| 15-110 Recitation Week 7 |
| --- |

**Reminders**

* 3/14 Tue - Check3/HW3 revisions due (Tuesday after break)
* Have a restful and rejuvenating break!

**Overview**

* Big-O Exercise
* For Loop Review
* Dictionary Review
* Tree Code Writing
* Dictionary Code Writing

| Problems |
| --- |

# **BIG-O EXERCISE**

Calculate the Big-O for the following examples:

| Returning the last character in a string |   |
| --- | --- |
| def powersOfTwo(n): # n = n m = 1 while m <= n:  print(m) m \*= 2 |  |
| def foo(L): # len(L) = n if L == []: return 0 else: L.append(L[0])  n = L.index(10) L.pop(0) return n# .index(), .pop() are O(n) worst case! |  |
| #You are guaranteed L is a nxn 2D listdef tripleLoop(L): for i in range(20): for row in L: for elem in row: print(elem) |  |

#

# **FOR EACH LOOP REVIEW**

Notes on Loops::

|  |
| --- |

Problem:

Use the following code to answer the questions:

s = "15-110"

for i in range(len(s)):

 print(i)

for i in s:

 print(i)

What does the code print?

What is the type of i for each loop?

#

# **DICTIONARY REVIEW**

Notes on dictionaries:

|  |
| --- |

**Here is an example of a type of problem that uses dictionaries. Read through the problem statement and solution and note the key points of the code.**

Problem:

Kelly’s Bakery is doing an inventory of their freshly baked goods. This morning, they baked new items and now they need to update their inventory to represent these items. You are given a dictionary that represents the inventory at Kelly’s Bakery, which maps the name of the item to how many items of that baked good are available. Write the function updateInventory(d, newItems) that takes the current inventory and a new dictionary called newItems and updates it accordingly. The function should also handle the case that there is an item in newItems that doesn’t exist in d.

Solution:

def updateInventory(d, newItems):

 for item in newItems:

 if item in d:

 d[item] += newItems[item]

 else:

 d[item] = newItems[item]

 return

#

# **TREE CODE WRITING**

Write the function addEvenLeaves(t) that takes in a dictionary representation of a tree and returns a sum of **only** the even values held by leaves.

def addEvenLeaves(tree):

 # base case: if tree is empty

 if \_\_\_\_\_\_\_\_\_\_\_\_\_:

 # What should you return if the tree is empty?

 return \_\_\_\_\_\_\_\_\_\_\_\_\_

 # base case: leaf node

 if \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_:

 # check if leaf’s value is even

 if \_\_\_\_\_\_\_\_\_\_\_\_\_\_:

 # returns the leaves value

 return \_\_\_\_\_\_\_\_\_\_\_\_\_\_

 else:

 # what should you return if the leaf isn’t even?

 return \_\_\_\_

 else:

 value = 0

 # recursive case if left subtree is not None

 if \_\_\_\_\_\_\_\_\_\_\_\_:

 value += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # recursive case if right subtree is not None

 if \_\_\_\_\_\_\_\_\_\_\_\_:

 value += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 return value

#

#

# **DICTIONARY CODE WRITING**

Given a dictionary that maps teams like CMU, Pitt, OSU, PennState, and another unspecified number of football teams, to the number of wins they have, we return the team with the most wins. There will be no ties. For example,

mostWins({ "CMU" : 1, "Pitt" : 1, "OSU" : 3, "PennState" : 1 }) returns “OSU”.

def mostWins(wins):

 **# Initialize variables to store the team that has won**

 **# the most so far and how many times they had won**

 mostWinTeam = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 mostWins = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 **# Loop through the dictionary**

 for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:

 **# What do we do if the current team has won more**

 **# than the team with the most wins so far?**

 if \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ > mostWins:

 mostWins = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 mostWinTeam = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 **# Return the team that has won the most**

 return mostWinTeam