CS1010S — Programming Methodology
National University of Singapore

Practical Examination

17 Nov 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 : Floyd's Triangle [10 marks]

Floyd's triangle is a right-angled triangular array of natural numbers, used in computer science education. It is named after Robert Floyd. It is defined by filling the rows of the triangle with consecutive numbers, starting with a 1 in the top left corner:

```
 1
 2   3
 4   5   6
 7   8   9   10
11  12  13  14  15
```

Source: Wikipedia

A. Implement the function `floyd_row(n)` which takes in a positive integer n and returns a tuple of integers that represent the nth row of the Floyd's triangle. [5 marks]

Sample execution:

```
>>> floyd_row(1)
(1,)

>>> floyd_row(2)
(2, 3)

>>> floyd_row(5)
(11, 12, 13, 14, 15)
```

B. Implement the function `floyd_sum(n)` which takes in a positive integer n and returns the sum of all the numbers in the Floyd's triangle until and including row n.

You should solve it computationally taking at least $O(n)$ time, i.e., not simply using a formula. [5 marks]

Sample execution:

```
>>> floyd_sum(1)
1

>>> floyd_sum(2)
6

>>> floyd_sum(5)
120
```

Question 2 : Video Game Sales [10 marks]

Important note: You are provided with a data file `vgsales.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.

Video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device. Early video games include Pong and Asteroids on 2600 platform while the latest platforms include PS4, XBOX One, and Nintendo Switch.

As you are planning to take CS3247: Game Development, you want to analyse the video game sales data to know which games are popular. Besides the basic game information such as name, platform, year, genre, and published, the 6^{th} to 9^{th} columns contain sales data for North America (NA), Europe (EU), Japan (JP), and other regions. The data contains the following column in specific order:

Name	Platform	Year	Genre	Publisher	NA	EU	JP	Other	Total

However, note that some of the data are unknown. For the unknown data, the value in the cell will be `'N/A'`.

A. Your first job in analysing the sales data is to analyse the regional sales data. Given a platform and year, you are to find the sales for each region as well as the total global sales for the given platform in the given year.

Implement the function `regional_sales` which takes as input the name of the data file, the name of the platform, and the year. It returns a tuple of 5 elements where each element is the sum of sales for the respective region for the given platform in the given year. The first element is for North America, the second for Europe, the third for Japan, the fourth for Others, and the fifth the combined global total sales.

Each element should be rounded to 2 decimal places. Note that you can use the function `round(val, dp)` to round the value `val` to the specified decimal places `dp`.

Sample execution:

```
>>> regional_sales("vgsales.csv", "X360", 2016)
(0.36, 0.4, 0.0, 0.07, 0.83)

>>> regional_sales("vgsales.csv", "3DS", 2012)
(17.04, 11.59, 19.92, 2.55, 51.1)
```

B. A more involved analysis is to find the up and coming genre. This analysis is good to predict the trend in gaming as game development often take years.

Your task for this analysis is to find in a given platform, genre with the highest percentage increase from the previous year. For example, on the PS3 platform, the Sports genre has combined sales of `12.63` in 2012 and `15.18` in 2013. The percentage increase is $\frac{15.18-12.63}{12.63} \times 100\% = 20.19\%$ **rounded off** to 2 decimal places.

If there are no or 0 sales figure available for a genre in the previous year, then you should not calculate the genre for that particular year Similarly, if the year or genre is `'N/A'`, you should not include the row in the computation.

You should then repeat the process of finding the genre with the highest percentage increase for each year for a given platform.

Implement the function `trending_genre` which takes as input the name of the data file and the name of the platform. It returns a dictionary where the key is the year and the value is the genre with the highest percentage increase (or lowest decrease if every genre suffers negative growth) from the given year. If there are multiple genre with the same percentage increase, then choose the genre with the alphabetically smaller name.

For instance, the genre Misc on PS3 has 0.53 sales in 2012 and 1.68 sales in 2013. It is in fact, the genre with the highest percentage increase for 2013 of 216.98%.

Sample execution:

```
>>> trending_genre("vgsales.csv", "PS3")
{2013: 'Misc', 2014: 'Misc', 2015: 'Simulation',
 2016: 'Role-Playing'}

>>> trending_genre("vgsales.csv", "PC")
{2013: 'Misc', 2014: 'Simulation', 2015: 'Strategy',
 2016: 'Adventure'}
```

Question 3 : Infinity Wars [10 marks]

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

— BACKGROUND (Okay to skip) —

> *"From the dawn of the universe, there was nothing. Then... boom. The Big Bang sent six elemental crystals, hurtling across the virgin universe. These Infinity Stones each control an essential aspect of existence. 'Space, Reality, Power, Soul, Mind and Time'."*
>
> **– Source: Wong and Doctor Strange**

> *The Infinity Stones are six immensely powerful objects tied to different aspects of the universe, created by the Cosmic Entities. Each of the stones possess unique capabilities that have been enhanced and altered by various alien civilizations for millennia.*
>
> *Only beings of immense power can directly wield the Stones, such as Celestials, or the Mad Titan Thanos. Lesser beings face dire consequences: Jane Foster became ill after being exposed to the Aether and Carina exploded when she touched the Power Stone; however, these effects can be reduced if a group is sharing the power among themselves. It is also possible to place an Infinity Stone inside a container to prevent a person from directly touching it, allowing the user to wield the Stone's power without suffering the normal repercussions. Examples include the Space Stone inside the Tesseract, the Mind Stone inside the Scepter, the Power Stone inside the Cosmi-Rod, the Time Stone inside the Eye of Agamotto, and all the Infinity Stones inside the Infinity Gauntlet.*

– Source: http://marvelcinematicuniverse.wikia.com

— END OF BACKGROUND —

For this question, we will be modelling both Infinity Stones and the artefacts used to wield the power of the stones without suffering the repercussions.

An *Infinity Stone* has a name and a set of powers. An intact Infinity Stone may i) be imbued with additional powers, ii) disarmed from its known powers, or iii) destroyed completely (*as is the case when the Scarlet Witch destroyed the Mind Stone from Vision*).

An Infinity Stone may be attached to an *artefact*. When this happens, the artefact gains all the powers of the attached Infinity Stone. The artefact may also lose the Infinity Stone either when taken by someone (*Thanos*) or when the Infinity Stone is destroyed. Artefacts may be combined with other artefacts. For simplicity, artefacts cannot be separated once they are combined.

An Infinity `Stone` is created with one or more inputs: its name (a string) and an arbitrary number of powers (each is a string) the stone contains. `Stone` supports the following methods:

- `imbue(power)` takes in a power (which is a string) and returns a string based on the following conditions:
  - `'<stone name> already destroyed'` if the Infinity `Stone` is already destroyed.
  - `'<Stone name> already imbued with the power <power name>'` if the Infinity `Stone` already contains the power.
  - otherwise, `power` is added to the `Stone`'s current powers and the string `'<Stone name> is now imbued with the power <power name>'` is returned.
- `disarm(power)` takes in a power (which is a string) and returns a string based on the following conditions:
  - `'<stone name> already destroyed'` if the Infinity `Stone` is already destroyed.
  - `'<stone name> is not imbued with the power <power name>'` if the Infinity `Stone` does not contain the power.
  - otherwise, `power` is removed from the `Stone`'s current powers and the string `'<Stone name> is no longer imbued with the power <power name>'` is returned.
- `destroy()` returns a string based on the following conditions:
  - `'<stone name> already destroyed'` if the Infinity `Stone` is already destroyed.
  - otherwise, the `Stone` is now destroyed and removed from any artefact it is currently residing in. The string `'<stone name> is now destroyed'` is returned.
- `list_powers()` returns a `tuple` containing:
  - nothing (empty `tuple`) if the Infinity `Stone` is already destroyed.
  - otherwise, the current powers of the Infinity `Stone` currently in any order.

An `Artefact` is created with one or more inputs: its name (a string) and an arbitrary number of `Stone`s it currently contains. `Artefact` supports the following methods:

- `add_stone(stone)` takes in an Infinity `Stone` as input and returns a string based on the following conditions:
  - `'<stone name> is already destroyed'` if the `Stone` is already destroyed.
  - `'<artefact name> already has <stone name>'` if the `Artefact` already contains `Stone`.
  - `'<stone name> is already part of <other artefact name>'` if the Infinity `Stone` is already contained in another artefact.
  - otherwise, the `Stone` is added to the `Artefact` and the string `'<stone name> is added to <artefact name>'` is returned.
- `remove_stone(stone)` takes in an Infinity `Stone` and returns a string based on the following conditions:
  - `'<artefact name> does not contain <stone name>'` if the artefact does not contain the given stone.
  - Otherwise, the stone is removed from the artefact and the string `'<stone name> is removed from <artefact name>'` is returned.
- `combine(other)` takes in an `Artefact` and returns a string based on the following conditions:
  - `'Cannot combine <artefact name> with itself'` if the other artefact is the same instance as the current artefact.
  - `'<artefact name> is already combined with <other name>'` if the other artefact is already part of the same combined artefacts.
  - otherwise, the two artefacts combine together and the string `'<artefact name> combines with <other name>'` is returned.
- `invoke()` returns a `tuple` of unique powers from all artefacts combined with this. The order of the powers does not matter.

Provide an implementation for the classes `Stone` and `Artefact`.

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:

```
power_stone   = Stone("Power Stone", "Attack", "Defense")
mind_stone    = Stone("Mind Stone", "Brainwash")
time_stone    = Stone("Time Stone")
reality_stone = Stone("Reality Stone", "Illusion")

>>> power_stone.imbue("Attack")
"Power Stone already imbued with the power Attack"

>>> power_stone.imbue("Strength")
"Power Stone is now imbued with the power Strength"
```

```
>>> power_stone.list_powers()
('Attack', 'Defense', 'Strength')

>>> time_stone.imbue("Repeat")
"Time Stone is now imbued with the power Repeat"

>>> time_stone.imbue("Undo")
"Time Stone is now imbued with the power Undo"

vision          = Artefact("Vision", mind_stone)
gauntlet        = Artefact("Gauntlet", power_stone)
eye_of_agamotto = Artefact("Eye of Agamoto")

>>> vision.add_stone(mind_stone)
"Vision already has Mind Stone"

>>> vision.remove_stone(power_stone)
"Vision does not contain Power Stone"

>>> vision.remove_stone(mind_stone)
"Mind Stone is removed from Vision"

>>> vision.add_stone(mind_stone)
"Mind Stone is added to Vision"

>>> vision.invoke()
('Brainwash',)

>>> vision.add_stone(power_stone)
"Power Stone is already part of Gauntlet"

>>> mind_stone.disarm("AI")
"Mind Stone is not imbued with the power AI"

>>> mind_stone.destroy()
"Mind Stone is now destroyed"

>>> mind_stone.disarm("Brainwash")
"Mind Stone already destroyed"

>>> vision.remove_stone(mind_stone)
"Vision does not contain Mind Stone"

>>> eye_of_agamotto.add_stone(time_stone)
"Time Stone is added to Eye of Agamoto"

>>> vision.combine(eye_of_agamotto)
```

```
"Vision combines with Eye of Agamoto"

>>> eye_of_agamotto.combine(vision)
"Eye of Agamoto is already combined with Vision"

# We use the power of time stone to remake mind_stone
mind_stone_remade = Stone("Mind Stone", "Brainwash", "Illusion")

>>> gauntlet.add_stone(mind_stone)
"Mind Stone is already destroyed"

>>> gauntlet.add_stone(mind_stone_remade)
"Mind Stone is added to Gauntlet"

>>> gauntlet.invoke()
('Attack', 'Defense', 'Strength', 'Brainwash', 'Illusion')

>>> gauntlet.add_stone(reality_stone)
"Reality Stone is added to Gauntlet"

>>> gauntlet.invoke()
('Attack', 'Defense', 'Strength', 'Brainwash', 'Illusion')
# no duplicate Illusion

>>> gauntlet.combine(eye_of_agamotto)
"Gauntlet combines with Eye of Agamoto"

>>> gauntlet.combine(vision)
"Gauntlet is already combined with Vision"

>>> gauntlet.invoke()
('Repeat', 'Undo', 'Attack', 'Defense', 'Strength', 'Brainwash',
   'Illusion')
```

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 —