Practice Midterm 2 Solutions

You have 2 hours to complete this exam. This exam is meant solely for practice and topics that are not in this exam may be covered while topics in it may not be covered. This exam is out of 50 points.

Problem	Points
1	12
2	5
3	6
4	7
5	8
6	12
Total	50

1 WWPD (12 pts)

(a) For each of the expressions in the table below, write the output displayed by the interactive Python interpreter. Write Error if the result is an ERROR, if the result is a Function write FUNCTION, and if the output is a generator object output GENERATOR. Complete the table on the next page.

```python
class Human:
    villains_caught = []
    def __init__(self, name):
        self.health = 100
        self.name = name

    def __repr__(self):
        return self.name

class Hero(Human):
    catchphrase = "Holy Cow"
    villains_caught = []
    def __init__(self, name, partners = []):
        super().__init__(name)
        self.partners = partners

    def add_buddy(self, partner):
        self.partners.append(partner)
        for i in self.partners:
            i.partners.append(partner)
        print("We're all partners!", self.partners)

    def catch_villain(self, villain):
        print(self.catchphrase)
        self.villains_caught.append(villain)

    def teamup(self, members):
        for member in members:
            yield from members

class Villain(Human):
    catchphrase = "I'm Evil"
    heroes_attacked = []

    def __init__(self, name):
        super().__init__(name)

    def escape_prison(self):
        self.villains_caught.pop()

    def attack_hero(self,Hero):
        self.heroes_attacked + [Hero]

batman = Hero("Batman", [])
robin =  Hero("Robin", [])
joker = Villain("Joker")
```

Expression	Interpreter Output
Hero.catch_villain(batman,joker)	Holy Cow
print(batman) batman == ”Batman”	Batman False
joker.escape_prison()	ERROR

superman=Hero(”Superman”)
batman.add_buddy(superman)
	We’re all partners! [Superman]

league = [batman, superman, robin]
flash = Hero(”Flash”)
duo = Hero.teamup(flash, league)
duo
	GENERATOR

next(duo)	Batman
joker.attack_hero(batman) joker.heroes_attacked	[ ]

batman.villains_caught.pop()
Hero.catch_villain(joker, joker)
	Joker
	I’m Evil

joker.villains_caught	[Joker]
batman.villains_caught	[ ]
superman.add_buddy(batman)	We’re all partners! [Superman, Batman, Batman]

league.insert(2,flash)
league.append(league.pop(0))
print(league)
next(duo)
next(duo)
next(duo)
next(duo)
	[Superman, Flash, Robin, Batman]
	Flash
	Robin
	Batman
	Superman

(b) We are given the below function. Where arr is an input array of size N, what is the running time of this function (time this function takes to terminate) in terms of N?

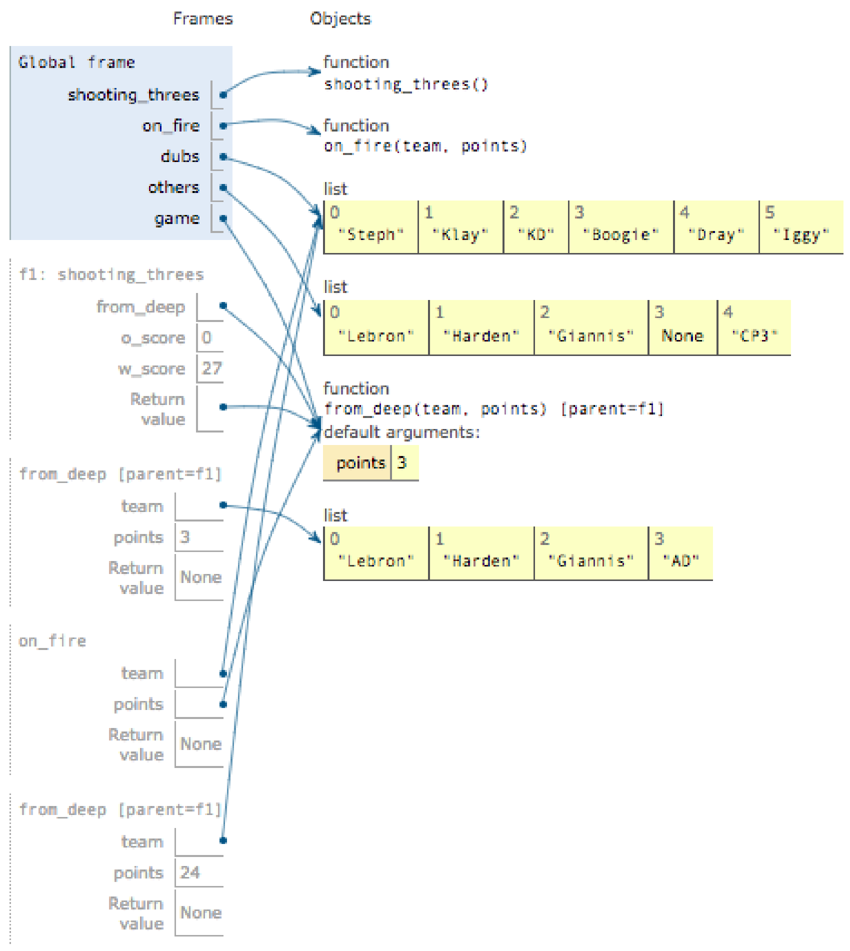
```
def iterate_through_members():
        lst = []
        arr = [1,2,3......N]
        superteamup = batman.teamup(arr)
        while True:
                lst.append(next(super_team_up)
```

(a) O(1) (b) O(log(N)) (c) O(N) **(d)O($\mathbf{N^2}$)** (e) O(2^N)

2 Got as Hot as a Toaster

Fill out the environment diagram below. There at most 4 frames to create. Write the parents, variable bindings, and return values for each frame.

```
def shooting_threes():
    w_score = 0
    o_score = 0
    def from_deep(team, points = 3):
        nonlocal w_score, o_score
        if team == others:
            o_score+=points
        else:
            w_score+=points
    return from_deep
def on_fire(team, points):
    points(team,24)
dubs = ["Steph", "Klay","KD","Boogie", "Dray"]
others = ["Lebron", "Harden", "Giannis"]
game = shooting_threes()
game(others + ["AD"])
others.append(dubs.append(["Iggy"][0]))
on_fire(dubs,game)
others.extend([["CP3","PG13"].pop(0)])
```



3 WarmUp for a Cool Day

You are given a list of n numbers, 1 through n, in an input array nums. Return a new list where every element in the returned list, output, is equal to the product of every element in nums excepts nums[i]. Assume that all the numbers in the initial nums list are non-zero.

You are forbidden from using the division operator.

```
def ProductExceptSelf(nums):
    """
    >>> ProductExceptSelf([1,2,3,4])
    [24,12,8,6]
    """
    firstPass = [1 for i in range(len(nums))]
    secondPass = [1 for i in range(len(nums))]
    for i in range(1,len(nums)):
        firstPass[i] = firstPass[i-1] * nums[i-1]
    for i in range(0,len(nums)-1)[::-1]: #[::-1] returns a reversed list
        secondPass[i] = secondPass[i+1]*nums[i+1]
    combined = [1 for i in range(len(nums))]
    for i in range(len(combined)):
        combined[i] = firstPass[i] * secondPass[i]
    return combined
print(ProductExceptSelf([10,9,8,7]))
```

4 Delivering with Zoidberg

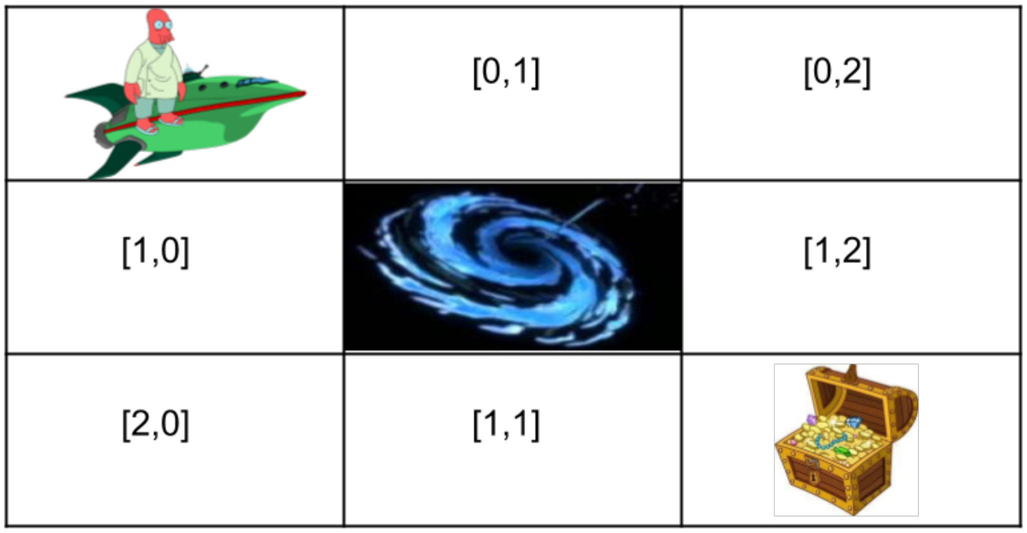
You are an alien trying to move through space. Space looks strangely like an n x n grid. You can move left, right, up, or down (as long as you don't go out of bounds in this n x n grid looking space), but there are some constraints:

1. There are some locations that are filled with black holes. As a result, if you go to that block, you will not be able to reach your goal. In the input grid, black holes will be given the denotation of 0 and all safe squares will be denoted by 1.
2. Because you are low on gas, you must reach your goal in k steps or less.
3. Your function inputs are a list of [x,y] coordinates for both your start and end, the amount of steps you can take (k), and a n x n matrix (represented as a 2d list) with 1's where you can travel and 0 where there are black holes.
4. Once you land on a square you can return to it (for example you can move down then move back up). The only exception to this rule is once you land on the goal square you cannot move anywhere.
5. You are guaranteed that your goal square will not be on a black hole.

Below is an example of an alien starting at [0,0] with a goal of [2,2]. They have 4 steps to get somewhere and there is a black hole at the point [1,1]. This is formulated as the following call:

```
>>> grid = [[1,1,1], [1,0,1], [1,1,1]
>>> uniquePaths([0,0],[2,2],4,grid)
2
```

After looking at the example, fill out the code on the next page.



```
def uniquePaths(start, end, k, grid):
        """
        >>> grd = [[1, 0],[1,1]]
        >>> uniquePaths([0,0],[1,1],2, grd)
        1
        >>> grd = [[1, 1],[1,1]]
        >>> uniquePaths([0,0],[1,1],4, grd)
        6
        """
        def check_valid(xpos, ypos):
            if xpos < 0 or ypos< 0 or xpos >= len(grid) or ypos >= len(grid):
                return False
            if grid[xpos][ypos] == 0:
                return False
            return True
        directions = []
        for i in [-1,0,1]:
            for j in [-1,0,1]:
                if i!=j and (i == 0 or j== 0):
                    directions.append((i,j))
        def findthePaths(pos,k):
            if pos[0] == end[0] and pos[1] == end[1]:
                return 1
            if not check_valid(pos[0], pos[1]) or k == 0:
                return 0
            return sum(findthePaths([pos[0]-i, pos[1]-j],k-1) for i,j in directions)
        return findthePaths(start,k)

grd = [[1, 1, 1],[1,0, 1], [1,1,1]]
print(uniquePaths([0,0],[2,2],4, grd))
```

5 Woody TreeWrecker

(a) You are a Flamboyant Tree Pecker. Your goal for this upcoming winter is to find how much fruit you can gather from some each level of a tree. In order to solve this problem you have decided that you will utilize your Python abilities to make a program that will take in a tree and output should be the a LinkedList with nodes corresponding to the sum of each level (with the head of the LinkedList acting as the sum of the topmost level of the Tree). If your Tree is None you may assume that you will return a None node.

```python
class Tree:
    """Tree with a value (entry) and a list of subtrees (branches)"""
    def __init__(self, entry, branches = None):
        self.entry = entry
        if branches:
            self.branches = branches
        else:
            self.branches = []
class LinkedList:
    def __init__(self,entry,next = None):
        self.entry = entry
        self.next = next
def BinaryTreeToSumLinkedList(root):
    queue,nextqueue, sum_so_far = [], [], 0
    queue.append(root)
    start = LinkedList(0) #dummy
    LL = start
    while queue:
        node = queue.pop()
        nextqueue.extend(node.branches)
        sum_so_far += node.entry
        if not queue:
            LL.next = LinkedList(sum_so_far)
            LL = LL.next
            sum_so_far = 0
            queue = nextqueue
            nextqueue = []
```

6 I Left, I Did Nothing, I returned

You are a famous unnamed Roman Emperor. You want to prove that all roads do in fact lead to Rome. You had once had a Tree that had a root which was Rome, but it was taken by some Gauls, very sad. However, you do have a list LinkedLists. Each LinkedList represents the path that you can traverse starting at some node and ending up at Rome. In other words, you starts somewhere in a LinkedList and at the end of each LinkedList is the root of our old Tree.

Given this list of LinkedLists, your goal is to construct a Tree that is composed of the paths that you are given. THERE ARE NO DUPLICATE NAMES IN OUR PATHS (so if we have a place called ”Pillowgia” that will be the only ”Pillowgia”. Additionally, there can from a certain location, you can only go to one location. The converse is not true. That is, if you can directly go from ”Pillowgia” to ”Bedtopia” in one LinkedList, there exists no LinkedList where ”Pillowgia” would go anywhere but ”Bedtopia”. However, you can reach ”Bedtopia” from places that are not ”PillowGia”.

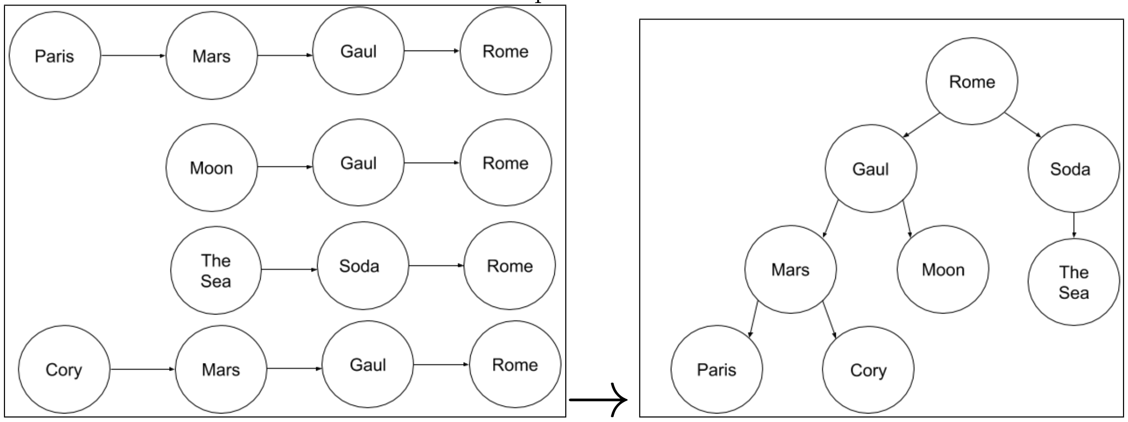
(a) Before we start we need to fix our LinkedList and Tree classes. Below is the code for the Tree class. Identify the error that could occur and give a brief way (English or Code is fine) on how you can fix it in the space next to it.

```python
class Tree:
    """Tree with a value (entry) and a list of subtrees (branches)"""
    """You'll need to change the input. If you make it [] then all branches
    that don't set children will share the same branches.
    """
    def __init__(self, entry, branches = None):
        self.entry = entry
        if branches:
            self.branches = branches
        else:
            self.branches = []
        """The bove 4 lines provide checks to see if the branches were passed in.
        If so, initialize self.branches with the passed in value. Otherwise, initialize
        self.branches to a new list"""

    def add_child(self, child):
        self.branches.append(child)

    def get_entry(self):
        return self.entry
```

(b) Great, now that we’ve done that let’s create a new method in our LinkedList class called traverse_to_end. This method should traverse to the end of the LinkedList and returns the last node’s entry.

```python
class LinkedList:
    def __init__(self,entry,next = None):
        self.entry = entry
        self.next = next
    def traverse_to_end(self):
       tmp = self
       while tmp.next:
           tmp = tmp.next
       return tmp.get_entry()
    def get_entry(self):
        return self.entry
```

(c) Now we're ready to actually create our function. Fill in the blanks with the appropriate phrases so that it follows the behavior the doctests and example below.



For the above example, note how there are no duplicate nodes even though certain nodes exist in more than 1 list. This means that there is more than one way to get to a node and you must account for this.

```
def treeFromRoads(roads):
    """
    >>> LL1=LinkedList("Paris",LinkedList("Mars",LinkedList("Gaul",LinkedList("Rome"))))
    >>> LL2=LinkedList("Moon",LinkedList("Kartik's Driveway",LinkedList("Rome")))
    >>> res = LinkedListsToBST([LL1, LL2])
    >>> res.entry
    Rome
    >>> res.branches[0].entry
    Gaul
    >>> res.branches[1].entry
    Kartik's Driveway
    """
    if not roads:
        return None
    nodes = {}
    def constructTree(element):
        currval = element.get_entry()
        if not element.next and currval not in nodes:
            nodes[currval] = Tree(currval)
        if currval not in nodes:
            constructTree(element.next)
            parent = element.next.get_entry()
            parentTree = nodes[parent]
            currTree = Tree(currval)
            nodes[currval] = currTree
            parentTree.add_child(currTree)
    for i in roads:
        constructTree(i)
    return nodes[roads[0].traverse_to_end()]
```