CS1010S — Programming Methodology
National University of Singapore

Practical Examination

16 November 2013

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**.
2. This practical exam consists of **four** questions, out of which you are expected to answer **three**. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks** and will consist of the best three scores out of the answers for the four questions (if all four questions are attempted). Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. The total score for this quiz is capped at **30 marks**. Marks in excess of your practical exam grade (because you answer more than 3 questions) will be added to the marks for the midterm exams, which is capped at **100 marks**.
5. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
6. You should use the template `practical-template.py` provided, but rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number. Please double check to ensure that there is no mistake. You are to upload your answers to IVLE Practical Exam folder at end of the exam and you should only upload one file. If you upload multiple files, we will choose one at random for grading. Marks will be deducted if you fail to follow these instructions.
7. Please note that it shall be your responsibility to ensure that the correct file is uploaded to IVLE at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam. You have been warned.
8. Please note that while sample executions are given, it is not sufficient to write programs that simply satisfy the given examples. Your programs will be tested on other inputs and they should exhibit the required behaviours to get full credit.

GOOD LUCK!

Question 1 : Counting Letters [10 marks]

A. Write a function `letter_count` that takes in number of words (strings) and returns a dictionary with the frequency counts of the various letters. Upper and lower-case characters are considered to be different characters. [5 marks]

Sample execution:

```
>>> letter_count("this")
{'s': 1, 'h': 1, 'i': 1, 't': 1}

>>> letter_count("this","is")
{'s': 2, 'h': 1, 'i': 2, 't': 1}

>>> letter_count("this","Is")
{'s': 2, 'h': 1, 'i': 1, 'I': 1, 't': 1}

>>> letter_count("this","is","a")
{'s': 2, 'h': 1, 'i': 2, 'a': 1, 't': 1}

>>> letter_count("this","is","a","good")
{'h': 1, 'i': 2, 'o': 2, 'd': 1, 's': 2, 'a': 1, 'g': 1, 't': 1}

>>> letter_count("this","is","a","good","day")
{'h': 1, 'i': 2, 'o': 2, 'd': 2, 's': 2, 'a': 2, 'y': 1, 'g': 1, 't': 1}

>>> letter_count("baa", "baa","banana")
{'b': 3, 'a': 7, 'n': 2}
```

B. Write a function `most_frequent` that takes in number of words (strings) and returns a list of the most frequently-occurring letters in the words. Upper and lower-case characters are considered to be different characters. **Hint:** You probably want to re-use `letter_count` from Part A. [5 marks]

Sample execution:

```
>>> most_frequent("this")
['i', 'h', 's', 't']

>>> most_frequent("this","is")
['i', 's']

>>> most_frequent("this","Is")
['s']

>>> most_frequent("this","is","a")
['i', 's']
```

```
>>> most_frequent("this","is","a","good")
['i', 'o', 's']

>>> most_frequent("this","is","a","good","day")
['a', 'd', 'i', 'o', 's']

>>> most_frequent("baa", "baa","banana")
['a']
```

Note: Part B is the *real* question. Part A is really a hint to help students along. Students will get full credit for this question if they are able to get Part B right, even if there are mistakes in Part A.

Question 2 : Exam Time!! [10 marks]

You are given two data files `exams.csv` and `modules.csv`, both in CSV format. `exams.csv` contains the final exam date information while `modules.csv` contains CORS bidding information.

You are to write a function `highest_quota_no_exams` that takes as input the two files and an integer k returns a list of the k module codes of the modules with the highest quota for a particular Group (column 2 of `modules.csv`) that has *no scheduled exams*. Group can refer to a given lecture or tutorial group. Note that to get the quota for a particular Group in `modules.csv`, you are expected to sum the quotas for the Group.

If there are ties, i.e. the $(k+1)$ module has the same quota as the kth module, you should return as many modules as it takes to include all the modules with the same quota as the kth module.

You are provided with two sets of `exams.csv` and `modules.csv` files and the following is the expected results for the two sets of files. Note that your code will be tested with other data files with the same input format, so just because your code works with the files provided does not necessarily mean that your code is necessarily correct.

Sample execution:

```
>>> highest_quota_no_exams("exams1.csv","modules1.csv",3)
('CS1280', 'CS1281', 'CS2250')

>>> highest_quota_no_exams("exams2.csv","modules2.csv",5)
('CS1281', 'CS2250', 'CS2261', 'CS3243', 'CS1105')
```

Hint: instead of trying to solve this problem directly, you might wish to solve the following two problems separately, before combining them into your final solution:

1. What is the list of modules that do not have final exams scheduled?
2. How do we find the groups with the highest quota?

Hint 2: You probably want to check your work as you go along instead of writing all your code and trying to debug everything at once. Consider yourself warned.

Question 3 : Inception – *Back by Popular Demand!* [10 marks]

— BACKGROUND STORY (Okay to skip) —

Dominick Cobb (Leonardo DiCaprio) and business partner Arthur (Joseph Gordon-Levitt) perform corporate espionage using an experimental military technology to infiltrate the subconscious of their targets and extract information while experiencing shared dreaming. Their latest target is Japanese businessman Saito (Ken Watanabe). Tiered "dream within a dream" strategies are used and dreamers awaken by a "kick" such as dying in the dream or falling. If the dreamer is the one who awakens, the dream "collapses". Each "extractor" carries a totem, a small object whose behavior only its owner can predict, used to determine whether a dreamer is in someone else's dream. Cobb's totem is implied to be a spinning top that spins perpetually in the dream state. The extraction from Saito fails when sabotaged by a memory of Cobb's deceased wife Mal (Marion Cotillard). Saito reveals, after Cobb's and Arthur's associate sells them out, that he was actually auditioning the team to perform the difficult act of "inception": planting an idea in a person's subconscious.

Saito wishes to break up the energy conglomerate of his ailing competitor Maurice Fischer (Pete Postlethwaite) by planting the idea in his son and heir Robert Fischer (Cillian Murphy) to disintegrate his father's company. Should Cobb succeed, Saito would use his influence to clear a murder charge against Cobb, so he can return to the United States and his children. Cobb accepts the offer and assembles his team: Eames (Tom Hardy), a conman and identity forger; Yusuf (Dileep Rao), a chemist who concocts the powerful sedative for a stable "dream within a dream" strategy; Ariadne (Ellen Page), an architecture student tasked with designing the labyrinth of the dream landscapes; and Arthur. Saito accompanies so he can verify the team's success.

When the elder Fischer dies in Sydney and his body is flown back to Los Angeles, the team share the flight with Robert Fischer. Cobb sedates him, bringing him into a shared dream with the extractors. At each level in the layered dreaming, the person generating the dream stays behind to set up a "kick" that will be used to awaken the other, sleeping team members who have entered another dream layer deeper. In the first level, Yusuf's rainy downtown dream, the team abducts Fischer; however, Fischer's trained subconscious projections attack, wounding Saito severely. Eames temporarily takes the appearance of Fischer's godfather, Peter Browning (Tom Berenger), to suggest Fischer reconsider his father's will. Yusuf drives the team in a van as they are sedated into the second level, a hotel dreamed by Arthur. Here, the extractors recruit Fischer, convincing him that his kidnapping was orchestrated by Browning and that they are Fischer's subconscious defence. In the third level, a snowy mountain fortress dreamed by Eames, Fischer is told they are in Browning's subconscious, but they are really going deeper into Fischer's. Yusuf, under assault in the first level, initiates his kick too soon by driving off a bridge, removing the gravity of Arthur's dream world and causing an avalanche in Eames' dream, a kick which all missed. Arthur improvises a new kick using an elevator that will be synchronized with the van hitting the water, while the team in Eames' dream races to finish the job before the new round of kicks.

Due to the effects of heavy sedation and multi-layered dreaming, death during this mission will result in entering Limbo, dream space of unknown content where the dreamer could be trapped. Elapsed time in each dream level is roughly 20 times greater than in the level above it; in Limbo, the deepest level of all, the planned ten hours of outer-world time would be experienced as almost two centuries. Cobb reveals to Ariadne that he spent "50 years" with Mal in Limbo constructing a world from their shared memories while seemingly growing old together. Returning to the waking world, they found less than three hours had passed, but Mal, convinced she was still dreaming, committed suicide, all the while trying to persuade Cobb to do so by incriminating him in her death. He fled the U.S. and left his children behind, ostensibly in the care of his father-in-law, Prof. Stephen Miles (Michael Caine). Saito succumbs to his wounds, and Cobb's projection of Mal sabotages the plan by killing Fischer, sending them both into Limbo. Cobb and Ariadne enter Limbo to find Fischer and Saito, while Eames remains on his dream level to set up a kick by rigging the fortress with explosives. Cobb confronts his projection of Mal, who tries convincing him to stay in Limbo. Cobb refuses and confesses guilt for Mal's suicide: using inception to plant the idea in her mind that the world they had been living in for 50 years was not real, Cobb had convinced her to leave Limbo with him through suicide, but once back in the real world, she was unable to discern dreaming from reality. Mal attacks Cobb but Ariadne shoots her. Through his confession, Cobb attains catharsis and chooses to remain in Limbo to search for Saito. Ariadne pushes Fischer off a balcony, resuscitating him at the mountain fortress, where he enters a safe room to discover and accept the planted idea: that his father wishes him to be his "own man", and that splitting up the conglomerate might not be a radical notion.

All team members other than Cobb and Saito ride the synchronized kicks back to reality: Ariadne jumps off a balcony in Limbo, Eames detonates the explosives in the fortress, Arthur blasts an elevator containing the team's sleeping bodies up an elevator shaft, and the van in Yusuf's dream hits the water. Cobb eventually finds an aged Saito in Limbo and the two remember their arrangement, presumably shooting themselves and awakening to outer-world reality on the airplane. Saito honors the arrangement and Cobb passes through U.S. customs once the plane lands in Los Angeles. Before reuniting with his children, Cobb tests reality with his spinning top, but he turns away to greet them before observing the results.

– Inception 2010, source: Wikipedia

— START OF ACTUAL QUESTION —

In this problem, you will implement two objects `Dream` and `Person` to model the world of *Inception*. In this world, a person can dream and other persons can enter the same dream. Furthermore, it is possible to have a dream within a dream and for the people who are in the same dream to enter those dreams. In Figure 1, we have a simple graphical representation of the actors and dreams in *Inception* to illustrate this. The dream bubbles indicate the *main* dreamer for each dream.

When a dreaming person receives a kick, he will be kicked out of his current highest level dream. If the person kicked out of a dream is the main dreamer of the dream, that dream collapses and all the other people in the dream will get kicked out as well. If there are higher level dreams, those dreams will also end. The various new configurations if different persons in Figure 1 are kicked are illustrated in Figures 2 to 5.

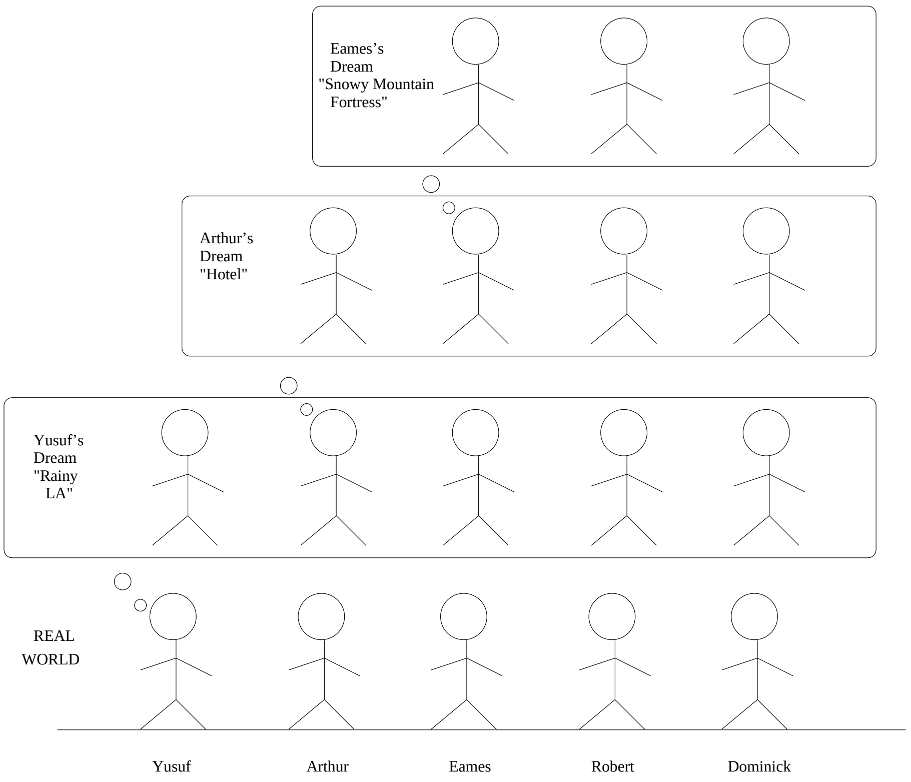


Figure 1: Dream model for *Inception*.

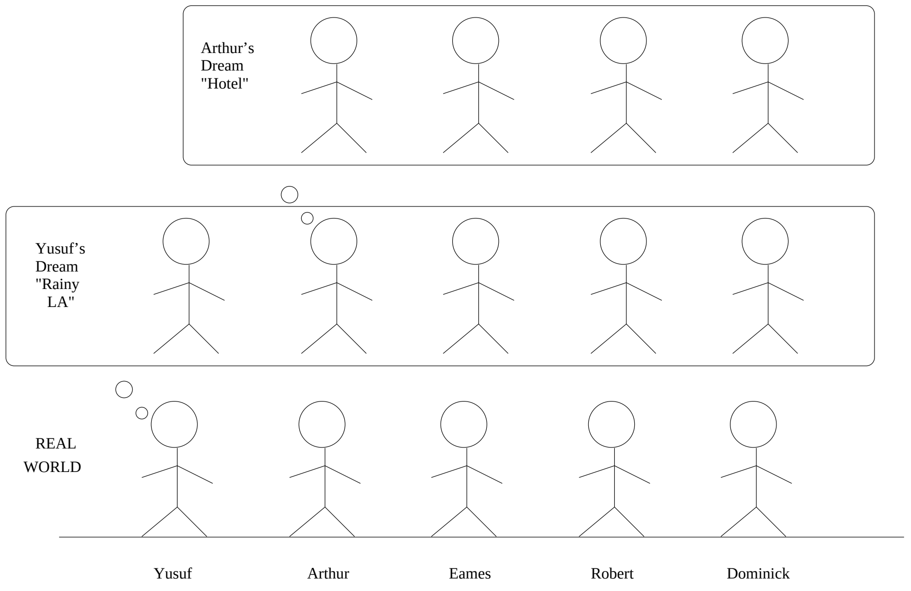


Figure 2: New configuration for Figure 1 if Eames gets kicked.

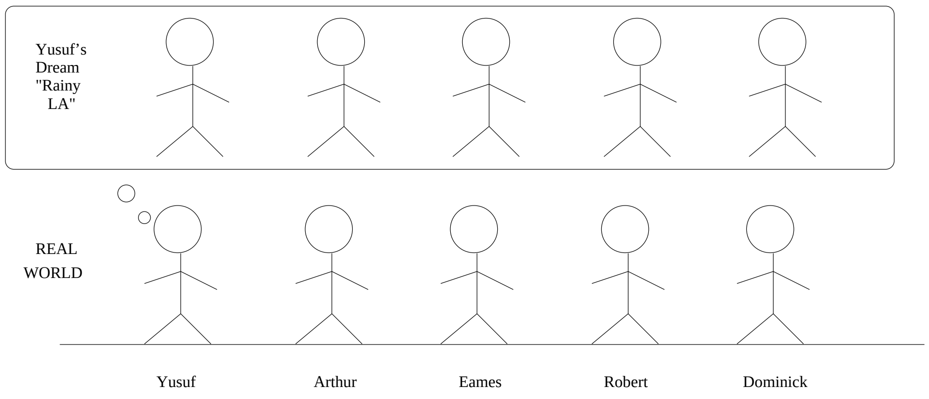


Figure 3: New configuration for Figure 1 if Arthur gets kicked.

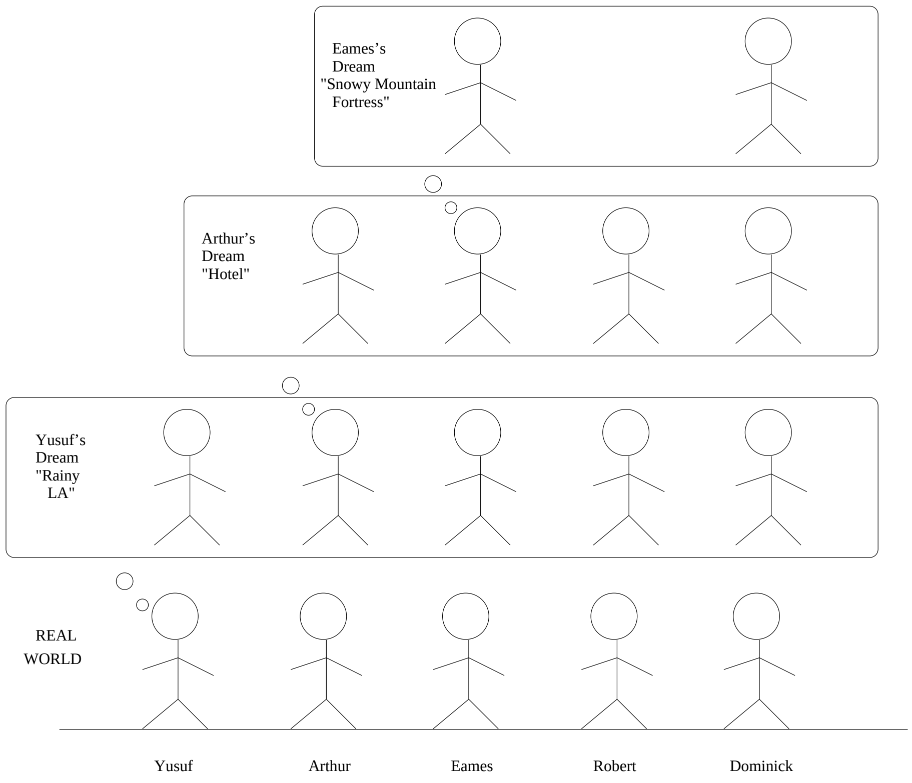


Figure 4: New configuration for Figure 1 if Robert gets kicked.

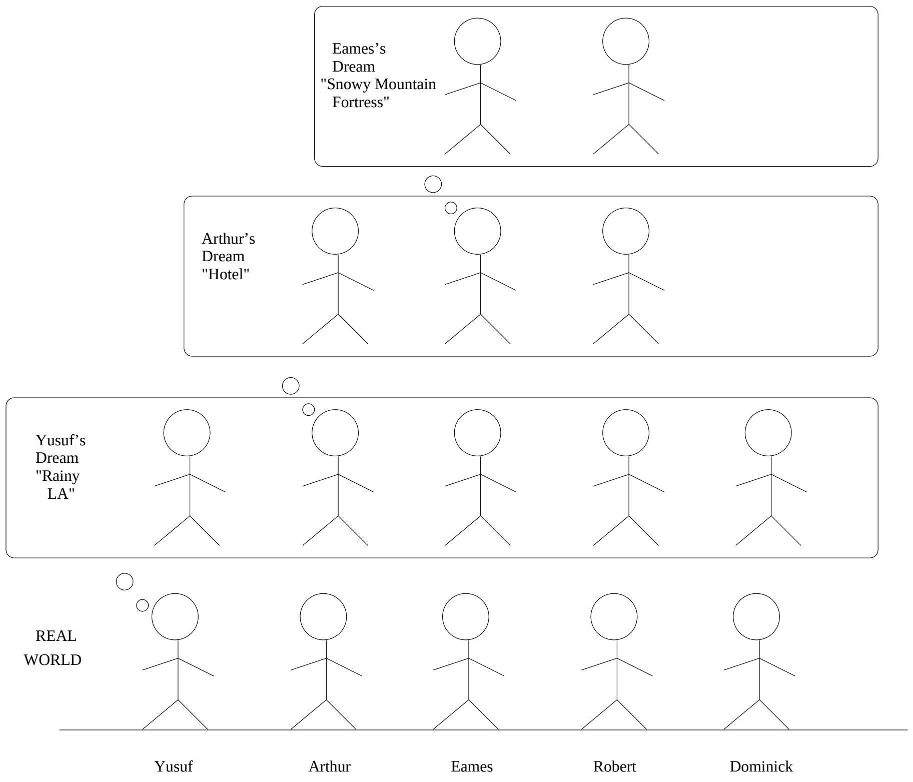


Figure 5: New configuration for Figure 1 if Dominick gets kicked **twice**.

It is possible for the same person to dream again at a higher level as illustrated in Figure 6, and it is possible for different people to have different dreams simultaneously. However, if two people are not in the same higher level dream, then they cannot be in the same even higher level dream. For example, in Figure 6, Robert cannot be a part of Yusuf's dreams because he is already in Dominick's "Camping" dream.

The `Person` object is created with a name (String) as input and supports the following **four** methods:

1. `get_name()`: will return person's name.
2. `dream_level()`: will return the level of the highest level dream that the person is currently in. `dream_level()` will return 0 for a person who is not currently dreaming (duh!).
3. `dream(<Person A>, <Person B>, ···, <Person B>)` : this returns a new `Dream` at the next higher level for this person, that will also include the persons who are arguments to the new dream. If one or more of the persons included as arguments is not a valid person for the new dream, this method will fail by returning `None`, without modifying anything. To be a valid argument, a person must currently be in the same highest-level dream as the main dreamer, or both of them must not be dreaming.
4. `kick()`: this will kick the person out of his current highest-level dream. If the person is the main dreamer of that dream, that dream will collapse and all the other persons involved will get kicked out also, and their higher-level dreams will all collapse.

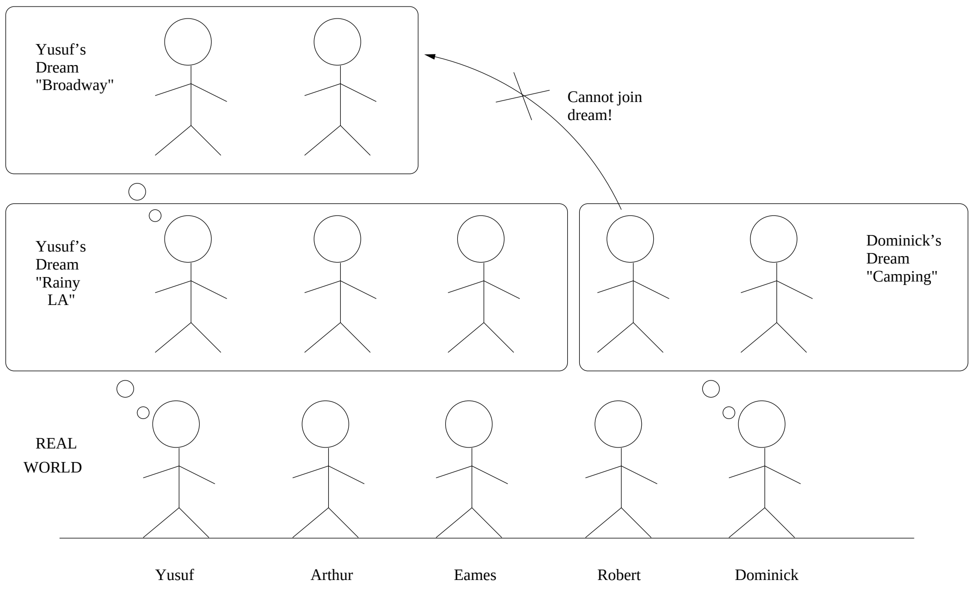


Figure 6: Persons with separate dreams.

Since the `Dream` object is created by the `dream(···)` method in `Person`, you get to decide on the input arguments. `Dream` however has to support the following **two** methods:

1. `dreamers()`: will return a list of the current dreamers.
2. `active()`: will return `True` if the current dream is still active, or `False` otherwise. A collapsed dream (because the main dreamer got kicked) is not active (of course!)

Note: You are allowed to define your own methods for `Person` and `Dream` in addition to the specified methods as you see fit. All the state in this question should be contained within the `Person` and `Dream` classes. You are **not allowed** to use **global** variables.

Hint: This question looks complicated, but it is not quite as difficult as it looks. Take some time to think about what state you need to keep in order to keep track of the state of the various objects before you start coding. More thinking, less coding!

Sample execution:

```
>>> yusuf = Person("Yusuf")
>>> eames = Person("Eames")
>>> arthur = Person("Arthur")
>>> robert = Person("Robert")
>>> dominick = Person("Dominick")
>>> jewels = Person("Jewels")


>>> yusuf.dream(eames,arthur,robert,dominick)
>>> yusuf.get_name()
Yusuf
```

```
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
1
>>> robert.dream_level()
1
>>> eames.dream_level()
1
>>> dominick.dream_level()
1

>>> dream = arthur.dream(jewels,eames,robert,dominick) # Jewels not in dream!
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
1
>>> robert.dream_level()
1
>>> eames.dream_level()
1
>>> dominick.dream_level()
1
>>> dream
None

>>> dream = arthur.dream(eames,robert,dominick)
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
2
>>> robert.dream_level()
2
>>> eames.dream_level()
2
>>> dominick.dream_level()
2
>>> dream.active()
True

>>> dream = jewels.dream(robert,dominick) # Jewels not in dream!
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
2
>>> robert.dream_level()
2
```

```
>>> eames.dream_level()
2
>>> dominick.dream_level()
2
>>> dream
None

>>> dream = eames.dream(robert,dominick)
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
2
>>> robert.dream_level()
3
>>> eames.dream_level()
3
>>> dominick.dream_level()
3
>>> dream.active()
True

>>> arthur.kick() # Major collapse
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
1
>>> robert.dream_level()
1
>>> eames.dream_level()
1
>>> dominick.dream_level()
1
>>> dream.active()
False

>>> arthur.kick() # One man kicked out
>>> yusuf.dream_level()
1
>>> arthur.dream_level()
0
>>> robert.dream_level()
1
>>> eames.dream_level()
1
>>> dominick.dream_level()
1
```

```
>>> yusuf.kick() # Major collapse
>>> yusuf.dream_level()
0
>>> arthur.dream_level()
0
>>> robert.dream_level()
0
>>> eames.dream_level()
0
>>> dominick.dream_level()
0
```

Question 4 : Logic Gate Mania (*Nasty*) [10 marks]

Logic gates compute boolean functions. There are three common types of logic gates, AND, OR and NOT. The following are the truth tables for the AND and OR gates:

AND	Input 2		Output
Input 1	False	False	False
	False	True	False
	True	False	False
	True	True	True

OR	Input 2		Output
Input 1	False	False	False
	False	True	True
	True	False	True
	True	True	True

The NOT gate has one input and simply inverts the input. These gates can be connected together to form larger boolean operator objects.
In this question, you will implement three different classes `AndGate`, `OrGate` and `NotGate`. The first two will support the following methods:

1. `connect (position, gate)`: This will connect the `gate` object at the input indicated by the position (which can be 0 or 1) and return the new boolean operator object. Note that an output can only be connected to **one** input that is not already connected to another output. Attempts to make an invalid connection will return `None`.

2. `compute (boolean_tuple)`: This will compute the boolean function of the current boolean operator object that this gate is attached to. The boolean tuple is assumed to have a number of terms equal to the number of inputs for the current boolean operator object.

The NOT gate supports two similar methods except it does not have a position argument in `connect` since it has only one input. While unconnected gates will simply compute the simple truth table, connected gates will all refer to the same boolean operator object that they are connected to and the `compute` function must be given a tuple of boolean values with the correct number of arguments corresponding to the number of inputs.

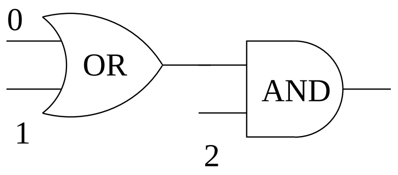
The ordering of the input boolean tuple is well-defined according to the inputs of the gates, for example, the 0th input of a AND gate will come before the 1th input. The following sample execution should make this uniqueness clear.

Important Note: Little to no credit will be given for the implementation of naive (non-connected) gates, because that's completely trivial. This challenge in this question is to be able to deal with connecting gates together.

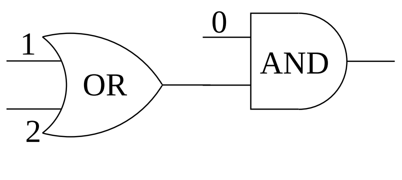
Sample execution:

```
>>> and1 = AndGate()
>>> and1.compute([False,False])
False

>>> and1.compute([False,True])
False

>>> and1.compute([True,False])
False
```

```
>>> and1.compute([True,True])
True

>>> notg = NotGate()
>>> notg.compute([False])
True

>>> notg.compute([True])
False

>>>or_and = AndGate().connect(0,OrGate())
```



```
>>> or_and.compute([False,False,False])
False

>>> or_and.compute([False,False,True])
False

>>> or_and.compute([False,True,False])
False

>>> or_and.compute([False,True,True])
True

>>> or_and.compute([True,False,False])
False

>>> or_and.compute([True,False,True])
True

>>> or_and.compute([True,True,False])
False

>>> or_and.compute([True,True,True])
True

or_and2 = AndGate().connect(1,OrGate())
```



```
>>> or_and2.compute([False,False,False])
False

>>> or_and2.compute([False,False,True])
False

>>> or_and2.compute([False,True,False])
False

>>> or_and2.compute([False,True,True])
False

>>> or_and2.compute([True,False,False])
False

>>> or_and2.compute([True,False,True])
True

>>> or_and2.compute([True,True,False])
True

>>> or_and2.compute([True,True,True])
True
```

Holiday Fun (not graded – to be attempted at own time): Once you've figured out how to do this, consider solving a multiple output version of this problem where we can tap the output of a gate to use as inputs for several different gates. This means that the output will be a list of booleans. We will however have to come up with a convention to ensure that the ordering of the outputs is uniquely-defined. For this advanced problem, memoization is probably helpful as well.