MIDTERM 2 REVIEW

COMPUTER SCIENCE MENTORS 61A

March 12, 2017

1 List Mutation

1. Draw the Box and Pointer!

```
>>> corgi = [3, 15, 18, 7, 9]
>>> husky = [8, 21, 19, 11, 25]
>>> poodle = corgi.pop()
>>> corgi += husky[-3:]
```

2. Draw the Box and Pointer!

```
>>> pom = [16, 15, 13]
>>> pompom = pom * 2
>>> pompom.append(pom[:])
>>> pom.extend(pompom)
```

2 OOP

1. The fOOd Chain

```
class Animal:
    kingdom = []
    location = 'Earth'
    def __init__(self, type, prey):
        self.type = type
        self.prey = prey
        self.kingdom.append(self)
        self.food_chain = [self]
    def eat(self):
        self.location = 'Mars'
    def __repr__(self):
        return self.type

class Predator(Animal):
    def eat(self):
        for animal in self.kingdom:
            if repr(animal) == self.prey:
                self.food_chain.extend(animal.food_chain)
                self.kingdom.remove(animal)
    def __str__(self):
        return '{}s eat {}s.'.format(self.type, self.prey)

grasshopper = Animal('Grasshopper', 'Grass')
sparrow = Predator('Sparrow', 'Grasshopper')
hawk = Predator('Hawk', 'Sparrow')
```

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If more than 3 lines are displayed, just write the first 3. If an error occurs, write Error. If evaulation would run forever, write Forever.
Assume that you have started `python3` and executed the following statements:

`>>> sparrow.kingdom`	
`>>> print(grasshopper)`	

`>>> Animal.eat(grasshopper)`
`>>> grasshopper.location`	

`>>> hawk.location`	
`>>> print(hawk)`	

`>>> sparrow.eat()`
`>>> sparrow.food_chain`	

`>>> hawk.eat()`
`>>> hawk.location`	

`>>> hawk.kingdom`	
`>>> hawk.food_chain`	
`>>> grasshopper.food_chain`	

2. The fOOd Chain

```
class Bird:
    def __init__(self, call):
        self.call = call
        self.can_fly = True
    def fly(self):
        if self.can_fly:
            return 'Don't stop me now!'
        else:
            return 'Ground control to Major Tom...'
    def speak(self):
        print(self.call)
class Chicken(Bird):
    def speak(self, other):
        super().speak()
        other.speak()

class Penguin(Bird):
    can_fly = False
    def speak(self):
        call = 'Ice to meet you'
        print(call)

andre = Chicken('cluck')
gunter = Penguin('noot')
```

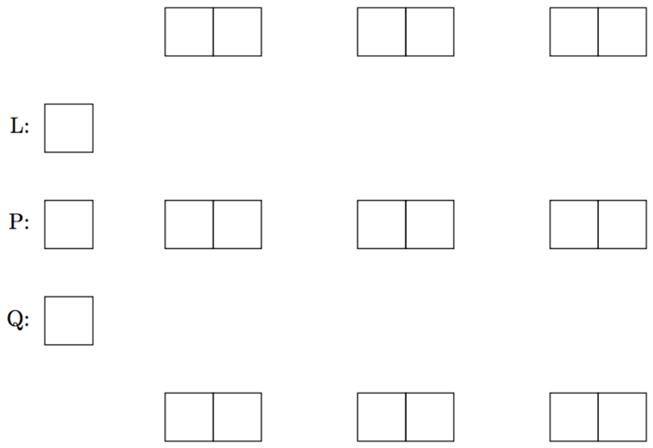
For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If more than 3 lines are displayed, just write the first 3. If an error occurs, write Error. If evaulation would run forever, write Forever.
Assume that you have started `python3` and executed the following statements.

`>>> andre.speak(Bird('coo'))`	
`>>> andre.speak()`	
`>>> gunter.fly()`	
`>>> andre.speak(gunter)`	
`>>> Bird.speak(gunter)`	

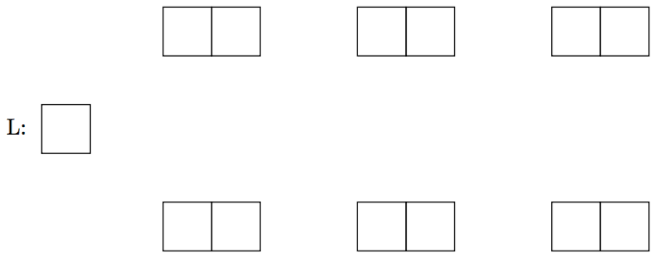
3 Linked Lists and Trees

1. For each of the following code fragments, add arrows and values to the object skeletons to the right to show the final state of the program. Single boxes are variables that contain pointers. Double boxes are Links. Not all boxes need be used.

```
L = Link(1, Link(2))
P = L
Q = Link(L, Link(P))
P.rest.rest = Q
```



```
L = Link.empty
for i in range(3):
    L = Link(i, L)
```



2. WWPD? Try drawing a box-and-pointer diagram!

```
def wild(lnk):
    if lnk is Link.empty or lnk.rest is Link.empty:
        return lnk
    breath = wild(lnk.rest)
    lnk.rest.rest = lnk
    lnk.rest = NULL
    return breath
```

`>>> triforce = Link(3, Link(1, Link(4)))`
`>>> wild(triforce)`	

`>>> triforce`	

3. The depth of a node from the root of a tree is measured by its distance from the root. Nodes that have the same depth are said to be on the same 'level'. Write a function that returns a dictionary, where each key is a level and each value is a list of all labels on that level.

```
def one_more_level(t):

    level_dict = ____________________

    def traverse(t, level):

        if level in queue:

            queue[level] = ____________________

        else:

            queue[level] = ____________________

        for ____________________:

            traverse(____________________, ____________________)

    traverse(____________________, __________________________)

    return ____________________
```

4 Iterators/Generators

1. Write an iterator that takes a Linked List and iterates through it. For instance:

```
>>> li = ListIterator(Link(1, Link(2, Link(3))))
>>> next(li)
1
>>> next(li)
2
>>> next(li)
3
>>> next(li)
StopIteration

class LinkIterator:
```

2. Write a generator that outputs the a decoded run length sequence.

```
def run_length_decoder(encoding):
    """
    >>> rld = run_length_decoder([("h", 1), ("e", 1), ("l", 2),
        ("o", 1)])
    >>> lst(rld)
    ["h", "e", "l", "l", "o"]
    """
```

5 Orders of Growth

1. What do these runtimes simplify to?

O($30n$)	O(10000)	O($n^2 + 10n + 1$)	O($100 + 2^n + n^50$)

2. What is the runtime?

```python
def m(n):
    if n % 7 == 0:
        return m (n - 1) + m(n - 2) * m (n -3)
    else:
        return n
```

```python
def rec(n):
    if n == 1:
        return 1
    return rec(n - 1)
```

```python
def rec(n):
    if n == 1:
        return 1
    return rec(n - 1) + rec(n - 1)
```

```python
def rec(n):
    if n == 1:
        return 1
    return rec(n//2)
```

```python
def wow(n):
    if n == 1:
        return 1
    while n != 0:
        if n % 2 == 1:
            return wow(n // 2) + 1
        n -= 1
```

```python
def pow(n):
    for x in range(50 * n):
        print("powwow")
    for x in range(n ** 2):
        wow(n ** 2)
```

3. Find the runtime in terms of m and n, where m is the `len(lst1)` and `n` is the `len(lst2)` Assume append and * are constant time operations

```
def createMatrix(lst1, lst2):
    """
    >>> createMatrix([1, 2], [3, 4, 5])
    [[3, 4, 5], [6, 8, 10]]
    """
    matrix = []
    for elem in lst1:
        row = []
        For elem in lst2:
            row.append(lst1 * lst2)
            matrix.append(row)
        return matrix
```

Circle one of the options below:

- O$(n + m)$
- O(m)
- O$(n^2 + m^2)$
- O$(n * m)$

4. Runtime with Linked Lists
Let n be the number of elements IN a linked list.
Let m be the number of elements ADDED to the linked list.

    1. What is the runtime of adding 1 link to the beginning of a linked list?

    2. What is the runtime of adding 1 link to the end of a linked list?

    3. What is the runtime of adding m links to the beginning of a linked list?

    4. What is the runtime of adding m links to the end of empty linked list?