CS1010S — Programming Methodology
National University of Singapore

Re-Practical Examination

24 Nov 2017

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. The total score for this test is capped at **16 marks** for those attempting the exam for the second time.
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. 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.
7. 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.
8. 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.
9. 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 : Chemistry [10 marks]

A chemical formula is a way of expressing information about the proportions of atoms that constitute a particular chemical compound or molecule, using a single line of chemical element symbols, numbers, and sometimes also other symbols, such as parentheses, dashes, brackets, commas and plus (+) and minus (–) signs. These are limited to a single typographic line of symbols, which may include subscripts and superscripts.

Source: Wikipedia

A molecule is made up of atoms, which is represented in a chemical formula as a string of characters and digits. For the purpose of this question, atoms are represented by a **single** character, with the quantity represented by a **single** digit immediately preceding it. If no digit precedes the atom, then the quantity is 1.

For example, the formula for sulphuric acid is `'H2SO4'`. It has two `'H'` atoms, one `'S'` atom and four `'O'` atoms, a total of 7 atoms.

A. Implement the function `num_atoms(molecule)` which takes in a chemical formula of a molecule, and returns the number of atoms in the molecule.

Hint: You can make use of the string function `str.isdigit()` and `str.isalpha()` to check if a string is a digit or an alphabetical character. [5 marks]

Sample Execution:

```
>>> num_atoms("H2SO4")  # sulphuric acid
7
>>> num_atoms("CH3CH2OH")  # ethanol
9
>>> num_atoms("NH4CO2NH2") # ammonium carbamate
11
```

B. The molar mass of each atom can be obtained from the periodic table. It is then simple to compute the molar mass of a molecule by simply summing up the mass of all the atoms.

Implement the function `molar_mass(molecule, table)` which takes in a molecule and a table (which is a `dict`) and returns the molar mass of the molecule. [5 marks]

Sample Execution:

```
>>> table = {'H':1.008,  'B':10.81,  'C':12.011, 'N':14.007,
             'O':15.999, 'F':18.998, 'P':30.974, 'S':32.06,
             'K':39.098, 'V':50.942, 'Y':88.906, 'I':126.9,
             'W':183.84, 'U':238.03}

>>> molar_mass("H2SO4", table)
98.072
>>> molar_mass("CH3CH2OH" ,table)
46.069
>>> molar_mass("NH4CO2NH2", table)
78.071
```

Question 2 : Box Office Hits [10 marks]

Important note: You are provided with a data file `box-office.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.

For the past sixteen weeks, you have been so busy with school work and had "no life" outside school (in no part due to CS1010S). Having missed all the blockbusters released in recent months, you decide to binge watch blockbuster hits the for entire Christmas Holidays.

Using your new-found Python magic, you scraped the weekly box-office takings of all Hollywood releases for the past decade off `www.boxofficemojo.com` to help you decide the best movies to watch.

A. Given a certain range of weeks in a year, you wish to know what were the top 10 highest grossing titles within those weeks. Implement the function `top_10_titles(fname, year, weeks)` that takes as input a filename (a `str`), a year (an `int`) and a sequence of weeks (a sequence of `int`).

The function returns the top ten highest grossing titles along with their earnings as a tuple of pairs. The total gross of a title is the sum of the title's weekly gross that lies in the given weeks. For example, if weeks is `(1,2,3,4)`, the total gross would be the sum of the gross in weeks 1, 2, 3 and 4 of the given year. [5 marks]

Sample Execution:

```
# First 5 weeks of 2016
>>> top_10_titles('box-office.csv', 2016, (1, 2, 3, 4, 5))
(('Star Wars: The Force Awakens', 247104200),
 ('The Revenant', 141730884),
 ("Daddy's Home", 79075678),
 ('Ride Along 2', 72686830),
 ('Kung Fu Panda 3', 48050957),
 ('13 Hours: The Secret Soldiers of Benghazi', 44892592),
 ('The Hateful Eight', 39324631),
 ('The Big Short', 37955298),
 ('Sisters', 37170845),
 ('Alvin and the Chipmunks: The Road Chip', 27779504))

# top 10 for 2010
>>> top_10_titles('box-office.csv', 2010, range(1,54))
(('Avatar', 466141929),
 ('Toy Story 3', 415004880),
 ('Alice in Wonderland (2010)', 334191110),
 ('Iron Man 2', 312128345),
 ('The Twilight Saga: Eclipse', 300531751),
 ('Inception', 292576195),
 ('Harry Potter and the Deathly Hallows Part 1', 285308372),
 ('Despicable Me', 251266265),
 ('Shrek Forever After', 238395990),
 ('How to Train Your Dragon', 217581231))
```

```
# What you missed this semester
>>> top_10_titles('box-office.csv', 2017, range(32, 50))
(('It', 326748251),
 ('Thor: Ragnarok', 253213000),
 ('Annabelle: Creation', 102092201),
 ('Kingsman: The Golden Circle', 99554816),
 ('Blade Runner 2049', 88650463),
 ("The Hitman's Bodyguard", 75468583),
 ('The LEGO Ninjago Movie', 58683641),
 ('Murder on the Orient Express (2017)', 55632000),
 ('Happy Death Day', 55469000),
 ('American Made', 51111000))
```

B. Looking at all these big numbers, you wonder how much the studios make off each film. Now you need not wonder any longer because you have the data and the magic of Python.

Implement the function `top_10_studios(fname, year)` that takes as input a filename (a `str`) and a year (an `int`), and returns the top ten studios with the highest average earnings per title for the given year as a tuple of pairs. Each pair is the name of the studio and the average earnings rounded to the nearest integer. [5 marks]

Sample Execution:

```
>>> top_10_studios('box-office.csv', 2010)
 (('P/DW', 143422577), ('BV', 87448476), ('Par.', 86072709),
 ('Fox', 75084303), ('WB (NL)', 71678164), ('Sony', 65610505),
 ('WB', 57031440), ('Sum.', 51862791), ('Uni.', 50432098),
 ('MGM', 50287556))

>>> top_10_studios('box-office.csv', 2015)
(('BV', 159580610), ('Uni.', 106650216), ('W/Dim.', 76223578),
 ('WB (NL)', 61690814), ('SGem', 60718966), ('Fox', 55193409),
 ('Sony', 49808077), ('Par.', 47957831), ('LG/S', 39740147),
 ('WB', 37740410))

>>> top_10_studios('box-office.csv', 2016)
(('BV', 191325159), ('Uni.', 72451381), ('Fox', 67792415),
 ('WB (NL)', 57091987), ('WB', 55875409), ('Par.', 49544556),
 ('SGem', 43301204), ('LG/S', 38679482), ('TriS', 34818558),
 ('Sony', 34686427))
```

Question 3 : Constantinople [10 marks]

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

In a marketplace of Constantinople, rumours can spread like wildfire as merchants mingle about while doing their business. The marketplace is made up of places where merchants gather to gossip or trade.

The class `Place` models a place in the marketplace, and is created with one input, a name (which is a string). It supports one method: `get_merchants()` which returns a tuple of the names of the merchants currently in that place.

The class `Merchant` models a merchant, and is created with three or more inputs: its name (which is a string), the amount of Lari in possession (which is an integer), and a place (which is a `Place`) which is the initial location of the merchant. Additional arbitrary number of inputs are rumours (which are strings) that the merchant knows.

`Merchant` supports the following methods:

- `get_location()` returns the name of the place that the merchant is in.
- `get_lari()` returns the amount of lari that the merchant currently has.
- `get_rumours()` returns a tuple containing the rumours that the merchant knows.
- `go_to(place)` takes as input a `Place` and:
  - returns the string `'<merchant name> is already at <place name>'` if the merchant is currently in the given place.
  - otherwise, the merchant moves to the given place and the string `'<merchant name> moves from <old place name> to <new place name>'` is returned.
- `gossip(other)` takes as input a `Merchant` and:
  - returns `'<merchant name> cannot gossip with himself'` if the input is himself.
  - returns `'<merchant name> and <other merchant name> are not at the same place'` is returned if the both merchants are not at the same place.
  - otherwise, the each merchants will exchange one rumour unknown to the other party. If there are more than one rumour unknown to the other party, any one can be arbitrarily chosen to be shared. The string `'<merchant name> gossips with <other merchant name>'` is returned.
- `shout(rumour)` takes a rumour as an input, and:
  - returns `'<merchant name> does not know that <rumour>'` if the merchant does not know the rumour.
  - otherwise, the merchant announces the rumour to all other merchants in the same place and all other merchants will now know about this rumour. The string `'<merchant name> shouts "Did you know that <rumour>?!"'` is returned.

Merchants can also lend Lari, which is their currency, to other merchants. This is supported by the following methods in `Merchant`:

- `lend(other, amount, interest)` takes as input a Merchant, an amount of Lari, and an interest (in percentage), and:
  - returns `'<merchant name> cannot lend to himself'` if other is the Merchant.
  - returns `'<merchant name> and <other merchant name> are not at the same place'` if other and the merchant are not at the same place.
  - returns `'<merchant name> does not have enough Lari to lend'` if the merchant has less than the given amount of Lari.
  - otherwise, the merchant lends the given amount of Lari to the other merchant with the given one-time interest amount. For example, if the amount lent is 200 and the interest is 1%, then the amount to be repaid is 202. The function returns the string `'<merchant name> lends <amount> Lari to <other merchant name> at <interest>% interest'`.
- `settle()` takes no inputs, and settles all the merchant's debt and credit with all other merchants in the current place using the following procedure:
  1. First, the merchant will attempt to collect from every debtor in the same place. If the debtor does not have enough Lari to return the full outstanding loaned amount, no collection is done.
  2. Second, the merchant will attempt to repay all his creditors starting with his largest debt, i.e., the creditor whom he owes the most Lari. If he does not have enough Lari to repay the creditor, no debt is repaid and he goes to try to repay the next largest creditor.

  If no transaction takes place, the string `'<merchant name> has nothing to settle'` is returned. Otherwise, the string of the following format is returned: `'<merchant name> collects <amount> Lari from <debtor1>, and collects <amount> Lari from <debtor2>, ... , and repays <amount> Lari to <creditor1>, and repays <amount> Lari to <creditor2>, ...'` for as many debts that were collected or credits that were repaid.

  [10 marks]

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:

```
>>> plaza    = Place('Plaza')
>>> market   = Place('Market')
>>> bazaar   = Place('Bazaar')
>>> fountain = Place('Fountain')

>>> arslan   = Merchant('Arslan', 100, bazaar,
                        'Istanbul is not Constantinople',
                        'Ottoman conquered the Byzantine')
>>> burhan   = Merchant('Burhan', 20,  market,
                        'banana is having a discount today',
                        'an apple a day keeps the medicine man away')
>>> doruk    = Merchant('Doruk',  85,  market)
>>> mustafa  = Merchant('Mustafa', 5,  fountain,
                        'today is Friday',
                        'the repractical exam is today',
                        'the repractical will lasts for 2 hours')

>>> market.get_merchants()
('Burhan', 'Doruk')

>>> burhan.go_to(market)
"Burhan is already at Market"

>>> burhan.go_to(plaza)
"Burhan moves from Market to Plaza"

>>> market.get_merchants()
('Doruk',)

>>> burhan.shout('today is Friday')
"Burhan does not know that today is Friday"

>>> burhan.shout('banana is having a discount today')
'Burhan shouts "Did you know that banana is having a discount today?!"'

>>> doruk.get_rumours()
()

>>> mustafa.go_to(plaza)
"Mustafa moves from Fountain to Plaza"

>>> mustafa.get_rumours()
('today is Friday', 'the repractical exam is today', 'the repractical
   will lasts for 2 hours')

>>> mustafa.shout('the repractical exam is today')
'Mustafa shouts "Did you know that the repractical exam is today?!"'

>>> burhan.get_rumours()
('banana is having a discount today', 'an apple a day keeps the medicine
   man away', 'the repractical exam is today')

>>> arslan.go_to(plaza)
"Arslan moves from Bazaar to Plaza"

>>> burhan.gossip(doruk)
"Burhan and Doruk are not at the same place"
```

```
>>> doruk.go_to(plaza)
"Doruk moves from Market to Plaza"

>>> burhan.gossip(doruk)
"Burhan gossips with Doruk"

>>> burhan.gossip(burhan)
"Burhan cannot gossip with himself"

>>> plaza.get_merchants()
('Burhan', 'Mustafa', 'Arslan', 'Doruk')

>>> burhan.shout('the repractical exam is today')
'Burhan shouts "Did you know that the repractical exam is today?!"'

>>> doruk.get_rumours()
('banana is having a discount today', 'the repractical exam is today')

>>> burhan.get_rumours()
('banana is having a discount today', 'an apple a day keeps the medicine
    man away', 'the repractical exam is today')

>>> arslan.get_rumours()
('Istanbul is not Constantinople', 'Ottoman conquered the Byzantine',
    'the repractical exam is today')

>>> burhan.gossip(arslan)
"Burhan gossips with Arslan"

>>> burhan.get_rumours()
('banana is having a discount today', 'an apple a day keeps the medicine
    man away', 'the repractical exam is today', 'Istanbul is not
    Constantinople')

>>> arslan.get_rumours()
('Istanbul is not Constantinople', 'Ottoman conquered the Byzantine',
    'the repractical exam is today', 'banana is having a discount today')

>>> arslan.gossip(burhan)
"Arslan gossips with Burhan"

>>> burhan.get_rumours()
('banana is having a discount today', 'an apple a day keeps the medicine
    man away', 'the repractical exam is today', 'Istanbul is not
    Constantinople', 'Ottoman conquered the Byzantine')

>>> arslan.get_rumours()
('Istanbul is not Constantinople', 'Ottoman conquered the Byzantine',
    'the repractical exam is today', 'banana is having a discount today',
    'an apple a day keeps the medicine man away')

>>> doruk.go_to(bazaar)
"Doruk moves from Plaza to Bazaar"

>>> doruk.lend(burhan, 100, 1)
"Doruk and Burhan are not at the same place"

>>> doruk.lend(doruk, 1, 0)
"Doruk cannot lend to himself"

>>> doruk.go_to(plaza)
"Doruk moves from Bazaar to Plaza"

>>> doruk.lend(burhan, 80, 1)
"Doruk lends 80 Lari to Burhan at 1\% interest"

>>> doruk.lend(arslan, 6, 2)
"Doruk does not have enough Lari to lend"

>>> burhan.lend(arslan, 40, 2)
"Burhan lends 40 Lari to Arslan at 2\% interest"

>>> burhan.lend(mustafa, 50, 2)
"Burhan lends 50 Lari to Mustafa at 2\% interest"

>>> burhan.go_to(fountain)
"Burhan moves from Plaza to Fountain"

>>> doruk.settle()
"Doruk has nothing to settle"

>>> burhan.settle()
"Burhan has nothing to settle"

>>> burhan.go_to(plaza)
"Burhan moves from Fountain to Plaza"

>>> burhan.settle()
"Burhan collects 51.0 Lari from Mustafa, and collects 40.8 Lari from
    Arslan, and repays 80.8 Lari to Doruk"
```

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 —