**Python Assignment 1**

This assignment consisted of writing a program to generate a list of abbreviations that were deemed ‘acceptable’ by a number of criteria, provided to us in the assignment brief. The script would generate a list of all possible abbreviations, as well as their scores (calculated according to the criteria provided), filter out non-unique abbreviations, and writing the list of acceptable abbreviations to a file. Furthermore, the script should be able to be run from the command line, resulting in an *if \_\_name\_\_ == ‘\_\_main\_\_’* clause.

When receiving a list of names to reduce to abbreviations, the script separates each name in the list into a concatenated, and a disjoined form, both capitalised. This is such that the input “James Hunt” would produce “JAMESHUNT”, and “JAMES HUNT”. This capitalisation is as required in the brief. A list of unique abbreviations is also created. Then, using the regex method, the first letter of the concatenated form is found, and stored in a list. The script then enters two layers of looping. The first will find the second letter, and the second will find the third. It will then calculate the composite score of the abbreviation, and store it and its score in the list of unique abbreviations. The abbreviation and score will then be written to the file. The program loops again, and again, until all possible abbreviations have been created. Along the way, the script checks the abbreviation it currently stores, against the list of unique abbreviations. If the abbreviation does not occur in the list, it is added to both the list and the file. If it occurs in the list, then it and its score are thus discarded, and the program moves on. For a name like “James Hunt”, this would result in the first abbreviation “JAM, 36”, the second abbreviation “JAE, 66”, the third “JAS, 31”, continuing on through the last name, and coming around again for “JME, 50”, “JMS, 15”, and so on so forth. Each step of the way, the score of the currently-being-constructed abbreviation is tracked, computing the score due to the first letter, adding it to the score of the second letter, and then finally the third, before resetting back to zero, popping the necessary items from the list, and starting again.

When checking each letter for its score, we were instructed to take note of its position, and also its general prevalence within the English language. For example, if the letter captured was at the end of the word, *and* it was a capital E, that was given a score of 20. If it was the end of a word and *not* E, it was given a score of 5. Also, if the letter happened to be the start of the word, it scored 0. Else, the character was scored based on its position in the word, and the prevalence in the language. This was done using the python regex library, *re*’s .search() method on the disjoined version of the word, capturing the start of the match, and checking its position relative to the end of the word. This was done for the start of the word.

To get the positional score of the letter in the word, the disjoined copy of the word was passed to an evaluation function, which used python’s inbuild .find() method, along with a match-case (included in python 3.10 and later) to score the position of the word.

To retrieve each letter’s individual score, the values were read in from a .csv, and imported into a dictionary. The letter in question was passed to the function, and an output returned.

During the process of building the program, care was taken to ensure integrity of the inputs, ignoring all non-alphabetical characters, and catching errors where they might occur. Testing was also done on each input to make sure that the program was executing in correct fashion, in this case it would be producing abbreviations that were not duplicated, and scoring them correctly.

Lastly, to ensure that the script could be run from the command line of a machine, an if ‘\_\_name\_\_’ == ‘\_\_main\_\_’: clause is used, this is boilerplate code used to store code that should only be run when the script is run as just that, a script, and not when imported as a module.