CS1010S — Programming Methodology
National University of Singapore

# Re-Practical Examination

21 April 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 **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. 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 : Check digit [10 marks]

A *check digit* is a number or letter that is used to validate a series of numbers. It is frequently the last character in an identification number, like your student number.

To calculate the check digit for the NUS student number, the digits of the number is summed up, and the modulo (remainder after division) $r$ of a prime number $p$ is taken. The character is determined by using $r$ to look up a matching table.

For the NUS student number, $p$ is 13 and the following lookup table is used:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Y | X | W | U | R | N | M | L | J | H | E | A | B |

For example, the check digit for the number `0113093` would be calculated as $0+1+1+3+0+9+3=17$ and $17 \bmod 13 = 4$. Looking up the table, 4 represents R, so the correct student number is `0113093R`.

**A.** The function `check_digit` takes as inputs an identification number (which is a string of digits) and a lookup up table (which is a dictionary). It outputs the respective check digit (character) of the identification number.

Examples:

```
nus_matric = {
    0: 'Y',
    1: 'X',
    2: 'W',
    3: 'U',
    4: 'R',
    5: 'N',
    6: 'M',
    7: 'L',
    8: 'J',
    9: 'H',
    10: 'E',
    11: 'A',
    12: 'B'
}

check_digit("0113093", nus_matric)
'R'

check_digit("0129969", nus_matric)
'E'
```

Implement the function `check_digit`. The prime number $p$ is simply the number of elements of the input lookup dictionary. You can check it with your own student number too!

[5 marks]

**B.** Oftentimes, instead of simply summing up the digits on the identification number, each digit is multiplied by a weight before being added.

For example, for the Singapore NRIC number, the digits are multiplied by the weights 2,7,6,5,4,3,2, i.e., the leftmost digit is multiplied by 2, the next by 7, then 6 and so on.

The number `9702743` would be summed by doing: $9\times2+7\times7+0\times6+2\times5+7\times4+4\times3+3\times2=123$. In this example, $p=11$ so 123 mod 11 = 2 which is "I" according to its lookup table:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| J | Z | I | H | G | F | E | D | C | B | A |

The function `weighted_check_digit` takes as inputs an identification number (which is a string of digits), a lookup up table (which is a dictionary) and the weights (which is a string of digits). It outputs the respective check digit (character) of the identification number.

Examples:

```
>>> sg_nric = dict(enumerate("JZIHGFEDCBA"))
>>> sg_weights = "2765432"

>>> weighted_check_digit("9702743", sg_nric, sg_weights)
'I'

>>> weighted_check_digit("9875133", sg_nric, sg_weights)
'E'
```

Provide an implementation for the function `weighted_check_digit`. You may assume that the length of the weights will match the length of the digit. You can check it with your own NRIC number too. [5 marks]

## Question 2 : COE bidding [10 marks]

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

Before owning a vehicle in Singapore, one needs to obtain a Certificate of Entitlement (COE) for the vehicle. Bidding for a COE takes place twice a month in different categories.

To estimate the demand on vehicle ownership, we are only interested in the number of bids received and successful bids for this.

The first line of the data file is a header which describes each column of data. You may assume that all data files follow this same fixed order of columns and there are no duplicate and conflicting rows.

**A.** Your new bosses at LTA wants to know the average over-subscription rate for each category in a year. The over-subscription rate for a bidding round is defined as the (number of bids received / number of successful bids).

Implement the function `oversub_avg` that takes as input a filename and a year (as strings). The function returns a dictionary which keys are the category obtained from the data file and the values are the yearly average (rounded to 3 dp) of all the over-subscription rates for the corresponding category. You can use the function `round(n, d)` to round $n$ to $d$ decimal places.

Note, you should not assume that there is always 2 bidding rounds per month and that the data contains every month in the year. [5 marks]

Sample execution:

```
>>> oversub_avg("coe.csv", "2013")
{'A': 2.032, 'B': 1.922, 'C': 1.4, 'D': 1.252, 'E': 1.834}

>>> oversub_avg("coe.csv", "2015")
{'A': 1.536, 'B': 1.499, 'C': 1.614, 'D': 1.319, 'E': 1.604}
```

**B.** LTA also wants to know which category was the most over-subscribed in each bidding round.

Implement the function `most_oversub` that takes as input a filename and a year (as strings). The function returns a dictionary which keys are the categories obtained from the data file and the values are number of times the category was the most oversubscribed category of the bidding round in the given year.

You should still assume that the categories are not fixed and there can be any number of bidding rounds in a month. [5 marks]

Sample execution:

```
>>> most_oversub("coe.csv", "2013")
{'A': 9, 'B': 8, 'C': 2, 'D': 0, 'E': 5}
```

```
>>> most_oversub("coe.csv", "2015")
{'A': 5, 'B': 2, 'C': 10, 'D': 0, 'E': 7}
```

## Question 3 : Monsters Inc. [10 marks]

– BACKGROUND STORY (Okay to skip) –

*The city of Monstropolis in the monster world is powered by energy from the screams of human children. At the Monsters, Inc. factory, skilled monsters employed as "scarers" venture into the human world to scare children and harvest their screams, through doors that activate portals to the children's bedroom closets. It is considered dangerous work, as human children are believed to be "toxic". Energy production is falling because children are becoming less easily scared, and company chairman Henry J. Waternoose is determined to find a solution. James P. "Sulley" Sullivan is the organization's top scarer, but his chief rival, chameleon-like monster Randall Boggs, is close behind.*

*One day, Sulley discovers that Randall has left a door activated on the scare floor, and a small girl has entered the factory. After several desperate failed attempts to put her back, Sulley conceals her and takes her out of the factory. He interrupts his best friend Mike Wazowski's date with his girlfriend Celia, and chaos erupts when the child is discovered. Sulley and Mike manage to escape with the child before the Child Detection Agency (CDA) quarantines the restaurant, and back at their home, discover that she is not actually toxic after all. Sulley grows attached to her and calls her "Boo", while Mike is just anxious to be rid of her.*

*When they smuggle her back into the factory in an attempt to send her home, Randall discovers her and tries to kidnap her, but mistakenly kidnaps Mike instead. He straps Mike to a large machine called "The Scream Extractor", which he intends to use to revolutionize the scaring industry and solve the monster world's energy problems by forcefully extracting screams from kidnapped human children, though not without harming them in the process. Before Randall can use the machine on Mike, Sulley intervenes and reports Randall to Waternoose. Waternoose, secretly in league with Randall, instead exiles Mike and Sulley to the human world in the Himalayas. The two are taken in by a Yeti, who suggests they travel to a nearby village to return to the monster world. Sulley prepares to leave, but Mike angrily refuses to go with him. Meanwhile, Randall is preparing to use the Scream Extractor on Boo, but Sulley arrives and saves her. Randall and Sulley fight, and after Mike returns and the two reconcile, they overpower Randall, take Boo and flee.*

*Randall pursues them to the giant door vault, and a wild chase ensues among the millions of doors as they move in and out of the storage vault on rails to the factory floor. Boo's laughter causes all the doors to activate, allowing the chase to pass in and out of the human world. Randall attempts to kill Sulley, but Boo overcomes her fear and attacks him. Sulley and Mike trap Randall in the human world, where two residents at a trailer park beat him with a shovel, mistaking him for an alligator. Sulley and Mike then trick Waternoose into revealing his plot with Randall, while Mike secretly records the entire conversation. The CDA then arrests Waternoose, and Sulley and Mike say goodbye to Boo and return her home before her door is shredded to prevent any more contact with her.*

*With the factory temporarily shut down, Sulley is named the new CEO of Monsters, Inc. Under his leadership, the energy crisis is solved by harvesting children's laughter instead of screams, as laughter has been found to be much more potent. Mike takes Sulley aside, revealing he has rebuilt Boo's door, and only needs one final piece, which Sulley took as a memento. Sulley enters and joyfully reunites with Boo.*

**Source: Wikipedia**

Figure 1: Mike and Sully

— START OF ACTUAL QUESTION —

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

In this question, you will model and implement two classes: `Door` and `Monster`.

A `Door` represents the door that monsters use to gain access into a child's bedroom. Since each door leads to only one child's bedroom, we will treat `Door` to be synonymous with the child in the room. Each child has a certain level of bravery which determines how loud they scream when they are scared by the monster. Their bravery increases by each monster they encounter.

Every `Monster` has a certain scariness factor. Monsters will then enter doors and scare the child in the room. It is possible for more than one monster to be in a room at the same time.

When a monster in a room decides to scare the child, every monster in the room will spring into action and participate in the scaring. The effective scariness of a monster is determined by how many times the child has been scared by the monster. Suppose $n$ is the number of times the child is being scared by the monster, and $s$ is the monster's scariness, its effective scariness will be $\frac{s}{n}$. In other words, the first time a monster scares a child, it will have its original scariness. The second time it will be halved, the third time it will be one third, the fourth will be one quarter and so on.

The energy of the scream obtained from the child when being scared is computed by subtracting the the bravery of the child from the sum of the effective scariness of all the monsters currently in the room. The energy obtained is then divided equally amongst

the monsters and added to their energy tally. The monsters will then exit the room and the child's bravery will increase by the number of monsters that scared them.

`Door` is created with two inputs, its name (which is a string) and a bravery value (which is an integer). It supports the following methods:

- `get_bravery` returns the current bravery of the child behind the door.
- `get_monsters` returns a comma separated string of the names of the monsters currently in the room *sorted in ascending order*, e.g. `"<Monster 1 name>, <Monster 2 name>, ..."`. In other words, the monsters names are separated by `", "`.

  Hint: You can use the `join` function for strings to join a list of strings, for example `" and ".join(["John", "Mary", "Jane"])` will return the string `"John and Mary and Jane"`. Experiment with it in the IDLE shell.

`Monster` is created with two inputs: its name (which is a string) and a scariness value (which is an integer). It supports the following methods:

- `get_name()` returns the name of the monster.
- `get_energy()` returns the total energy tally obtained by the monster.
- `get_location()` returns the string `"<Monster name> is in <Door name>"` if the monster is currently in the room behind the door it entered. Otherwise the string `"<Monster name> is on the Scream Floor"` is returned.
- `enter(door)` takes a `Door` as its input. It returns a string based on the following conditions:
  - If the monster is currently in the room behind the same door, the string `"<Monster name> has already entered <Door name>"` is returned.
  - If the monster is in another room behind another door, the string `"<Monster name> is currently in <Other Door name>"` is returned.
  - Otherwise, the monster enters the door and the string `"<Monster name> enters <Door name>"` is returned.
- `scare()` takes no input and returns a string while performing some actions based on the following conditions:
  - If the monster is not currently in any room behind a door, the string `"<Monster name> has not entered any door"` is returned.
  - Otherwise, all the monsters in the room which the monster is in will scare the child in union. The energy obtained from the child's scream is computed by subtracting the bravery of the child from the sum total of the *effective* scariness of every monster, as described above.

    After scaring, the child's bravery will increase by the number of monsters present and the monsters will all exit the room.

If the energy obtained is 0 or less, then the string `"<names of monsters> failed to obtain energy from <Door name>"` is returned, where `<names of monsters>` is a comma separated string of the names of the monsters sorted in ascending order.

Otherwise, the energy will be equally divided amongst the monsters and added to their energy tally and the string `"<names of monsters> got <total energy> energy from <Door name>"` is returned, where `<names of monsters>` is a comma separated string of the names of the monsters sorted in ascending order.

Provide an implementation for the classes `Door` and `Monster`.

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.

A function `leaderboard` has been defined that ranks the monsters according to their energy tally for you.

Sample Execution:

```
>>> leaderboard(sully, mike, randall)
"Sully: 0
Mike: 0
Randall: 0"

>>> sully.get_energy()
0

>>> sully.get_location()
"Sully is on the Scream Floor"

>>> sully.enter(mary)
"Sully enters Mary's Room"

>>> sully.get_location()
"Sully is in Mary's Room"

>>> mary.get_monsters()
"Sully"

>>> sully.scare()
"Sully got 11.0 energy from Mary's Room"
# Sully's scariness (12) - Mary's bravery (1)

>>> leaderboard(sully, mike, randall)
"Sully: 11.0
Mike: 0
Randall: 0"
```

```
>>> mary.get_bravery()
2  # Mary got a bit bravier

>>> sully.get_location()
"Sully is on the Scream Floor"

>>> randall.scare()
"Randall has not entered any door"

>>> randall.enter(mary)
"Randall enters Mary's Room"

>>> randall.scare()
"Randall got 6.0 energy from Mary's Room"
# Randall's scariness (8) - Mary's bravery (2)

>>> sully.enter(mary)
"Sully enters Mary's Room"

>>> sully.enter(ted)
"Sully is currently in Mary's Room"

>>> mary.get_bravery()
3

>>> sully.scare()
"Sully got 3.0 energy from Mary's Room"
# Scully's scariness is now 6 because Mary has seen him once before

>>> mary.get_bravery()
4

>>> sully.enter(mary)
"Sully enters Mary's Room"

>>> sully.scare()
"Sully failed to obtain energy from Mary's Room"
# Sully's scariness is now 12/3=4

>>> randall.enter(mary)
"Randall enters Mary's Room"

>>> randall.scare()
"Randall failed to obtain energy from Mary's Room"

>>> mary.get_bravery()
6  # Mary got braver even though Randall failed
```

```
>>> randall.get_location()
"Randall is on the Scream Floor"

>>> sully.enter(ted)
"Sully enters Ted's Room"

>>> mike.enter(ted)
"Mike enters Ted's Room"  # Mike goes to understudy Sully

>>> leaderboard(sully, mike, randall)
"Sully: 14.0
Randall: 6.0
Mike: 0"

>>> ted.get_monsters()
"Mike, Sully"

>>> ted.get_bravery()
1

>>> mike.scare()
"Mike, Sully got 12.0 energy from Ted's Room"
# Sully (12) + Mike (1) - Ted bravery(1)

>>> leaderboard(sully, mike, randall)
"Sully: 20.0
Mike: 6.0
Randall: 6.0"  # 12 energy equally divided between Sully and Mike

>>> ted.get_bravery()
3  # Ted's bravery up by 2

>>> sully.enter(ted)
"Sully enters Ted's Room"

>>> mike.enter(ted)
"Mike enters Ted's Room"

>>> randall.enter(ted)
"Randall enters Ted's Room"

>>> randall.scare()
"Mike, Randall, Sully got 11.5 energy from Ted's Room"
# 0.5 + 8 + 6 - 3 = 11.5
# Ted has seen Mike and Sully once before but it's his first
# time seeing Randall

>>> ted.get_bravery()
```

```
6  # Ted's bravery up by 3

>>> sully.enter(boo)
"Sully enters Boo's Room"

>>> mike.enter(boo)
"Mike enters Boo's Room"

>>> randall.enter(boo)
"Randall enters Boo's Room"

>>> sully.scare()
"Mike, Randall, Sully failed to obtain energy from Boo's Room"
# Nobody can ever scare Boo

>>> mike.get_energy()
9.833333333333334

>>> leaderboard(sully, mike, randall)
"Sully: 23.83
Mike: 9.83
Randall: 9.83"  # Sully remains top scarer
```

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 —