6.01 Midterm 1 Solutions: Fall 2010

Name:	**Section:**

Enter all answers in the boxes provided.

This solution is not correct for people who took the make-up exam on Wednesday morning starting at 8AM.

During the exam you may:

- read any paper that you want to
- use a calculator

You may not

- use a computer, phone or music player

For staff use:

1.	/12
2.	/16
3.	/16
4.	/21
5.	/20
6.	/15
total:	/100

1 OOP (12 points)

The following definitions have been entered into a Python shell:

```
class Account:
    chargeRate = 0.01
    def __init__(self, start):
        self.value = start
    def debit(self, amount):
        debitAmt = min(amount, self.value)
        self.value = self.value - debitAmt
        return debitAmt
    def deposit(self, amount):
        self.value += amount
    def fee(self, baseAmt):
        self.debit(baseAmt * self.chargeRate)
    def withdraw(self, amount):
        if self.value >= 10000.0:
            self.fee(amount/2.0)
        else:
            self.fee(amount)
        return self.debit(amount)

class Checking(Account):
    chargeRate = 0.05
    def deposit(self, amount):
        if self.value <= 1:
            Account.deposit(self, (1-self.chargeRate) * amount)
        else:
            Account.deposit(self, amount)
```

Assume that the following expressions have been evaluated:

```
Eric = Checking(4000.0)
Ellen = Account(4000.0)
```

Write the values of the following expressions. Write `None` when there is no value; write `Error` when an error results and explain briefly why it's an error. Assume that these expressions are evaluated one after another (all of the left column first, then right column).

`Eric.withdraw(3000.0)`

3000.0

`Eric.value`

850.0

`Eric.withdraw(1000.0)`

800.0

`Eric.value`

0

`Eric.deposit(5000.0)`

None

`Eric.value`

4750.0

`Ellen.withdraw(3000.0)`

3000.0

`Ellen.value`

970.0

`Ellen.withdraw(1000.0)`

960.0

`Ellen.value`

0.0

`Ellen.deposit(5000.0)`

None

`Ellen.value`

5000.0

2 So who kicks b*** (16 points)

Here are some class definitions, meant to represent a league of football teams.

```
class Team:
    def __init__(self, name, wins, losses, pointsFor, pointsAgainst):
        self.name = name
        self.wins = wins
        self.losses = losses
        self.pointsFor = pointsFor
        self.pointsAgainst = pointsAgainst

class League:
    def __init__(self):
        self.teams = {}
    def addTeam(self, team):
        self.teams[team.name] = team
    def updateGame(self, teamName, ptsFor, ptsAgin):
        # to be filled in
    def computeStat(self, proc, filt):
        return [proc(team) for team in self.teams.values() if filt(team)]
```

Imagine instantiating these classes by:

```
Pats = Team('Pats', 5, 1, 150, 100)
Ravens = Team('Ravens', 4, 2, 80, 30)
Colts = Team('Colts', 1, 4, 100, 200)

NFL = League()
NFL.addTeam(Pats)
NFL.addTeam(Ravens)
NFL.addTeam(Colts)
```

We would like to be able to update our information by adding in new game data. For example, if the Pats beat the Colts, by a score of 30 to 15, the record for the Pats should include another win, and an updated record of points for and points against; similarly for the Colts. We would do this by calling

```
NFL.updateGame('Pats', 30, 15)
NFL.updateGame('Colts, 15, 30)
```

Write a Python procedure that will complete the definition of `updateGame`. You may assume that all teams exist in the instance of the league, and that there are no ties, only wins and losses. Please make sure that the indentation of your written solution is clear.

```
def updateGame(self, teamName, ptsFor, ptsAgin):
    team = self.teams[teamName]
    if ptsFor > ptsAgin:
        team.wins += 1
    else:
        team.losses += 1
    team.pointsFor += ptsFor
    team.pointsAgainst += ptsAgin
```

Assume that the NFL has been defined as above, but with more teams. Write an expression using computeStat, to be evaluated by the Python interpreter, that will return a list of wins for all teams in the NFL. (It is okay to define and use helper functions.) For the example instance defined above, your expression should return the list:

```
[5, 4, 1]
```

```
NFL.computeStat(lambda y: y.wins, lambda x: True)
```

Write an expression using computeStat, to be evaluated by the Python interpreter, that will return a list of the losses for the Pats and the Colts, where each entry includes the name of the team. (It is okay to define and use helper functions.) For the example instance defined above, your expression should return the list:

```
[['Pats', 1], ['Colts', 4]]
```

```
NFL.computeStat(lambda y: [y.name, y.losses],
                lambda x: x.name == 'Pats' or x.name == 'Colts')
```

3 Will it or won't it? (16 Points)

For each difference equation below, say whether, for a unit sample input signal:

- the output of the system it describes will diverge or not as n approaches infinity, assuming that $x[n], y[n] = 0$ for $n < 0$,
- the output of the system it describes (a) will always be positive, (b) will alternate between positive and negative, or (c) will have a different pattern of oscillation

Part 1:

$$5y[n] + 2y[n-1] = x[n-2]$$

diverge? Yes or No: No

Why? root's magnitude less than 1

positive/alternate/oscillate: Alternate

Why? root's sign is negative

Part 2:

$$y[n] = -2y[n-1] - 5y[n-2] + x[n-1]$$

diverge? Yes or No: Yes

Why? largest root's magnitude greater than 1

positive/alternate/oscillate: Oscillates

Why? pole is complex

4 Grow, baby, grow (21 Points)

You and your colleague in the Biology Department are growing cells. In each time period, every cell in the bioreactor divides to yield itself and one new daughter cell. However, due to aging, half of the cells die after reproducing (don't worry about the details of how accurately this models real cell division).

We can describe this system with the following difference equation. We let P_o denote the number of cells at each time step.

Then

$$P_o[n] = 2P_o[n-1] - 0.5P_o[n-2]$$

Suppose that $P_o[0] = 10$ and $P_o[n] = 0$ if $n < 0$. What are the first few values for the number of cells (note that while not physically realistic, our model might provide fractional answers)?

$P_o[0] = 10$

$P_o[1] = 20$

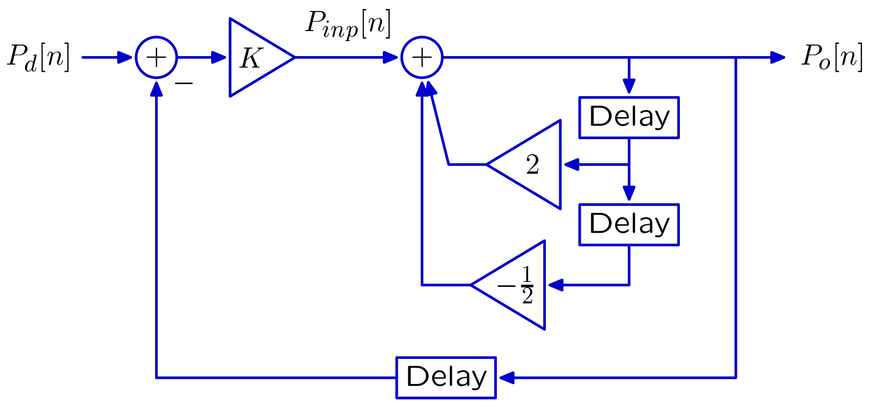
$P_o[2] = 35$

$P_o[3] = 60$

Your goal is to create a constant population of cells, that is, to keep P_o constant at some desired level P_d. You are to design a proportional controller that can add or remove cells as a function of the difference between the actual and desired number of cells. Assume that any additions or deletions at time n are based on the measured number of cells at time $n-1$. Denote the number of cells added or removed at each step P_{inp}. Derive the difference equations that govern this closed loop system.

$P_o[n] = 2P_o[n-1] - .5P_o[n-2] + P_{inp}[n]$
$P_{inp}[n] = k(P_d[n] - P_o[n-1])$

Draw a block diagram that represents this system, using delays, adders/subtractors and gains.



What is the system function that characterizes $\frac{P_o}{P_d}$? Use k to denote the gain in your system.

$\frac{k}{0.5R^2+(k-2)R+1}$

5 Predicting Growth (20 Points)

The following Python classes differ only in the boxed regions.

```
class GrowthA(SM):
    startState = (0,0)
    def getNextValues(self,state,input):
        (s0,s1)   =  state
        output    =  input + s0 + 2*s1
        newState  =  (s1,output)
        return (newState,output)
```

```
class GrowthB(SM):
    startState = (0,0)
    def getNextValues(self,state,input):
        (s0,s1)   =  state
        output    =  input + 2*s0 + s1
        newState  =  (s1,output)
        return (newState,output)
```

```
class GrowthC(SM):
    startState = (0,0)
    def getNextValues(self,state,input):
        (s0,s1)   =  state
        output    =  input + s0 + 2*s1
        newState  =  (s1,input)
        return (newState,output)
```

```
class GrowthD(SM):
    startState = (0,0)
    def getNextValues(self,state,input):
        (s0,s1)   =  state
        output    =  input + 2*s0 + s1
        newState  =  (s1,input)
        return (newState,output)
```

Part a. Determine which (if any) of GrowthA, GrowthB, GrowthC, and GrowthD generate state machines whose output $y[n]$ at time n is given by

$$y[n] = x[n] + x[n-1] + 2x[n-2]$$

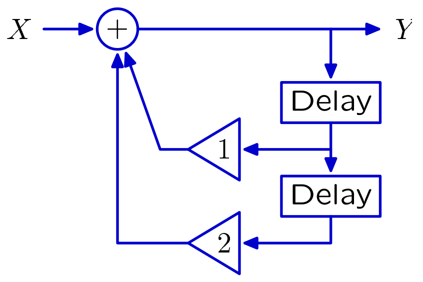
for times $n \geqslant 0$, when the input $x[n] = 0$ for $n < 0$.

Circle all of the classes that satify this relation, or circle **none** if none satisfy it.

GrowthA GrowthB GrowthC GrowthD **none**

GrowthD

Part b. Determine which (if any) of GrowthA, GrowthB, GrowthC, and GrowthD generate state machines whose input-output relation can be expressed as the following block diagram.



Circle all of the classes that satify this relation, or circle **none** if none satisfy it.

GrowthA	GrowthB	GrowthC	GrowthD	**none**

GrowthB

Part c. Let $\mathrm{H_A}$, $\mathrm{H_B}$, $\mathrm{H_C}$, and $\mathrm{H_D}$ represent the system functions associated with the state machines generated by GrowthA, GrowthB, GrowthC, and GrowthD, respectively. Fill in the following table to indicate the number and locations of the poles of $\mathrm{H_A}$, $\mathrm{H_B}$, $\mathrm{H_C}$, and $\mathrm{H_D}$. Pole locations can be listed in any order. Leave unnecessary entries blank.

system	# of poles	pole 1 location	pole 2 location	pole 3 location
H_A				
H_B				
H_C				
H_D				

2, 1 + root(2), 1 - root(2)

2, 2, -1

0

0

6 I need a caffeine jolt (15 Points)

Taking exams is hard work, and it would be nice if there were a caffeine dispenser next to every student's desk. You are to create a state machine that dispenses caffeine jolts (unfortunately these are expensive, and cost 3 dollars each!). This machine accepts dollar bills in different denominations, but does not make change. Hence, your machine should have the following behavior:

- The inputs to the machine are positive integers (representing different denominations of dollars);
- If the input plus the current amount of money deposited in the machine is greater than 3, the machine outputs the current input;
- If the input plus the current amount of money deposited in the machine is exactly 3, the machine outputs a jolt and resets its internal state;
- If the input plus the current amount of money deposited in the machine is less than 3, the machine adjusts its state, outputs `None` and waits for the next input.

Here are some examples:

```
>>>v = Vending()
>>>v.transduce([1,1,1])
[None, None, 'jolt']

>>>v.transduce([1,2])
[None, 'jolt']

>>>v.transduce([5,1,4,2])
[5, None, 4, 'jolt']
```

Feel free to change the `startState` if it simplifies your solution. Here is an outline of our state machine:

```
class Vending(sm.SM):
    startState = None
    def getNextValues(self, state, inp):
```

Complete the definition of `getNextValues`

```
class Vending(sm.SM):
    startState = 0    # Note the choice of startState
    def getNextValues(self, state, inp):
        if state + inp > 3:
            return (state, inp)
        elif state + inp == 3:
            return (0, 'jolt')
        else:
            return (state + inp, None)
```