CS1010S — Programming Methodology
National University of Singapore

Practical Examination

21 Apr 2018

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **three** questions. The time allowed for solving this test is **2 hours**.
3. The maximum score of this test is **30 marks**. 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. While you are also provided with the template `practical-template.py` to work with, your answers should be submitted on Coursemology. Note that you can **only run the test cases on Coursemology for a limited number of tries** 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 depend only on the provided test cases. Do ensure that you submit your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org and we are not able to upload the answers to Coursemology.org, you will be required to name your file `<mat no>.py` where `<mat no>` is your matriculation number and leave the file on the Desktop. If your file is not named correctly, we will choose any Python file found at random.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology and correctly left in the desktop folder 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.
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!

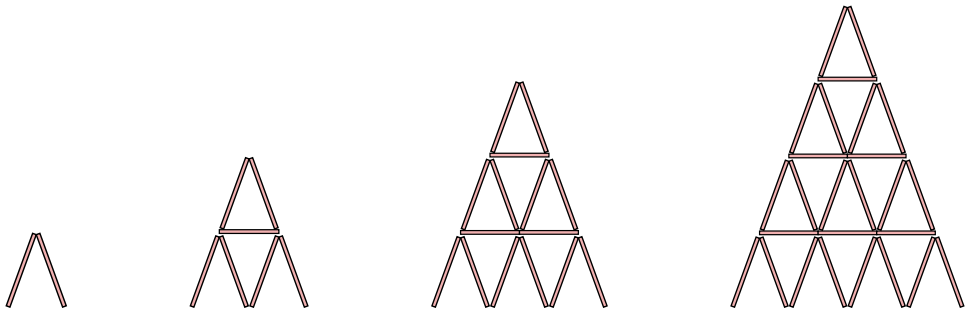
Question 1 : House of Cards [10 marks]

A house of cards (also known as a card tower) is a structure created by stacking playing cards on top of each other. –Source: Wikipedia

A 4-level classic card tower

The tower is created by layers and each layer is created by stacking cards horizontally on top of pairs of supporting cards. As the tower height increases, so does the width of the bottom-most layer.

The following illustrates card towers of heights 1 to 4. Note that there are no cards at the base of the tower.

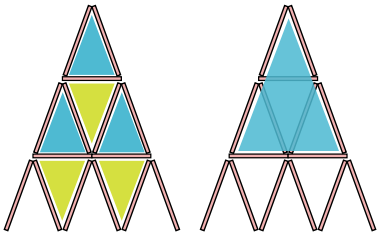


A. Implement the function `num_cards` which takes a non-negative integer h, and returns the number of cards needed to build a tower of height h. Your function should solve the problem computationally, i.e. using recursion or iteration and not simply use a formula. [5 marks]

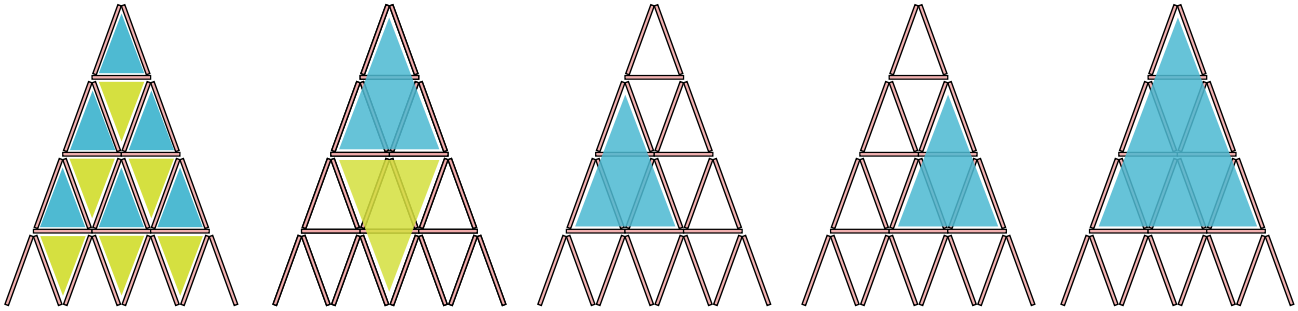
Sample execution:

```
>>> num_cards(1)
2

>>> num_cards(2)
7

>>> num_cards(3)
15

>>> num_cards(4)
26
```

B. Jerryl sees that a card tower is made up of overlapping triangles of different sizes. For example, he counts 7 triangles (4 upright and 3 inverted) in a card tower of height 3:



and 17 triangles (10 upright and 7 inverted) in a card tower of height 4:



Note that as with before, there are no cards on the base to make triangles there.

Implement a function `num_triangles`, which takes a non-negative integer h, and returns the number of triangles that can be formed from a tower of height h. Your function should solve the problem computationally, i.e. using recursion or iteration and not simply use a formula.

Hint: You may want to consider counting upright and inverted triangles separately. [5 marks]

Sample execution:

```
>>> for i in range(1, 10):
        num_triangles(i)

0
2
7
17
33
57
90
134
190
```

Question 2 : IRAS [10 marks]

Important note: You are provided with a data file `tax_data.csv` for this question for testing, but your code should work correctly for *any* data file with the same format and it *will* be tested with other data files.

Every year, around this time of the year, is tax day. Millions of working people in Singapore file their taxes to IRAS as well as submit any tax rebate. Tax rebates come in various categories, which are specified in the data file.

As an intern in IRAS, you are tasked to simplify their reporting system. IRAS database contains each persons' unique NRIC number, the year of tax filing, their yearly income for the given year, the tax paid in that year, as well as the three rebates. These data are given in the data file in this specific column order, with **arbitrary many categories of rebates**:

NRIC	Year	Annual Income	Tax Paid	*Rebates...

A. The basic report that you need to generate is the personal statistics. Given a person's NRIC number, you are to find the first and latest years he/she filed taxes, as well as the smallest, largest, and average tax (rounded to 2 decimal places) paid by the person in the database.

Implement the function `yearly_stats` which takes as input the name of the data file, and a person's NRIC number. It returns the statistics required above as a tuple.

Sample execution:

```
>>> yearly_stats("tax_data.csv", "G2407965J")
(2014, 2017, 400, 4300, 1567.5)

>>> yearly_stats("tax_data.csv", "G2380503K")
(2014, 2017, 300, 1000, 595.0)
```

B. The second report is more complex and it is intended for the higher ups. One of government policies is to set the effective tax rate. For a given person in a given year, the effective tax rate is the ratio of the tax paid over the given year's income minus all the rebate he filed that year. i.e., $\frac{tax\_paid}{annual\_income - \sum rebates}$.

For instance, someone earning $40,000 per year but filed $600 house rebate, $500 child rebate, and $1,600 education rebate has an effective taxable income of $3,7300. If he/she paid $400 in tax, the tax rate is effectively $400/37,300 = 1.07\%$.

The government wants to know the mean effective tax rate for each year. To compute this, you need to compute the effective tax rate of everyone for a given year and find the average.

Implement the function `effective_tax` that takes the name of the data file as its only input. It returns a dictionary where the keys are the years (in `str`) and the values are the average effective tax rate for the corresponding year, rounded to 2 decimal places.

Sample execution:

```
>>> effective_tax("tax_data.csv")
{'2017': 1.95, '2016': 1.018, '2015': 1.017, '2014': 1.013}
```

Question 3 : Pacific Rim Uprising [10 marks]

Warning: Please read the entire question carefully and plan well before starting to write your code.

— BACKGROUND (Okay to skip) —

Ten years after the Battle of the Breach, former Jaeger pilot Jake Pentecost—son of deceased Kaiju War hero Stacker Pentecost—makes a living by stealing and selling Jaeger parts on the black market in the Los Angeles area. After he tracks part of a disabled Jaeger's power core to the secret workshop of fifteen years old Jaeger enthusiast Amara Namani, both are arrested by the Pan-Pacific Defense Corps after an altercation between Amara's small, single-pilot Jaeger Scrapper and the Jaeger November Ajax. Jake's adoptive sister and PPDC Secretary General Mako Mori gives Jake a choice between the prison or return to PPDC as an instructor with Amara as his recruit.

Upon arriving at a Shatterdome in China, Jake starts training Jaeger program cadets with his estranged former co-pilot Nate Lambert. Nate and Mako reveal to him that the Jaeger program is threatened by Shao Corporation's drone program, which offers to mass produce Kaiju-Jaeger hybrid drones developed by Liwen Shao and Dr. Newton Geiszler. Mako is due to deliver a final assessment to determine the authorization of the drones at a PPDC council meeting in Sydney, but is killed by rogue Jaeger Obsidian Fury before she can report. Her death prompts the PPDC council to authorize the drone program and order their immediate deployment. Moments before her death, Mako transmitted the location of a defunct Jaeger production facility in Siberia. Jake and Nate travel to the area in their own Jaeger, but Obsidian Fury destroys the complex and engages them in battle. Upon destroying its reactor, they find that Obsidian Fury was controlled by a Kaiju's secondary brain, which testing shows was grown on Earth.

When the drones reach their respective locations, they are taken over by cloned Kaiju brains and simultaneously attack Shatterdomes worldwide, inflicting heavy casualties on the PPDC forces and incapacitating almost all Jaegers. Hermann Gottlieb seeks out Geiszler for help, only to discover that Geiszler is the mastermind behind the attack. Geiszler's mind has been taken over by the Precursors, the alien race who created the Kaiju, due to his regularly drifting with Kaiju brains. Seeking to destroy the world for the Precursors, Geiszler, now the Precursor Emissary, commands the drone-Kaiju hybrids to open new breaches all over the world. Although Shao is able to destroy the hybrids, three powerful Kaiju—Raijin, Hakuja, and Shrikethorn—emerge from the breaches and unite in Tokyo. The team realizes that the Precursors' goal is to activate the Ring of Fire by detonating Mount Fuji with the Kaiju's chemically reactive blood, spreading toxic gas into the atmosphere and wiping out all life on Earth, terraforming the planet for Precursor colonization.

The cadets are mobilized while Gottlieb and Shao repair the PPDC's four remaining Jaegers; Gottlieb invents Kaiju-blood-powered rockets, which launch the team to Tokyo. Although the Jaegers initially repel the Kaiju, the Precursor Emissary merges them into one gigantic beast that quickly overpowers the team, killing Suresh, wounding Nate, and leaving Gipsy Avenger the only operational Jaeger.

> *Jake and Amara pilot it against the Mega-Kaiju, with Shao remote piloting Scrapper aiding them by locating and welding a rocket to Gipsy's right hand and sending the larger Jaeger into the Mega-Kaiju which kills the creature. Jake and Amara survive by transferring into Scrapper prior to the collision. Nate takes the Precursor Emissary into custody.*
>
> *The captive Precursor Emissary threatens that his masters will attack the world over and over again. Jake replies that next time, humanity will be the ones attacking the Precursors.*
>
> **– Source: Wikipedia**

— END OF BACKGROUND —

For this question, we will be modelling Kaijus, which are giant alien monsters.

A *Kaiju* has a name, some health points (HP) and a set of known attack moves. It can only attack using a move that it knows. A Kaiju can take damage which will reduce its HP. When its HP reaches 0, the Kaiju dies and is no longer able to attack.

Kaiju's can also merge with other Kaiju's to become a *Mega-Kaiju*. The Mega-Kaiju collectively knows the moves of all the Kaijus that it is merged with. Its HP is also the sum total of all the merged Kaijus. Like a Kaiju, it will also die when its HP is reduced to 0.

A `Kaiju` is created with three inputs, its name (a string), a category (an integer) and a sequence of attack moves (which are strings). The starting HP of a Kaiju is determined as 10 raised by its category, i.e.: $hp = 10^{category}$.

`Kaiju` supports the following methods:

- `get_health()` returns the current number of health points (HP) of the Kaiju.
- `damage(amount)` takes in an integer and returns a string based on the following conditions:
  - `'<Kaiju name> is already dead'` if the Kaiju is currently dead.
  - `'<Kaiju name> takes <amount> damage and dies'` if the Kaiju dies if its HP is less than or equal to the `amount` of damage.
  - otherwise, `'<Kaiju name> takes <amount> damage'` is returned and the Kaiju's HP is reduced by `amount`.
- `attack(move)` takes as input an attack move (which is a string) and returns a string based on the following conditions:
  - `'<Kaiju name> is already dead'` if the Kaiju is currently dead.
  - `'<Kaiju name> does not know <move>'` if the Kaiju does not know this attack `move`.
  - otherwise, `'<Kaiju name> performs <move> attack'` is returned.
- `merge(other1, other2, ...)` takes in an arbitrary number of other `Kaijus` and merge all of them together with the `Kaiju` itself to create a `MegaKaiju`.

  If any of the Kaijus is already a `MegaKaiju`, then the merge fails and the string `'<Kaiju original name> is already a Mega-Kaiju'` is returned.

Otherwise the merge is successful and a new `MegaKaiju` is created from all the Kaijus and returned. The Mega-Kaiju's HP and known attack moves will be the collective total of all the participating Kaijus. All the participating Kaijus will also no longer act as individual Kaijus but respond as if they were the MegaKaiju.

See sample run for details.

`MejaKaiju` is a subclass of `Kaiju` and it behaves the same as a `Kaiju`. The only difference is that its method `merge` does not return a new `MegaKaiju` but instead merge the list of Kaijus with itself, increasing its HP and assimilating its knowledge of moves in the process.

Provide an implementation for the classes `Kaiju` and `MegaKaiju`.

For simplicity, you do not have to worry about data abstraction and can access the properties of both classes directly. Take careful note of the characters in the returned strings, especially spaces and punctuation.

Sample Execution:

```
knifehead   = Kaiju("Knifehead", 3, ("Headbutt", "Snapping jaws"))
rajin       = Kaiju("Rajin", 5, ("Electric jaw", "Morphic skull",
                                                 "Shredding bite"))
hajuka      = Kaiju("Hajuka", 4, ("Molten blood",))
shrikethorn = Kaiju("Shrikethorn", 4, ("Weaponized spines",))
ripper      = Kaiju("Ripper", 1, ())  # no attack move

>>> knifehead.attack('Headbutt')
"Knifehead performs Headbutt attack"

>>> knifehead.attack('Molten blood')
"Knifehead does not know Molten blood"

>>> knifehead.damage(500)
"Knifehead takes 500 damage"

>>> knifehead.get_health()
500

>>> knifehead.damage(500)
"Knifehead takes 500 damage and dies"

>>> knifehead.attack('Headbutt')
"Knifehead is already dead"

>>> knifehead.damage(500)
"Knifehead is already dead"

>>> rajin.attack('Electric jaw')
"Rajin performs Electric jaw attack"
```

```
>>> rajin.damage(50000)
"Rajin takes 50000 damage"

>>> rajin.get_health()
50000

>>> hajuka.get_health()
10000

>>> shrikethorn.get_health()
10000

>>> rajin.attack('Molten blood')
"Rajin does not know Molten blood"

>>> mega = rajin.merge(hajuka, shrikethorn)
# merge into a Mega-Kaiju
# rajin, hajuka and shrikethorn now behaves as one MegaKaiju

>>> mega.get_health()
70000  # MegaKaiju has combined HP

>>> rajin.get_health()
70000  # individual Kaijus now behave like the Mega-Kaiju

>>> hajuka.get_health()
70000  # individual Kaijus now behave like the Mega-Kaiju

>>> shrikethorn.attack('Molten blood')
"MegaKaiju performs Molten blood attack"  # MegaKaiju's name is
   MegaKaiju

>>> shrikethorn.attack('Headbutt')
"MegaKaiju does not know Headbutt"

>>> ripper.merge(rajin)
"Rajin is already a Mega-Kaiju"

>>> rajin.merge(ripper)

>>> ripper.get_health()
70010

>>> mega.damage(50000)
"MegaKaiju takes 50000 damage"

>>> ripper.damage(3000)
"MegaKaiju takes 3000 damage"
```

```
>>> rajin.attack('Weaponized spines')
"MegaKaiju performs Weaponized spines attack"

>>> shrikethorn.damage(18000)
"MegaKaiju takes 18000 damage and dies"

>>> mega.attack('Weaponized spines')
"MegaKaiju is already dead"
```

You are advised to solve this problem incrementally. Even if you cannot fulfil all the desired behaviours, you can still get partial credit for fulfilling the basic behaviours.

— E N D O F P A P E R —