

Practice Midterm 2

You have 2 hours to complete this exam. This exam is meant solely for practice and is in no way representative of how the actual Midterm 2 will be. 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 (12 pts) What would Penguin Print

(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. For each line, consider that it runs to completion and that each line runs in sequential order (disregarding errors). The first box has been completed for you. (10 pts)

```

1  class Human:
2      villains_caught = []
3      def __init__(self, name):
4          self.health = 100
5          self.name = name
6
7      def __repr__(self):
8          return self.name
9
10 class Hero(Human):
11     catchphrase = "Holy_Cow"
12     villains_caught = []
13     def __init__(self, name, partners = []):
14         super().__init__(name)
15         self.partners = partners
16
17     def add_buddy(self, partner):
18         self.partners.append(partner)
19         for i in self.partners:
20             i.partners.append(partner)
21         print("We're all partners!", self.partners)
22
23     def catch_villain(self, villain):
24         print(self.catchphrase)
25         self.villains_caught.append(villain)
26
27     def teamup(self, members):
28         for member in members:
29             yield from members
30
31 class Villain(Human):
32     catchphrase = "I'm Evil"
33     heroes_attacked = []
34
35     def __init__(self, name):
36         super().__init__(name)
37
38     def escape_prison(self):
39         self.villains_caught.pop()
40
41     def attack_hero(self, Hero):
42         self.heroes_attacked + [Hero]
43
44 batman = Hero("Batman", [])
45 robin = Hero("Robin", [])
46 joker = Villain("Joker")

```

Expression	Interpreter Output
joker.catchphrase	"I'm Evil"
Hero.catch_villain(batman,joker)	Holy Cow
print(batman)	
batman == "Batman"	
joker.escape_prison()	
superman=Hero("Superman")	
batman.add_buddy(superman)	
league = [batman, superman, robin]	
flash = Hero("Flash")	
duo = Hero.teamup(flash, league)	
duo	
next(duo)	
joker.attack_hero(batman)	
joker.heroes_attacked	
batman.villains_caught.pop()	
Hero.catch_villain(joker, joker)	
joker.villains_caught	
batman.villains_caught	
superman.add_buddy(batman)	
league.insert(2,flash)	
league.append(league.pop(0))	
print(league)	
next(duo)	
next(duo)	
next(duo)	
next(duo)	

(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? (2 pts)

```

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

```

(a) $O(1)$

(b) $O(\log(N))$

(c) $O(N)$

(d) $O(N^2)$

(e) $O(2^N)$

2 (5 pts) 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.

```
1 def shooting_threes():
2     w_score = 0
3     o_score = 0
4     def from_deep(team, points = 3):
5         nonlocal w_score, o_score
6         if team == others:
7             o_score += points
8         else:
9             w_score += points
10    return from_deep
11 def on_fire(team, points):
12     points(team, 24)
13 dubs = ["Steph", "Klay", "KD", "Boogie", "Dray"]
14 others = ["Lebron", "Harden", "Giannis"]
15 game = shooting_threes()
16 game(others + ["AD"])
17 others.append(dubs.append(["Iggy"][0]))
18 on_fire(dubs, game)
19 others.extend(["CP3", "PG13"].pop(0))
```

3 (6 pts) 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.

```

1 def ProductExceptSelf(nums):
2     """
3     >>> ProductExceptSelf([1,2,3,4])
4     [24,12,8,6]
5     """
6     firstPass = -----
7
8     secondPass = -----
9
10    for ----- in range(-----):
11
12        firstPass[i] = -----
13
14    for ----- in range(-----)[::-1]:
15        #[::-1] returns a reversed list
16
17        secondPass[i] = -----
18
19    #Hint, firstPass, secondPass, and combined are very similar
20
21    combined = -----
22
23    for ----- in range(-----):
24
25        combined[i] = -----
26
27    return -----

```

4 (7 pts) Delivering with Zoidberg



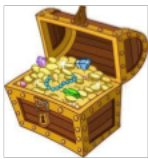
You are an alien trying to move through space. Space looks strangely like an $n \times n$ grid. You can move left, right, up, or down (as long as you don't go out of bounds in this $n \times n$ grid looking space), but there are some constraints:

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. In a step, you may move up, down, left, or right. You CANNOT move diagonally or remain still.
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 $n \times n$ 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.
6. You may use any built in Python Functions (max, sum, min, any, abs etc).

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:

```
1 >>> grid = [[1,1,1], [1,0,1], [1,1,1]]
2 >>> uniquePaths([0,0],[2,2],4,grid)
3 2
```

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

	$[0,1]$	$[0,2]$
$[1,0]$		$[1,2]$
$[2,0]$	$[1,1]$	

```

1 def uniquePaths(start, end, k, grid):
2     """
3     >>> grd = [[1, 0],[1,1]]
4     >>> uniquePaths([0,0],[1,1],2, grd)
5     1
6     >>> grd = [[1, 1],[1,1]]
7     >>> uniquePaths([0,0],[1,1],4, grd)
8     6
9     """
10    def check_valid(xpos, ypos):
11
12        if -----:
13            return False
14
15        if -----:
16            return False
17        return True
18    directions = []
19    for i in [-1,0,1]:
20        for j in [-1,0,1]:
21
22            if -----:
23                directions.append((i,j))
24
25    def findthePaths(pos,k):
26
27        if -----:
28            return 1
29
30        if -----:
31            return 0
32
33        return -----
34
35    return findthePaths(-----)

```

5 (8 pts) Woody TreeWrecker

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.

```

1 class Tree:
2     """Tree with a value (entry) and a list of subtrees (branches)"""
3     def __init__(self, entry, branches = None):
4         self.entry = entry
5         if branches:
6             self.branches = branches
7         else:
8             self.branches = []
9 class LinkedList:
10     def __init__(self, entry, next = None):
11         self.entry = entry
12         self.next = next
13 def BinaryTreeToSumLinkedList(root):
14     """
15     >>>t = Tree(1, [Tree(2, [Tree(8), Tree(10)]), Tree(3, [Tree(14)])])
16     >>>L = BinaryTreeToSumLinkedList(link)
17     >>>L
18     LinkedList(1, LinkedList(5,LinkedList(32)))
19     """
20     queue, nextqueue, sum_so_far = [], [], 0
21
22     ----- .append( ----- )
23     start = LinkedListNode(0) #dummy node helps get rid of edge case
24     LL = start
25
26     while -----:
27
28         node = -----
29
30         ----- .extend( ----- )
31
32         sum_so_far += -----
33
34         if not queue:
35
36             LL.next = -----
37
38             sum_so_far = 0
39
40             queue = -----
41
42             nextqueue = -----
43
44             LL = LL.next
45
46     return start.next

```


6 (12 pts) 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 start 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. We have the following constraints

1. THERE ARE NO DUPLICATE NAMES IN OUR PATHS (so if we have a place called "Pillowgia" that will be the only "Pillowgia").
2. From one location, you can only go to one location. That is, if you can directly go from "Soda" to "Cory" in some LinkedList, there exists no LinkedList where "Soda" would go anywhere but "Cory".
3. It is possible to reach one location from various locations. Following from the prior example, you can reach "Cory" from places that are not "Soda".

Examples of this are on the next page. You do not need this information for the first 2 parts of the problems.

(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. (1 pts)

```

1 class Tree:
2     def __init__(self, entry, branches = []):
3         self.entry = entry
4         self.branches = branches
5
6     def add_child(self, child):
7         self.branches.append(child)
8
9     def get_entry(self):
10        return self.entry

```

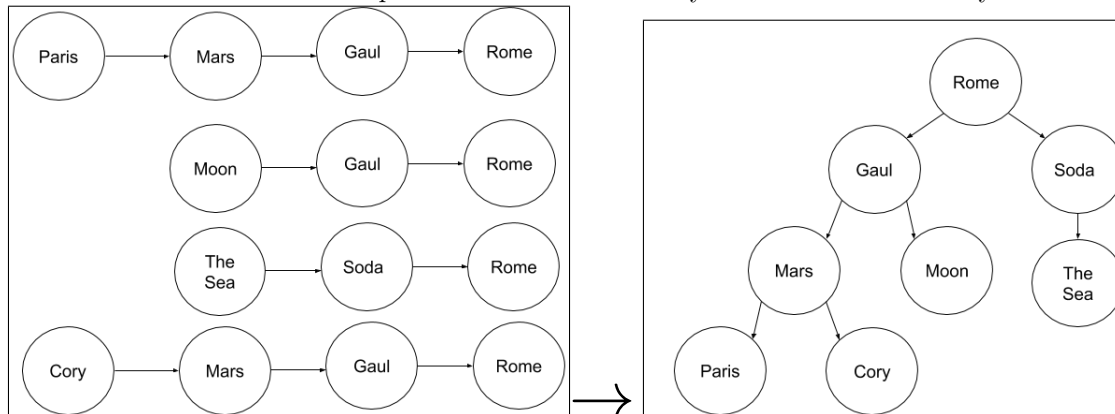
(b) Great, now that we've done that let's create a new method in our LinkedList class called `traverse_to_end`. This method should traverse to the end of the LinkedList and returns the last node's entry. (1 pts)

```

1     def traverse_to_end(self):
2         """
3         Function traverses to the end of the LinkedList and returns the last
4         node's entry. Respect Abstraction.
5         >>> L = LinkedList("A", "B", "C", "D")
6         >>> L.traverse_to_end()
7         "D"
8         """
9         tmp = -----
10
11        while -----:
12
13            tmp = -----
14
15        return -----
16    def get_entry(self):
17        return self.entry

```

(c) We're ready to fill in our function. Fill in the blanks with the appropriate phrases so that it follows the behavior the doctests and example below. Assume that any LinkedLists in the array are not None.(10 pts)



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. (8 pts)

```

1 def treeFromRoads(roads):
2
3     if not _____:
4         return None
5
6     nodes = _____
7
8     def constructTree(element):
9
10        currval = _____
11
12        if not _____ and _____:
13
14            _____ = Tree(currval)
15
16            _____
17
18        if _____:
19
20            _____
21
22            parent=_____
23
24            parentTree=_____
25
26            currTree= Tree(currval)
27
28            _____
29
30            _____
31
32        for i in _____:
33
34            _____
35
36        return _____
  
```