Assignment 1

COMP9021, Trimester 1, 2020

1. General matters

1.1. **Aim.** The purpose of the assignment is to:

- develop your problem solving skills;
- design and implement the solutions to problems in the form of small sized Python programs;
- practice the use of arithmetic computations, tests, repetitions, lists, dictionaries, strings, Unicode characters.

1.2. **Submission.** Your programs will be stored in files named `patience.py` and `poker_dice.py`. After you have developed and tested your program, upload it using Ed (unless you worked directly in Ed). Assignments can be submitted more than once; the last version is marked. Your assignment is due by March 29, 10:00pm.

1.3. **Assessment.** The assignment is worth 10 marks. It is going to be tested against a number of inputs. For each test, the automarking script will let your program run for 30 seconds.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

The outputs of your programs should be **exactly** as indicated.

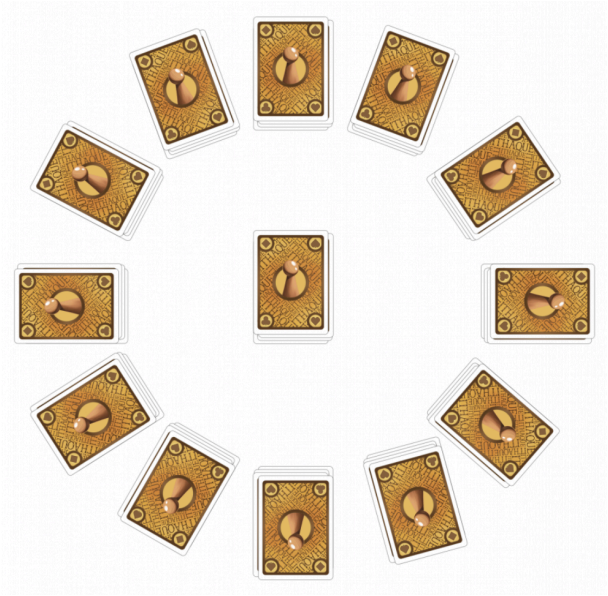
1.4. **Reminder on plagiarism policy.** You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of algorithms, not code. But you must implement the solution on your own. Submissions are routinely scanned for similarities that occur when students copy and modify other people's work, or work very closely together on a single implementation. Severe penalties apply.

2. Patience

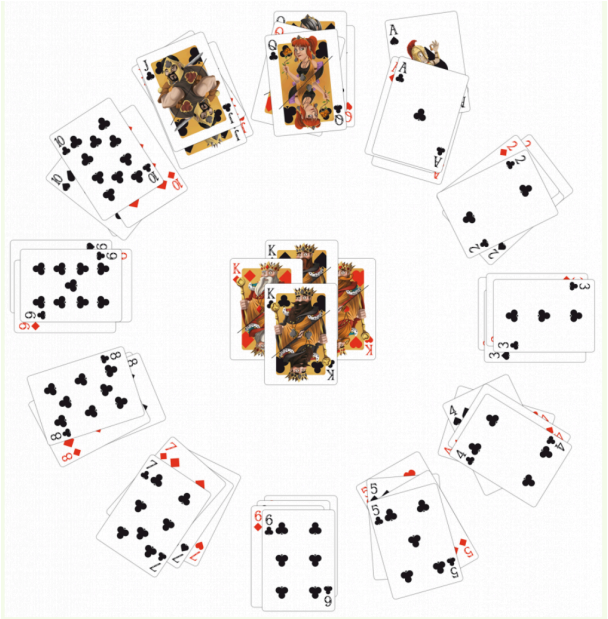
A deck of 52 cards, in random order, is distributed as follows:

- 4 cards for each of the 12 "hours" marked along a circle, making up the *dial*;
- 4 cards at the centre of the circle, making up the *centre*.

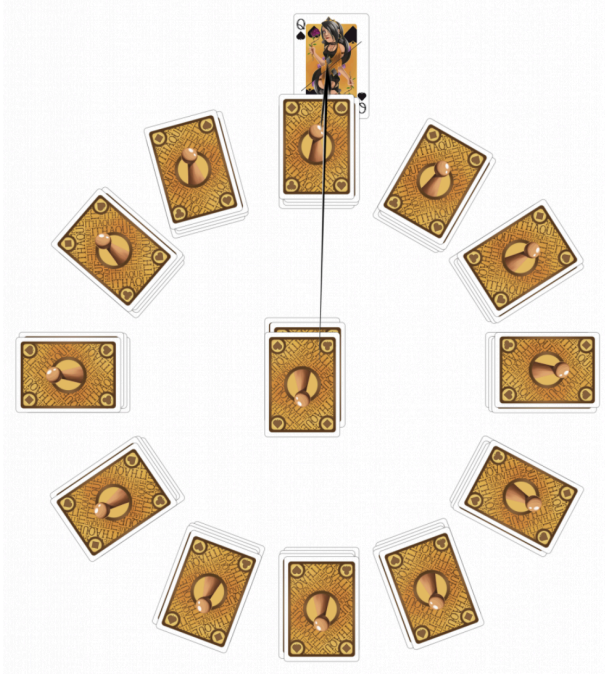
All cards are facing down:



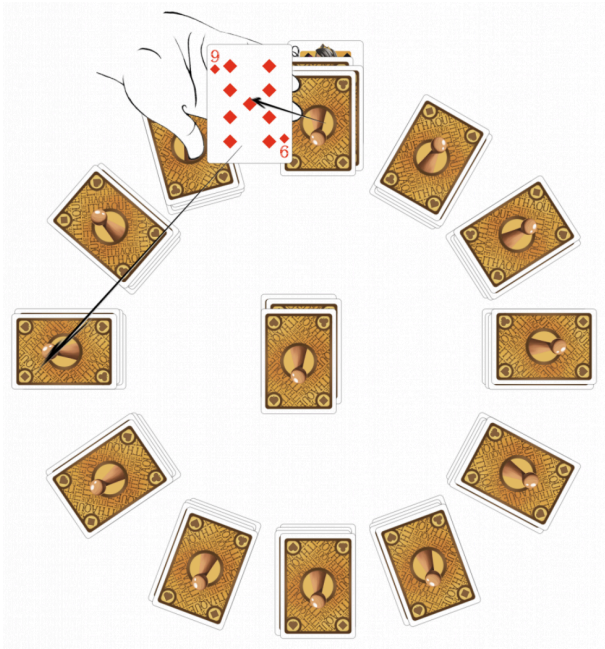
Positions 1 o'clock to 10 o'clock are meant to be where the cards of rank 1 to 10 eventually end up, respectively; 11 o'clock where the jacks eventually end up; 12 o'clock where the queens eventually end up; the centre where the kings eventually end up:



To start with, the card C at the top of the centre is taken off the centre, and it is placed, facing up, under the stack of cards, S, at the location that corresponds to the rank of C (so S now consists of 5 cards: one card facing up, and above it, 4 cards facing down). Below is the illustration for the case where C is the queen of spades:



Then the card C' at the top of S is taken off S (so S now consists of 4 cards: one card facing up, and above it, 3 cards facing down) and similarly to what was done with C, C' is placed, facing up, under the stack of cards at the location that corresponds to the rank of C'. Below is the illustration for the case where C' is the 9 of diamonds:



The game is won if the player is not stuck before all cards have reached (facing up) their intended destinations.

In the program `patience.py`, a function, `generate_dial_and_centre(for_seed)`, has been implemented that returns a tuple with 2 elements: a dictionary, say `dial`, whose keys are the hours and whose values are the lists of 4 cards located at the corresponding location, from the one on the table to the one at the top of the stack; a list, say `centre`, of the 4 cards are the centre. It is easy to see from the stub and sample outputs how the cards are represented.

You have to implement:

- `initial_hour(hour, dial)`, a function that shows the (hidden) cards at location `hour` (meant to be a number between 1 and 12) before the game starts;

- `hour_after_playing_from_beginning_for_at_most(hour, nb_of_steps, dial, centre)`, a function that displays the cards at location `hour` (meant to be a number between 1 and 12) after `nb_of_steps` many cards have been taken off the top of a stack and put under the appropriate stacks, if it is possible to play for at least that many steps; otherwise the function prints out `Could not play that far...`;
- `kings_at_end_of_game(dial, centre)`, a function that displays the kings at the centre at the end of the game, unless the game could not go all the way to the end, in which case the function prints out `No success...`.

Below is a possible interaction.

```
$ python3
...
>>> from patience import *
>>> dial, centre = generate_dial_and_centre(0)
>>> for hour in dial:
...     print(f'hour:2', dial[hour], sep=': ')
...
 1: ['H3', 'H10', 'D9', 'D1']
 2: ['Ck', 'D8', 'H2', 'Hq']
 3: ['S7', 'D2', 'Sq', 'D10']
 4: ['S3', 'Sk', 'Dj', 'H5']
 5: ['Hk', 'H4', 'C4', 'D7']
 6: ['C8', 'H9', 'S5', 'Dk']
 7: ['C6', 'S6', 'C10', 'H6']
 8: ['Hj', 'S1', 'S9', 'H7']
 9: ['C2', 'Cq', 'C7', 'D4']
10: ['Sj', 'S4', 'S2', 'C3']
11: ['H8', 'D5', 'D6', 'H1']
12: ['C1', 'D3', 'C9', 'S10']
>>> centre
['C5', 'Cj', 'S8', 'Dq']
>>> initial_hour(12, dial)
hidden🂡  hidden🃃  hidden🃉  hidden🂪
>>> initial_hour(10, dial)
hidden🂫  hidden🂤  hidden🂢  hidden🃓
>>> initial_hour(3, dial)
hidden🂧  hidden🃂  hidden🂠  hidden🃊
>>> initial_hour(2, dial)
hidden🃛  hidden🃈  hidden🂲  hidden🂰
>>> initial_hour(9, dial)
hidden🃒  hidden🃀  hidden🃗  hidden🃄
>>> initial_hour(4, dial)
hidden🂣  hidden🂮  hidden🃋  hidden🂵
>>> initial_hour(5, dial)
hidden🂾  hidden🂴  hidden🃔  hidden🃇
>>> initial_hour(7, dial)
hidden🃖  hidden🂦  hidden🃚  hidden🂶
>>> initial_hour(6, dial)
hidden🃘  hidden🂹  hidden🂥  hidden🃎
>>> initial_hour(8, dial)
hidden🂻  hidden🂡  hidden🂩 hidden🂷
>>> hour_after_playing_from_beginning_for_at_most(12, 1, dial, centre)
🃀  hidden🂡  hidden🃃  hidden🃉  hidden🂪
>>> hour_after_playing_from_beginning_for_at_most(12, 2, dial, centre)
🃀  hidden🂡  hidden🃃  hidden🃉
```

```
>>> hour_after_playing_from_beginning_for_at_most(10, 2, dial, centre)
🂪 hidden🂫 hidden🂤 hidden🂢 hidden🃃
>>> hour_after_playing_from_beginning_for_at_most(10, 3, dial, centre)
🂪 hidden🂫 hidden🂤 hidden🂢
>>> hour_after_playing_from_beginning_for_at_most(3, 3, dial, centre)
🃃 hidden🂧 hidden🂢 hidden🂠 hidden🃊
>>> hour_after_playing_from_beginning_for_at_most(3, 4, dial, centre)
🃃 hidden🂧 hidden🂢 hidden🂠
>>> hour_after_playing_from_beginning_for_at_most(10, 4, dial, centre)
🂪 🃊 hidden🂫 hidden🂤 hidden🂢
>>> hour_after_playing_from_beginning_for_at_most(10, 5, dial, centre)
🂪 🃊 hidden🂫 hidden🂤
>>> hour_after_playing_from_beginning_for_at_most(2, 5, dial, centre)
🂢 hidden🃅 hidden🃈 hidden🂲 hidden🂰
>>> hour_after_playing_from_beginning_for_at_most(2, 6, dial, centre)
🂢 hidden🃅 hidden🃈 hidden🂲
>>> hour_after_playing_from_beginning_for_at_most(12, 6, dial, centre)
🂰 🃌 hidden🃁 hidden🃓 hidden🃉
>>> hour_after_playing_from_beginning_for_at_most(9, 7, dial, centre)
🃉 hidden🃂 hidden🃀 hidden🃇 hidden🃄
>>> hour_after_playing_from_beginning_for_at_most(4, 8, dial, centre)
🃄 hidden🃃 hidden🃅 hidden🃋 hidden🂵
>>> hour_after_playing_from_beginning_for_at_most(5, 9, dial, centre)
🂵 hidden🃅 hidden🂴 hidden🃔 hidden🃇
>>> hour_after_playing_from_beginning_for_at_most(7, 10, dial, centre)
🃇 hidden🃆 hidden🃆 hidden🃊 hidden🂶
>>> hour_after_playing_from_beginning_for_at_most(6, 11, dial, centre)
🂶 hidden🃈 hidden🂹 hidden🃕 hidden🃍
>>> hour_after_playing_from_beginning_for_at_most(8, 13, dial, centre)
🂨 hidden🂻 hidden🂡 hidden🂩 hidden🂷
>>> hour_after_playing_from_beginning_for_at_most(10, 45, dial, centre)
🂺 🃊 🃊 🂪
>>> hour_after_playing_from_beginning_for_at_most(10, 46, dial, centre)
Could not play that far...
>>> kings_at_end_of_game(dial, centre)
No success...
>>> dial, centre = generate_dial_and_centre(18)
>>> hour_after_playing_from_beginning_for_at_most(1, 52, dial, centre)
🃁 🃁 🃁 🃁
>>> hour_after_playing_from_beginning_for_at_most(10, 53, dial, centre)
🃊 🂺 🃊 🂪
>>> kings_at_end_of_game(dial, centre)
🂾 🃎 🃞 🂮
```

3. Poker dice

Write a program named `poker_dice.py` that simulates the roll of 5 dice, at most three times, as described at

http://en.wikipedia.org/wiki/Poker_dice

as well as a given number of rolls of the 5 dice to evaluate the probabilities of the various hands.

The next three pages show a possible interaction.

The following observations are in order.

- Your program has to define the functions `play()` and `simulate()`, but of course it will likely include other functions.
- What is input from the keyboard is displayed in red; what is black is output by the program (in the pdf, for clarity, not at the prompt…).
- The hands are displayed in the order Ace, King, Queen, Jack, 10 and 9.
- All dice can be kept by inputting either `all`, or `All`, or the current hand in any order.
- No die is kept by inputting nothing.
- We have control over what is randomly generated by using the `seed()` function at the prompt, but do not make use of `seed()` in the program.
- To roll a die, we use `randint(0, 5)` **with 0, 1, 2, 3, 4 and 5 corresponding to Ace, King, Queen, Jack, 10 and 9, respectively**.

:

```
$ python3
...
>>> from random import seed
>>> import poker_dice
>>> seed(0)
>>> poker_dice.play()
The roll is: Ace Queen Jack Jack 10
It is a One pair
Which dice do you want to keep for the second roll? all
Ok, done.
>>> poker_dice.play()
The roll is: Queen Queen Jack Jack Jack
It is a Full house
Which dice do you want to keep for the second roll?  All
Ok, done.
>>> poker_dice.play()
The roll is: King King Queen 10 10
It is a Two pair
Which dice do you want to keep for the second roll?  King 10 Queen King 10
Ok, done.
>>> poker_dice.play()
The roll is: Ace King Queen 10 10
It is a One pair
Which dice do you want to keep for the second roll? 10 11
That is not possible, try again!
Which dice do you want to keep for the second roll? ace
That is not possible, try again!
Which dice do you want to keep for the second roll? 10 10
The roll is: King 10 10 10 9
It is a Three of a kind
Which dice do you want to keep for the third roll? all
Ok, done.
>>> poker_dice.play()
```

```
The roll is: Ace Ace Queen 9 9
It is a Two pair
Which dice do you want to keep for the second roll? Ace
The roll is: Ace Ace Queen Jack 10
It is a One pair
Which dice do you want to keep for the third roll? Ace
The roll is: Ace Queen Queen Jack 10
It is a One pair
>>> seed(2)
>>> poker_dice.play()
The roll is: Ace Ace Ace King Queen
It is a Three of a kind
Which dice do you want to keep for the second roll? Ace Ace Ace
The roll is: Ace Ace Ace 9 9
It is a Full house
Which dice do you want to keep for the third roll? all
Ok, done.
>>> poker_dice.play()
The roll is: King Queen Queen 10 10
It is a Two pair
Which dice do you want to keep for the second roll?
The roll is: Ace King Jack 10 9
It is a Bust
Which dice do you want to keep for the third roll?
The roll is: Queen Jack 10 9 9
It is a One pair
>>> seed(10)
>>> poker_dice.play()
The roll is: Ace Jack Jack 10 10
It is a Two pair
Which dice do you want to keep for the second roll? Jack 10 Jack 10
The roll is: Ace Jack Jack 10 10
It is a Two pair
Which dice do you want to keep for the third roll? Jack 10 Jack 10
The roll is: King Jack Jack 10 10
It is a Two pair
>>> seed(20)
>>> poker_dice.play()
The roll is: King Queen 9 9 9
It is a Three of a kind
Which dice do you want to keep for the second roll? 9 King 9 9
The roll is: King 9 9 9 9
It is a Four of a kind
Which dice do you want to keep for the third roll? 9 9 9 9
The roll is: Ace 9 9 9 9
It is a Four of a kind
```

```
>>> seed(0)
>>> poker_dice.simulate(10)
Five of a kind : 0.00%
Four of a kind : 0.00%
Full house     : 10.00%
Straight       : 0.00%
Three of a kind: 0.00%
Two pair       : 20.00%
One pair       : 60.00%
>>> poker_dice.simulate(100)
Five of a kind : 0.00%
Four of a kind : 0.00%
Full house     : 3.00%
Straight       : 4.00%
Three of a kind: 14.00%
Two pair       : 28.00%
One pair       : 45.00%
>>> poker_dice.simulate(1000)
Five of a kind : 0.10%
Four of a kind : 2.50%
Full house     : 3.80%
Straight       : 3.40%
Three of a kind: 17.20%
Two pair       : 20.60%
One pair       : 46.30%
>>> poker_dice.simulate(10000)
Five of a kind : 0.08%
Four of a kind : 1.99%
Full house     : 3.93%
Straight       : 3.33%
Three of a kind: 14.88%
Two pair       : 23.02%
One pair       : 46.58%
>>> poker_dice.simulate(100000)
Five of a kind : 0.08%
Four of a kind : 1.94%
Full house     : 3.85%
Straight       : 3.17%
Three of a kind: 15.47%
Two pair       : 23.02%
One pair       : 46.31%
```