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

**Reminders**

* 10/22 Tues - Check3/HW3 revisions due (Tuesday after break)
* [Recitation feedback form](https://forms.gle/519bDVtMU6mGbH5JA)
* Have a restful and rejuvenating break!

**Overview**

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

‘

| Problems |
| --- |

# **BIG-O EXERCISE**

Calculate the Big-O for the following examples:

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

#

# **FOR-ITERABLE LOOP REVIEW**

Notes:

|  |
| --- |

Use this code to answer the following questions:

s = "15-110"

for i in range(len(s)):

 print(i)

for i in s:

 print(i)

What does each loop print?

What is the data type of i in each loop?

#

# **DICTIONARY CODE WRITING**

We’re given a dictionary that maps some number of football teams (e.g. CMU, Pitt, OSU, PennState) to the number of wins and losses they have (represented as [wins, losses]), and an integer representing the minimum number of games to be considered. We want to return the team with the highest percentage of wins and that has played the minimum number of games. There will be no ties.

E.g. bestTeam({ "CMU" : [1, 10], "Pitt" : [7, 10], "OSU" : [10, 6], "PennState" : [2, 1] }, 5) returns “OSU”

def bestTeam(winsLosses, minGames):

 bestTeam = \_\_\_\_\_\_\_\_\_\_\_\_

 bestPercent = \_\_\_\_\_\_\_\_\_\_\_

 for team in winsLosses:

 wins = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 losses = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 gamesPlayed = \_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_

 # check if team played enough games

 if \_\_\_\_\_\_\_\_\_\_\_\_\_ >= minGames:

 winPercent = \_\_\_\_\_\_\_\_\_\_\_\_ / gamesPlayed

 if \_\_\_\_\_\_\_\_\_\_\_\_\_ > bestPercent:

 bestPercent = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 bestTeam = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

# **TREE CODE WRITING**

Write the function addEvenLeaves(t) that takes in a dictionary representation of a tree (you can assume it will have at least 1 node) and returns a sum of **only** the even values held by leaves.

def addEvenLeaves(tree):

 # base case: empty tree

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

 # what should we return?

 return \_\_\_\_

 # recursive case

 else:

 result = 0

 # check if we’re at a leaf

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

 # check if its value is even

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

 result += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # recursively add the even leaves of the subtrees

 result += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 return result

#