Ministry of Education, Singapore
Computing Teachers' Content Upgrading Course 2020

Practical Assessment 2

17 June 2020

Time allowed: 3 hours

Instructions to candidates:

1. This is an **open-book** exam.
2. Answer all **three** questions.
3. You may complete your solutions in any IDE first before copying them into this Jupyter Notebook for submission.
4. Input validation is not required
5. Submit this Jupyter Notebook online before the test ends. You may submit multiple times, but only last submission before test end time will be accepted. https://driveuploader.com/upload/JDtXaQiUmX/ (https://driveuploader.com/upload/JDtXaQiUmX/)
6. Please note that the sample test cases may not be enough to test your program. Your programs will be tested with other inputs and they should exhibit the required behaviours to get full credit.

Name & Email

- Rename your jupyter notebook to "YourName" using menu `File > Rename`
- Enter your name and your email address in following cell

```
# YOUR NAME
# YOUR EMAIL
```

Question 1: Processing CSV File

The `"graduates.csv"` file in the `"data"` folder contains the number of university degree graduates by year, sex and type of course.

```
year,sex,type_of_course,no_of_graduates
1993,Males,Education,na
1993,Males,Applied Arts,na
1993,Males,Humanities & Social Sciences,481
1993,Males,Mass Communication,na
1993,Males,Accountancy,295
1993,Males,Business & Administration,282
```

(reference: https://data.gov.sg/dataset/graduates-from-university-first-degree-courses-by-type-of-course (https://data.gov.sg/dataset/graduates-from-university-first-degree-courses-by-type-of-course))

A) Write a function `load_graduate_data(file)` , where `file` is the path to the CSV file, to read the file and return a list of records.

- Each record (data point) in the list is a tuple, e.g. `(1993,'Males','Humanities & Social Sciences',481)` .
- The `year` and `no_of_graduates` are converted to integer values.
- Exclude header row in the returned list.
- Exclude records whose `no_of_graduates` column are non-numeric values, e.g. `na` .

Note: Use `str.isnumeric()` function to check if a string is of numeric values.

In [1]:

```
# WRITE YOUR CODE HERE

```

Test Case

In [2]:

```
file_name = 'data/graduates.csv'
table = load_graduate_data(file_name)

print("Record count: ", len(table))
print(table[:2])
print(table[-2:])
```

Expected output:

```
Record count:  600
[(1993, 'Males', 'Humanities & Social Sciences', 481), (1993, 'Males', 'Accountanc
y', 295)]
[(2014, 'Females', 'Engineering Sciences', 1251), (2014, 'Females', 'Services', 21
9)]
```

B) Write a function `graduates_by_course(table, start_year, end_year)` to find the total number of graduates from `table` between `start_year` and `end_year` (both inclusive) for all courses.

- The `table` data is the same data returned from `load_graduate_data()` function.
- It returns a dictionary whose key is the course type, and value is the total graduate number between the years.

In [3]:

```
# WRITE YOUR CODE HERE

```

Test Case

In [4]:

```
# The following 3 lines are needed if you have not completed 1(A).
import pickle
with open('raw/graduates.pickle', 'rb') as handle:
    table = pickle.load(handle)

result = graduates_by_course(table, 2000, 2004)
print(len(result))
print(result)
```

Expected output:

```
14
{'Education': 1359, 'Humanities & Social Sciences': 9264, 'Mass Communication': 60
9, 'Accountancy': 3620, 'Business & Administration': 5502, 'Law': 744, 'Natural, P
hysical & Mathematical Sciences': 5423, 'Medicine': 870, 'Dentistry': 165, 'Health
Sciences': 325, 'Information Technology': 2821, 'Architecture & Building': 1706,
 'Engineering Sciences': 17931, 'Applied Arts': 46}
```

C) Write a function `save_graduates_by_course(data, file)` to save data to a CSV file, where `data` is a dictionary returned from function `graduates_by_course()`, and `file` is the output file path.

- Each row in the CSV file is in `course, count` format. There is no header row.

In [5]:

```
# WRITE YOUR CODE HERE

```

Test Case

In [6]:

```
# The following 3 lines are needed if you have not completed 1(A).
import pickle
with open('raw/graduates.pickle', 'rb') as handle:
    table = pickle.load(handle)

result = graduates_by_course(table, 2000, 2004)
save_graduates_by_course(result, 'data/result.csv')

with open('data/result.csv') as f:
    data = [r.strip() for r in f.readlines()]
    data.sort()
    print(len(data))
    print(data)
```

Expected output:

```
14
['"Natural, Physical & Mathematical Sciences",5423', 'Accountancy,3620', 'Applied
 Arts,46', 'Architecture & Building,1706', 'Business & Administration,5502', 'Dent
istry,165', 'Education,1359', 'Engineering Sciences,17931', 'Health Sciences,325',
'Humanities & Social Sciences,9264', 'Information Technology,2821', 'Law,744', 'Ma
ss Communication,609', 'Medicine,870']
```

Question 2: Sorting and Searching

The file "hdb_flats_constructed.csv" provides annual number of flats constructed by Housing And Development Board (HDB) from 1977 to 2017.

A) Implement the class `FlatData` which meets following requirements:

- It has 2 instance attributes, `_year` and `_num`, representing a year and the number of flats constructed in that year.
- Its `__init__()` function takes in 2 parameters `_year` and `_num` to initialize the above two instance variable.
- Its `__repr__()` function returns a string in the format of `_year:_num`, e.g. `"1977:30498"`.

Note: It is OK if your class implements other additional functions which may be useful in solving part C and part D questions.

In [7]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [8]:

```
i1 = FlatData(1977, 30498)
print(i1)
i2 = FlatData(1978, 29742)
print(i1._num == i2._num)
print(i1._num < i2._num)
print(i1._num > i2._num)
```

Expected output:

```
1977:30498
False
False
True
```

B) Implement a function `load_flat_data(file)`, where `file` is the path to the CSV file "hdb_flats_constructed.csv", to read the file and returns list of `FlatData` objects.

- Upon reading a record, a `FlatData` object is created and added to a list.
- Convert `_year` and `_num` to integer values.

- Exclude header row in the returned list.

Note: You may assume all records in the file are valid. No data validation is required.

In [9]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [10]:

```
file_name = 'data/hdb_flats_constructed.csv'
table = load_flat_data(file_name)

print("Record count: ", len(table))
print(table[:2])
print(table[-2:])
```

Expected output:

```
Record count:  41
[1977:30498, 1978:29742]
[2016:26025, 2017:35210]
```

C) Implement a function `find_flat_by_year(data, year)` which find a `FlatData` object by year using **BinarySearch** algorithm.

- The list returned by `load_flat_data()` is already in ascending order by year because the data in the CSV file is sorted by year.

In [11]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [12]:

```
# The following 3 lines are needed if you have not completed 2(B).
import pickle
with open('raw/hdb_flats.pickle', 'rb') as handle:
    table = pickle.load(handle)

print(find_flat_by_year(table, 1977))
print(find_flat_by_year(table, 2000))
print(find_flat_by_year(table, 2017))
print(find_flat_by_year(table, 2020))

```

Expected output:

```
1977:30498
2000:27678
2017:35210
None
```

D) Implement a function `sort_flat_data(data)` which sorts a list of `Flat` objects in **descending** order by its `_num` using **QuickSort** algorithm.

In [13]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [14]:

```
# The following 3 lines are needed if you have not completed 2(B).
import pickle
with open('raw/hdb_flats.pickle', 'rb') as handle:
    table = pickle.load(handle)

result = sort_flat_data(table)
print('Years with most constructed flats:', result[:2])
print('Years with least constructed flats:', result[-2:])
```

Expected output:

```
Years with most constructed flats: [1984:67017, 1985:46370]
Years with least constructed flats: [2008:3154, 2006:2733]
```

Question 3: Card Game

A standard 52-card deck includes 13 ranks in each of the four suits: clubs, diamonds, hearts and spades. In the game for this question, only **ranks** are used, whereas **suits** are ignored. You will defines two classes `Game` and `Player` to simulate this game.

How the Game Works:

1. At the beginning of the game, each player receives the same number of random cards from the deck in a particular order. Players are not allowed to change the order of cards in their hand from left to right.
2. When it is a player's turn, he plays his **right-most** card onto the table. Newly played cards are stacked on top of existing cards on the table.
    - Cards on the table are kept in a single stack with older cards at the bottom and newer ones on top.
3. In either of following scenarios, the player takes some or all of the cards from the table.
    - If the player plays a card with rank "J", the player takes all the cards on the table.
    - If the player plays a card with rank equal to another card on the table, the player takes all the cards between the two cards, including the two cards of same rank.

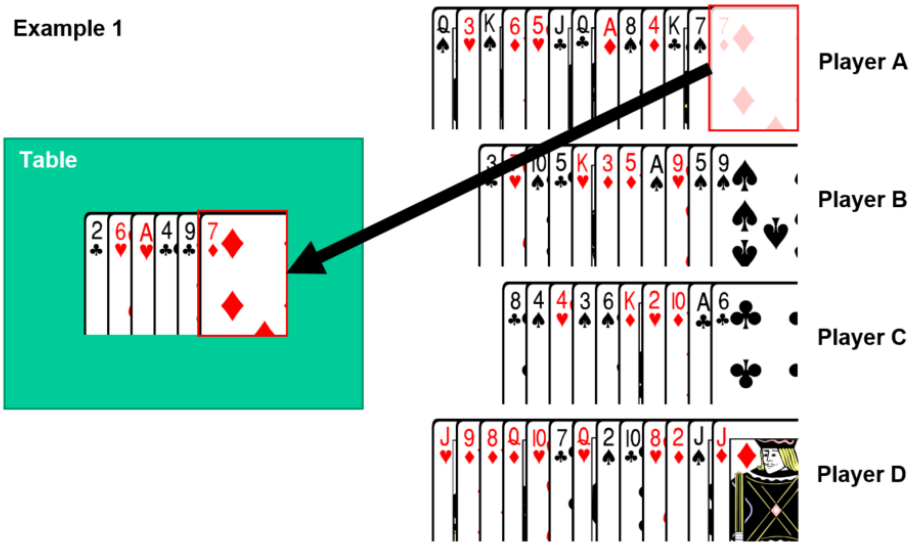
4. When player takes cards from the table, those cards are added to the **left** of player's existing cards. The order of the added cards must not be changed (i.e., the bottom-most card that was taken becomes the player's new left-most card).

5. The game ends when one player has no more cards. The player with the most cards wins.

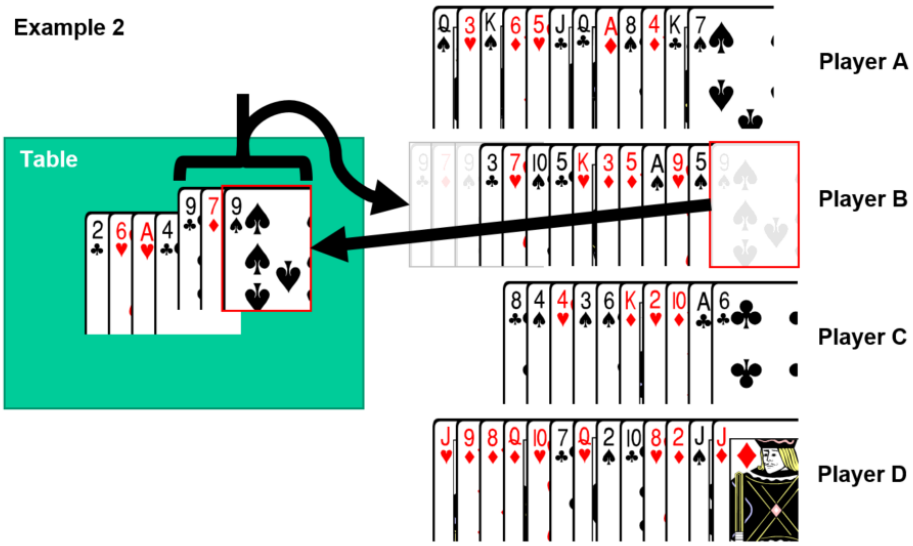
Game Example:

Suppose the existing cards on the table are `[2, 6, 'A', 4, 9]`, where `2` is the oldest card and `9` is the most recently played card.

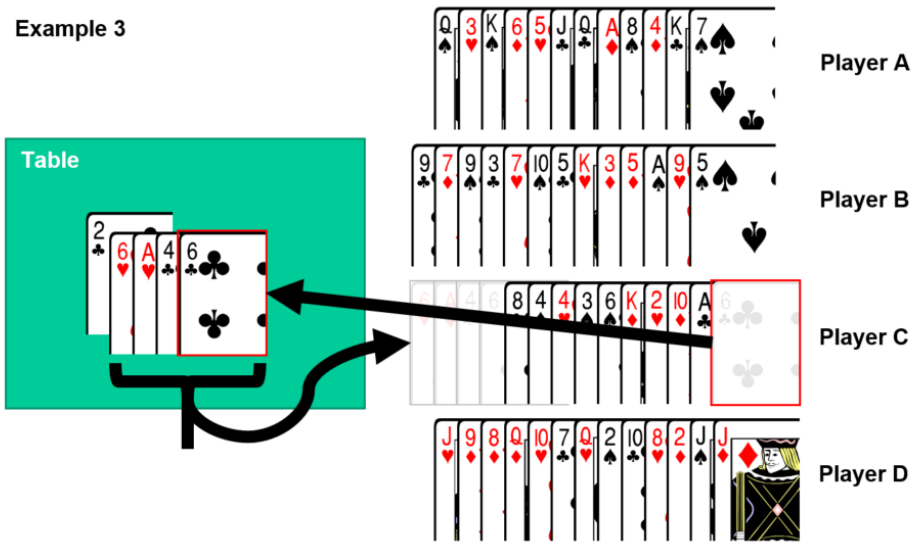
- Example 1: Player A plays card `7`. He doesn't get to take any cards off the table.



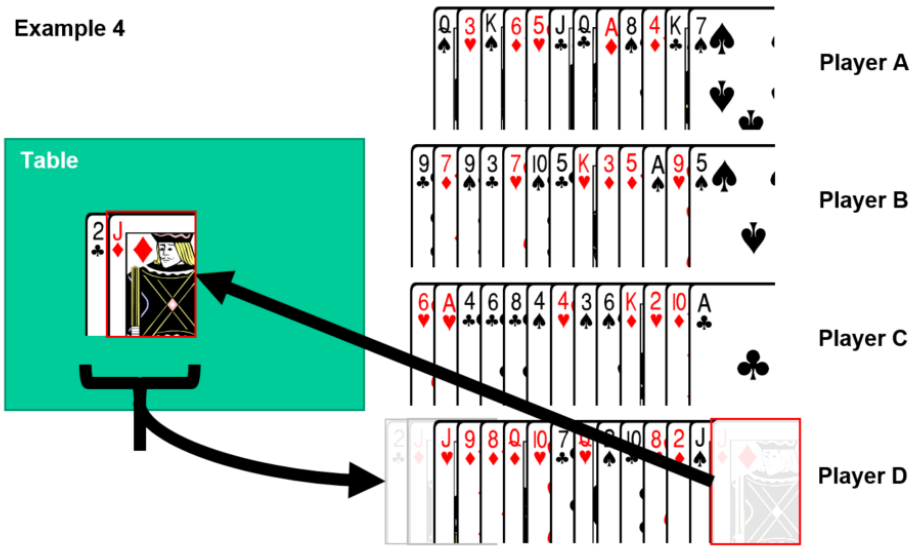
- Example 2: Player B then plays card `9`. He gets to take three cards `[9, 7, 9]` and adds them to the left of his existing cards. Remaining cards on table are `[2, 6, 'A', 4]`.



- Example 3: Player C then plays card `6`. He gets to take four cards `[6, 'A', 4, 6]`. Remaining card on table is `[2]`.



- Example 4: Player D then plays card `J`, which is a special card. He takes all cards on the table, i.e. `[2, 'J']`. There are no more cards on the table.



Player Example:

When a player plays a card, he removes his *right-most* card from the card list.

- For example, if player has cards `[3, 4, 2]`, card `2` will be the next card to be played.

When a player takes cards from the table, he adds them to the *left* side of his card list.

- For example, if the cards he takes from the table are `[6, 'A', 4, 9, 6]` and his existing cards are `[3, 4, 2]`, the combined cards will be `[6, 'A', 4, 9, 6, 3, 4, 2]`

A) Define the `Game` class as follows:

- Implement its `__init__(table_cards=None)` function to initialize an attribute `_table_cards` using the parameter `table_cards`. The `_table_cards` attribute is used to keep track of which cards are on the table. If the parameter `table_Cards` is `None`, then there are initially no cards on the table.

- Implement its `__str__()` function to return a string representation of the Game, for example, `Game([2, 5, 3])` where `[2, 5, 3]` is the value of `_table_cards` .
- Implement a method `play(card)` which simulates a player playing the card `card` .
  - It returns a list of cards removed from the table. If there is no card to be returned, it returns an empty list.
  - It prints a message `"Play card <card>"` , e.g. `"Play card 9"`
  - It also modifies `_table_cards` to correctly reflect the remaining cards on the table.

In [15]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [16]:

```
game = Game()
game.play('A')
game.play(2)
game.play(3)
game.play(8)
print(game)
cards = game.play(8)
print("Remove cards:", cards)
print(game)
```

Expected output:

```
Play card A
Play card 2
Play card 3
Play card 8
Game(['A', 2, 3, 8])
Play card 8
Remove cards: [8, 8]
Game(['A', 2, 3])
```

Test Case 2

In [17]:

```
game = Game()
game.play('A')
game.play(2)
game.play(3)
print(game)
cards = game.play(2)
print("Remove cards:", cards)
print(game)
```

Expected output:

```
Play card A
Play card 2
Play card 3
Game(['A', 2, 3])
Play card 2
Remove cards: [2, 3, 2]
Game(['A'])
```

Test Case 3

In [18]:

```
game = Game()
game.play('A')
game.play(2)
game.play(3)
cards = game.play('J')
print("Remove cards:", cards)
print(game)
```

Expected output:

```
Play card A
Play card 2
Play card 3
Play card J
Remove cards: ['A', 2, 3, 'J']
Game([])
```

Test Case 4

In [19]:

```
game = Game([2, 6, 'A', 4, 9])
print(game)

for card in [7, 9, 6, 'J']:
    cards = game.play(card)
    print("Remove cards:", cards)
    print(game)
```

Expected output:

```
Game([2, 6, 'A', 4, 9])
Play card 7
Remove cards: []
Game([2, 6, 'A', 4, 9, 7])
Play card 9
Remove cards: [9, 7, 9]
Game([2, 6, 'A', 4])
Play card 6
Remove cards: [6, 'A', 4, 6]
Game([2])
Play card J
Remove cards: [2, 'J']
Game([])
```

B) Define a class `Player` as follows:

- Implement its `__init__(name, cards=None)` function to initialize attributes `_name` and `_cards` using the parameters `name` and `cards`. If `cards` is `None`, the player initially has no cards.
- Implement its `__str__()` function to returns a string representation of the Player in the format of `"Player(<name>, <cards>)"`, e.g. `"Player(Mark, [1,2,3])"`.
- Implement its `dequeue()` function which removes last item in the `_cards` list. It returns `None` if player has no more cards.
- Implement its `enqueue_all(cards)` function which inserts `cards` to the front of the `_cards` list. For example, if `cards = [4,5]` and `_cards = [1,2,3]`, then the updated `_cards` will be `[4,5,1,2,3]`.

In [20]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [21]:

```
player = Player('X', [1,2,3])
print(player)
print(player.dequeue())
```

Expected output:

```
Player(X, [1, 2, 3])
3
```

Test Case 2

In [22]:

```
player = Player('X', [1,2,3])
player.enqueue_all([4,5])
print(player)
```

Expected output:

```
Player(X, [4, 5, 1, 2, 3])
```

Test Case 3

In [23]:

```
player = Player('X')
print(player.dequeue())
player.enqueue_all([6,7,8,9])
print(player)
print(player.dequeue())
```

Expected output:

```
None
Player(X, [6, 7, 8, 9])
9
```

C) Define a class `Game2` which inherits from `Game` class and adds the following functions.

- Implement a property `card_count` which returns the current number of cards on the table.
- Add a class attribute `RANKS` whose value is `['A', 2, 3, 4, 5, 6, 7, 8, 9, 10, 'J', 'Q', 'K']`.
- Implement a class method `distribute_cards(player_count)` which simulates shuffling a full deck of cards and dividing it evenly between the given number of players.
  - The function returns a list of sublists, where each sublist contains the cards for one player. Number of sublists equals to the total number of players.
  - The `player_count` is the total number of players.
  - For example, when `player_count = 2`, it returns a list of 2 sublists, where each sublist contains 26 cards; when `player_count = 3`, it returns 17 cards for each player with one card not used in the output.
  - Use class attribute `RANKS` to generate full deck (with ranks and without suits).
  - Use following code to shuffle the deck before distribution.

```
random.seed(0)
random.shuffle(cards)
```

In [24]:

```
# WRITE YOUR CODE HERE

```

Test Case 1

In [25]:

```
game = Game2()
game.play('A')
game.play(2)
print('Total cards on table:', game.card_count)
```

Expected output:

```
Play card A
Play card 2
Total cards on table: 2
```

Test Case 2

In [26]:

```
game = Game2()
game.distribute_cards(10)
```

Expected output:

```
[[3, 'K', 7, 3, 'K'],
 [8, 6, 'J', 2, 'J'],
 [8, 'A', 5, 10, 8],
 [2, 'K', 4, 9, 6],
 ['A', 'Q', 4, 5, 3],
 ['J', 9, 2, 'Q', 'J'],
 [4, 5, 10, 9, 7],
 [2, 6, 9, 8, 'A'],
 ['Q', 10, 5, 7, 'K'],
 [6, 7, 4, 3, 'A']]
```