# Assignment 1

COMP9021, Trimester 3, 2024

## 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, fundamental Python data types, exceptions, Unicode characters;
- print carefully.

### 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 October 21, 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 Shuffling cards

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

- 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.

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, we could do

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

- and to shuffle the deck of all 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 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

The aim is to put aside the 12 pictures (Jacks, Queens and Kings), considering in stages a group of 16 cards taken off the deck and placed facing up.

With all cards in the deck facing down, so with the first card in the deck at the bottom and with the last card in the deck at the top, the top 16 cards are taken off the deck and placed facing up in four rows of 4 cards each, from the top row to the bottom row and for each row, from left to right. So the last card in the deck is placed as the leftmost card of the first row, and the $16^{\text{th}}$ to last card in the deck is placed as the rightmost card of the fourth row. All pictures amongst those 16 cards, if any, are put aside. If $n$ cards are put aside with $n \geq 1$, then the top $n$ cards in what remains of the deck are taken off the deck and replace the cards that have been put aside, from the top row to the fourth row, and for a given row from left to right. Unless no picture can be put aside to start with, the procedure is repeated again and again until no picture can be put aside (so there are between 1 and 12 repetitions), which brings an end to the first round of the game. If all 12 pictures have been put aside then the game is won. Otherwise, the cards that could not be put aside are combined together with those that are left in the deck, making up a new deck of size 52 minus the number of cards that were put aside; that deck is shuffled before the procedure that has been described starts again for a second round. The game stops when all 12 pictures have been put aside within at most 4 rounds, in which case you can call it a win, or when the end of the fourth round is reached and at least one picture has not been put aside, in which case the game is lost.

## 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 52 cards.

Every round starts by shuffling cards, so that is the shuffle that takes place at the beginning of the first round. If there is a second, third or fourth round then there will be a second, third or fourth shuffle, respectively, with fewer and fewer cards. Recall that the cards involved in a given shuffle have to be **sorted**. What is fed to `seed()` is **increased by 1** at every new shuffle. So if the user input 275,

- before the first round, `seed()` will be passed 275 and then `shuffle()` will be called,
- before the second round, in case there is one, `seed()` will be passed 276 and then `shuffle()` will be called,
- before the third round, in case there is one, `seed()` will be passed 277 and then `shuffle()` will be called, and
- before the fourth round, in case there is one, `seed()` will be passed 278 and then `shuffle()` will be called.

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 second line of output represents the 52 card deck, with all cards facing down (52 `]`s).

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

```
Starting first round...
```

The other rounds are announced by an empty line followed by a line that reads

```
After shuffling, starting _ round...
```

with `_` being one of `second`, `third` and `fourth`.

For a given round, the output consists of at least one group of lines, each group being structured as follows.

- The first line in the group is empty.
- The second line in the group reads

  `Drawing and placing _ card`*`[s]`*`:`

  with `_` some number, necessarily equal to 16 when the round starts, and equal to the number $i$ of cards that have just been put aside (of course, it is `card` if $i = 1$ and `cards` otherwise).
- The third line in the group represents the current state of the deck from the top of which cards have been taken off. It is a sequence of `]`s, as many `]`s as there are cards left in the deck. In a given round, the deck decreases in size.
- The next 4 lines in the group represent the 4 rows of 4 cards that are facing up.
- The last line in the group is an empty line.

The group is complete if no card is put aside. Otherwise, there are 6 extra lines in the group:

- an empty line;
- a line that reads

  `Putting _ picture`*`[s]`*` aside:`

  with `_` some number $i$ (of course, it is `picture` if $i = 1$ and `pictures` otherwise);
- 4 lines that represent the 4 rows of 4 cards that are facing up and that are not pictures.

At the very end, the output ends with an empty line followed by a line that reads either

```
You uncovered all pictures, you won!
```

or

```
You uncovered no pictures, you lost!
```

or

`You uncovered only _ picture`*`[s]`*`, you lost!`

with `_` some number $i$ between 1 and 11 (of course, it is `picture` if $i = 1$ and `pictures` otherwise).

Each card in a row of cards facing up 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, and of course after the Unicode characters for any preceding card on the row. **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()`, and then, as necessary, shuffling a smaller and smaller deck, with
  - if there are at least 2 shuffles in this game, $i+1$ given to `seed()` for the second shuffle,
  - if there are at least 3 shuffles in this game, $i+2$ given to `seed()` for the third shuffle,
  - if there are 4 shuffles in this game, $i+3$ given to `seed()` for the fourth shuffle;
- if $n \geq 2$, the second time shuffling the deck of all cards with $i+1$ given to `seed()`, and then, as necessary, shuffling a smaller and smaller deck, with
  - if there are at least 2 shuffles in this game, $i+2$ given to `seed()` for the second shuffle,
  - if there are at least 3 shuffles in this game, $i+3$ given to `seed()` for the third shuffle,
  - if there are 4 shuffles in this game, $i+4$ given to `seed()` for the fourth shuffle;
- ...
- the $n^{\text{th}}$ and last time, shuffling the deck of all cards with $i+n-1$ given to `seed()`, and then, as necessary, shuffling a smaller and smaller deck, with
  - if there are at least 2 shuffles in this game, $i+n$ given to `seed()` for the second shuffle,
  - if there are at least 3 shuffles in this game, $i+n+1$ given to `seed()` for the third shuffle,
  - if there are 4 shuffles in this game, $i+n+2$ given to `seed()` for the fourth shuffle.

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 uncovered pictures 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 uncovered pictures.

There is a single space to the left and to the right of the separating vertical bar, with all lines consisting of precisely 40 characters.

# 4 Second solitaire game

## 4.1 Game description

The aim is to place the 52 cards onto 8 stacks, organised as two rows for, from left to right, the Hearts, the Diamonds, the Clubs and the Spades.

- The top row should consist of increasing sequences of cards based on Aces.
- The bottom row should consist of decreasing sequences of cards based on Kings.

For instance,

- the top row could consist of Hearts stacked up from Ace up to Four, to the left of the Ace of Diamonds, to the left of Clubs stacked up from Ace up to Jack, to the left of Spades stacked up from Ace up to Five,
- in which case the bottom row should consist of Hearts stacked up from King down to Five, to the left of Diamonds stacked up from King down to Two, to the left of the Queen of Clubs on top of the King of Clubs, to the left of Spades stacked up from King down to Six.

With all cards in the deck facing down, so with the first card in the deck at the bottom and with the last card in the deck at the top, the top three cards are taken off the deck and placed facing up, so with the penultimate card, say $B$, sandwiched between the last card, say $A$, at the bottom and the third to last card, say $C$, at the top. If the latter is an Ace or a King, it starts the stack it defines; $B$, now being uncovered, can start another stack if it is also an Ace or a King, or it can increase the first stack if it can go above $C$, because it is a Two and $C$ is the Ace of the same suit, or because it is a Queen and $C$ is the King of the same suit. If both $C$ and $B$ can be placed, then $A$ gets a chance to be placed too. The procedure continues, taking off the next 3 cards off the top of the deck again and again, except maybe at the end, if there are only 1 or 2 cards left to empty the deck. Very likely, not all cards could be placed when this first round of the game ends. If no card could be placed (by the way, your program will estimate that probability), then the game is lost. Otherwise, a new round can start: the cards that were taken off the original deck and that could not be placed make up a new (incomplete) deck that is put upside down, and the procedure that has been described starts again. Either there is a round during which no card is placed, in which case the game is lost, or all cards make it to one of the 8 stacks after some number of rounds, in which case you can call it a win.

## 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 deck of 52 cards.

The output starts with an empty line followed by one of

```
All cards have been placed, you won!
```

or

```
_ cards could not be placed, you lost!
```

with _ a number between 3 and 52 (as it is obviously impossible that only 1 or 2 cards remain...). The output ends 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. Your program should wait for your input on the next line, aligned under q and the three leftmost as above. Until q is input, the program should output an empty line and prompt the user again and again, doing what it is requested to do if the input is correct, doing nothing otherwise. 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 $m^{\text{th}}$ and $n^{\text{th}}$ lines, $m^{\text{th}}$ and $n^{\text{th}}$ lines included, with $m$ and $n$ being the numbers provided as input.

Here is a possible interaction.
When the input is correct, the first line of collected output reads

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

and the second line represents the 52 card deck, with all cards facing down (52 ]s); the third and the fourth to last lines of collected output are empty.
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.
The key pattern in the collected output consists of four lines.

- The first line represents the current state of the deck from the top of which cards have been taken off. It is a sequence of ]s, as many ]s as there are cards in the deck. In a given round, the deck decreases in size and eventually becomes empty, with correspondingly, an empty line in the collected output.
- The second line represents the cards that have been taken off the deck and that have not been placed. As those cards are facing up, the last card is displayed with the appropriate Unicode character; the cards underneath, if any, are represented by [s. There can be no such cards (because all cards that could be placed have been placed), with correspondingly, an empty line in the collected output. Overall, in a given round, the number of such cards increases (recall that at the end of a round, if the game continues, then those cards make up the deck for the next round).
- The third line represents the increasing sequences built on top of Aces. The line starts empty. As the cards are facing up in a given stack, the last one is displayed while the cards underneath, if any, are represented by [s.

- The fourth line represents the decreasing sequences built on top of Kings. The line starts empty. As the cards are facing up in a given stack, the last one is displayed while the cards underneath, if any, are represented by `[`s.

When a card is placed, a line just before the first of the four lines just described is saved in the collected output, that reads either

```
Placing one of the base cards!
```

or

```
Making progress on an increasing sequence!
```

or

```
Making progress on a decreasing sequence!
```

The beginning of each round is announced with a line that reads

```
Starting to draw 3 cards (if possible) again and again for the _ time...
```

where `_` is one of `first`, `second`, `third`, `4th`, `5th`, `6th`...; that line is surrounded by empty lines.

When not empty, both lines with the increasing and decreasing sequences of cards start with 4 spaces, the starts of two consecutive stacks being distant of 15 characters. **No line has any trailing space, nor is there any tab character anywhere**.

### 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 $n^{\text{th}}$ 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.