Assignment 1

COMP9021, Trimester 3, 2025

1 General matters

1.1 Aim

The purpose of the assignment is to:

- develop your problem solving skills;
- practice carefully reading specifications and following them;
- design and implement solutions to problems as small Python programs;
- practice arithmetic computations, conditional tests, loops, exceptions, fundamental Python data types, and Unicode characters;
- gain control over print statements.

1.2 Submission

Your programs should be stored in files named `solitaire_1.py` and `solitaire_2.py`. After developing and testing your programs, upload them via Ed (unless you worked directly in Ed). Assignments can be submitted multiple times; only the last submission will be graded. Your assignment is due on October 27 at 11:59am.

1.3 Assessment

The assignment is worth 13 marks and will be tested against multiple inputs. For each test, the automarking script allows your program to run for 30 seconds.

Assignments may be submitted up to 5 days after the deadline. The maximum mark decreases by 5% for each full late day, up to a maximum of five days. For example, if students A and B submit assignments originally worth 12 and 11 marks, respectively, two days late (i.e., more than 24 hours but no more than 48 hours late), the maximum mark obtainable is 11.7. Therefore, A receives $\min(11.7, 12) = 11.7$ and B receives $\min(11.7, 11) = 11$.

The outputs of your programs must exactly match the expected outputs. **You are required to save your outputs in `.txt` files and use the `diff` command to identity any differences between your outputs and the provided expected outputs (also in `.txt` files). You are responsible for any failed tests caused by formatting errors that `diff` would have detected.**

1.4 Reminder on plagiarism policy

You are encouraged to discuss strategies for solving the assignment with others; however, discussions must focus on algorithms, not code. You must implement your solution independently. Submissions are routinely scanned for similarities that arise from copying, modifying others' work, or collaborating too closely on a single implementation. Severe penalties apply.

2 Decks, shuffling

2.1 Decks

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

- Numbers 0 to 7 represent the Hearts, from Seven to Ace.
- Numbers 8 to 15 represent the Diamonds, from Seven to Ace.
- Numbers 16 to 23 represent the Clubs, from Seven to Ace.
- Numbers 24 to 31 represent the Spades, from Seven to Ace.

For example, 6 represents the King of Hearts, and 26 represents the Nine of Spades.

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

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

For example, 16 represents the Four of Diamonds, and 36 represents the Jack of Clubs.

2.2 Shuffling

Both exercises require shuffling a deck of 32 or 52 cards, using the following convention. By *shuffling a deck of cards*, we mean randomising the associated set of numbers by providing the list of numbers, **arranged in increasing order**, as an argument to the `shuffle()` function from the `random` module. For instance, to shuffle the 52-card deck, we could do

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

To ensure predictable results, the `seed()` function from the `random` module should be called with a specified argument just before invoking `shuffle()`.

By *shuffling the 52-card deck with 678 passed to `seed()`*, we mean performing the following steps:

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

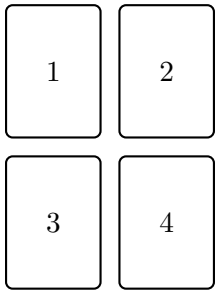
It lets `cards` denote

```
[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. After the deck has been shuffled, the cards in the deck are face down, with the first card at the bottom and the last card on top. The top four cards are revealed and placed at the following locations:



As long as there are at least two cards of the same suit (Hearts, Diamonds, Clubs or Spades), such pairs are discarded simultaneously and replaced with cards drawn from the top of the deck.

- If exactly two cards of a suit are present, both are discarded.
- If exactly three cards of a suit are present, the two in the lowest-numbered locations are discarded.
- If there are two cards each of two different suits, or if all four cards share the same suit, then all four cards are discarded.

Empty locations are filled with cards drawn from the top of the deck, starting with the lowest-numbered locations. Note that at the very end, all four cards may be discarded even if only two cards remain in the deck; these are then placed at locations 1 and 2.

The game is won when the deck is empty. The game is lost if one card of each suit is on the table while cards remain in the deck.

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 with a single space):

```
Enter an integer to pass to the seed() function:
```

with the program awaiting your input, which can be assumed to be an integer. Your program will pass that integer to `seed()` before calling `shuffle()`, as described in Section 2, to shuffle the 32-card deck.

Here is an example interaction for a game that is lost.

Here is an example interaction for a game that is won.

Here is another example interaction for a game that is won.

Note the following:

- There are no tabs in the output.
- No line ends with trailing spaces.
- Each line of "squares" is preceded by an empty line.
- Each "square" is indented with 4 spaces.
- Consecutive cards in a "square" are separated by a space.
- Each line contains at most six "squares" of cards.

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 allows you to call the `simulate()` function, which takes two arguments.

- The first argument, n, is a strictly positive integer representing the number of games to play.
- The second argument, i, is an integer.

The function simulates the playing of the game n times,

- the first time shuffling the 32-card deck with i passed to `seed()`,
- if $n \geq 2$, the second time shuffling the 32-card deck with $i + 1$ passed to `seed()`,
- …
- the n^{th} and last time, shuffling the 32-card deck with $i + n - 1$ passed to `seed()`.

Here is an example interaction.

Here is another example interaction.

The number of rounds corresponds to the number of "squares" shown in the game display.

Pay attention to the formatting. There are no tab characters anywhere in the output, and no line ends with trailing spaces. The text is right-aligned in the first and third columns, and left-aligned in the second.

Frequencies are computed as floating-point numbers and formatted to two digits after the decimal point. Only strictly positive frequencies and the corresponding number of cards left are output, including cases smaller than 0.005%, which are displayed as `0.00%`. The output is ordered in ascending order of cards left, and for equal values, in ascending order of rounds.

4 Second solitaire game

4.1 Game description

It is played with 52 cards. They are shuffled once, immediately before the game begins. The aim is to create four columns of Ace, Three, Five, Seven, Nine, Jack, and King in alternating colours, and four rows of Two, Four, Six, Eight, Ten, and Queen in alternating colours. For instance, a Seven of Hearts or Diamonds can be followed by a Nine of Clubs or Spades, and a Six of Clubs or Spades can be followed by an Eight of Hearts or Diamonds. Up to three rounds are allowed to place all the cards. At each round, the remaining cards (the whole deck at the start of the first round) are stacked face down, and cards are drawn one by one from the top until the stack is empty.

An Ace or Two drawn from the stack starts a new column or row. Columns are created left to right, and rows top to bottom. A card drawn from the top of the stack extends a column or row if possible. Otherwise, it is placed face up on the discard pile. When a card drawn from the stack can extend a column or row, the top card of the discard pile, if any, is used to extend a column or row whenever possible. This process continues with the new top card, and so on. When a card can extend two columns or two rows (because both end with black cards or both with red cards of the same value), the leftmost column or the topmost row is chosen.

When the stack becomes empty, one of three things happens:

- If all cards have been placed on the table, the game is won.
- If any cards remain and fewer than three rounds have been played, the discard pile is turned over to become the new stack, and play proceeds exactly as in the previous round.
- If any cards remain and this was the third round, the game is lost.

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 with a single space)

```
Enter an integer to pass to the seed() function:
```

with the program awaiting your input, which can be assumed to be an integer. Your program will pass that integer to `seed()` before calling `shuffle()`, as described in Section 2, to shuffle the 52-card deck.

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 <= _)
```

In all cases, `_` denotes the same number. The program should wait for input on the next line, aligned with the leftmost non-space characters of the three preceding lines. Until `q` is input, the program should execute the requested action if the input is valid, output an empty line, and prompt the user again. The program exits when `q` is input. Input is valid only if it strictly follows the required format, with integers within the specified ranges (note that positive numbers must not include a leading `+`). Spaces are permitted at the beginning or end of the input, before the first `-` in a range, and after the second `-` in a range. No space is allowed between a minus sign and the digits of a negative number.

- The first kind of input outputs the first n lines of the collected output, where n is the number entered.
- The second kind of input outputs the last n lines, where the input is $-n$.
- The third kind of input outputs the portion of the collected output from the m^{th} to the n^{th} line (inclusive), where m and n are the numbers entered.

Here is an example interaction for a game that is lost.

Here is an example interaction for a game that is won.

Each round begins with a line of the form

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

where `_` is `first`, `second` or `third`. (Note: there may be one or two rounds only.)

When a card is drawn from the stack, the program outputs one of the following messages depending on the card:

- Ace:
  Starting a column 😊
- Two:
  Starting a row 😊
- Can extend a column:
  Extending a column by using the card drawn from the stack 😊
- Can extend a row:
  Extending a row by using the card drawn from the stack 😊
- Cannot extend a column or a row and the card is neither an Ace nor a Two:
  Cannot use the card drawn from the stack ☹

After a card has been drawn from the stack that either starts or extends a column or a row, if the discard pile is not empty, the program outputs one of the following messages depending on the top card of the discard pile:

- Can extend a column:
  Extending a column by using the top card of the discard pile 😊
- Can extend a row:
  Extending a row by using the top card of the discard pile 😊

- Cannot extend a column or a row:

  Cannot use the top card of the discard pile ☹

Of the eight previous messages, each of the first seven is followed by:

- A line representing the stack (one `]` for each card remaining; this line may be empty if the stack is empty).
- A line representing the discard pile (one `[` for each card in the pile except the top one, followed by the top card; this line may also be empty if the discard pile is empty).

If a card starts or extends a column or row, these two lines are then followed by seven lines showing the current state of the columns and rows.

Pay attention to the formatting:

- No line ends with trailing spaces.
- The start of the second and third rounds is preceded by two empty lines.
- All messages described previously are preceded by an empty line.
- The second, third and fourth columns are preceded by one, two, and three tabs, respectively.
- Each row is preceded by five tabs.
- Consecutive cards in a row are separated by a 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 allows you to call the `simulate()` function, which takes two arguments.

- The first argument, n, is a strictly positive integer representing the number of games to play.
- The second argument, i, is an integer.

The function simulates the playing of the game n times,

- the first time shuffling the 52-card deck with i passed to `seed()`,
- if $n \geq 2$, the second time shuffling the 52-card deck with $i + 1$ passed to `seed()`,
- …

- the n^{th} and last time, shuffling the 52-card deck with $i + n - 1$ passed to `seed()`.

Here is an example interaction.

Here is another example interaction.

Pay attention to the formatting. There are no tab characters anywhere in the output, and no line ends with trailing spaces. The text is right-aligned. Frequencies are computed as floating-point numbers and formatted to two digits after the decimal point. Only strictly positive frequencies and the corresponding number of cards left are output, including cases smaller than 0.005%, which are displayed as `0.00%`. The output is ordered in ascending order of cards left.