure execution flow in Python using Functions
- The other key component of programs is data
- How do we structure data?

Ordering Food at A Restaurant

- Table 1 orders
  - *Soup*
  - *Kebabs*
  - *Chicken Tikka Masala*
  - *2 Rotis*
  - *Ice cream*
- Table 2 orders
  - *Samosas*
  - *Chai*

- Sequence of delivery is important
  - *2 Rotis*
  - *Ice cream*
  - *Kebabs*
  - *Chicken Tikka Masala*
  - *Soup*
- Orders must be delivered to the right table
  - *What happens if Chicken Tikka Masala gets delivered to Table 2*

The waiter would naturally write down the orders as two lists

- [soup, kebabs, [chicken tikka masala, 2 rotis], ice cream]
- [samosas, chai]

Lists in Python

- Lists are the most common data structure in Python
- A list is an ordered sequence of items

```
instructor = [‘Milind Gandhe’, ‘A219’]
students= [[‘RollNo1’, ‘Name1’, ‘CS’], [‘Roll no2’, ‘Name2’, ‘ECE’]]
course = [‘ES112’, ‘A106’, instructor, students]
```

- A list may have items of different types
  - *Possible, but not recommended*
- A list can be embedded inside another list

A list is a collection of objects with an inherent notion of order and next

Indexing and Slicing

- Indexing and slicing work the same as on strings

```
x = [‘This’, ‘is’, [‘a’, ‘strange’, ‘sentence’], 5, True]
x[0]       ->  ‘This’
x[1:4]     ->  [‘is’, [‘a’, ‘strange’, ‘sentence’],5]
x[0:5:2]   ->  [‘This’, [‘a’, ‘strange’, ‘sentence’], True]
x[-3,-1]   ->  [['a', 'strange', 'sentence'], 5]
```

Changing Elements of a List

- We cannot change individual characters or slices in a string

```
x = 'cat’
x[1] = ‘u’
TypeError: 'str' object does not support item assignment
```

- We can change individual items in a list

```
y = [‘c’,’a’,’t’]
y[1] = ‘u’
```

- `y` is now `[’c’,’u’,’t’]`
  - *Note that this is the same object, not a new object*

Lists are Mutable

- Also works on slices

```
y[0:1] = [‘b’, ‘a’]
```

y is now `[’b’,’a’,’t’]`

- Size of the slice may not match the list being assigned to it
  - *The slice is deleted*
  - *The assigned list is inserted in its place*

```
x = [0, 1, 2, 3, 4]
x[2:4] = [5, 6, 7]
```

Looking Under the Hood: Aliasing

- The following code results in y getting a value 2

```
x = 2
y = x
x = 3
```

- However, the following code gives y a value of [4, 2, 3]

```
x = [1, 2, 3]
y = x
x[0] = 4
```

Assigning a list to a variable is the equivalent of assigning a pointer in C