Lists and Recursion

(yay!)

Lists

List Basics

- **Data structure** that holds ordered set of data values.
  - Ex: L = [“I”, “love”, 15110]
  - Lists in Python can contain different data types
- **Indexing**: we can access each data values based on numerical position in list.
  - L[0] = “I”
  - L[-1] = 15110
- **Slicing**: we can access sublists within lists!
  - L[a:b:c], where a= start point (inclusive), b = end point (exclusive), c = step size
  - L[0:2] = [“I”, “love”]
  - L[1:] = [“love”, 15110]
- **List Looping**
  - for elem in L: #use this when you don’t know/care about list indexes
  - For in in range(len(L)) #use this when you do want to have access to list indexes

List Basics (cont'd.)

- List concatenation and basic methods. Note differences!
  - Adding Lists: ["Flako", "Gabriel"] + ["Minjoo", "Shivi"] = ["Flako", "Gabriel", "Minjoo", "Shivi"]
  - L.append(x): Adds an element x to end of list. Returns None.
  - L.extend(A): Adds all elements in list A to the end of list L. Returns None.
  - L.pop(i): Removes the ith element from list L. Returns value at popped index.
- Other helpful methods to review…
  - len(L) → length of list
  - min(lst) → smallest element of list
  - max(lst) → largest element in list
  - sum(lst) → sum of all elements in list
  - random.choice(lst) → returns random value in list

List Aliasing / Destructive and Non-Destructive

- Variables aliased when they are set equal
- Destructive: modify all aliased values also
  - Append, extend, insert, remove, pop
- Non-destructive: makes new list/new reference, breaks alias
  - Concatenation, slicing
  - Note that lst += and lst = lst + behaves differently

Destructive vs. Non-destructive Example

Write the non-destructive version of the following function:

```
def removeDuplicates(L):
    index = 0
    while index < len(L):
        item = L[index]
        if L.count(item) > 1:
            L.remove(item)
        index += 1
    return L
```

List Code Tracing Example 1

Trace through the code below. What are the values of A, B, C, D, E, and F? Which share the same memory space?

```
A=’cat’
B=A
A=’dog’
C=[1,2,3,4,5]
D=C[::]
E=C
D.insert(0, 10)
E[1]=’pie’
F=C.remove(1)
```

Solution

A=’dog’
B=’cat’
C=[’pie’,3,4,5]
D=[10,1,2,3,4,5]
E=[‘pie’,3,4,5]
F=None

C and E are aliased/same memory space.

List Code Tracing Example 2

```
def mys(lst):
    newLst = lst
    i = 0
    while i < len(newLst):
        if type(lst[i]) != str:
            newLst.pop(i)
        else:
            i = i+1
    print(newLst)
lst = ["a", "hey", 3, "d", True]
mys(lst)
```

Solution

What does the function print with the function call? `["a", "hey", "d"]`

In general terms, what does this function do? `Given a list, it prints the list with any non-string elements removed`

2D Lists

- 2D lists are lists of lists
  - Outer list elements are lists
  - Inner list elements are individual values
- Index with double brackets
- Nested loops to iterate through both lists

2D Lists

cities = [["Pittsburgh", "Allegheny", 302407],

["Philadelphia", "Philadelphia", 1584981],

["Allentown", "Lehigh", 123838],

["Erie", "Erie", 97639],

["Scranton", "Lackawanna", 77182]]

a. Cities[1] = ["Philadelphia", "Philadelphia", 1584981]
b. Cities[1][0] = “Philadelphia”
c. What happens if we run cities[3][0] = “Harrisburg”

2D List Code Writing Example

Let's say that you are in charge of processing student information from a poll. You receive your information in the form of a list of strings, as so:

```
#lst=[name, favorite_color, age, favorite_season, has_pet]
lst=["Michael, blue, 25, summer, True",
     "Lily, green, 18, winter, False"]
```

You need to convert this list so that it is useful for your needs. Write a function, pollAnswers, that takes in the list lst and transforms it into a 2D list, like so:

```
lst_2d=[
    ['Michael', 'blue', '25', 'summer', 'True'],
    ['Lily', 'green', '18', 'winter', 'False']
    ]
```

2D List Code Writing (cont'd.)

How would you use that same lst (modified by pollAnswers) and now specified by the variable lst_2d (lst_2d=lst), in order to find/return the average age of the people you have polled? Write it in the function findAverage, which takes in the parameter lst_2d.

Solution

```
def findAverage(lst_2d):
    sum=0
    For i in range(len(lst_2d)):
        sum+=int(lst_2d[i][2])
    Return sum/len(lst_2d)
```

Recursion

Recursion Basics

- General algorithm:
  - Base case of when the input is the smallest value
  - Determine how to make the problem slightly smaller
  - return *something* combined with recursive call on smaller problem
- Base case builds the answer by ending the continual recursive calls with a solid return value
- Return types must match!!
- Infinite recursion → recursion error. Don’t forget to make the problem smaller!!!

Recursion and Binary Search

- Multiple recursive calls become recursive call trees
- Examples:
  - Towers of Hanoi
  - Fibonacci
- Recursive binary search divides the amount needed to search through by half, given that the input list is sorted
  - If middle is less than target, search to the right
  - If middle is greater than target, search to the left

```
def binarySearch(lst, target):
    if lst == [ ]:
        return False
    else:
        midIndex = len(lst) // 2
        if lst[midIndex] == target:
            return True
        elif target < lst[midIndex]:
            return binarySearch(lst[:midIndex], target)
        else: # lst[midIndex] < target
            return binarySearch(lst[midIndex+1:], target)
```

- Base case for binary search? How is problem made smaller?
- binarySearch([2, 4, 6, 9], 9) = ?
- binarySearch([2, 4, 6, 9, 10], 0) = ?

Recursion Code Trace

Trace the following function. What is this function doing in general?

```
def x(n):
  if n == 0:
    return 0
  else:
    return n % 10 + x(int(n / 10))

print(x(345))
print(x(45))
```

Solution

$x(345) = 12$, $x(45) = 9$

The function finds the sum of the digits of a number.

Recursive Function Writing Example

Write a recursive function `reverseOdds` that takes in a list of integers `lst` and a an integer `target` as input and returns all of the integers in `lst` less than `target` in reverse order. For example,

`reverseOdds([1,2,3,4,5,6], 4)` returns `[3,2,1]`
`reverseOdds([27,13,9,15,12,34], 14)` returns `[12,9,13]`
`reverseOdds([2,4,6,8,10],1)` returns `[]`

Solution

```
def reverseOdds(lst, target):
    if lst == []:
        return []
    else:
        first = lst[0]
        rest = lst[1:]
        result = reverseOdds(rest, target)
        if first < target:
            return result + [first]
        else:
            return result
```

Thanks for coming! :D

Other things to review:

- HW Problems
- Lecture Notes
- Practice Tests
- Small Group Problems
- These slides