6.01 Final Exam: Fall 2010

Name:	**Section:**

Enter all answers in the boxes provided.

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

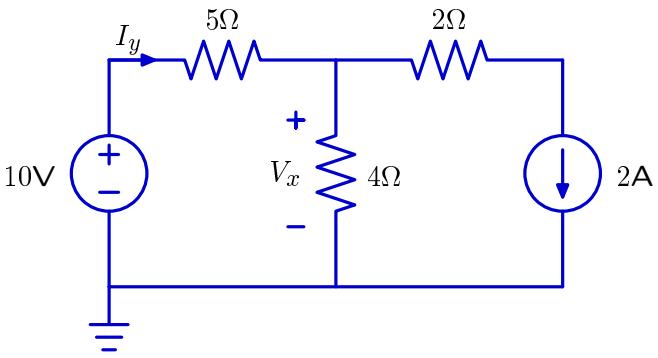
For staff use:

1.	/12
2.	/10
3.	/14
4.	/8
5.	/20
6.	/20
7.	/16
total:	/100

1 Circuits (12 points)

1.1 Finding voltages and currents

Consider the circuit shown below:



Determine I_y.

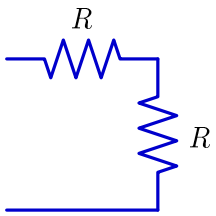
$I_y =$

Determine V_x.

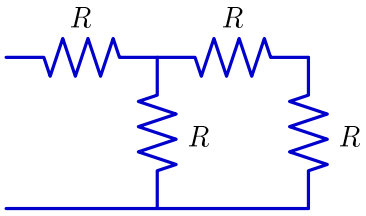
$V_x =$

1.2 Resistive Ladders

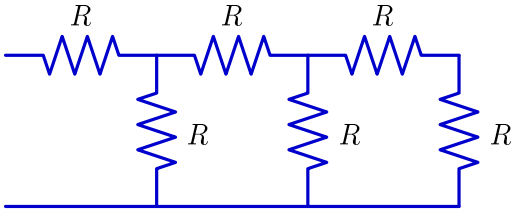
Determine the equivalent resistance of each of the following combinations of resistors.



equivalent resistance =



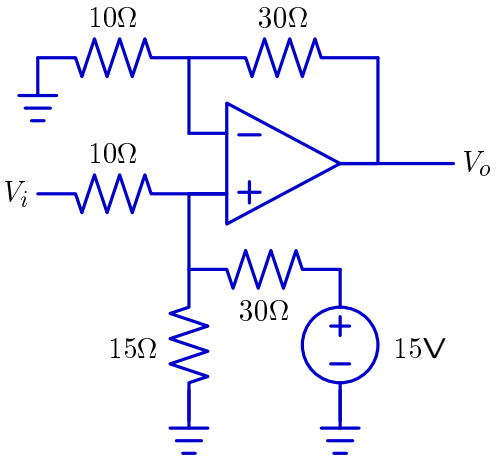
equivalent resistance =



equivalent resistance =

2 Op-Amp Circuit (10 points)

Consider the following circuit. Assume that the op amp is ideal, except that its output is limited by the power supply to $0 < V_o < 10V$.



Let V_+ and V_- represent the voltages at the positive and negative input terminals of the op amp.

Determine an expression for V_- in terms of V_o.

$V_- =$

Determine an expression for V_+ in terms of V_i.

$V_+ =$

Determine an expression for V_o in terms of V_i, assuming no limits on V_o.

$V_o =$

Determine the range of input voltages V_i for which the output of the op amp, V_o, is not saturated.

range=

3 OOPS (14 points)

You are writing the software version of a popular card game. Each player's objective is to gain levels; a player gains a level each time they defeat another player in battle. The winner of each battle is the player that currently has the highest "strength", where strength is the sum of the player's current level (indicating her experience) and the current strength of any items she possesses. In the case of a tie, the challenging player loses.

You have decided to use Python classes to represent the players and the items they can carry. You hire a Harvard student to write the code, but he never took 6.01, so his code is not perfect. Moreover, his dog ate part of the code in transit, so you have to fill in some missing pieces.

Here is his current code:

```
class Player:
  name = ""
  level = 1
  equipment = []
  potions = []
  def __init__(self, name):
    self.name = name
  def currentStrength(self):
    strength = self.level + sum([item.strength() for item in self.equipment])
    return strength
  def battle(self, other):
    myStrength = self.currentStrength()
    otherStrength = other.currentStrength()
    print self.name, "challenges", other.name
    print self.name, "has a total strength of", myStrength
    print other.name, "has a total strength of", otherStrength
    if myStrength > otherStrength:
      print self.name, "wins!"
      self.level += 1
      other.level -= 1
    else:
      print other.name, "wins!"
      other.level += 1
      self.level -= 1

class Item:
  coreStrength = 0
  def __init__(self, name):
    self.name = name
  def strength(self):
    return self.coreStrength

class BucklerOfSwashing(Item):
  coreStrength = 2

class FreezingExplosive(Potion):
  coreStrength = 5
```

Here is a brief interaction using this system:

```
>>>alyssa = Player("A. Python Hacker")
>>>ben = Player("Ben Bitdiddle")
>>>ben.battle(alyssa)
Ben Bitdiddle challenges A. Python Hacker
Ben Bitdiddle has a total strength of 1
A. Python Hacker has a total strength of 1
A. Python Hacker wins!

>>>alyssa.equipment.append(BucklerOfSwashing("excalibyr"))
>>>alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 4
Ben Bitdiddle has a total strength of 0
A. Python Hacker wins!
```

3.1 Where am I keeping things?

Someone points out that your Harvard friend is not very comfortable with the distinction between class and instance variables. He notes that in the above example, evaluating

```
for e in ben.equipment:
   print e.name
```

results in

```
excalibyr
```

which doesn't seem right, since `alyssa` is the person possessing this item, not `ben`.

Describe what is wrong with Santa's code, and indicate what you would change to correct this. Do this with minimal change to code.

3.2 I need a drink

Assuming that your correction in the previous part is made, we now need to worry about potions – unfortunately Rudolph ate the code for the Potion class. Potions are items that can only be used once (afterwards, they have no "strength"). If a person is engaged in a battle, and has an unused potion, it is consumed during the battle, contributing it strength to the overall strength. Potions thus have a 'usedUp' method, which returns a boolean indicating whether or not the potion has been used.

Our plan is to have the `Potion` class inherit from the `Item` class. To use potions, we need to decide how to compute its strength. If a potion inherits a strength method from the `Item` class, then it will always contribute to the strength computation. However, we want behaviors such as:

```
>>>alyssa = Player("A. Python Hacker")
>>>ben = Player("Ben Bitdiddle")
>>>ben.battle(alyssa)
Ben Bitdiddle challenges A. Python Hacker
Ben Bitdiddle has a total strength of 1
A. Python Hacker has a total strength of 1
A. Python Hacker wins!

>>>alyssa.equipment.append(BucklerOfSwashing("excalibyr"))
>>>ben.potions.append(FreezingExplosive())
>>>alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 4
Ben Bitdiddle has a total strength of 5
Ben Bitdiddle wins!

>>>alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 3
Ben Bitdiddle has a total strength of 1
A. Python Hacker wins!
```

Notice that in the last two battles, Alyssa loses the first one because Ben has a potion, which is expended. Thus in the next battle, Alyssa wins because Ben no longer has that strength.

Write the Potion class; be sure to inherit from the Item class, and do not repeat code unnecessarily.

Provide a new method below for `currentStrength` (in `Player`) to achieve the described behavior.

3.3 I need a lot of drinks

While some potions are "use once", it would be nice to have a potion that never runs out. Create a new class, called `InfinitePotion`, that inherits from the Potion class. This version of a potion never expires. For example:

```
class FreezingCanada(InfinitePotion):
  coreStrength = 5

>>>alyssa = Player("A. Python Hacker")
>>>ben = Player("Ben Bitdiddle")
>>>ben.battle(alyssa)
Ben Bitdiddle challenges A. Python Hacker
Ben Bitdiddle has a total strength of 1
A. Python Hacker has a total strength of 1
A. Python Hacker wins!

>>>alyssa.equipment.append(BucklerOfSwashing("excalibyr"))
>>>ben.potions.append(FreezingCanada())
>>>alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 4
Ben Bitdiddle has a total strength of 5
Ben Bitdiddle wins!

>>>alyssa.battle(ben)
A. Python Hacker challenges Ben Bitdiddle
A. Python Hacker has a total strength of 3
Ben Bitdiddle has a total strength of 6
Ben Bitdiddle wins!
```

Write the InfinitePotion class; be sure to inherit from the Potion class, and do not repeat code unnecessarily. You should be able to obtain the behavior above without changing any other code.

4 State machines (8 points)

Consider the following procedure:

```
def ImOdd(inList):
    outList = []
    y = 0
    for i in range(len(inList)):
        y = y +inList[i]
        if y%2 == 1:
            outList.append(None)
            y = 0
        else:
            outList.append(y)
    return outList
```

What is the value of

```
ImOdd([1,2,4,1,2,4,6,7,4,3,8,4,3])
```

Write a state machine class `OddSM` such that `OddSM().transduce(inList)` will return the same result as `ImOdd(inList)`, provided `inList` is a list of integers.

```
class OddSM(sm.SM):
    startState = 0
    def getNextValues(self, state, inp):
```

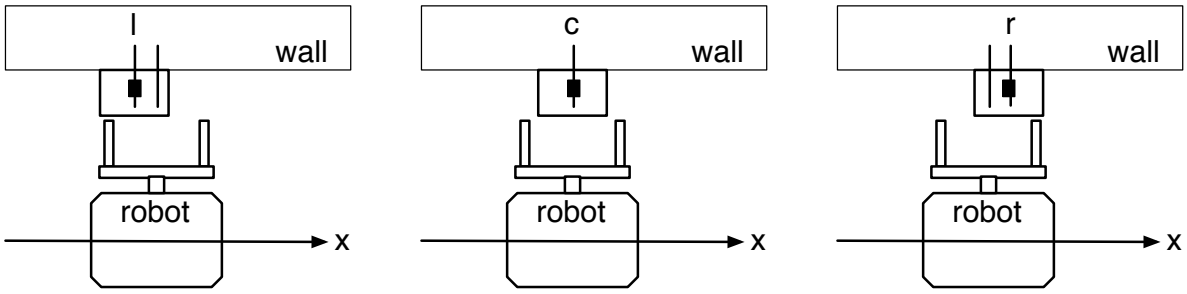
5 A touching experience (20 points)

Consider adding a gripper to the side of one of our robots as shown in the figure below. The gripper can be extended (toward the wall in the figure), retracted (away from the wall), and it can open and close.

We're trying to grasp a block of known width which is placed against a wall, but we're initially uncertain of the location of the block's center (the black square) relative to the center of the robot's gripper. We start with the gripper open to a width slightly wider than the block's width.

To keep things very simple, we'll assume that there are only three states, corresponding to when the block center is:

- a little to the left of the gripper center (call this `'l'`), or
- centered on the gripper (call this `'c'`), or
- a little to the right of the gripper center (call this `'r'`).



Note that these represent positions of the **center of the block relative to the center of the gripper**.

Assume that there's a sensor on each finger tip that can detect contact. The robot gets observations by extending the gripper forward a fixed distance which is sufficient to reach the wall; the gripper will stop moving when it hits either the wall or the block. If both sensors trigger then they are touching the wall behind the block and we can close the gripper and grasp. If only one sensor triggers, it is touching the block and we need to pull the gripper back, move the robot either right or left as appropriate and try again. The sensors are not perfect, they may miss a contact with some probability.

Let's say that the robot's moves to the left and right can be noisy; it might move to the adjacent location (in the desired direction) or it might leave the robot in its current location. Note that a move to the right from `l` or a move to the left from `r` leaves the robot in the same location.

To simplify things, the state of the robot will simply be represented by `'l'`, `'c'` or `'r'`, and we won't worry about commanding the robot to retract the gripper, move to the side, and then extend the gripper; we will simply treat these as all part of a move to the left or right.

5.1 Observation Model

Assume that each sensor has an independent probability of 0.25 of missing an actual contact. The sensors will never trigger without a contact.

Write a Python procedure to encode the **observation model** for this situation. Represent observations as a tuple of the form (L, R) where L is `True` if the left finger sensed a contact and R is `True` if the right finger sensed a contact.

```
def pOGivenS(s):
```

5.2 Belief update - left contact

For this part, assume that

- the probability of moving in the commanded direction is 0.5 and the probability of staying in the current state is 0.5.
- the probability of missing a contact on each finger is (independently) 0.25
- the initial belief state (for `('l', 'c', 'r')`) is $(1/4, 1/2, 1/4)$

We observe a single contact on the left finger (observation = (True, False)). What is the belief state after this observation (use fractions)?

`'l'`	`'c'`	`'r'`

Now, we command a motion to the right. What is the resulting belief state (use fractions)?

`'l'`	`'c'`	`'r'`

If you show your work neatly below, we may be able to give you partial credit.

5.3 Belief update - no contact

Make the same assumptions as above and starting from the **same initial belief state**: $(1/4, 1/2, 1/4)$.

We observe that neither finger detects a contact (observation = (False, False)). What is the belief state after this observation (use fractions)?

`'l'`	`'c'`	`'r'`

We command a motion to the right. What is the resulting belief state (use fractions)?

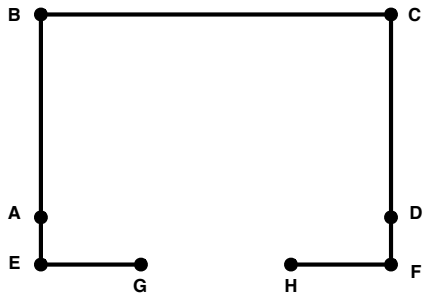
`'l'`	`'c'`	`'r'`

If you show your work neatly below, we may be able to give you partial credit.

6 Trail blazing (20 points)

You are trying to plan a path for a vehicle moving through an environment without obstacles. The environment has several cities, some of which are connected by straight roads. The vehicle starts at a city A and has to move to another city D. It can travel on-road or off-road. If it goes on a road of length d, then the cost is d. If it goes off-road for a distance d, then the cost is 2d.

Here is an example world, with lines indicating roads, and a matrix of **Euclidean** ("airline") distances between each pair of cities (this figure is drawn to scale):



	A	B	C	D	E	F	G	H
A	0.0	4.0	8.1	7.0	1.0	7.1	2.2	5.1
B	4.0	0.0	7.0	8.1	5.0	8.6	5.4	7.1
C	8.1	7.0	0.0	4.0	8.6	5.0	7.1	5.4
D	7.0	8.1	4.0	0.0	7.1	1.0	5.1	2.2
E	1.0	5.0	8.6	7.1	0.0	7.0	2.0	5.0
F	7.1	8.6	5.0	1.0	7.0	0.0	5.0	2.0
G	2.2	5.4	7.1	5.1	2.0	5.0	0.0	3.0
H	5.1	7.1	5.4	2.2	5.0	2.0	3.0	0.0

We assume that when moving between one city and another, the vehicle uses the road if it's available on its direct path. So, for example:

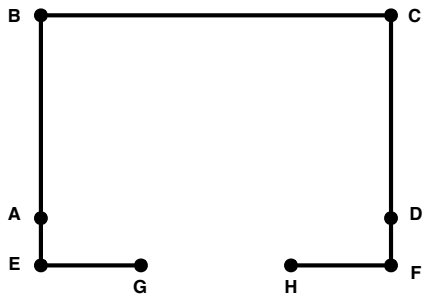
- The cost from E to A is 1
- The cost from E to F is $10 = \text{dist}(EG) + 2 * \text{dist}(GH) + \text{dist}(HF)$
- The cost from E to D is $14.2 = 2 * \text{dist}(ED)$

!!!! Note that the goal is city D !!!!

6.1 Cost

What is the cost of the path ABD

Scratch Paper



	A	B	C	D	E	F	G	H
A	0.0	4.0	8.1	7.0	1.0	7.1	2.2	5.1
B	4.0	0.0	7.0	8.1	5.0	8.6	5.4	7.1
C	8.1	7.0	0.0	4.0	8.6	5.0	7.1	5.4
D	7.0	8.1	4.0	0.0	7.1	1.0	5.1	2.2
E	1.0	5.0	8.6	7.1	0.0	7.0	2.0	5.0
F	7.1	8.6	5.0	1.0	7.0	0.0	5.0	2.0
G	2.2	5.4	7.1	5.1	2.0	5.0	0.0	3.0
H	5.1	7.1	5.4	2.2	5.0	2.0	3.0	0.0

6.2 Depth-First with DP

Some assumptions for this part:

- The AD off-road path is not allowed; all other paths on and off road are allowed.
- The successors (children) of a state are pushed onto the agenda in REVERSE ALPHABETICAL order.

What is the path found from A to D by Depth First search with DP? What is its cost? What states are expanded by the search.

- Path:
- Cost:
- Expanded:

6.3 Breadth First with DP

Some assumptions for this part:

- The AD off-road path is not allowed; all other paths on and off road are allowed.
- The successors (children) of a state are pushed onto the agenda in ALPHABETICAL order.

What is the path found from A to D by Breadth First search with DP? What is its cost? What states are expanded by the search?

- Path:
- Cost:
- Expanded:

6.4 Uniform Cost with DP

Some assumptions for this part:

- All paths on and off road are allowed.
- The successors (children) of a state are pushed into the agenda in ALPHABETICAL order.

What is the path found from A to D by Uniform Cost search with DP? What is its cost? What states are expanded by the search?

- Path:
- Cost:
- Expanded:

6.5 Heuristic

Describe a (non-zero) admissible heuristic for this domain.

7 System functions (16 points)

Each of the following systems have one input signal $x[n]$ and one output signal $y[n]$. Determine expressions for the poles of each system Also determine all values of the parameter k for which the system is stable (or write "none" if there is no choice for which the system is stable).

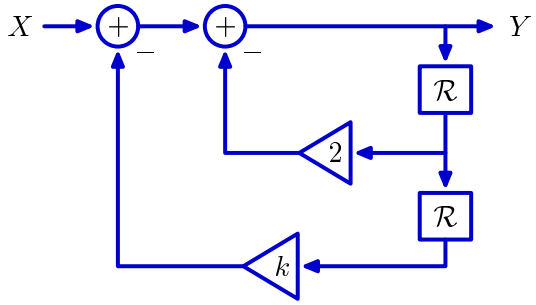
System 1:

$$y[n] = x[n] + u[n] + v[n]$$
$$u[n] = ky[n-1]$$
$$y[n] = v[n-1]$$

poles:

stable k:

System 2:



poles:

stable k:

System 3:

```
sf.FeedbackSubtract(sf.Cascade(sf.Gain(k),sf.FeedbackAdd(sf.R(),sf.Gain(1))),sf.Gain(1))
```

poles:

stable k:

System 4:

$$\frac{Y}{X} = \frac{R^2}{(1-3R)(1-2kR+4R^2)}$$

poles:

stable k:

6.01SC Introduction to Electrical Engineering and Computer Science

Spring 2011