Assignment 1

COMP9021, Trimester 1, 2025

1 General matters

1.1 Aim

The purpose of the assignment is to:

- develop your problem solving skills;
- let you carefully read specifications and follow them;
- let you design and implement the solutions to problems in the form of small sized Python programs;
- let you practice the use of arithmetic computations, tests, repetitions, fundamental Python data types, Unicode characters;
- have control over print statements.

1.2 Submission

Your programs will be stored in files named `solitaire_1.py` and `solitaire_2.py`. After you have developed and tested your programs, upload them 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 31, 10:00am.

1.3 Assessment

The assignment is worth 13 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.

Assignments can be submitted up to 5 days after the deadline. The maximum mark obtainable reduces by 5% per full late day, for up to 5 days. Thus if students A and B hand in assignments worth 12 and 11, both two days late (that is, more than 24 hours late and no more than 48 hours late), then the maximum mark obtainable is 11.7, so A gets $\min(11.7, 12) = 11.7$ and B gets $\min(11.7, 11) = 11$.

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 Decks, shuffling

2.1 Decks

The first exercise simulates a solitaire game that is played with 32 cards, namely, the Ace, Seven, Eight, Nine, Ten, Jack, Queen and King of each of the 4 suits, with the following convention.

- Numbers from 0 to 7 denote the Hearts, from the Ace of Hearts up to the King of Hearts.
- Numbers from 8 to 15 denote the Diamonds, from the Ace of Diamonds up to the King of Diamonds.
- Numbers from 16 to 23 denote the Clubs, from the Ace of Clubs up to the King of Clubs.
- Numbers from 24 to 31 denote the Spades, from the Ace of Spades up to the King of Spades.

So for instance, 6 denotes the Queen of Hearts, and 26 denotes the Eight of Spades.

The second exercise simulates a solitaire game that is played with 52 cards, with the following convention.

- Numbers from 0 to 12 denote the Hearts, from the Ace of Hearts up to the King of Hearts.
- Numbers from 13 to 25 denote the Diamonds, from the Ace of Diamonds up to the King of Diamonds.
- Numbers from 26 to 38 denote the Clubs, from the Ace of Clubs up to the King of Clubs.
- Numbers from 39 to 51 denote the Spades, from the Ace of Spades up to the King of Spades.

So for instance, 16 denotes the Four of Diamonds, and 36 denotes the Jack of Clubs.

2.2 Shuffling

Both exercises require to shuffle a deck of cards, either the full deck (of 32 or 52 cards) or a subset of the full deck. For that purpose, the following convention is followed.

By shuffling a deck of cards, we mean randomising the corresponding set of numbers by providing the list of those numbers, **in increasing order**, as an argument to the `shuffle()` function of the `random` module. For instance,

- to shuffle the whole deck of 52 cards, we could do

```
>>> cards = list(range(52))
>>> shuffle(cards)
```

- and to shuffle the deck of all 52 cards except for the Four of Diamonds and the Jack of Clubs, we could do

```
>>> cards = sorted(set(range(52)) - {16, 36})
>>> shuffle(cards)
```

To make sure that results are predictable, just before calling the `shuffle()` function, the `seed()` function of the `random` module should be called with a given argument. By *shuffling the deck of all (52) cards with 678 given to `seed()`*, we mean doing something equivalent to:

```
>>> cards = list(range(52))
>>> seed(678)
>>> shuffle(cards)
```

which by the way, lets `cards` denote

```
[11, 12, 22, 38, 15, 16, 14, 28, 4, 34, 46, 48, 33,
   18, 5, 17, 27, 37, 50, 51, 31, 41, 9, 1, 39, 3,
       29, 40, 43, 23, 25, 13, 19, 35, 26, 42, 24, 32, 44,
           45, 6, 36, 8, 47, 2, 30, 10, 49, 21, 0, 20, 7]
```

3 First solitaire game

3.1 Game description

It is played with 32 cards. The aim is to get rid of enough cards and be left with the 4 Aces in sequence, possibly together with other cards before or after the 4 Aces. Elimination of cards proceeds over 3 stages, with all cards still in play being distributed over 4 stacks for the first stage, 3 stacks for the second stage, and 2 stacks for the third and last stage. The cards are shuffled only once, just before the game begins.

At the start of the first stage, the cards in the deck are facing down, so with the first card in the deck at the bottom and with the last card in the deck at the top, and distributed from the top to the bottom of the deck over 4 stacks, so the first, second, third and fourth cards at the top of the deck become the cards at the bottom of the first (leftmost), second, third and fourth (rightmost) stacks, respectively, the fifth, sixth, seventh and eight cards at the top of the deck become the cards directly above the cards at the bottom of the first, second, third and fourth stacks, respectively, etc. The first stack is turned upside down so its cards are now facing up. All cards at the top of the stack are discarded until an Ace is uncovered, unless there is no Ace in the first stack in which case the whole stack is discarded. If an Ace has been uncovered then what is left of the stack is turned upside down and then put aside, so with the Ace now facing down on the table. The same is done with the second, third and fourth stacks, each time turning what is left of the stack, if anything, upside down and putting it aside above the cards that have been previously kept, if any.

For the second stage, the same procedure is followed, except that the cards that have been put aside are distributed over 3 stacks instead of 4. There will be one more card in the first stack than in the third stack if the number of cards that is left is not a multiple of 3, and there will be one more card in the second stack than in the third stack if the number of cards that is left is equal to 2 modulo 3. Note that the last card that is distributed (facing down) is the first Ace that has been uncovered during the first stage.

The same is done for the third stage, except that the cards that have been put aside are distributed over 2 stacks.

At the end of the third stage, the cards that have been last put aside are taken from top to bottom and displayed from left to right facing up. The 4 Aces are necessarily all there but the game is won only if they occur in sequence, without any other card between two of them. Of course if there are only 4 cards left then the game is known to be won before they are revealed.

3.2 Playing a single game (3.5 marks)

Your program will be stored in a file named `solitaire_1.py`. Executing

```
$ python3 solitaire_1.py
```

at the Unix prompt should produce the following output (ending in a single space):

```
Please enter an integer to feed the seed() function:
```

with the program now waiting for your input, which should be an integer, and which you can assume will be an integer. Your program will feed that integer to `seed()` before calling `shuffle()`, as described in Section 2, to shuffle the deck of 32 cards.

Here is a possible interaction for a game that is lost.

Here is a possible interaction for a game that is won.

The output starts with an empty line followed by a line that reads:

```
Deck shuffled, ready to start!
```

The next line of output represents the 32 card deck, with all cards facing down (32 `]`s). It is followed with an empty line.

The beginning of the first round is announced by a line that reads:

```
Distributing the cards in the deck into 4 stacks.
```

The next two rounds are announced by a line that reads:

```
Distributing the cards that have been kept into _ stacks.
```

with `_` being `3` for the second round and `2` for the third round. That line is a followed by 5 lines:

- a representation of the stacks that remain, the starts of two adjacent stacks being 12 characters away;
- an empty line (whose purpose will be explained further down);
- a representation of all cards that have been discarded, facing up, using `[` for all of them except for the last (top) one, that is properly displayed—that line being empty in the first stage;
- an empty line (whose purpose will be explained further down);
- an empty line.

Then, for each stage, the output consists of groups of 12 or 13 lines, each group being structured as follows.

- The first line in the group reads as one of the following:
  - `No ace in _ stack, after it has been turned over.`
    with `_` one of `first`, `second`, `third` and `fourth`, or
  - `_ [(and last)] card in _ stack, after it has been turned over, is an ace.`
    with
    - the first `_` one of `First`, `Second`, `Third`, `Fourth`, `Fifth`, `Sixth`, `Seventh` and `Eighth`,
    - the second `_` one of `First`, `Second`, `Third` and `Fourth`,
    - `(and last)` added when the card is indeed the last one in the stack.
- The second line in the group depicts the stacks that remain with for the one being processed, what is left of it after it has been turned upside down and all cards that do not have an Ace above them have been discarded one after the other; so either there is nothing left of the stack or what is left has an Ace at the top.
- The third line in the group depicts all cards that have been discarded one after the other from the stack being processed, with at the top the last card in the stack if no Ace has been found, and the card just above the Ace that has been found otherwise.

- The fourth line in the group depicts the cards that have been discarded up to then, facing up.
- The fifth line in the group depicts the cards that have been put aside up to then, facing down.
- The sixth line in the group is empty.
- The seventh line in the group reads as one of the following:
  - `Discarding|Adding to the cards that have been discarded all cards in the stack.`
    or
  - `Discarding|Adding to the cards that have been discarded the card before the ace.`
    or
  - `Discarding|Adding to the cards that have been discarded the _ cards before the ace.`
    with `_` an integer at least equal to 2.

  If nothing had been discarded yet, Discarding is used; otherwise, Adding to the cards that have been discarded is used.
- In case an Ace has been found, the next line reads as one of the following:
  - `Keeping|Also keeping the ace, turning it over.`
    or
  - `Keeping|Also keeping the ace and the card after, turning them over.`
    or
  - `Keeping|Also keeping the ace and the _ cards after, turning them over.`
    with `_` an integer at least equal to 2.

  If nothing had been put aside yet, Keeping is used; otherwise, Also keeping is used.
- The line that follows depicts the stacks that remain to be processed, if any.
- The line that follows is empty.
- The line that follows depicts the cards that have now been discarded (possibly unchanged).
- The line that follows depicts the cards that have now been put aside (possibly unchanged).
- The last line in the group is empty.

The output ends with a group of 6 lines:

- The first line in the group reads:

  `Displaying the _ cards that have been kept.`

  with `_` an integer (necessarily at least equal to 4).
- The second line in the group reads: You lost! or You won!
- The next two lines in the group are empty.
- The penultimate line in the group depicts the cards that have been discarded over the game.
- The last line in the group depicts all cards that have been put aside at the end of the game, displayed facing up next to each other.

Note that **there is no tab anywhere** in the output and **no line has any trailing space**.

3.3 Playing many games and estimating probabilities (3 marks)

Executing

```
$ python3
```

at the Unix prompt and then

```
>>> from solitaire_1 import simulate
```

at the Python prompt should allow you to call the `simulate()` function, that takes two arguments.

- The first argument, say n, is meant to be a strictly positive integer, and you can assume that it is a strictly positive integer, that represents the number of games to play.
- The second argument, say i, is meant to be an integer, and you can assume that it is an integer.

The function simulates the playing of the game n times,

- the first time shuffling the deck of all cards with i given to `seed()`,
- if $n \geq 2$, the second time shuffling the deck of all cards with $i + 1$ given to `seed()`,
- …
- the n^{th} and last time, shuffling the deck of all cards with $i + n - 1$ given to `seed()`.

Here is a possible interaction.

Probabilities are computed as floating point numbers and formatted with 2 digits after the decimal point. Only strictly positive probabilities and the corresponding number of cards left when winning are output (including the cases, if any, when they are smaller than 0.005%, and so output as `0.00%`). The output is sorted in increasing number of cards left when winning.

There is a single space to the left and to the right of the separating vertical bar, with all lines consisting of precisely 45 characters.

4 Second solitaire game

4.1 Game description

It is played with 52 cards. The Sevens are removed from the deck and placed on the table, facing up, with from left to right, the Seven of Diamonds, the Seven of Clubs, the Seven of Spades, and the Seven of Hearts, making sure there is enough space on the table to place above the Sevens all cards from the Eights up to the Kings, and below the Sevens all cards from the Sixes down to the Aces, with all cards belonging to the same suit ending up in the same column. Up to 3 stages are allowed to eventually place all cards. For each stage, all cards that remain are stacked facing down, and the card at the top is taken off the stack, again and again until there is no card left. A card taken off the top of the stack is placed at the location where it has to be if the card just above or the card just below in its suit has been placed already, and in case that is not possible, it is put aside, facing up, above all cards already put aside, if any. If the card could be placed, we then check whether the card at the top of the cards that have been put aside, if any, can itself extend the column for its suit, and it if can, place it at the location where it has to be and again, check the card at the top of the cards that have been put aside, if any, stopping when there is no card left amongst those that have been put aside or when there are some cards left but the one at the top cannot extend the column for its suit, at which point we take off the card at the top of the stack of cards left to process, if any. At the time the stack has become empty, either all cards have been appropriately placed on the table and the game is won, or there is at least one stage left, in which case the stack of cards put aside is turned upside down and becomes the stack of cards to process, proceeding exactly as during the previous stage. So the game is lost if the game is played over 3 stages and not all cards have been appropriately placed on the table at the end of the third stage. The 48 cards (the whole deck with all Sevens removed) are shuffled only once, just before the game begins.

4.2 Playing a single game (3.5 marks)

Your program will be stored in a file named `solitaire_2.py`. Executing

```
$ python3 solitaire_2.py
```

at the Unix prompt should produce the following output (ending in a single space)

```
Please enter an integer to feed the seed() function:
```

with the program now waiting for your input, which should be an integer, and which you can assume will be an integer. Your program will feed that integer to `seed()` before calling `shuffle()`, as described in Section 2, to shuffle the 52 cards minus the four Sevens.

The output starts with an empty line followed by

```
There are _ lines of output; what do you want me to do?

Enter: q to quit
       a last line number (between 1 and _)
       a first line number (between -1 and -_)
       a range of line numbers (of the form m--n with 1 <= m <= n <= _)
```

with all occurrences of `_` denoting the same number. The program should wait for the input on the next line, aligned under `q` and the three leftmost `a`s above. Until `q` is input, the program should output an

empty line, do what it is requested to do if the input is correct, output an empty line and prompt the user again. The program exits when `q` is input. The input is correct if it is exactly as required, including integers being within the required ranges (noting that positive numbers should not be preceded with `+`), except that there can be any amount of space at the beginning of the input, at the end of the input, before the first `-` if entering a range, and after the second `-` if entering a range (no space between the minus sign and the digits of a negative number...).

- The first kind of input will let the program output the first n lines of the collected output, with n being the number provided as input,
- the second kind of input will let the program output the last n lines of the collected output, with $-n$ being the number provided as input, and
- the third kind of input will let the program output that part of the collected output that ranges between the m^{th} and n^{th} lines, m^{th} and n^{th} lines included, with m and n being the numbers provided as input.

Here is a possible interaction.

Here is a possible interaction that displays the complete collected output for a game that is lost.

Here is a possible interaction that displays the complete collected output for a game that is won.

When the input is correct, the first line of collected output reads

```
All 7s removed and placed, rest of deck shuffled, ready to start!
```

and the second line of collected output represents the 48 cards, with all cards facing down (48 `]`s). It is followed by en empty line (whose purpose will be explained further down), followed by 6 empty lines to accommodate the Kings, Queens, Jacks, Tens, Nines and Eights. Then comes a line that depicts the Sevens, followed by 6 empty lines to accommodate the Sixes, Fives, Fours, Threes, Twos and Aces, followed by an empty line.

The rest of the collected output consists of lines that read

```
Starting _ round...
```

with `_` being `first`, or `second` if there is a second stage, or `third` if there is a third stage, followed by an empty line. That is followed by a sequence of lines that are structured as follows.

- First is a line that reads

  Cannot place card from top of stack of cards left ☹

  or

  Cannot place card from top of stack of cards put aside ☹

  or

  Placing card from top of stack of cards left 😊

  or

  Placing card from top of stack of cards put aside 😊
- In all cases except for Cannot place card from top of stack of cards put aside ☹, then comes a line that depicts the stack of cards that remain to be processed, facing down, followed by a line that depicts all cards that have been taken off the stack and could not be placed, facing up.

- In case a card could be placed, either from the stack of cards left or from the stack of cards put aside, come the 13 lines for the Kings down to the Aces to display the cards that have been placed up to now.
- Last comes an empty line.

When Placing card from top of stack of cards left 😊 is used, the stack of cards to process decreases by one. When Placing card from top of stack of cards put aside 😊 is used, the stack of cards that could not be placed decreases by one. In both cases, the card that can be placed is displayed at the intended location. In the first case, the card is "discovered" as the card just added to those already placed, whereas in the second case, the card is known since it was facing up and so we know where to look to find it as the card just added to those already placed.

Cannot place card from top of stack of cards left ☹ is used when a card is taken off the stack to process and cannot be placed. Cannot place card from top of stack of cards put aside ☹ is used when a card has just been placed and the stack of cards that could not be placed is not empty while the card at its top still cannot be placed. In the first case, that card is "discovered" and becomes the new card at the top of the stack of cards that could not be placed.

As soon as all cards have been placed, a last line is added to the collected output that reads:

```
You placed all cards, you won 👍
```

If there is a third stage and it ends with some cards still waiting to be placed, a last line is added to the collected output that reads:

```
You could not place _ cards, you lost 👎
```

with `_` the number of cards that could not be placed.

Each placed card is displayed with the appropriate Unicode character **after one, two, three or four tab characters** depending on whether the **position** of the card on the row is the first, the second, the third or the fourth, respectively, as determined by its suit, and of course after the Unicode characters for any preceding card on the row. **No line has any trailing space**.

4.3 Playing many games and estimating probabilities (3 marks)

Executing

```
$ python3
```

at the Unix prompt and then

```
>>> from solitaire_2 import simulate
```

at the Python prompt should allow you to call the `simulate()` function, that takes two arguments.

- The first argument, say n, is meant to be a strictly positive integer, and you can assume that it is a strictly positive integer, that represents the number of games to play.

- The second argument, say i, is meant to be an integer, and you can assume that it is an integer.

The function simulates the playing of the game n times,

- the first time shuffling the deck of all cards with i given to `seed()`,
- if $n \geq 2$, the second time shuffling the deck of all cards with $i + 1$ given to `seed()`,
- …
- the n^{th} and last time, shuffling the deck of all cards with $i + n - 1$ given to `seed()`.

Here is a possible interaction.

Probabilities are computed as floating point numbers and formatted with 2 digits after the decimal point. Only strictly positive probabilities and the corresponding number of cards left are output (including the cases, if any, when they are smaller than 0.005%, and so output as `0.00%`). The output is sorted in decreasing number of cards left.

There is a single space to the left and to the right of the separating vertical bar, with all lines consisting of precisely 32 characters.