STATISTICAL PROGRAMMING – PYTHON
Individual Programming Exercise (IPE)

How are you going to be assessed?

Your grade starts from 10.0

1. Maximum **- 10.0** if you copy part of your colleague's code.
2. Maximum **- 8.0** if your code does not run correctly when professor tests it.
3. Maximum **-6.0** if your code does not implement the whole game (hard/easy; many rounds, infinite games until killing).
4. Maximum **-4.0** if your code does not follow the instructions on how to implement the game: random function, the counts and the computer strategy, reading inputs and parameters from player, etc...
5. Maximum **-2.0** if your code does not follow the output exactly as requested.
6. Maximum **-1.0** if your code is hard to understand and/or uses variable/function names hard to make out what they do.
7. Maximum **-0.5** if your code does not include comments to help understanding it.

First, please watch this 15 minute video recorded by the professor helping you to start with the exercise - https://www.youtube.com/watch?v=RX51zO5G0n4

The exercise: Human behavior prediction

The study of the predictability of human behavior is an intense field of research, mainly for its applications in stock exchanges, in games, in government treatises, sports and also to identify which student will give up the python course 😉 Recently, computational methods have been surpassing the human capacity to predict behavior [1].

In this programming exercise, we will implement a program that "predicts" human behavior using a simple game.

The game consists of a human player and the computer. Both choose either the value 0 or the value 1. If both choose the same numbers, the computer earns a point. If the numbers chosen are different, the human player earns a point. This is repeated n times. The winner is the one who accumulates more points at the end of n moves. The strategy to be implemented for the computer to "predict" the human player's bid is based on a simple "machine learning" algorithm.

This learning assumes that every choice the player makes is influenced by his/her previous move. In this way four counts will be made:

(1) throw00 = count of the number of times the human player chose 0 given that in the previous bid his/her bid was 0,
(2) throw01 = count of the number of times the human player chose 0 given in the bid previous his/her bid was 1,
(3) throw10 = count of the number of times the human player chose 1 given that his/her previous bid was 0,
(4) throw11 = count of the number of times the human player chose 1 given his/her previous bid was 1.

So, the following cases may occur:

If the player's last throw was 0:

1. If throw10 > throw00: then the computer chooses 1
2. If throw10 < throw00: then the computer chooses 0
3. If throw10 = throw00: then the computer chooses randomly 0 or 1.

If the player's last throw was 1:

4. If throw11 > throw01: then the computer chooses 1
5. If throw11 < throw01: then the computer chooses 0
6. If throw11 = throw01: then the computer chooses randomly 0 or 1.

To avoid cheating, in your code, always make the computer choice 0 or 1 before asking the user for its choice of move 0 or 1.

For the first throw of the computer, there is no previous throw of the player, so computer chooses randomly 0 or 1.

For the random throw, proceed as follows. Generate a random number using the linear congruences method (see Appendix). If the random number is less than or equal to $2^{32}/2$ ($=2^{31}$), the computer will play 0. Otherwise, it will play 1. To facilitate the structure of your program, generate a new random number at the beginning of each move, and use it only if necessary according to the rules above. If the random number is not used it will be discarded at the beginning of the next move when a new number is generated.

Only when you initialize your python game, it will require an initial seed x0, manually choose an integer value of your choice between 1 and 2^{31}.

After n rounds, wins who gets the most points (player or computer).

Your Individual Programming Exercise (IPE) should receive as input (via keyboard) the number of moves to be made (positive integer) and difficulty of the game (easy or difficult).

In "easy" mode, the computer has no learning, i.e., all computer throws are random, using the method of linear congruences. If the random number generated is less than or equal to $2^{32}/2$ ($=2^{31}$), the computer will choose 0 and otherwise, 1.

In "difficult" mode, the computer will use the learning method described above.

During the match, your IPE should show graphically how the game is going, i.e. print a bar that illustrates the score of the computer and the score of the player. The score will be simulated by the impression of asterisks.

Example:
PLAYER: *
COMPUTER: ***

(in this example, the player won once and the machine three times)

At the end of the game (after the n-th0 round), you must print on the screen who was the winner or if there was a tie.

Example 1:
You won!

Example 2:
Computer won!

Example 3:
It was a tie!

Play against the computer and see if you are able to beat it!

See below an example of playing the game twice and killing it. Your program should run new games until the user kills the program or the user chooses to stop playing with an option that you provide.

The output must be exactly the same format as in the example below. The red part will not be in the print massage, they are examples of inputs entered by the player. (Replace "Your Name" with your actual name)

Welcome to Human Behavior Prediction by Your Name

Choose the type of game (1: Easy; 2: Difficult): 2
Enter the number of moves: 5

Choose your move number 1 (0 or 1): 1
player = 1 machine = 0 – Player wins!
PLAYER: *
COMPUTER:

Choose your move number 2 (0 or 1): 0
player = 0 machine = 0 – Computer wins!
PLAYER: *
COMPUTER: *

Choose your move number 3 (0 or 1): 1
player = 1 machine = 1 – Computer wins!
PLAYER: *
COMPUTER: **

Choose your move number 4 (0 or 1): 0
player = 0 machine = 0 – Computer wins!
PLAYER: *
COMPUTER: ***

Choose your move number 5 (0 or 1): 0
player = 0 machine = 1 – Player wins!
PLAYER: **
COMPUTER: ***

Difficult game is over, final score: player 2 - 3 computer - The COMPUTER won!

Choose the type of game (1: Easy; 2: Difficult): 1
Enter the number of moves: 5

Choose your move number 1 (0 or 1): 1
player = 1 machine = 0 – Player wins!
PLAYER: *
COMPUTER:

Choose your move number 2 (0 or 1): 1
player = 1 machine = 0 – Player wins!

```
PLAYER: **
COMPUTER:
---
Choose your move number 3 (0 or 1): 1
player = 1 machine = 1 – Computer wins!
PLAYER: **
COMPUTER: *
---
Choose your move number 4 (0 or 1): 1
player = 0 machine = 0 – Computer wins!
PLAYER: **
COMPUTER: **
---
Choose your move number 5 (0 or 1): 1
player = 0 machine = 1 – Player wins!
PLAYER: ***
COMPUTER: **
---
Easy game is over, final score: player 3 - 1 computer – You won!

Choose the type of game (1: Easy; 2: Difficult):
```

(the last part did not have a number, because I killed it using CRLT+C)

Here is a list of helpful videos covering important Python concepts:

I do recommend you to go through the 31 videos of Socratica YouTube Channel - https://www.youtube.com/watch?v=bY6m6_IIN94&list=PLi01XoE8jYohWFPpC17Z-wWhPOSuh8Er-

If you do not have time now to watch them all, the specific ones that cover concepts that could help you with the individual exercise are:

2- https://www.youtube.com/watch?v=KOdfpbnWLVo
5- https://www.youtube.com/watch?v=_87ASgggEg0
7- https://www.youtube.com/watch?v=Aj8FQRIHJSc
9- https://www.youtube.com/watch?v=9OK32jb_TdI
11- https://www.youtube.com/watch?v=f4KOjWS_KZs
12- https://www.youtube.com/watch?v=NE97ylAnrz4
14- https://www.youtube.com/watch?v=ohCDWZgNIU0
31- https://www.youtube.com/watch?v=nlCKrKGHSSk

Bonus Part (careful there is a lot of work here and only worth extra participation)

8. Extra participation point - if your code never breaks whatever the input entered by the user (if it is not an allowed number, if it is text, if it is empty, if it is float).
9. Extra participation point if you implement the function linear_congruences in a file called demo.py, save it inside a directory called mypackage and imports this file to use the function as explained in this video - https://www.youtube.com/watch?v=qmsTqQbcBNM.

```
ep1.py
mypackage
├──── __init__.py
└──── demo.py
```

Python interprets "import mypackage.demo" as a include the code ./mypackpage/demo.py in my current code; in other words, mypackage is a folder which must contain a file demo.py and the folder mypackage must be in the same directory of the code ep1.py.

Appendix:

Method of linear congruences

Given an initial number x_0, known as seed, the number of the sequence x_1 is given by $x_1=(ax_0+b) \bmod (m)$. In general form, the x_{i+1} number is obtaining from the previous number x_i using the formula $x_{i+1}=(ax_i+b) \bmod (m)$. mod is the modulo operator[1]

For example, for a = 7, b = 1, m = 13 and $x_0=3$, the sequence of numbers generated is: 9, 12, 7, 11, 0, 1, 8, 5, 10, 6, 4, 3, 9, ...

In this IPE, you must use as parameters the values a=22695477, b=1 e $m=2^{32}$, which are used by known systems.

How to print result bars:

To print * 5 time, simply use: print ("*"*5)

Bibliografia

[1] Kanter JM, Veeramachaneni K. Deep feature synthesis: towards automating data science endeavors. IEEE International Conference on Data Science and Advanced Analytics, 2015.

Common Questions and Answers:

Q: Which seed are you using so that we can test to see if our code is running correctly?
A: For testing, I will use 1234 as in the video: https://www.youtube.com/watch?v=RX51zO5G0n4.

Q: If I do the bonus part, session 9, what do I have to deliver?
A: A zip file with the whole package structure. ep1.py and directory with __init__.py and demo.py (note that __ is double underscore, not a single underscore)

Q: Can I use random.randint()?
A: No, the method of linear congruence is generating random numbers for you for both difficult = 1 and also when you need it for difficult = 2.

Q: In your example 1 above with difficult = 2, the computer move should be 0, how is it possible that the computer wins?
A: The example is correct. If you're getting a different result, it is possibly because you are using the current user move in the decision. You should never use the current move of the player in the decision, which is like cheating☹ In the 3rd move t00 = t10 = 0, therefore the 3rd move is also random as the 2 previous ones. To avoid risk of "cheating", I strongly recommend that in your code, you always make the computer choice 0 or 1 before asking the user for its choice of move 0 or 1.

Q: Should I count t00, t01, t10 and t11 when playing difficult = 1?
A: Not mandatory, but you would learn more if you do. No bonus given.

[1] https://en.wikipedia.org/wiki/Modulo_operation

Q: I play 2 games successively, should I restart t00 = 0, t01 = 0, t10 = 0 and t11 = 0?
A: No, you should never restart t00, t01, t10 and t11. It only restarts when you start the whole program again.

Q: I play 2 games successively. For Game 2 => last move value for the first round of game 2 should be considered as unknown or to be equal to the last move of Game 1 ?
A: Not mandatory, but you can consider it to be equal to the last move of Game 1. No bonus given.

Q: For difficult 2, first two moves have to be random?
A: I will consider correct if you do it random (as in difficult = 1) or if you consider the experience of previous games (taking into account t00, t01, t10 and t11).

Q: Should I record t00, t01, t10 and t11 in a file and read it every time the Python program start?
A: Don't! Please do not over complicate things. But I won't discount points if you do.