CS1010S — Programming Methodology
National University of Singapore

Practical Examination

12 November 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 pick a file 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 : Josepheus Problem [10 marks]

In computer science and mathematics, the Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game.

People are standing in a circle waiting to be executed. Counting begins at a specified point in the circle and proceeds around the circle in aspecified direction. After a specified number of people are skipped, the next person is executed. The procedure is repeated with the remaining people, starting with the next person, going in the same direction and skipping the same number of people, until only one person remains, and is freed.

- Source: Wikipedia

We will attempt to solve the Josepheus Problem with a series of steps.

A. Suppose you are given a sequence of bools, i.e., `True` or `False` values. Given an index (where the first index starts from 0), find the next index that has a `True` value. If the end of the sequence is reached without finding a `True` value, the search wraps around and continues from index 0.

For example, given a sequence (True, True, False, True, False). The result with the given index i will be:

i	output	Explanation
0	1	The next index contains `True`.
1	3	The next index is `False` so we continue until we reach a `True`.
2	3	Same as above.
3	0	We reach the end without finding a `True` so we wrap around
4	0	Same reason as above.

Implement the function `next_pos(seq, i)` that takes as inputs a sequence of bools and a non-negative starting index, and outputs the index that contains the next `True` value. You may assume that there is at least one `True` value in the sequence. [4 marks]

Examples:

```
>>> next_pos((True, True, False, True, False), 1)
3

>>> next_pos((True, True, False, True, False), 3)
0

>>> next_pos((False, False, True, False, False, False), 3)
2
```

B. The next step is to extend the previous function to take in another parameter k, and return the index of the next kth `True` value.

For example, using the same sequence (True, True, False, True, False).

If $i = 0$ and $k = 2$, we start from index 0. The next `True` value is at index 1. The 2nd `True` value is at index 3. So the output is 3.

If $i = 2$ and $k = 3$, then the 1st `True` value is at index 3, the 2nd at index 0 and the 3rd at index 1. So the output will be 1.

Implement the function `next_k(seq, i, k)` that takes as inputs a sequence of bools and a starting index i and number of steps k, where i and k are non-negative integers. The function outputs the index containing the next kth `True` value.

You may again assume that there is at least one `True` value in the sequence and that the function `next_pos` in part A is correct and given.

Hint: It might be easier to use recursion. [4 marks]

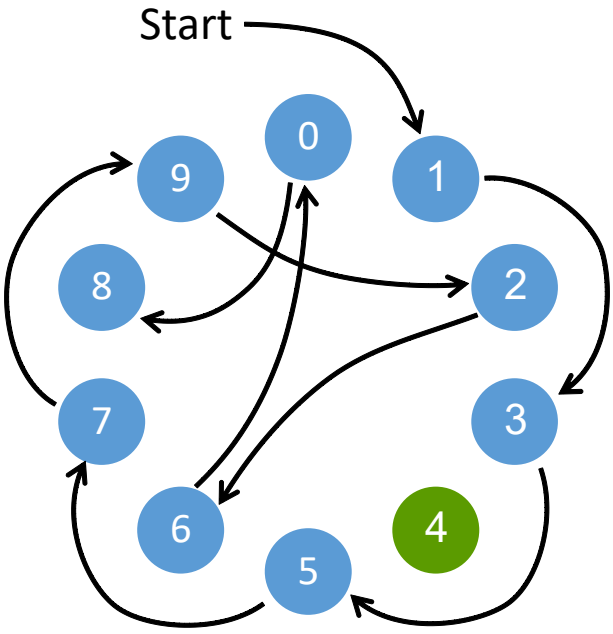
Examples:

```
>>> next_k((True, True, False, True, False), 0, 2)
3

>>> next_k((True, True, False, True, False), 2, 3)
1
```

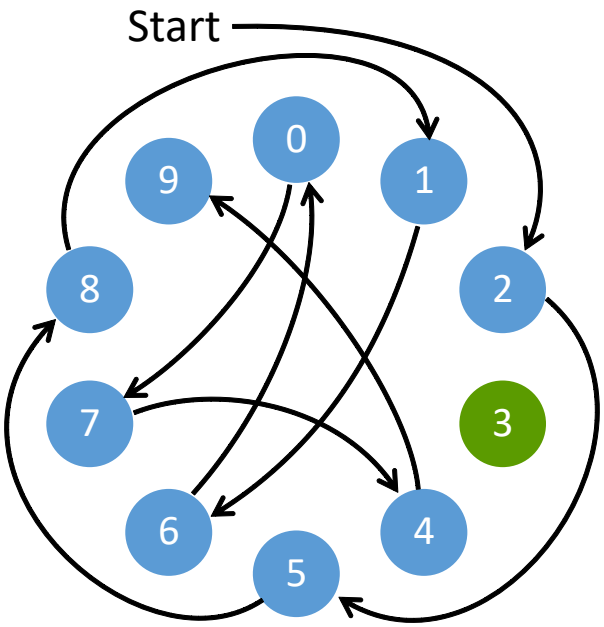
C. Now we will attempt to solve the Josepheus Problem using the functions we have just defined.

n number of objects are placed in a circle and labeled starting from 0 to $n-1$. For a given integer k, starting at 0, every kth object is removed until only one is left. For example, the image below shows when $n = 10$ and $k = 2$.



Because $k = 2$, index 0 is skipped and 1 is removed. Then skipping 2, 3 is removed. Then 5, 7 followed by 9 are removed. At this point, 0, 2, 4, 6 and 8 are left. Next, we skip 0 and the next number is 2 since 1 has already been removed. Then 6 followed by 0 then 8 is removed, leaving 4 as the only number left.

Another example below shows $n = 10$ and $k = 3$.



Skipping to the 3rd index, 2 is the first to be removed, followed by 5, then 8 then 1. This continues with every 3rd number removed and finally 3 is left.

Implement the function `josephus(n, k)` which takes as inputs integers $n > 0$ and $k \geq 1$, and outputs the last number remaining.

You are to solve the problem computationally without using a formula. You may assume the functions `next_pos` and `next_k` are correct and given. [2 marks]

Examples:

```
>>> josephus(10, 2)
4

>>> josephus(10, 3)
3

>>> josephus(41, 2))
18
```

Question 2 : Graduation [10 marks]

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

The Ministry of Education (MOE) is interested to track the trends in college education. Knowing that you are a Python guru, they have provided you data on the number of graduates for each course by gender for the past several years.

Each row of the data file reflects the year, sex, course and number of graduates. Some courses were not offered in some years, so the number of graduates is reflected as `'na'`. You can use the string function `str.isdigit()` to check if a string is a digit.

A. MOE is interested to know for a particular course, which year had the highest number of graduates.

Implement the function `highest_year(fname, course)` that takes as input the name of the data file and a course. The function will return a tuple containing the year with the highest number of graduates and the corresponding number of graduates.

Note that the years are formatted as strings. [5 marks]

Sample execution:

```
>>> highest_year("graduates.csv", "Law")
('2013', 368)

>>> highest_year("graduates.csv", "Medicine")
('2014', 261)

>>> highest_year("graduates.csv", "Dentistry")
('2013', 48)
```

B. Another interesting demographic that people are curious about is the gender ratio in each course.

Implement a function `gender_ratio(fname, course)` that takes as inputs a filename, and a course. The function will return a dictionary where the keys are years, and the values are the ratio of male to female graduates, to 2 decimal places. You can assume that both male and female records for a given course is found in the datafile. Also, if a course has no female graduates, then the resulting value will be `float("inf")`.

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

Sample execution:

```
>>> gender_ratio("graduates.csv", "Law")
{'1993': 0.99, '1994': 0.88, '1995': 0.94, '1996': 0.71,
 '1997': 0.93, '1998': 0.68, '1999': 1.19, '2000': 0.94,
 '2001': 0.6,  '2002': 0.95, '2003': 0.73, '2004': 0.62,
 '2005': 0.5,  '2006': 0.52, '2007': 0.68, '2008': 0.82,
 '2009': 0.78, '2010': 1.54, '2011': 0.58, '2012': 0.68,
```

```
 '2013': 1.06, '2014': 1.02}

>>> gender_ratio("graduates.csv", "Accountancy")
{'1993': 0.74, '1994': 0.72, '1995': 0.73, '1996': 0.61,
 '1997': 0.5,  '1998': 0.54, '1999': 0.62, '2000': 0.51,
 '2001': 0.53, '2002': 0.42, '2003': 0.42, '2004': 0.6,
 '2005': 0.43, '2006': 0.38, '2007': 0.45, '2008': 0.55,
 '2009': 0.58, '2010': 0.69, '2011': 0.65, '2012': 0.76,
 '2013': 0.58, '2014': 0.73}

>>> gender_ratio("graduates.csv", "Services")
{'2007': inf,  '2008': 0.33, '2009': 0.36, '2010': 0.83,
 '2011': 0.48, '2012': 0.82, '2013': 0.34, '2014': 0.62}
```

Question 3 : Pokémon Go! [10 marks]

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

For this question, you are to design and model the trainers and pokémons in the popular game of Pokémon.

Trainers are humans who go around catching pokémons. Once a pokémon is caught, it is owned by the trainer until given away to another trainer.

Trainers and pokémons have a level. Trainers cannot catch or receive pokémons that have a higher level than them. Pokémons can power-up their level, but not above that of their trainer, unless all the trainer's pokémons are at the trainer's level, then the trainer will level up instead.

Certain pokémons can also evolve to a different pokémon.

The class `Trainer` will model a trainer and the class `Pokemon` will model a pokémon.

A `Trainer` is created with one input: a name, which is a string. This will create a trainer with the given name with a level of 1. `Trainer` supports the following methods:

- `curr_level()` which returns the string `"<Trainer name> is currently at level <Trainer level>"`.
- `list_pokemons()` which returns a tuple containing the names of the pokémons in the trainer's possession.
- `catch(pokemon)` takes as input a `Pokemon` and returns a string based on the following conditions:
  - If `pokemon` is already owned by a trainer, the string `"<pokemon name> is already owned by <owner name>"`, where `<owner name>` is the name of `pokemon`'s owner.
  - If `pokemon`'s level is higher than the trainer's level, the string `"<pokemon name> got away!"` is returned.
  - Otherwise, `pokemon` is caught by the trainer and the string `"<trainer name> has caught <pokemon name>"` is returned.
- `give(pokemon, new_trainer)` takes as input a `Pokemon` and a `Trainer`, and returns a string based on the following conditions:
  - If the trainer does not own `pokemon`, the string `"<trainer name> does not own <pokemon name>"` is returned.
  - If the trainer is the `new_trainer`, the string `"Cannot give to self"` is returned.
  - If `pokemon`'s level is higher than `new_trainer`'s level, the string `"<new_trainer>'s level is not high enough"` is returned.
  - Otherwise, `pokemon` is transferred from the current trainer to `new_trainer`, and the string `"<pokemon name> transfers from <trainer name> to <new_trainer name>"` is returned.

A `Pokemon` is created with at least two inputs, a name (which is a string) and a level (which is an integer), followed by an arbitrary number of names (which are strings). This creates a pokemon with the given name and level, and if more names are specified, the names indicate the evolution order of the pokemon. For example, `Pokemon("Charmander", 3, "Charmelon", "Charlizard")` will create a pokemon named "Charmander", which can evolve into a pokemon named "Charmelon", which can then further evolve into a pokemon named "Charlizard".

`Pokemon` supports the following methods:

- `curr_level()` returns the string `"<pokemon name> is currently at level <pokemon level>`.
- `owned_by()` returns the string `"<pokemon name>'s owner is <trainer name>"` where `<trainer name>` is the name of the trainer that owns the pokemon.

  Otherwise, if the pokemon is not owned by any trainer, the string `"<pokemon name> has no trainer"` is returned.
- `power_up()` returns a string based on the following conditions:
  - If the pokemon is not owned by any trainer, the string `"<pokemon name> needs a trainer to power up"` is returned.
  - If the pokemon's level is lower than its trainer's level, the pokemon's level will increase by 1, and the string `"<pokemon name> powers up to level <new level>"` is returned, where `<new level>` is the new level of the pokemon.
  - If all its trainer's pokemons are at the trainer's level, then the trainer's level will increase by 1, and the string `"<trainer name> levels up to level <new trainer level>"` is returned.
  - Otherwise, the string `"<pokemon name> cannot exceed trainer's level"` is returned.
- `evolve()` takes no input and either:
  - returns a new `Pokemon` if the pokemon is able to evolve, i.e., there is another name given in the optional inputs when the pokemon was created. The name of the new pokemon will be the evolved name and the level will be the same as the current pokemon.

    If the pokemon has a trainer, then the trainer will now own the new pokemon and no longer own the current pokemon. Note that a new `Pokemon` object is being created, and not the pokemon simply renamed. You do not need to worry about deleting the current pokemon.
  - raise an `EvolveError` if the pokemon cannot evolve further. `EvolveError` should contain a property `pokemon` that refers to the current pokemon that tried to evolve, and when printing the error, the string `"<pokemon name> cannot evolve further"` will be displayed. Hint: The constructor of the class `Exception` takes a string as input.

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 strings, especially spaces and punctuation.

Sample Execution:

```
>>> ash = Trainer("Ash")
>>> misty = Trainer("Misty")
>>> pikachu = Pokemon("Pikachu", 1)
>>> staryu = Pokemon("Staryu", 1)

>>> ash.curr_level()
"Ash is currently at level 1"

>>> ash.list_pokemons()
()

>>> pikachu.curr_level()
"Pikachu is currently at level 1"

>>> pikachu.owned_by()
"Pikachu has no trainer"

>>> pikachu.power_up()
"Pikachu needs a trainer to power up"

>>> ash.catch(pikachu)
"Ash has caught Pikachu"

>>> ash.list_pokemons()
('Pikachu',)

>>> pikachu.owned_by()
"Pikachu's owner is Ash"

>>> pikachu.power_up()  # All Ash's pokemon are at his level
"Ash levels up to level 2"

>>> ash.curr_level()
"Ash is currently at level 2"

>>> misty.catch(staryu)
"Misty has caught Staryu"

>>> ash.catch(staryu)
"Staryu is already owned by Misty"

>>> p1 = Pokemon("Pidgey", 1)
>>> p2 = Pokemon("Pidgey", 1)
```

```
>>> ash.catch(p1)
"Ash has caught Pidgey"

>>> ash.list_pokemons()
('Pikachu', 'Pidgey')

>>> pikachu.power_up()
"Pikachu powers up to level 2"

>>> pikachu.power_up()  # not all of Ash's pokemon are at his level
"Pikachu cannot exceed trainer's level"

>>> p1.power_up()
"Pidgey powers up to level 2"

>>> ash.catch(p2)
"Ash has caught Pidgey"

>>> ash.list_pokemons()  # Ash has two pokemons with the same name
('Pikachu', 'Pidgey', 'Pidgey')

>>> ash.give(p1, ash)
"Cannot give to self"

>>> ash.give(staryu, misty)
"Ash does not own Staryu"

>>> ash.give(p1, misty)
"Misty's level is not high enough"

>>> ash.give(p2, misty)
"Pidgey transfers from Ash to Misty"

>>> ash.list_pokemons()
('Pikachu', 'Pidgey')

>>> misty.list_pokemons()
('Staryu', 'Pidgey')

>>> p2.owned_by()
"Pidgey's owner is Misty"

>>> ash.catch(charmander)
"Charmander got away!"

>>> p1.power_up()
"Ash levels up to level 3"
```

```
>>> charmander = Pokemon("Charmander", 3, "Charmelon", "Charizard")
>>> charmelon = charmander.evolve()
>>> charmelon == charmander
False

>>> ash.catch(charmelon)
"Ash has caught Charmelon"

>>> ash.list_pokemons()
('Pikachu', 'Pidgey', 'Charmelon')

>>> charizard = charmelon.evolve()
>>> ash.list_pokemons()
('Pikachu', 'Pidgey', 'Charizard')

>>> charizard.owned_by()
"Charizard's owner is Ash"

>>> charmelon.owned_by()
"Charmelon has no trainer"

>>> try:
...     charizard.evolve()
... except EvolveError as e:
...     p = e.pokemon
...     print(p.curr_level())
...     print(e)

"Charizard is currently at level 3"
"Charizard cannot evolve further"
```

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 —