CS1010X — Programming Methodology
National University of Singapore

Practical Examination

8 June 2019

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**.
2. This practical exam consists of **three** questions. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks** and you will need to answer all three questions to achieve the maximum score. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. 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.
5. Your answers should be submitted on Coursemology.org. You will be allowed to run some public tests cases to verify that your submission is correct. Note that you can run the test cases on Coursemology.org **up to 5 times** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness using IDLE and not on Coursemology.org. Do however ensure that you submit your answers correctly by running the test cases at least once.
6. You are also provided with the template `practical-template.py` to work with. If Coursemology.org fails, you will be required to rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number and submit that file.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly. 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 as specified by the problems to get full credit. There is no need for you to submit test cases.

GOOD LUCK!

Question 1 : Smallest Number [10 marks]

A. [**Warm-Up**] Write a function `smallest` that takes an arbitrary number of integer digits $(0,\cdots,9)$ and returns the smallest integer that can be constructed using all the digits. The returned integer must contained all the digits, so it clearly cannot begin with 0, unless the number is 0. [5 marks]

Sample execution:

```
>>> smallest(9,1,3)
139

>>> smallest(1,3,9,0,0)
10039

>>> smallest(2,1,1,3,9,0)
101239
```

B. Write a function `second_smallest` that takes an arbitrary number of integer digits $(0,\cdots,9)$ and returns the second smallest integer that can be constructed using all the digits. The returned integer must contained all the digits, so it clearly cannot begin with 0, unless the number is 0. If there is no possible second smallest integer, i.e. there is only one possible integer, then return `None`. [5 marks]

Sample execution:

```
>>> second_smallest(9,1,3)
193

>>> second_smallest(1,3,9,0,0)
10093

>>> second_smallest(2,1,1,3,9,0)
101293

>>> second_smallest(1,1,1)
None
```

Question 2 : Picking the Best Course [10 marks]

Your cousin has just completed her A-Levels and she's trying to decide which course to study for college. She comes to you for advice. Since you learnt how to process data in CS1010X, you decided that instead of giving her your own opinions, you would instead use your newfound skills to study the available data to help her with the decision.

The data was provided as comma-separated file `graduates-by-first-degree.csv`, where each entry contains the following information:

- year
- gender
- course
- number of graduates

Some rows are invalid and have `"na"` in the number of graduates column. You should ignore these rows. Note that you cannot assume that your code will be run on the `graduates-by-first-degree.csv`file provided. Your code will also be tested on private data sets with the same format.

A. **[Warm Up]** Write a function `most_common_major`, that takes in the name of the csv file containing the data and a specified year and returns major that had the most number of graduates in the specified year. If there is no data for the specified year, return `None`. [3 marks]

Sample execution:

```
>>> most_common_major("graduates-by-first-degree.csv", 1993)
"Engineering Sciences"

>>> most_common_major("graduates-by-first-degree.csv", 2000)
"Engineering Sciences"

>>> most_common_major("graduates-by-first-degree.csv", 2010)
"Engineering Sciences"
```

B. **[New Courses]** Your cousin would like to understand what new courses were introduced over the years. Write a function `new_courses`, that takes in the name of the csv file containing the data, a start year and an end year and returns a list of the new courses that are introduced in the period with the year of introduction. Basically, these courses should <u>not</u> have any enrollment up to the start year (inclusive), but have non-zero enrollment at some point after the start year and before the end year (inclusive). [3 marks]

Sample execution:

```
>>> new_courses("graduates-by-first-degree.csv",1993,2000)
[('Education', 1995), ('Mass Communication', 1997)]
```

```
>>> new_courses("graduates-by-first-degree.csv",2001,2010)
[('Applied Arts', 2003), ('Services', 2008)]

>>> new_courses("graduates-by-first-degree.csv",1993,2020)
[('Education', 1995), ('Mass Communication', 1997),
('Applied Arts', 2003), ('Services', 2008)])
```

C. **[Top-*k* Promising Courses]** After some discussion with your cousin, you decide that a new course doesn't mean that it is a good course. A better way to choose might be to see which courses have had the highest growth in enrollment over a period to predict rising popularity. Write a function `topk_growing_major`, that takes in the name of the csv file containing the data, a positive integer *k*, a start year and an end year and returns a list of the top *k* courses with the biggest increase in growth (in percentage terms) between the start year and the end year. They should be ordered in descending order. Basically, these courses should have non-zero enrollment in both the start year and end year (inclusive). Function will return an empty list if there are no courses between the specified years. **[4 marks]**

Sample execution:

```
>>> topk_growing_major("graduates-by-first-degree.csv",3,1993,2000)
['Engineering Sciences', 'Dentistry', 'Humanities & Social Sciences']

>>> topk_growing_major("graduates-by-first-degree.csv",2,2000,2010)
['Health Sciences', 'Education']

>>> topk_growing_major("graduates-by-first-degree.csv",3,2000,2014)
['Health Sciences', 'Education', 'Law']
```

Question 3 : Avengers: EndGame [10 marks]

— BACKGROUND (Okay to skip) —
(Possible Spoilers!)

Twenty-three days after Thanos used the Infinity Gauntlet to disintegrate half of all life in the universe, Carol Danvers (Captain Marvel) rescues Tony Stark and Nebula from deep space and returns them to Earth. They reunite with the remaining Avengers Bruce Banner, Steve Rogers, Rocket, Thor, Natasha Romanoff, and James Rhodes and find Thanos on an uninhabited planet. They plan to retake and use the Infinity Stones to reverse the disintegrations, but Thanos reveals he destroyed them to prevent further use. An enraged Thor decapitates Thanos.

Five years later, Scott Lang escapes from the quantum realm. He travels to the Avengers compound, where he explains to Romanoff and Rogers that he experienced only five hours while trapped. Theorizing the quantum realm could allow time travel to the past, the three ask Stark to help them retrieve the Stones from the past to reverse Thanos' actions in the present, but Stark refuses to help. After talking with his wife, Pepper Potts, Stark relents and works with Banner, who has since merged his intelligence with the Hulk's strength, and the two successfully build a time machine. Banner warns that changing the past does not affect their present and any changes instead create branched alternate realities. He and Rocket go to the Asgardian refugees' new home in Norway - New Asgard - to recruit Thor, now an overweight alcoholic, despondent over his failure in stopping Thanos. In Tokyo, Romanoff recruits Clint Barton, now a ruthless vigilante following the disintegration of his family.

Banner, Lang, Rogers, and Stark travel to New York City in 2012. Banner visits the Sanctum Sanctorum and convinces the Ancient One to give him the Time Stone. Rogers successfully retrieves the Mind Stone, but Stark and Lang inadvertently allow 2012 Loki to escape with the Space Stone. Rogers and Stark travel to S.H.I.E.L.D. headquarters in 1970, where Stark obtains an earlier version of the Space Stone and encounters a young Howard Stark in the process, while Rogers steals several Pym Particles from Hank Pym to return to the present. Rocket and Thor travel to Asgard in 2013, extracting the Reality Stone from Jane Foster and retrieving Thor's hammer, Mjolnir. Nebula and Rhodes travel to Morag in 2014 and steal the Power Stone before Peter Quill can. Rhodes returns to the present with the Power Stone, but Nebula is incapacitated when her cybernetic implants link with those of her past self. Through this connection, 2014 Thanos learns of his future success and the Avengers' attempts to undo it. Thanos captures her and sends past-Nebula to the present in the former's place. Barton and Romanoff travel to Vormir, where the Soul Stone's keeper, the Red Skull, reveals it can only be acquired by sacrificing someone they love. Romanoff sacrifices herself, allowing Barton to obtain the Soul Stone.

Reuniting in the present, the Avengers fit the Stones into a Stark-created gauntlet, which Banner uses to resurrect all those whom Thanos disintegrated. Past-Nebula uses the time machine to transport past-Thanos and his warship to the present, where

> *he attacks the Avengers' compound, planning to destroy and then rebuild the universe with the Stones. Nebula convinces past-Gamora to betray Thanos and kills her past self. Stark, Rogers, and Thor fight Thanos but are outmatched. Thanos summons his army to devastate the Earth, but a restored Stephen Strange arrives with other sorcerers, the restored Avengers and Guardians of the Galaxy, the armies of Wakanda and Asgard, and the Ravagers to fight Thanos and his army alongside Danvers, who destroys Thanos' warship as she arrives. After overpowering the heroes, Thanos seizes the gauntlet, but Stark steals the Stones back and uses them to disintegrate Thanos and his army, then dies from the energy emitted in the process.*
>
> *Following Stark's funeral, Thor appoints Valkyrie as the king of New Asgard and joins the Guardians of the Galaxy, while Quill searches for 2014 Gamora. Rogers returns the Infinity Stones and Mjolnir to their original places in time and remains in the past to live with Peggy Carter. In the present, an elderly Rogers passes on his shield and mantle to Sam Wilson.*
>
> **– Source: Wikipedia**

Time travelling is tricky business. In this problem, we will implement a simple one reality model of time travelling for the Marvel universe. In our model, there are 2 objects (classes): `Timeline` and `Person`.

A new `Timeline` is created without any arguments. It supports 2 functions (though you are free to add helper functions as you see fit):

- `born(name, year, lifespan)` returns a new `Person` object for a person with the specified name, to be born in the specified year and with a specified lifespan (in years).

- `get_people(year)`: returns a list of the people who exist in the specified year in the timeline. The list is a list of pairs of name and "identity". The identity term allows us to specify specific instances of a person existing at the same time (for example, to disambiguate between a 'Thor' belonging to the current timeline, and another 'Thor' from the future that has travelled back in time). This should be clear from the sample execution below.

The `Person` class supports supports 2 functions (though you are free to add other functions if you think they are helpful for you):

- `jump(from_year, to_year, identity)` allows the person to make a jump in the timeline from the `from_year` to the `to_year`. Because there can be many versions of the same person in a given year, we use the `identity` field to disambiguate between them. The "identity" of the new version of the same person created by the jump is the `from_year`. You can assume that the "identity" of all persons NOT created by a jump (i.e. just `born` is the year in which they were born.)

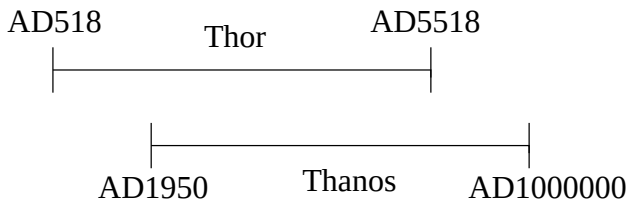
- `kill(year, person, identity)` kills the specified person in the specified year. Again `identity` is used to disambiguate different version of the person who can be killed in the specified year. Return `True` if successful, `False` otherwise. Killing can fail if the killer is not actually around during the specified time or if the victim is not around during that time. Just because a person is available at the specified

year doesn't mean that he will be killed if the specified version of him/her is not available during that time.

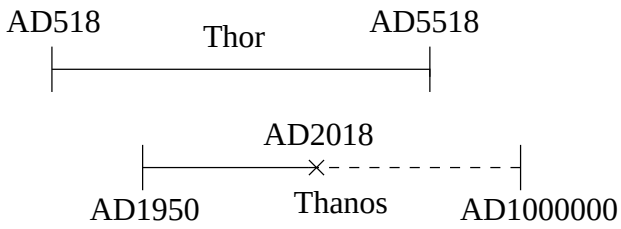
Note that a jump or a death would invalidate future events (both death and jumps). The timeline after a jump or death is reset to the nature lifespan, minus any jumps or death. To keep things consistent and simple, the semantics is not inclusive of the jump/death year once a jump/death happens. Basically a person is no longer in the `from_year` once he jumps and he appears in the `to_year`. When a person is killed in Year X, he is no longer present when we do `timeline.get_people(X)`. We adopt the convention that time segments are not inclusive of the `to` field. This will be clear in the sample execution below.

You are to make sure that you understand what needs to be done and how it all works before you start working on the problem.

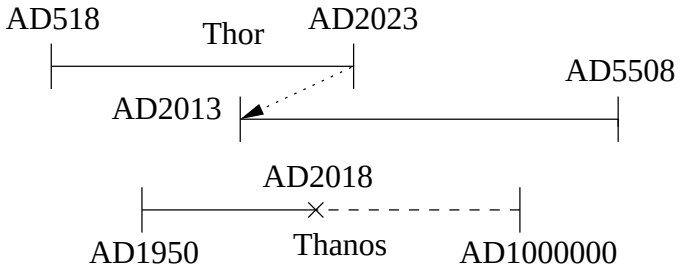
Let us illustrate how this all works with an example. First, Thor and Thanos are both born. Thor, as an Asgardian has an expected lifespan of 5000 years. Thanos is more or less immortal, so we give him a very long lifespan just for kicks:



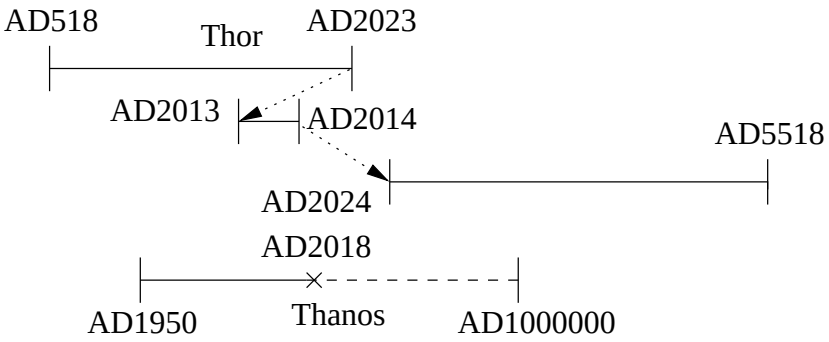
Then Thor decapitates Thanos in 2018.



Nope, that didn't help, so Thor jumps back from 2023 to 2013 to retrieve the Reality stone.

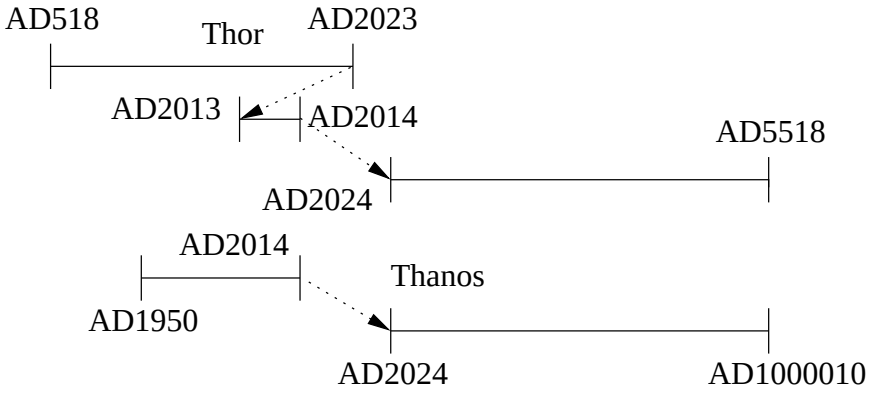


We assume that a character will spend at least a year where they go for simplicity and Thor jumps back to 2024 after retrieving the Reality stone.



Note that in 2013, there are 3 characters: `("Thor", 518)`, `("Thor", 2023)`, and `("Thanos", 1950)`. In 2024, there is only `("Thor", 2013)` because Thanos is dead. Note that the `identity` is the year where a character jumps from. Yes, it's hard to keep track of time travellers to figure out who is who. We try our best.

But then Thanos manages to jump from 2014 to the future too!



Because Thanos jumped before he was killed, he is no longer dead in 2018, though he currently doesn't exist in 2018. And now in 2024, there is both `("Thor", 2013)` and `("Thanos", 2014)`. In other words, jumping can undo the effect of being killed.

Sample execution (newlines are added for readability):

```
>>> t = Timeline()
>>> thor = t.born("Thor",518,5000)
>>> thanos = t.born("Thanos",1950,1000000)

>>> t.get_people(2017)
[('Thor', 518), ('Thanos', 1950)]

>>> thor.kill(2018,thanos,1950) # whoops. Violence. :'(
True
>>> thor.kill(2018,thanos,1950) # Can't kill him twice!
False
>>> t.get_people(2018) # Thanos dead.
[('Thor', 518)]
```

```
>>> thor.jump(2023,2013,518)
>>> thor.jump(2014,2024,2023)

>>> t.get_people(2013)
[('Thor', 2023), ('Thor', 518), ('Thanos', 1950)]

>>> t.get_people(2014)
[('Thor', 518), ('Thanos', 1950)]

>>> t.get_people(2022)
[('Thor', 518)]

>>> t.get_people(2023)  # Thor left for 2014
[]

>>> t.get_people(2024)
[('Thor', 2014)]

>>> thanos.jump(2014,2024,1950)
>>> print(t.get_people(2024))
[('Thor', 2014), ('Thanos', 2014)]

>>> t.get_people(2014) # New Thor and old Thanos
                       # jumped so only old Thor left
[('Thor', 518)]

>>> t.get_people(2017)
[('Thor', 518)]

>>> thor.kill(2018,thanos,1950)  #Thanos is no longer around to die.
False
```