Information and Communication Technologies — Test

Python Test — Part 1

For those exercises, write your answers using Python.

Exercise 1

Write a program that goes through all numbers in an array of integers and creates two new arrays. The first array will contain only the even numbers from the initial one, and the second array will contain only the odd numbers from the initial one.

- a call on the array `[32, 5, 12, 8, 3, 75, 2, 15]` must build two arrays:
  `evens` that will be equal to `[32, 12, 8, 2]`
  `odds` that will be equal to `[5, 3, 75, 15]`.

Exercise 2

Write a function `even_sum` that takes as input an array of numbers, and outputs the sum of the numbers which are located on even indexes.

- a call `even_sum([1, 2, 3, 4, 5])` will output 9.

Exercise 3

You are going to move out from your old flat, and in order to do so, you rent a little truck that can transport up to 8 furnitures. You choose a contract that costs 100€ and that allows you to drive 250 km (with the truck). Here is your organization:

- (by car) go to the warehouse to pick up your truck;
- (by truck) go back to your first flat;
- (by truck) do the moving with the furnitures;
- (by truck) after the last furnitures are in your new flat, you go back to the warehouse;
- (by car) you go back to your new flat.

Here are the distances (one-way):

- between old flat and warehouse: 15 km
- between old flat and new flat: 25 km
- between new flat and warehouse: 15 km

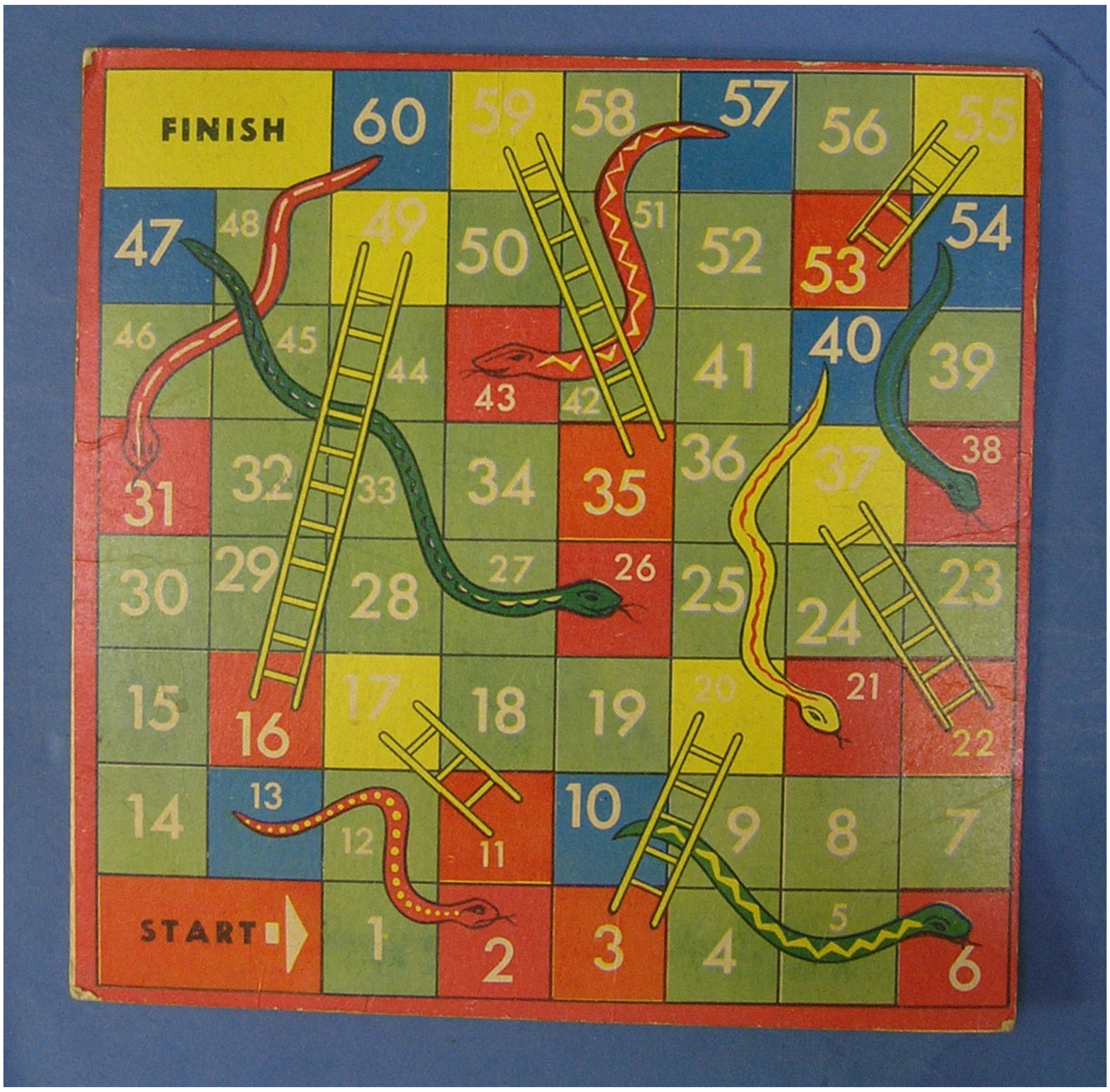
The oil consumption makes you spend an additional 1€ every 10 km.

1. What is the maximum number of furnitures you can move if you use this contract to move in your new flat?
2. Let us denote by n the number of furnitures you want to move. Compute the price you'll pay (taking into account everything) when...
   - $n = 5$
   - $n = 8$
   - $n = 19$
   - $n = 25$
3. Write a function with the following specifications:
   - Input: an integer `n` (the number of furnitures to move);
   - Output: the total cost to move `n` furnitures with the truck, with this contract;
   - Errors to catch: (a) you must first check that `n` is positive (> 0) and (b) don't forget to check that the contract allows to move the desired number of furnitures, and print an error if this is not the case.

Information and Communication Technologies — Test

Python Test — Part 2: Snakes and ladders

The game "Snakes and ladders" is a popular game where any number of players may play. Each player has a pawn that starts on the first cell of the game, and the goal is to be the first player whose pawn reaches the last cell of the game. At each turn, you throw a 6-sided die, and move your pawn according to the die roll (the 6 faces of the die show the 6 numbers from 1 to 6; if your pawn would move past the end cell, it stays on the end cell[1]). If your pawn ends on the lower part of a ladder, your pawn moves up the ladder (nothing particular happens if your pawn ends on a cell covered by a ladder if it's not the lower part of the ladder) and if your pawn ends on the tail of a snake, your pawn moves down the snake down to its mouth (nothing particular happens if your pawn ends on a cell covered by a snake if it's not the head of the snake). In this work, we will consider the following board game (the start cell can be labelled "0" and the end cell can be labelled "61"):



Source: https://commons.wikimedia.org/wiki/Category:Snakes_and_ladders

The objective of this project is to program different ways to play this game and to find some properties of this game.

[1]In the "Game of Goose", your pawn would go back, this is different here.

1 Without Python

To play this game, we consider an array `special_cells` which gives, for each special cell (lower parts of ladders, snakes tails), the cell where the pawn has to go if it ends on this cell. For example there is a ladder that starts on cell 3 and ends on cell 20, hence `special_cells[3]` has the value 20. For cells that are not special, we put the value 0.

We give in Figure 1 the content of this array:

$$special\_cells \leftarrow [0,0,0,20,0,0,0,0,0,0,6,17,0,2,0,0,49,0,0,0,0,0,37,0,0,0,0,0,0,0,0,0,0,0,0,59,0,0,0,0,21,\\ 0,0,0,0,0,0,26,0,0,0,0,0,55,38,0,0,43,0,0,31,0]$$

Figure 1: Content of the array "special_cells" (62 cells, from 0 to 61).

1. First, we want to count the number of ladders in the game, by looking at the `special_cells` array, without looking at the board game.

   (a) Let i denote the index of a cell. What test must be done on `special_cells[i]` to know whether there is a ladder starting from cell i?

   (b) The algorithm in Figure 2 is an attempt to count the number of ladders. In fact, this algorithm does not compute the number of ladders. What does this algorithm compute instead?

   (c) Explain how to correct the algorithm in Figure 2.

```
Variables:
$nb\_ladders$ and $i$ are two integers.
$special\_cells$ is an array of integers, defined in Figure 1.

Instructions of the function:
$nb\_ladders \leftarrow 0$
For $i$ from 0 to 61
    If ($special\_cells[i] \neq 0$), Then
        $nb\_ladders \leftarrow nb\_ladders + 1$
    End If
End For
Return $nb\_ladders$
```

Figure 2: Algorithm "ladder_count".

2. Now, we want to simulate a single-player version of the game. There is only one pawn on the board, that starts on cell 0. We wish to count how many die throws it takes to finish the board.

   (a) The function $randint(a, b)$ outputs an integer chosen uniformly at random between a (inclusive) and b (inclusive). How can we use this function to simulate a die roll?

   (b) Complete the algorithm in Figure 3 to simulate the single-player game.

   (c) Modify the algorithm to also count the number of die rolls needed to reach the finish cell.

   (d) BONUS: In fact, there is a high probability that this algorithm will crash, at the end of the game, in the likely case where `pawn_cell` will be greater than 61. Give an example where `pawn_cell` will be greater than 61 and explain what will make the algorithm crash.

```
Variables:
pawn_cell and die_roll are two integers.
special_cells is an array of integers, defined in Figure 1.

Instructions of the function:
pawn_cell ← 0
While (.................), Do
        die_roll ← randint(..., ...)
        pawn_cell ← pawn_cell + die_roll
        If (special_cells[pawn_cell] ≠ 0), Then
                pawn_cell ← special_cells[pawn_cell]
        End If
End While
```

Figure 3: Algorithm "single_player".

2 With Python

For this section, please start by downloading the following file, that contains the python implementation of the algorithms in the previous section:

http://www.barsamian.am/2022-2023/S6ICTC/Test_Snakes_and_ladders.py

The two first questions are independant. The BONUS question uses the results from both questions.

1. Implement the function `is_correct` that takes as parameter an array describing a board (the array *special_cells* defined in Figure 1 is one such array), and returns a boolean telling whether this is a valid board for the game. A valid board for this game must respect the following rules:

   - it has 62 cells
   - it has 6 ladders (you can use function "ladder_count" already given in the python file)
   - it has 7 snakes (you can write a function similare to "ladder_count" to count this)
   - the total number of cells "going up" on ladders must be equal to 97 (you can use function skipped_cells_count already given)
   - the total number of cells "going down" on snakes must be equal to 114 (you can use function lost_cells_count already given)

   Remark: the *special_cells* array is a valid array, so please check that your function returns true when this array is given as parameter.

2. We give the function `single_player_nb_die_rolls` that takes as parameter an array describing a board, and returns the number of die rolls needed to finish the board, on one specific game (the game simulated is different each time you call this function, because it uses random die rolls).

   Please fill the function `average_die_rolls` that takes as parameter an array describing a board, and returns the average number of die rolls needed to finish 10000 games.

BONUS Can you give another valid board such that the average number of die rolls needed to finish it is at least one more than for the *special_cells* board? Another valid board such that the average number of die rolls needed to finish it is at least one less than for the *special_cells* board?