CS1010S — Programming Methodology
National University of Singapore

Re-Practical Examination

22 April 2016

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 **18 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.org. Note that you can run the test cases on Coursemology.org **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 in the CS1010S folder 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 : M&M's Madness [10 marks]

You have been offered some M&M's candy.

A. Being the OCD that you are, you want all M&M's candy to be of the same colour. You are allowed to exchange as many pieces of candy as you want to the colour of your choice.

Suppose the candy that you have is represented by a string of characters, with each colour being represented by a character, e.g., the string `"rbygo"` represents candies of the colour red, blue, yellow, green and orange.

Implement the function `num_swaps` that takes as input a string representing the candies, and return the minimum number of candy you have to exchange in order to get all candies to be the same colour. [5 marks]

Examples:

```
>>> num_swaps("rbygo")
4  # change all candies to any one of the colours

>>> num_swaps("rrbbbg")
3  # change all the candies to blue

>>> num_swaps("rrrbbbggy")
6  # change the candies to either red or blue
```

B. Because blue is rad, you decide you want all your candies to be blue (`'b'`). But this time you cannot simply exchange any candy for a blue one. Specific colours can only be traded in for another specific colour based on a set of rules.

For example, one set of rules could be:

$$\begin{aligned} Red &\rightarrow Blue \\ Blue &\rightarrow Yellow \\ Yellow &\rightarrow Green \\ Green &\rightarrow Orange \\ Orange &\rightarrow Red \end{aligned}$$

So in order to exchange a green candy to a blue one, you first need to trade it for an orange one, then trade the orange for a red, and finally trade the red for a blue—which took 3 exchanges!

Suppose the rules are encoded in a Python dictionary, with the keys being the colour to trade in and the value the colour obtained from the trade. So the example above would be `{'r':'b', 'b':'y', 'y':'g', 'g':'o', 'o':'r'}`.

Implement a function `num_trades(candies, rules)` which takes as input a string which represents the candies, and a dictionary which represent the trading rules, and outputs the number of trades required to turn all the candies to blue (`'b'`).

You may assume that the trading rules given is sufficient to cover all the different colours of candies, and that it is always possible to end up with a blue candy. [5 marks]

Examples:

```
>>> num_trades('rrbbgg',
               {'r':'b', 'b':'y', 'y':'g', 'g':'o', 'o':'r'})
8  # it takes 1 trade to get red to blue, and 3 to get green to blue.

>>> num_trades('rbbryy', {'r':'b', 'b':'r', 'y':'r'})
6  # it takes 1 trade to get red to blue, and 2 to get yellow
   # to blue.

>>> num_trades('rygop',
               {'r':'b', 'b':'b', 'y':'b', 'g':'b', 'o':'b',
                'p':'b'})
5  # it only takes 1 trade to trade any colour to blue, even pink!
```

BONUS Suppose a particular set of trading rules will never give you a blue colour. If your function `num_trades` can detect and outputs infinity (produced by `float("inf")`), you will receive an addition **bonus of 3 marks** to be added to your total score.

Note: You cannot exceed 30 marks and the cap of 18 still applies if you are retaking the test.

Example:

```
>>> num_trades('rrbbgg',
               {'r':'b', 'b':'r', 'y':'g', 'g':'y'})
inf  # yellow will give green, and green will give yellow
```

```
     # you will never get to blue!

>>> num_trades('rbbb',
      {'r':'y', 'b':'r', 'y':'g', 'g':'g'})
inf  # All colours end up with green, which will always
      # trade back for green!
```

Question 2 : Rain Rain Go Away [10 marks]

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

Having been appointed as the chief statistician of the newly establish club Students Observing the Climate (SOC), you are given a record of the monthly rainfall in Singapore from 1975 to 2015.

A. The members are interested in knowing which month in a year has seen the highest rainfall (in mm) in a day, and how much rain was observed.

Implement the function `highest_rainfall(fname, start, stop)` that takes as inputs a filename, and a starting and ending year. The function will return a dictionary where the keys are years between the starting year (inclusive) and ending year (exclusive), and the values are tuples in the form `(month, rainfall)`, where `month` is the month containing the day of the highest rainfall, and `rainfall` is the amount of rainfall. [5 marks]

Sample execution:

```
>>> highest_rainfall('rainfall.csv', 2001, 2011)
{'2001': ('Dec', 211.1), '2002': ('Nov', 84.0),
 '2003': ('Jan', 194.4), '2004': ('Jan', 177.8),
 '2005': ('Nov', 133.8), '2006': ('Dec', 198.0),
 '2007': ('Dec', 159.0), '2008': ('Aug', 133.9),
 '2009': ('Apr', 86.6),  '2010': ('Jul', 121.1)}

>>> highest_rainfall('rainfall.csv', 1980, 1986)
{'1980': ('Jan', 133.5), '1981': ('Dec', 79.7),
 '1982': ('Aug', 121.5), '1983': ('Sep', 112.9),
 '1984': ('Feb', 154.4), '1985': ('Dec', 86.8)}
```

B. Another question the members want to answer is how many days in a year did it rain. More specifically, they want to know what percentage of the year were rainy days.

Implement a function `rainy(fname, start, stop):` that takes as inputs a filename, and a starting and ending year. The function will return a dictionary where the keys are years between the starting year (inclusive) and ending year (exclusive), and the values are the percentage of the year (rounded to two decimal places) which are rainy days.

Note that a leap year has 366 days while a non-leap year has 365 days. The function `isleap` from the `calendar` module can be used to check if a year is leap. It is also provided in Coursemology for this question without needing to import.

You can use the function `round(n, d)` to round n to d decimal places. [5 marks]

Sample execution:

```
>>> rainy('rainfall.csv', 1975, 1981)
{1975: 53.42, 1976: 43.99, 1977: 38.63, 1978: 46.85, 1979: 46.03,
 1980: 48.09}

>>> rainy('rainfall.csv', 2010, 2016)
{2010: 48.77, 2011: 51.51, 2012: 52.19, 2013: 56.44, 2014: 41.64,
 2015: 34.25}
# Last year seems to be pretty dry.
# Do you think it's global warming?
```

Question 3 : The Empire Strikes Back [10 marks]

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

Sith Lord was a title conferred upon individuals who followed the Sith tradition. Sith Lords drew upon the dark side of the Force, using it to gain power. After Darth Bane established the Rule of Two, Sith tradition dictated that there could only be two Sith Lords at any given time. *Source: Wookieepedia*

In this question, you will model and implement the class `Sith`. Each Sith has a name that begins with "Darth", and can learn and use force powers.

According to the established Rule of Two, a Sith can only have one master, or one apprentice at any time, not both. This means if a Sith has an apprentice, he cannot take on a master. Likewise, if a Sith has a master, he cannot take on an apprentice. The only way to break the master-apprentice relationship is for either one to die by fighting. (Usually the apprentice will fight his master.)

The winner of the fight is determined by the force powers each posses. For every force power a Sith knows that his opponent does not, he gets one point. For powers that both Siths knows, the Sith with the higher competency level of the power will get one point. After comparing all their powers, the Sith with the higher points will win and the loser will die. A dead Sith naturally cannot take new apprentices or perform any further actions.

To increase his chances in a fight, a Sith can choose to train his power, which will increase the competency level of that power. A Sith can also impart (teach) one of his power to his apprentice which his apprentice does not yet know. The apprentice will then posses this power with a competency level of 0.

A `Sith` is create with at least one input: a name (which is a string), and an optional number of addition of powers (which each is a string). For a Sith, the title "Darth" will be prepended to the front of the given name, and will form part of his name. `Sith` supports the following methods:

- `get_name()` which returns the name of the Sith (with Darth prepended).
- `get_powers()` which returns a list of pairs, with the power as the first element of the pair, and the competency level of the power as the second element.
- `get_master()` returns the name of the current master, or `None` if the Sith has no master at the moment.
- `get_apprentice()` returns the name of the current apprentice, or `None` if the Sith has no apprentice at the moment.
- `take_apprentice(apprentice)` takes as input an *apprentice*, which is a `Sith`, and returns a string based on the following conditions:
  - `"<name> is already dead"` if either Sith is dead. <name> would be the name of the dead Sith.
  - `"<Sith name> cannot take <apprentice name> as an apprentice"` if either Sith currently has a master or an apprentice.
  - Otherwise, the Sith will take on *apprentice* as his apprentice and the string `"<Sith name> takes <apprentice name> as an apprentice"`
- `impart()` will return a string based on the following conditions:
  - `"<Sith name> is already dead"` if the Sith is dead.
  - `"<Sith name> does not have an apprentice"` if the Sith does not currently have an apprentice.
  - It the Sith has a power which his apprentice does not know, then the Sith will impart that power to his apprentice. The string `"<Sith name> imparts <power> to <apprentice name>"` will be returned.
  - Otherwise, the apprentice knows every power that the Sith knows. The string `"<Sith name> has nothing to impart to <apprentice name>"` will be returned.
- `train(power)` takes a *power* (which is a string) as the only input, and returns a string based on the following:
  - `"<Sith name> is already dead"` if the Sith is dead.
  - `"<Sith name> has does not know <power>"` if the Sith does not know *power*.
  - Otherwise, the competency level of *power* will increase by one, and the string `"<Sith name> trains <power> to level <power level>"` will be returned.
- `fight(opponent)` takes in another `Sith` as the input, and will have both Siths fight each other. `"<dead Sith name> is already dead"` will be returned if either of the Siths is already dead before the battle.

  The outcome of the battle is based on scoring both Siths by:

– For each power one Sith knows that the other does not, the Sith will get 1 point.

– For powers which both Sith know, the Sith with the higher competency level for the power will get 1 point. In the event of a tie, nobody gets any points.

The winner will be the Sith with the higher score. The string `"<winner's name> kills <loser's name> in battle"` will be returned, and the losing Sith dies.

Otherwise, both Siths have the same points and the fight ends in a draw. Nothing happens and the string `"<Sith name> and <opponent name> are equally matched"` will be returned.

Sample execution:

```
>>> plagueis = Sith("Plagueis", "lightning", "choke")
>>> sidious = Sith("Sidious", "shadow")
>>> maul = Sith("Maul")
>>> tyranus = Sith("Tyranus")
>>> vader = Sith("Vader", "hate")

>>> plagueis.get_name()
'Darth Plagueis'
>>> plagueis.get_powers()
[('choke', 0), ('lightning', 0)]
>>> plagueis.train("choke")
'Darth Plagueis trains choke to level 1'
>>> plagueis.train("lightning")
'Darth Plagueis trains lightning to level 1'

>>> plagueis.take_apprentice(sidious)
'Darth Plagueis takes Darth Sidious as an apprentice'
>>> plagueis.get_apprentice()
'Darth Sidious'
>>> sidious.get_master()
'Darth Plagueis'

>>> plagueis.impart()
'Darth Plagueis imparts choke to Darth Sidious'

>>> sidious.train("lightning")
'Darth Sidious has does not know lightning'
# depends on what was imparted
>>> sidious.train("choke")
'Darth Sidious trains choke to level 1'
>>> sidious.get_powers()
[('choke', 1), ('shadow', 0)] # depends on what was imparted

>>> sidious.take_apprentice(maul)
'Darth Sidious cannot take Darth Maul as an apprentice'
# because Sidious currently has a master!
# He must kill his master to take an apprentice

>>> sidious.fight(plagueis)
'Darth Sidious and Darth Plagueis are equally matched'
# both knows one power that the other does not
# and the power that they both know, are at the same level

>>> plagueis.impart()
'Darth Plagueis imparts lightning to Darth Sidious'
>>> plagueis.impart()
'Darth Plagueis has nothing to impart to Darth Sidious'
# Plagueis imparts all his knowledge to Sidious
# He's gonna get a hurt real bad!

>>> sidious.get_powers()
[('shadow', 0), ('choke', 1), ('lightning', 0)]

>>> sidious.fight(plagueis)
'Darth Sidious and Darth Plagueis are equally matched'
# still equally matched because though Plagueis has one power
# that is higher level than Sidious, Sidious knows lightning
# that Plagueis does not

>>> sidious.train("shadow")
'Darth Sidious trains shadow to level 1'

>>> sidious.fight(plagueis)
'Darth Sidious and Darth Plagueis are equally matched'
# Training shadow does not help

>>> sidious.train("lightning")
'Darth Sidious trains lightning to level 1'
# Now this should do it

>>> sidious.fight(plagueis)
'Darth Sidious kills Darth Plagueis in battle'

>>> sidious.take_apprentice(maul)
'Darth Sidious takes Darth Maul as an apprentice'
# now he can take an apprentice

>>> plagueis.train("choke")
'Darth Plagueis is already dead'  # poor guy

>>> sidious.impart()
'Darth Sidious imparts shadow to Darth Maul'
>>> maul.train(maul.get_powers()[0][0])
'Darth Maul trains shadow to level 1'
```

```
>>> maul.train(maul.get_powers()[0][0])
'Darth Maul trains shadow to level 2'
>>> maul.train(maul.get_powers()[0][0])
'Darth Maul trains shadow to level 3'
# Maul trains hard

>>> maul.impart()
'Darth Maul does not have an apprentice'

>>> maul.take_apprentice(tyranus)
'Darth Maul cannot take Darth Tyranus as an apprentice'
# Maul cannot take an apprentice when he has a master
# He must kill his master!

>>> maul.fight(sidious)
'Darth Sidious kills Darth Maul in battle'  # @.@!!!
# He only knows one power

>>> sidious.take_apprentice(tyranus)
'Darth Sidious takes Darth Tyranus as an apprentice'
# Trivia: Darth Tyranus was formerly Count Dooku!

>>> vader.train("hate")
'Darth Vader trains hate to level 1'
# Vader has natural talent

>>> vader.fight(tyranus)
'Darth Vader kills Darth Tyranus in battle'
>>> sidious.take_apprentice(vader)
'Darth Sidious takes Darth Vader as an apprentice'
```

Note that the sample execution does not contain all possible cases and scenarios. You are expected to realize all corner cases and handle them accordingly.

You are advised to solve this problem incrementally. Even if you cannot fulfill 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 —