CS1010S — Programming Methodology
National University of Singapore

Practical Examination

16 April 2016

Time allowed: 2 hours

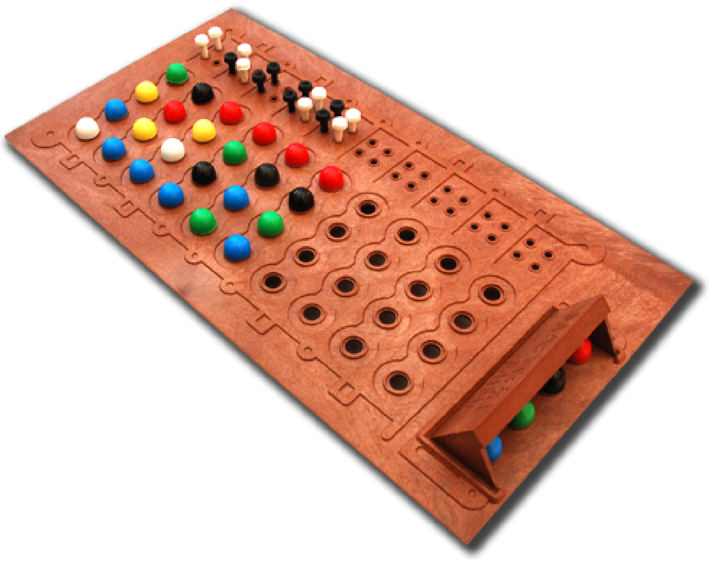
Instructions (please read carefully):

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. 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.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.
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 in the CS1010S folder 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 : Mastermind [10 marks]

Mastermind is a code breaking game for two players. One player, the *codemaker*, chooses a secret pattern of n pegs out of a set of coloured pegs, and the other player, the *codebreaker*, tries to guess the pattern.



For this question, the secret code will be a string with each character representing a peg, e.g., `"rbyg"` could represent a sequence of red, blue, yellow and green pegs. (It actually does not matter what colour the characters represent.) The codebreaker will then guess with a sequence of pegs to try and match the secret code. Note that while the game is commonly played with four pegs, we can play with any number of pegs in our version.

A. After making a guess, the codemaker will inform the codebreaker how many pegs of the guess are correct in both colour and position, i.e., the number of *exact matches*. Implement a function `correct_pegs(code, guess)` which takes as inputs two strings, the code and the guess, and returns the number of characters in the guess that match the code in both colour and position.

Assume the code and guess have the same length. [5 marks]

Examples:

```
>>> correct_pegs("rrbk", "rkbb")
2  # the 1st and 3rd characters match

>>> correct_pegs("ydgkyd", "yyyddd")
2  # the first and last characters match

>>> correct_pegs("rbygkp", "bygkpr")
0  # none of the characters match
```

B. In addition to the number of correct pegs, the codemaker will also inform the number of correct colour code peg **placed in the wrong position**. We call this the number of *partial matches*. If there are duplicated colours in the guess, the number of partial matches should correspond to the number of pegs that are in the wrong position without double counting.

For example, if the code is `"wwbbb"` and the guess is `"bbwww"`, the number of partial matches is 4. The first two 'b's in the guess are in the wrong position, and so are the next two 'w's. The last 'w' does not match any in the code because the first two 'w's in the code is already matched in the guess.

Implement a function `check_guess(code, guess)` that takes as inputs two strings, the code and the guess, and returns a tuple of 2 elements, the first being the number of exact matches and the second element the number of partial matches.

Assume the code and guess have the same length. [5 marks]

Examples:

```
>>> check_guess("wwbbb", "bbwww")
(0, 4)

>>> check_guess("rrbbyy", "rcbyrg")
(2, 2)

>>> check_guess("rbyg", "bbbb")
(1, 0)
```

Question 2 : Population Planning [10 marks]

Important note: You are provided with a data file `births.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 to the Ministry of Social and Family Development (MSF), you are given a record of the number of births in Singapore for each month from 1960 to 2015. A separate record of the number of births for each of the main races, Malay, Chinese, Indian and Others are also included.

A. Your bosses are interested in knowing for each year, within a certain range, which month had the highest total number of births for that year.

Implement the function `highest_month(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 is the month with the highest number of births for that year. [5 marks]

Sample execution:

```
>>> highest_month('births.csv', 2000, 2010)
{'2000': 'Nov', '2001': 'Oct', '2002': 'Oct', '2003': 'Oct',
 '2004': 'Oct', '2005': 'Sep', '2006': 'Oct', '2007': 'Nov',
 '2008': 'Sep', '2009': 'Oct'}
```

```
>>> highest_month('births.csv', 1965, 1975)
{'1965': 'Oct', '1966': 'Oct', '1967': 'Oct', '1968': 'Oct',
 '1969': 'Oct', '1970': 'Nov', '1971': 'Oct', '1972': 'Oct',
 '1973': 'Oct', '1974': 'Oct'}
```

B. One of the questions that we are all interested in is how the number of births for each of the races change annually. We can compute the change for a year as a percentage growth (or drop) from the previous year. For example, 8,309 Malays babies were born in year 2000 and 7,816 babies in 2001. Thus the number of babies born in 2001 is $7816/8309 = 94.07\%$ of babies born in 2000. So the percentage growth for 2001 is $94.07 - 100 = -5.93\%$. In other words, there is a decrease of 5.93% of babies born in 2001.

Implement a function `compare_growth(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 four races, and each value is a list of the percentage growth (rounded to two decimal places) of each increasing year.

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

Sample execution:

```
>>> compare_growth('births.csv', 1975, 1980)
# shows the growth of the years 1976, 1977, 1978 and 1979
{'Indians': [-0.6, -0.73, 7.15, 8.58],
 'Malays':  [-3.13, -1.94, 2.96, 4.2],
 'Chinese': [9.76, -12.56, 2.64, 2.48],
 'Others':  [2.64, -5.36, -1.63, 11.51]}

>>> compare_growth('births.csv', 1996, 2005)
{'Indians': [0.39, 1.36, -6.02, 5.45, -3.58, 2.77, -4.8, -2.51],
 'Malays':  [-3.88, -1.75, -5.69, 3.94, -5.93, -7.01, -7.86, -1.85],
 'Chinese': [-2.84, -11.29, 1.16, 10.33, -15.34, -1.04, -9.17,
    -1.09],
 'Others':  [1.33, 2.86, 2.1, 5.63, -0.38, 1.22, -2.25, 6.01]}
# apparently there was a sharp increase in Chinese births
# in year 2000, which happens to be the year of the Dragon. Hmm..
```

Question 3 : Return of the Jedi [10 marks]

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

"A Jedi was a Force-sensitive individual, most often a member of the Jedi Order, who studied, served, and used the mystical energies of the Force; usually, the light side of the Force." - *Source: Wookieepedia*

In this question, you will model and implement the class `Jedi`. Each Jedi has an overall skill level and is able to learn to use some force powers. Each of the force powers he knows has competency level, which cannot exceed his overall skill level.

A Jedi is also able to take on another Jedi who has the same or higher skill level as his master. He can then learn new powers from his master. Every new force power learnt will start with a competency level of 0. It is possible for a master to have several disciples at once.

Whenever a Jedi trains, he will increase the competency level of one of his powers by 1. The power that is chosen must **have the lowest level among the powers and be lower than his current overall skill level**. If all his powers are at the same level as his overall skill level, then his skill level will increase by 1.

When a Jedi's skill level exceeds that of his master, then he will swap places with his master, i.e., he will become his master's master, and the former master will now be his disciple. The former's master's master will then be his new master (See Figure 1).

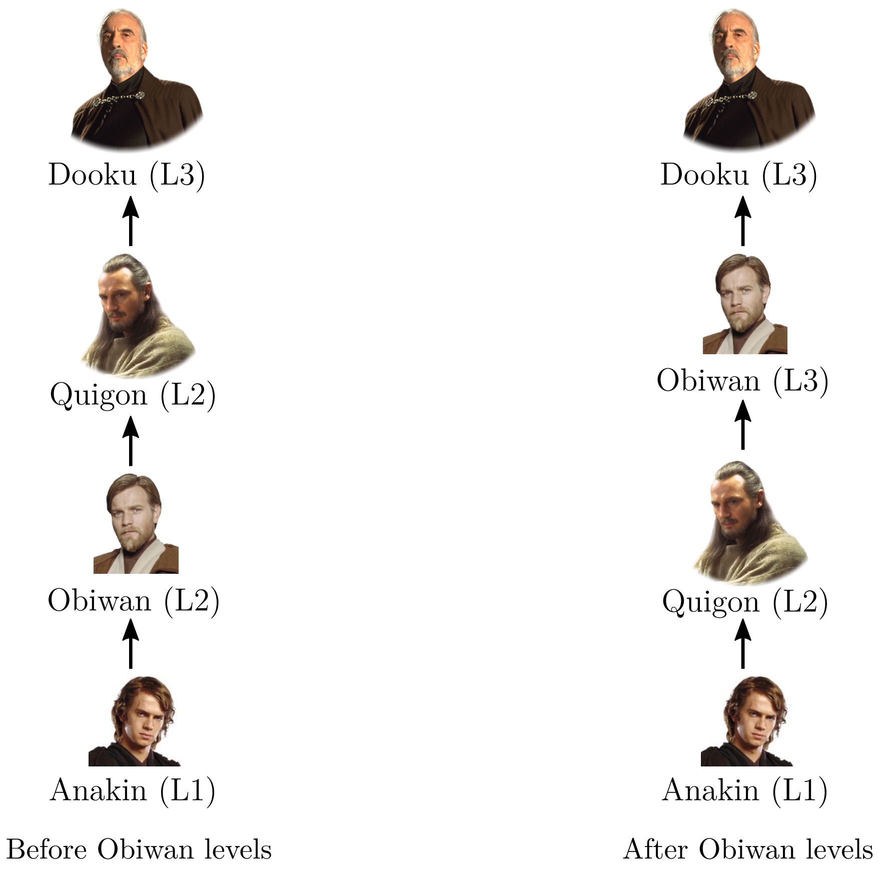


Figure 1: Suppose the arrow indicates a master-disciple relationship. If Obiwan levels above Quigon, but not over Dooku, then Obiwan will take over as Quigon's master, and Dooku will be Obiwan's master. Obiwan's disciple Anakin will then become Quigon's disciple.

A `Jedi` is created with two inputs: a name (which is a string) and a level (which is an integer), and a third optional input, a sequence of powers (which is a sequence of strings). This will create a Jedi of the given name, level and a list of powers the Jedi knows, if any. `Jedi` supports the following methods:

- `get_name()` which returns the name of the Jedi.

- `get_level()` which returns the overall skill level of the Jedi.
- `get_powers()` which returns a list of pairs, with the power as first element of the pair, and the level as the second element.
- `get_master()` returns the name of the current master, or `None` if the Jedi has no master at the moment.
- `set_master(master)` takes as input another Jedi, and returns a string based on the following conditions:
  - If the master has a lower level than the Jedi, the string `"<master name> has a lower level than <Jedi name>"` is returned.
  - If the master is currently the master of the Jedi, the string `"<master name> is already the master of <Jedi name>"` is returned.
  - Otherwise, the Jedi will take master as his new master, and the string `"<master name> takes <Jedi name> as a disciple` will be returned.
- `get_disciples()` will return a list of names of the current disciples of the Jedi.
- `train()` will return a string based on the following conditions:
  - If the Jedi has not learnt any powers, the string `"<Jedi name> has not learnt any powers"` will be returned.
  - If the Jedi has at least one power which level is less than his overall skill level, then any one of such power will increase its level by one. The string `"<Jedi name> trains <power> to level <new level>"` is returned.
  - Otherwise, all powers are at the same level as the Jedi's skill level. The Jedi's skill level will then increase by one and the string `"<Jedi name> levels up to level <level>"` will be returned.

    Note that If the Jedi's level exceeds his master's level, he will automatically take on his master's master as his new master.
- `learn()` will return a string based on the following conditions:
  - If the Jedi has no master, the string `"<Jedi name> does not have a master"` will be returned.
  - If the master has learnt at least one power that the Jedi has not learnt, then the Jedi will learn a new power from the master. The string `"<Jedi name> learns <power> from <master's name>"` will be returned.
  - Otherwise, if the Jedi has learnt every power that the master knows. The string `"<Jedi name> has nothing to learn from <master's name>"` will be returned.

Sample execution:

```
dooku  = Jedi("Dooku",  3, ("push", "jump"))
quigon = Jedi("Quigon", 2, ("mind trick",))
obiwan = Jedi("Obiwan", 1)  # starts out with no powers
anakin = Jedi("Anakin", 1)  #
```

```
luke   = Jedi("Luke",   1)  #


>>> dooku.get_name()
'Dooku'
>>> dooku.get_level()
3
>>> dooku.get_powers()
[('push', 0), ('jump', 0)]
>>> dooku.train()
'Dooku trains jump to level 1'  # he could also train push

>>> dooku.get_powers()
[('jump', 1), ('push', 0)]
>>> dooku.train()
'Dooku trains push to level 1'  # must train push as it is
                                # the lowest level
>>> dooku.get_powers()
[('jump', 1), ('push', 1)]  # both have to be at level 1

>>> dooku.set_master(quigon)
'Quigon has a lower level than Dooku'
>>> quigon.set_master(dooku)
'Dooku takes Quigon as a disciple'
>>> quigon.get_master()
'Dooku'
>>> quigon.train()
'Quigon trains mind trick to level 1'  # he only knows mind trick

>>> obiwan.train()
'Obiwan has not learnt any powers'
>>> obiwan.learn()
'Obiwan does not have a master'
>>> obiwan.set_master(quigon)
'Quigon takes Obiwan as a disciple'

>>> obiwan.learn()
'Obiwan learns mind trick from Quigon'  # since Quigon only knows
                                        # mind trick
>>> obiwan.get_powers()
[('mind trick', 0)]  # newly learned powers start at level 0

>>> obiwan.train()
'Obiwan trains mind trick to level 1'
>>> obiwan.get_level()
1
>>> obiwan.train()
'Obiwan levels up to level 2'  # since all his powers are at level 1
```

```
>>> obiwan.get_level()
2

>>> obiwan.get_master()
'Quigon'
>>> quigon.get_disciples()
['Obiwan']
>>> obiwan.learn()
'Obiwan has nothing to learn from Quigon'

>>> obiwan.train()
'Obiwan trains mind trick to level 2'

>>> anakin.set_master(obiwan)
'Obiwan takes Anakin as a disciple'
>>> luke.set_master(obiwan)
'Obiwan takes Luke as a disciple'
>>> obiwan.train()
'Obiwan levels up to level 3'
>>> obiwan.get_master()
'Dooku'  # Quigon is level 2, so Obiwan has exceed his master's level
         # So now he takes on Quigon's master as his new master

# The master-disciple hierarchy has changed as illustrated in Fig 1
>>> quigon.get_disciples()
['Anakin', 'Luke']
>>> obiwan.get_disciples()
['Quigon']
>>> dooku.get_disciples()
['Obiwan']

>>> obiwan.learn()
'Obiwan learns push from Dooku'   # he could also learn jump

>>> obiwan.get_powers()
[('mind trick', 2), ('push', 0)]  # or ('jump', 0)
>>> obiwan.train()
'Obiwan trains push to level 1'
```

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 —