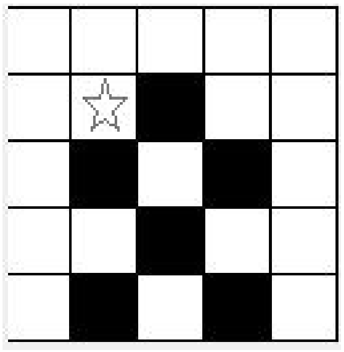
Practice Midterm 2

Short Answer

1. Write the numeric labels that result in each cell after the floodFill completes.

Here is an excerpt of the floodFill code:



```
def floodFill(row, col, color, depth=0):
   …
   floodFill(row-1, col, color, depth+1)
   floodFill(row+1, col, color, depth+1)
   floodFill(row, col-1, color, depth+1)
   floodFill(row, col+1, color, depth+1)
```

2. What was the purpose of the set visited in mazeSolver?

3. Draw a level 2 Sierpinksi Triangle.

4. State and explain the method used to make fibonacci(n) more efficient.

5. Explain the difference between type() and isinstance().

6. Why do we generally avoid setting default arguments of a function to an empty list when we define that function? (eg. def foo(L=[]))

7. Briefly describe what is wrong with the following piece of code. Given an example of an input that would cause this function to crash and state what error Python would throw:

```
def ohNo(x):
     if x == 0: return 1
     if x < 0: return 1 + ohNo(x - 1)
     if x > 0: return 1 + ohNo(x + 1)
```

8. Fill in the blanks:

```
     def powerset(a):
         #returns a list of all subsets of the list a
         if (len(a) == 0):
             return ______
         else:
             allSubsets = [ ]
             for subset in _____________:
                 allSubsets += [subset]
                 allSubsets += ______________
             return _____________

     print powerset([1,2,3])
```

Code Tracing

```
class Human(object):
     def __init__(self, name, age=42):
          self.name = name
          self.age = age
          self.children= []

     def child(self):
          c = Human(self.name + "son", age=self.age//2)
          c.parent = self
          self.children.append(c)
          return c

     def marry(self,other):
          self.spouse = other
          other.spouse = self

     def __repr__(self):
          return self.name+str(self.age)

     def family(self):
          print(self.spouse)
          print(self.children)


a = Human("bob")
b = Human("mary",35)
c = Human("tom",24)
d = Human("mary")
a.marry(d)
a.child()
b.child()
print(a.children == b.children)
a.children[0].marry(c)
c.marry(b.children[0])
b.marry(b)
a.family()
b.family()
for i in [a,b,c,d]: print(i.spouse)
```

```
def ct2(x, y):
    try:
        if y % 3 != 0:
            print (x, y)
            return ct2(x - 1, y + 1)
        else:
            return ct2(x, y)

    except:
        print (x, y)
        try:
            x = x + 1
            y = y - 1
            y = x / (2 * x - y -3)
            x = x + 2 * y
            return y / (2 * x - y)
        except:
            return 3 * x + y

print(ct2(5, 4))

##########################################

def d1(f):
    def g(h):
        return f(f(h))
    return g

@d1
def d2(f):
    def g(x, t):
        return f(x+1, t)
    return g

@d2
def ct3(x, t):
    if len(t) <= 2 or x <= 2:
        return x * t
    m = len(t)//2
    a = t[:m]
    b = t[m:]
    print(a, b)
    return ct3(x - 5, a) + ct3(x - 6, b)

print(ct3(3, "abbcd"))
```

Reasoning Over Code

```
def f(a):
    assert(type(a) == list)
    L = []
    n = 8
    result = []
    def g():
        for i in range(n):
            h = lambda j: j * L[j]
            result.append(hi))
    for item in a:
        L.append(item + 5)
        n -= 1
    assert(L[0] == n)
    g()
    return(result == [0, 6, 30, 120])
```

1. Free Response: longestCAChain

A CA-parent of a student is a CA who taught that student in 112 recitation. Since every 112 CA took 112, every 112 CA has at least one CA-parent (except for the very first 112 CAs). We now define a CA-chain as a list of CAs where each CA in the list appears directly before one of their CA-parents.
For example,

```
[“Austin Richardson”, “Christine Zhang”, “Ben Plaut”]
```
is a CA-chain, since Ben was a CA-parent of Christine, and Christine was a CA-parent of Austin.

Given a dictionary where keys are CAs and the value of a key is a list of the CA’s CA-parents, write the function longestCAChain(chainDict) that returns the longest CA-chain.
For example,

```
chainDict = {
             “Julia Yang” : [“Anthony Kuntz”],
             “Becca Stokes” : [“Tara Stentz”, “Nick Wilson”],
             “Anthony Kuntz” : [“Audrey Lao”]
            }
assert(longestCAChain(chainDict) ==
     [“Julia Yang”, “Anthony Kuntz”, “Audrey Lao”])
```

You may assume that a CA-chain ends once you encounter a CA who has no CA-parents, which means that CA is not a key in chainDict. You may also decide how you would like to resolve ties between two CA-chains of the same length.

2. Free Response: findPath

Help! Professor Kosbie is stuck in the Matrix and needs to find his way out without getting caught by any Agents that might be lurking in the Matrix looking for him.

We define a grid in the Matrix as a square 2D list of booleans, where each cell in the grid is True if there is an Agent in that cell, and False otherwise. Our starting point is the upper left corner, and our ending point is the lower right corner of the grid. Because of the way the Matrix is designed, Professor Kosbie can only move to the right and down to reach his destination. Write the function findPath(grid) that takes in the 2D grid and returns a list of (row,col) pairs indicating a path from the starting point to the ending point, while avoiding any Agents. If there is no valid path, then return None.

For example,

```
grid = [ [False,True,False,False],
         [False,False,False,True],
         [True,False,True,False],
         [False,False,False,False] ]
assert(findPath(grid) == [(0, 0), (1, 0), (1, 1), (2, 1), (3,
1), (3, 2), (3, 3)])
```

Note that in some cases, there may be more than one correct path. In those cases, you may decide how to break ties. In addition, you may assume that the starting cells and ending cells do not contain any Agents.

(The naive solution to the problem above runs in O(2**n), or exponential time, where n is the length of one side of the grid. Bonus points if you can figure out a polynomial time solution!)

3. OOP Free Response: VenmOop

You are asked to write software for the next big digital wallet app, VenmOOP. As the name of the company suggests, you need to use OOP religiously. With this in mind, create the classes User and History that passes the following tests: (in case of ambiguity, make sensible assumptions)

```
dan = User(1,25)
charles = User(2,15)
eric = User(3,50)

assert(dan.balance == 25.00)
assert(isinstance(charles,User))
assert(eric.id == 3)
assert(isinstance(charles.history,History))

dan.pay(eric,20,"dinner")
assert(eric.history.size == 1)
assert(dan.history.entry(1) ==
{"type":"pay","amt":20,"comment":"dinner","status":"success"}

dan.pay(charles,20,"")
assert(charles.balance == 15.00)
assert(charles.history.entry(1)["comment"] == None)
assert(charles.history.entry[“type”] == “charge”)
assert(dan.history.entry(2)[“status”] == “failed”)

dan.charge(eric,10.5,"thanks")
assert(dan.balance == 15.50)
assert(dan.history.size == 3)
assert(eric.history.entry(2)["type"] == "charge")
```

4. Animation Free Response: DotChase

Background: In the game of Dot Chase, there is a main dot that initially is blue and has a radius of 10. The main dot starts 100 pixels directly to the right of the center of the screen, and travels at some constant speed in anti-clockwise circular motion around the center of the screen, such that the distance between the center of the screen and the center of the main dot is always 100 pixels. When the user presses the “Right” key, the main dot speeds up, and when the user presses the “Left” key, the main dot slows down. When the user presses “d”, a new blue dot appears at the right side of the same circular path, with a random radius within the range of 5 to 20, inclusive. The new dot will also travel anti-clockwise around the path at some constant speed. If the main dot collides with any other dot on the screen, it “absorbs” the dot so that the other dot disappears and the main dot increases its radius by the radius of the other dot.

Write the OOPy animation DotChase that extends the Animation class, so you may assume that the run method is already written for you. To implement this, you may also want to write a Dot class.