University of St. Gallen

School of Management, Economics, Law, Social Sciences, International Affairs and Computer Science (HSG)

Documentation Python Coding Project
Python Character Tournament

Dario Patzi Chavez	20-609-467	DarioP
Marja Emilia Steuer	19-613-363	Emy
Mathilde Bianco	19-616-978	mtldb
Robert Josua Michels	18-615-286	therobo

Course:	4,799,1.00 Skills: Programming - Introduction Level
Lecturer:	Dr. Mario Silic
Group ID:	1142
Coding Language	Python

Table of Contents

1 Aim of the Project

The aim of the project was to simulate a fight between characters. At the beginning, the user is asked to choose their character amongst a predefined list of characters as well as to select an attribute for their character from a predefined list of attributes. Each character (Harry Potter, Ron Weasley, Draco Malfoy, Voldemort) has a distinct initial health and strength level. A higher health setting increases survival chances, while a high strength increases the damage caused to the opponents' health during the fight (stronger punches). As will be explained, the fighter who survives the longest (most punches) is the winner of the fight. The user must choose their character bearing this tradeoff between strength and health in mind. Additionally, the user is presented with a list of seven attributes to choose from (coffee, sword, protein shake, hat, pizza, skis and vision goggles), each having a different health and strength level. Based on the attribute chosen, the health and strength settings of the character are modified to reflect the sum of the characters settings and those of the attribute. While the user can choose a character and an attribute, the remaining characters are assigned an attribute at random. It is possible that multiple fighters have the same attribute. The user's configurated fighter will then fight another of the fighters. During the fight, the fighters punch the other fighter with a fraction of its strength, thus decreasing its opponent's health by the respective amount. Both fighters punch their opponent "simultaneously", such that at the end of the fight, both fighters threw an equal number of punches. The only way to win this fight is if a fighter still has positive health while the opponent's health is negative. Simultaneously to the users' fight, the remaining two characters of the list will fight each other. The winner of the two fights will then duel themselves to determine the ultimate winner of the tournament while the losers will fight for third place.

2 How it was Accomplished

To accomplish this aim, the code first defines a class for the character, the attribute and the fighter respectively. Subsequently, the list of characters and attributes are added to their respective class. In a next step, the code creates a function that allows the user to pick its character and another function to select the attributes. After choosing the character and attributes, a function was created that implements the selection and, if not possible, asks the user to reselect their fighter. Following this, another two functions are there to attribute the attributes to the other fighters at random and to select the fighters. Once the selection has been implemented successfully, the code calls on the next function which determines the winner and

subsequently, a function that illustrates the results of the tournament. The next function of the code programs the actual fight. Additionally, after running the fight the user is given the possibility to restart the tournament if so desired.

3 Project Code in Details

3.1 Prerequisite, Classes and importing Characters & Attributes

While the previous section explained how the code was written to enable the creation of a fictional tournament, this section will analyze the code in more depth. Before we begin, it is crucial to remember that, for the successful execution of this code, two libraries must be imported (numpy and random), as these will be required within the proceeding functions. As mentioned, the next step includes the creation of three distinct classes, one for the characters and attributes respectively and the third for the fighters which incorporates the information from the previous classes. Each class consists of a __init__, __str__, and __repr__ function, while the third class (Fighter) additionally includes a fight function. This fight function is what enables the code to execute the tournament and be able to retrieve the required information from the instances.

Looking more in depth at the fight function within the Fighter class, we see that this function requires two mandatory arguments: self and opponent. Accordingly, this function will only be able to execute given two fighters, representing the two fighters in the duel. The next line creates a variable called discounter, which is assigned the initial integer 9, which represents a discount factor of the total strength per punch of the fighter (i.e. each punch is a fraction of the fighters total strength). Within the proceeding infinite loop, the first if statement acts as a safety mechanism to prevent a fight amongst exact equals (fighters who have, including their attributes, exactly the same health and strength level). Since there will, by definition, never be a winner between these two fighters, this safety mechanism prevents unnecessary code execution and instead selects either fighter randomly as the winner (with justification of a “lucky punch”). This winner is treated identically later as if they had overcome their opponent. Next, if the if statement was not True, the discounter variable is increased by one. This serves as to alter weight of each punch in the scenario of no clear initial winners. Here again, a safety mechanism, build as an if statement, ensures to give an eventual ending point if a clear winner cannot be found. Precisely, after 9’990 iterations of this section, a random winner is selected based on the aforementioned “lucky punch” principle, since neither fighter managed to beat their opponent (even though their health’s and strengths are not identical). If neither of these if

statement is True, the next lines of code are executed, which represent the actual fighting mechanism. For this, new variables are created as the fight process should not alter the settings of the fighters (since the second stage of the tournament restarts with fresh health and strength). Accordingly, for each iteration, the value of a punch (damage_taken and damage_given) is different. The health of both fighters is systematically reduced by a punch of the opponent until either fighter's health is below zero, as seen in the while loop condition. Once eithers health is negative, the code checks whether the other fighter's health is positive in which case they are the winner while the loop restarts from the top (increasing the discounter hence altering the strength of each punch and restarting the fight process with initial health) if the health of both fighters is negative. This is since, as mentioned above, the fighter must survive the fight to be considered the winner. Accordingly, once this infinite loop ends, there is one clear winner for each fight in the stage and they each get awarded an increase of one to their gamewon argument, which becomes relevant later in determining the second stage of the tournament.

Following this, as mentioned above, the predefined characters and attributes are imported into their respective class. We chose to limit the tournament participants to four fighters and limit the attributes to seven, yet this could be changed by adding more instances of either class.

3.2 Functions

3.2.1 let_user_pick_C(characters)

This function, later used within the continue_with_character() function, lays the groundwork for the user to be able to choose which character they would like from the characters list. To do so, an infinite loop is created which asks the user to input a number which represents one of the characters. This loop will run until the user has selected an integer which corresponds to the number of characters available, in our case four characters.

3.2.2 let_user_pick_A(characters)

Similarly to let_user_pick_C(characters), this function, which is also used later within the continue_with_character() function, lays the groundwork for the user to be able to choose which attribute they would like to equip to their character. The user is again asked, with the input function, to insert an integer, within the range of the available options of attributes, which represents the chosen attribute. This loop again will run indefinitely until the code is either interrupted or an acceptable integer is inserted.

3.2.3 continue_with_character()

This function serves as to ensure that the user is satisfied with their chosen character and attribute. Initially, this function creates three global variables since these will be required later in the code, outside of this function. Thereafter, this function calls both let_user_pick_C() and let_user_pick_A() function to receive the user's choice, before asking the user whether they are satisfied with their choice. Here, it does not matter whether the user capitalizes their answer or does not, since the code transforms the string into uncapitalized when checking for identity. Only following the satisfaction of the user with their choice will the selection be saved as an instance of the Fighter class, aggregating the health (and strength) of the character with that of the attribute.

3.2.4 create_fighters()

This function serves as to assign a random attribute to the remaining characters, excluding the chosen character of the user. This is done by, for each character not chosen by the user, selecting a random integer which corresponds to the index of an attribute. Thereafter, the combination of character and attribute is converted into an instance of the Fighter class, aggregating the health (and strength) of the character with that of the attribute. As can be seen here, the selection of attributes is nonexclusive.

3.2.5 selection_to_fighter()

This function initially creates an empty list, which is saved as a global variable due to the necessity of later codes not within this function. Thereafter, continue_with_character() and create_fighters() are called executed, while saving the respective Fighter class instances into this newly created list. At the end, the list fighters includes all the fighters, as instances of the Fighter class, that will participate in the tournament, including their aggregate levels from combing the character settings with those of the attribute.

3.2.6 winner()

This function serves to determine the overall winner of the tournament, based on the value of gamewon. It additionally creates four global variables, one for each ranking, as these will be used within the standing() function. In order to determine the winner, this code checks which fighter has a higher gamewon, in the respective lists (created in the fighting() function). Since the winners of the first stage of the tournament are saved in a different list from the losers, checking for the winners of the second stage is easy. The winner of the tournament will have a gamewon equal to two (since they won both games), while the second and third both won one

game, and the last having won no game hence a gamewon equal to zero. Last, this function calls the function standing().

3.2.7 standing()

The purpose of this function is to illustrate to the user the outcomes of the tournament. It will additionally display a personalized message depending on the outcome of the user's chosen fighter. This function additionally also resets the gamewon's of all fighters, as this will be required if the user desires to start a new tournament.

3.2.8 fighting()

This function is at the core of the entire code, due to its creation of the fighting process. It begins by creating global variables, an empty list and a copy of the fighters list (which includes all instances of the Fighter class). It then randomly selects two integers, in order to randomly distribute the fighters of the fighters list into couples, which are saved as tuples in a list tournament. Since our code only created four characters, there are only two tuples. Next, the first stage of the fight is executed, using the fight function of the Fighter class between the fighters in the tuple. The winners, those with a gamewon equal to one, are added into an empty list (TMW) while the losers are added in another list (TML). The second stage of the fight, identical to the first, then fights the winners against each other and losers against each other. Last, this function calls on the function winner() to illustrate and determine the winner proceeded by executing the restart_fight() function.

3.2.9 restart_fight()

As the name already gives away, this function serves to give the user the option to restart the fight. The user is asked and, depending on their answer, the whole tournament selection is restarted or ended. In case the user desires to restart the tournament, the function tournament_duel() is executed.

3.2.10 tournament_duel()

This function only serves to facilitate the initiation of the tournament. It executes initially the selection_to_fighter() function followed by the fighting() function. Therewith, all functions listed above are called without requiring excessive lines of codes calling each separately. On the condition that all prerequisites are loaded, it is sufficient to run tournament_duel() to initiate the fighting procedure of the tournament.

4 Remarks

This program enables the user to play an interactive game, whereby their decisions and a bit of luck directly influence the outcome of the tournament. While different combinations of characters and attributes may result in higher probabilities of winning the tournament, this is not guaranteed. If the user would like to find out whether there is a definitive winning combination, all possible combinations of their opponents must be considered. A certain combination (e.g. Harry Potter with Coffee) may in some instances win first place yet may just as well place last depending on the combinations of the opponents. Accordingly, a further project could attempt to find a pattern of which attributes/characters give the highest probability (or certainty) of victory. In addition, this program gives the user the ability to personalize it, they could alter the levels of attributes/characters, change the naming of attributes/characters or even add new instances of characters or fighters. The latter would require minor tweaks in the code to accommodate the increased number of participants.