ES2B4 Computer Engineering and Programming: Assignment 2

SCHOOL OF ENGINEERING – UNIVERSITY OF WARWICK

NAME: PABLO NAVAJAS HELGUERO

ID: 1604648

SECTION: A1. Tuples, Lists and Dictionaries

A1.3 **Understanding List Comprehensions**

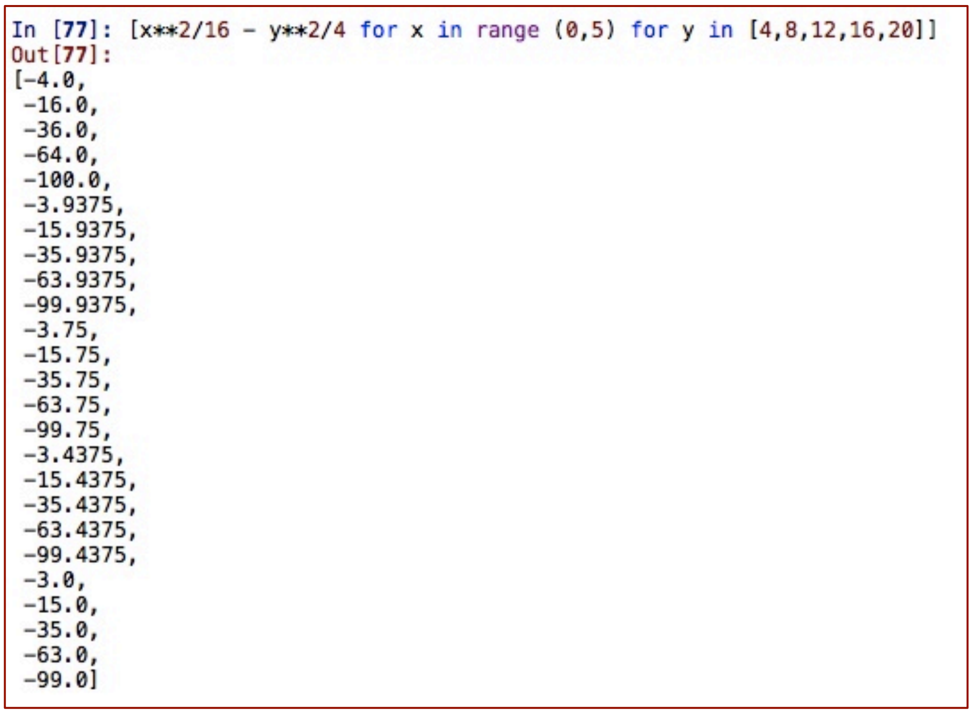
Exercise 1 –

This code calculates the result of the equation (equation1) for a set of values.

$$\frac{x^2}{16} - \frac{y^2}{4} \tag{1}$$

For each value of x (in a range from 0 to 5) it performs the calculation with all the values of 'y'. This means that for 'x=0' it calculates the result of all the cases of 'y'.

This is due to the fact that y is defined as a list of values.



```
In [77]: [x**2/16 - y**2/4 for x in range (0,5) for y in [4,8,12,16,20]]
Out[77]:
[-4.0,
 -16.0,
 -36.0,
 -64.0,
 -100.0,
 -3.9375,
 -15.9375,
 -35.9375,
 -63.9375,
 -99.9375,
 -3.75,
 -15.75,
 -35.75,
 -63.75,
 -99.75,
 -3.4375,
 -15.4375,
 -35.4375,
 -63.4375,
 -99.4375,
 -3.0,
 -15.0,
 -35.0,
 -63.0,
 -99.0]
```

Image 1: Output of first exercise (A1_3_1)

Exercise 2 –

In this exercise, I created a dictionary record type using list comprehension. First of all, I create the list of 'subject_code' and the list of 'studentrecords' with all the marks of each student.

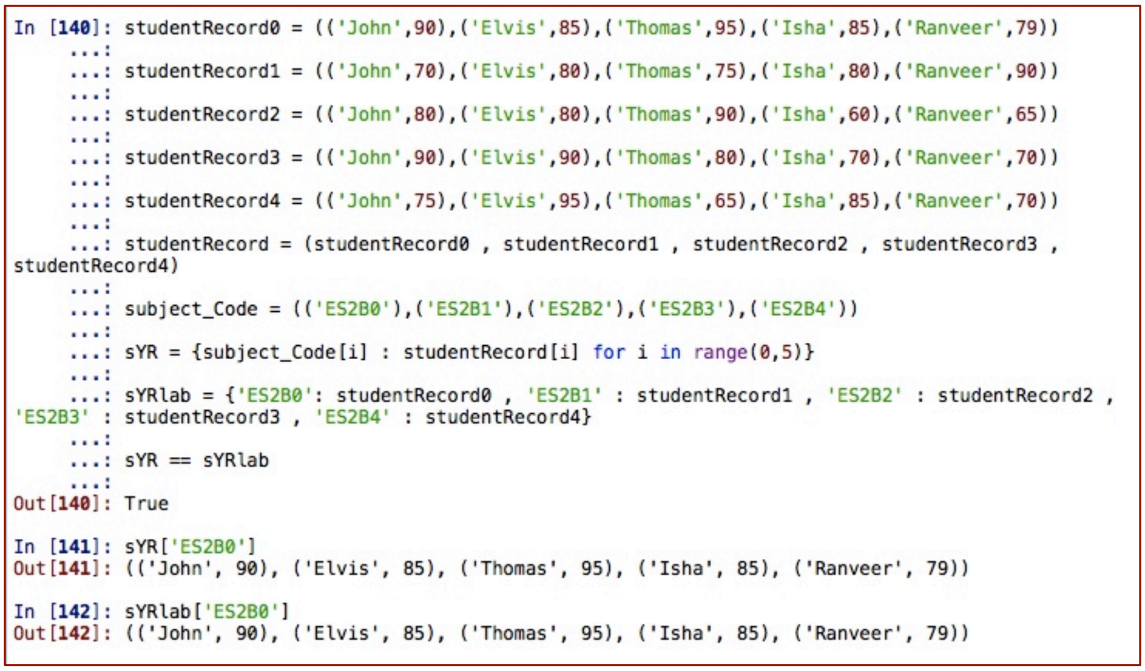
It is possible to create a list comprehension for dictionaries using the format:

```
>>> { tuple1 : tuple2 for condition}
```

Hence, I decided to make a dictionary with the ordered elements of the tuples:

```
>>> {subjectcode[i] : studentrecords[i] for i in range(0, 6)}
```

This code ensures that, each module will be matched with its corresponding set of marks, as I entered the modules and marks in order.



```
In [140]: studentRecord0 = (('John',90),('Elvis',85),('Thomas',95),('Isha',85),('Ranveer',79))
     ...:
     ...: studentRecord1 = (('John',70),('Elvis',80),('Thomas',75),('Isha',80),('Ranveer',90))
     ...:
     ...: studentRecord2 = (('John',80),('Elvis',80),('Thomas',90),('Isha',60),('Ranveer',65))
     ...:
     ...: studentRecord3 = (('John',90),('Elvis',90),('Thomas',80),('Isha',70),('Ranveer',70))
     ...:
     ...: studentRecord4 = (('John',75),('Elvis',95),('Thomas',65),('Isha',85),('Ranveer',70))
     ...:
     ...: studentRecord = (studentRecord0 , studentRecord1 , studentRecord2 , studentRecord3 ,
studentRecord4)
     ...:
     ...: subject_Code = (('ES2B0'),('ES2B1'),('ES2B2'),('ES2B3'),('ES2B4'))
     ...:
     ...: sYR = {subject_Code[i] : studentRecord[i] for i in range(0,5)}
     ...:
     ...: sYRlab = {'ES2B0': studentRecord0 , 'ES2B1' : studentRecord1 , 'ES2B2' : studentRecord2 ,
'ES2B3' : studentRecord3 , 'ES2B4' : studentRecord4}
     ...:
     ...: sYR == sYRlab
     ...:
Out[140]: True

In [141]: sYR['ES2B0']
Out[141]: (('John', 90), ('Elvis', 85), ('Thomas', 95), ('Isha', 85), ('Ranveer', 79))

In [142]: sYRlab['ES2B0']
Out[142]: (('John', 90), ('Elvis', 85), ('Thomas', 95), ('Isha', 85), ('Ranveer', 79))
```

Image 2: Exercise 2 code (A1_3_2)

As it can be seen in Image 2, when comparing both the new dictionary (sYR) and the dictionary created in the lab (sKYlab) the output indicates 'TRUE', meaning they are equal. And we can access the marks of each module using the Key (which obviously provides equal outcomes for both our dictionaries):

```
>>> sYR['ES2B0']
```

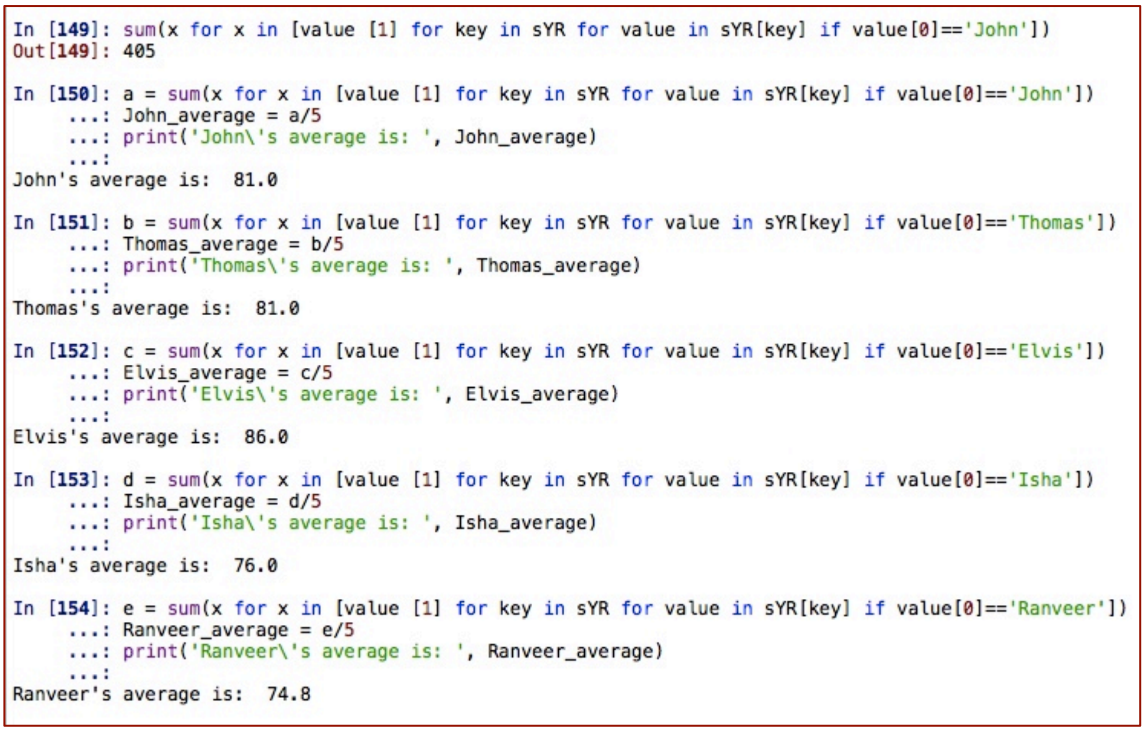
Exercise 3 –

This line of code calculates the sum of the marks that correspond to John for all five modules, using the data from our previous exercise.

```
>>>  sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0] == 'John'])
```

It is achieved thanks to the fact that this line calculates the sum of all the factors located in the second position (value[1]) for every list in all of our student year record dictionary's key (sYR) if the factor in the first position (value[0]) is equal to 'John'.

This technique can be used to calculate the average: dividing the sum by the total number of modules (shown in image 5). And can also be used to calculate the sum of marks and the average of all other students.



```
In [149]: sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='John'])
Out[149]: 405

In [150]: a = sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='John'])
     ...: John_average = a/5
     ...: print('John\'s average is: ', John_average)
     ...:
John's average is:  81.0

In [151]: b = sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='Thomas'])
     ...: Thomas_average = b/5
     ...: print('Thomas\'s average is: ', Thomas_average)
     ...:
Thomas's average is:  81.0

In [152]: c = sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='Elvis'])
     ...: Elvis_average = c/5
     ...: print('Elvis\'s average is: ', Elvis_average)
     ...:
Elvis's average is:  86.0

In [153]: d = sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='Isha'])
     ...: Isha_average = d/5
     ...: print('Isha\'s average is: ', Isha_average)
     ...:
Isha's average is:  76.0

In [154]: e = sum(x for x in [value [1] for key in sYR for value in sYR[key] if value[0]=='Ranveer'])
     ...: Ranveer_average = e/5
     ...: print('Ranveer\'s average is: ', Ranveer_average)
     ...:
Ranveer's average is:  74.8
```

Image 3: Code for exercise 3 (A1_3_3)

In this exercise, I learned that list comprehensions can take quite complex conditions, making it a very powerful tool for coding.

Exercise 4 –

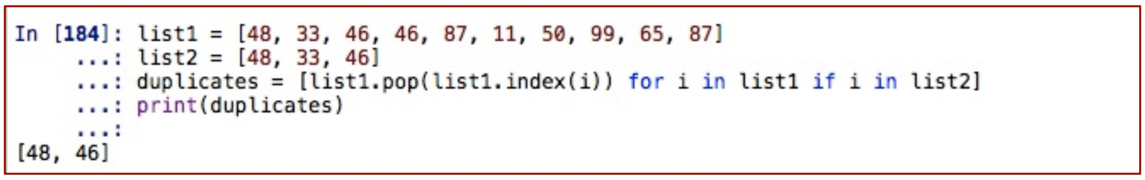
When seeing the list comprehension code introduced, it is logical to expect it to produce a list of all the elements common to both lists. Therefore, the expected outcome would be [48,33,46,46]. However, the output of the list is [48, 46].

The code introduced tries to do it by checking if the elements in each position of list1 (from the first position: position 0, to the end) are also in list2. Despite being a valid approach, the outcome is incorrect.

```
>>>     list1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
>>>     list2 = [48, 33, 46]
>>>     duplicates = [list1.pop(list1.index(i)) for i in list1 if i in list2]
```

The error is due to the fact that the tool used to create the list is '.pop()', which is a method that removes the value at a certain list index. The use of this method modifies the list and, therefore changes the expected output.

More precisely, the given code checks if the first value in list1 (48) is also in list2; and, after noting that it is in both lists, it removes it: modifying the list and moving '33' to the first position of the list. As it has already checked the first element of the list, it ignores the fact that it has changed; and checks the second value (46). And follows the exact same process. It removes it, modifying the list and ignoring what should have been the next value (46).



```
In [184]: list1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
     ...: list2 = [48, 33, 46]
     ...: duplicates = [list1.pop(list1.index(i)) for i in list1 if i in list2]
     ...: print(duplicates)
     ...:
[48, 46]
```

Image 4: Output of Code from exercise 4 (A1_3_4)

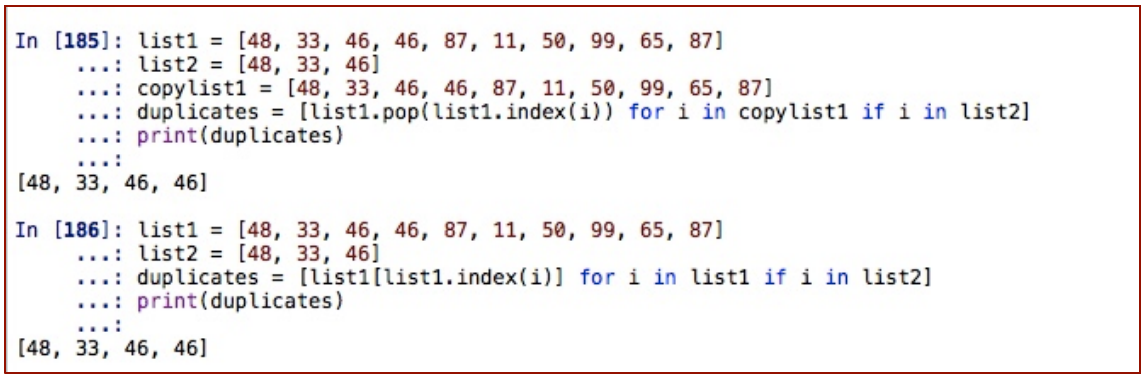
SOLUTIONS:

When creating a separate list (list1copy) equal to list1, and checking if the number is on that list and list2 (instead of checking if it is in the list that has been modified), we obtain the expected output. This is one solution to overcome our issue but changes the list, which is not desirable.

The best change to the existing code would be to change the '.pop()' tool for a less intrusive method, such as simply referring to the list's element through the format: "list1[Key]" :

```
>>> list1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
>>> list2 = [48, 33, 46]
>>> duplicates = [list1[list1.index(i)] for i in list1 if i in list2]
>>> print(duplicates)
```

The outcome and code for both solutions can be seen in image5:



```
In [185]: list1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
     ...: list2 = [48, 33, 46]
     ...: copylist1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
     ...: duplicates = [list1.pop(list1.index(i)) for i in copylist1 if i in list2]
     ...: print(duplicates)
     ...:
[48, 33, 46, 46]

In [186]: list1 = [48, 33, 46, 46, 87, 11, 50, 99, 65, 87]
     ...: list2 = [48, 33, 46]
     ...: duplicates = [list1[list1.index(i)] for i in list1 if i in list2]
     ...: print(duplicates)
     ...:
[48, 33, 46, 46]
```

Image 5: Solutions for exercise 4 (A1_3_4)

SECTION: A3. Classes – A Gibberish converter

A3.4 **Critical Thinking**

The pigLatin_converter of the base class can be invoked from within the gibberish_converter method in the inherited class by writing the following line of code inside its method (the initialization method in this case):

```
>>>class gibberish_class(pLatin_class):
>>>    def __init__(self,Sentence):
>>>            self.Sentence = Sentence
>>>    ➔       super().MethodName()    ⬅
```

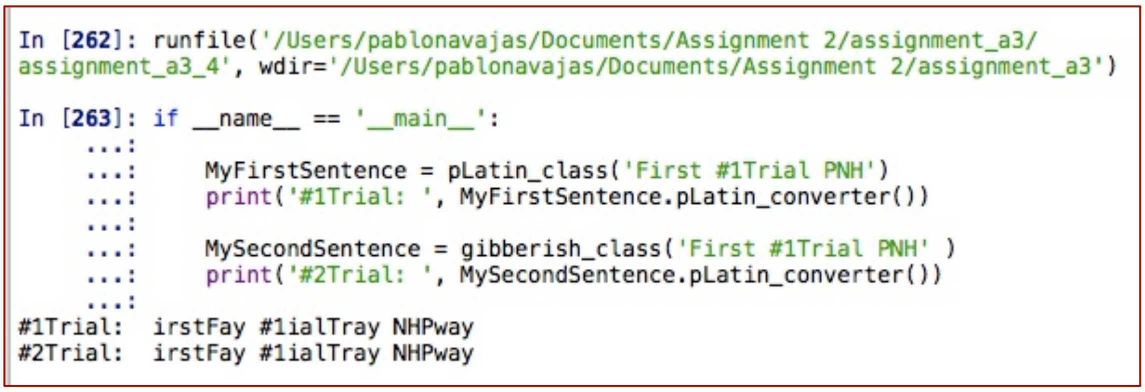
The Python 'Super()' Function is used for 'inheritance' cases. 'Inheritance' occurs when a new class is created based on an existing class. This means that the inherited class gets the methods and attributes of the base class (for the case of: class **gibberish_class**(pLatin_class) gibberish_class is the inherited class and pLatin_class the base class). The 'Super()' function allows the 'child' class to call methods from its 'parent' class using the format:

```
>>> super().MethodName()
```

In my program, I tested it by comparing the result of both methods when the same 'Sentence' input value was introduced. This was achieved using the code:

```
>>> if __name__ == '__main__':
>>>     MyFirstSentence = pLatin_class('First #1Trial PNH')
>>>     print('#1Trial: ', MyFirstSentence.pLatin_converter())
>>>     MySecondSentence = gibberish_class('First #1Trial PNH' )
>>>     print('#2Trial: ', MySecondSentence.pLatin_converter())
```

The outcome was the same (as seen in image 5), proving that the function works with both classes without having to write it twice nor having to refer to it using more complex lines of code.



```
In [262]: runfile('/Users/pablonavajas/Documents/Assignment 2/assignment_a3/
assignment_a3_4', wdir='/Users/pablonavajas/Documents/Assignment 2/assignment_a3')

In [263]: if __name__ == '__main__':
     ...:
     ...:     MyFirstSentence = pLatin_class('First #1Trial PNH')
     ...:     print('#1Trial: ', MyFirstSentence.pLatin_converter())
     ...:
     ...:     MySecondSentence = gibberish_class('First #1Trial PNH' )
     ...:     print('#2Trial: ', MySecondSentence.pLatin_converter())
     ...:
#1Trial:  irstFay #1ialTray NHPway
#2Trial:  irstFay #1ialTray NHPway
```

Image 6: Testing line of code Exercise A3_4