# LIN6209 Week 6 Assignment

Design, build and test the following functions:  
Submit your python script with the filename ‘w6\_yourID.py’.

## thermostat(temperature)

If the temperature is less that 19 turn the heat on, If it’s more than 24 turn the heat off. If it’s neither, leave the heat setting as it is. Return the string 'on', 'off' or 'stat' respectively.  
For example:

| temperature | return value |
| --- | --- |
| 21 | 'stat' |
| 16 | 'on' |
| 34 | 'off' |
| 19 | 'stat' |
| 24 | 'stat' |

**Hints:**  
A few if statements with possibly some elif and else, should get the job done.

## score\_2(dice\_1, dice\_2)

The parameters are guaranteed to be integers between 1 and 6 inclusive and represent the face values on the throw of two six-sided dice. Calculate the score according to this rule:

* The score is the sum of both dice
* unless it is a ‘double’ (both dice the same) in which case add one of the dice to the score again
* unless it is ‘double six’ (both dice six) in which case add both dice to the score again
* unless the sum of the two dice is seven in which case double it

For example:

| parameter values | return value |
| --- | --- |
| 1, 5 | 6 |
| 5, 1 | 6 |
| 3, 4 | 14 |
| 5, 2 | 14 |
| 2, 2 | 6 |
| 6, 6 | 24 |

**Hints:**  
A few if statements with possibly some elif and else, should get the job done.

## score\_4(red1, red2, blue1, blue2)

The parameters are guaranteed to be integers between 1 and 6 inclusive. They represent the face values on the throw of four six-sided dice, two coloured red and two coloured blue. Calculate the score according to the rule:

* The score is the face value of the lowest of all four dice
* Unless the sum of any one red dice and any one blue dice is seven in which case score 7
* Unless the sum of any one red dice and any one blue dice is seven, and the sum of the remaining red and blue dice is also seven, in which case score 14
* Unless the four dice are any permutation of (3,3,4,4), in which case score 21  
  For example:

| parameter values | return value |
| --- | --- |
| 1, 5, 6, 6 | 7 |
| 3, 2, 3, 3 | 2 |
| 2, 6, 1, 5 | 14 |
| 1, 1, 2, 6 | 7 |
| 1, 2, 3, 4 | 1 |
| 3, 3, 3, 4 | 7 |
| 3, 3, 4, 4 | 21 |
| 4, 3, 3, 4 | 21 |

**Hints:**  
Same again: if, elif and else. Just be careful you cover all the options and test them.

## fizzbuzz(an\_int)

The parameter is guaranteed to be a positive integer or zero. If an\_int is a multiple of 3 return 'Fizz' , if it is a multiple of 5 return 'Buzz' and if it is a multiple of 15 return 'FizzBuzz'. If it is none of those then return str(an\_int).  
For example:

| parameter value | return value |
| --- | --- |
| 1 | '1' |
| 5 | 'Buzz' |
| 33 | 'Fizz' |
| 45 | 'FizzBuzz' |
| 13 | '13' |

**Hints:**  
Same again: if, elif and else. Just be careful you cover all the options and test them.

## num\_seq\_str(an\_int)

The parameter is guaranteed to be a positive integer greater than zero. Return a string containing the integers 1 to an\_int inclusive. Successive integers are separated by a comma and a space. Note that there is no trailing comma or space at the end of the string.

| parameter value | return value |
| --- | --- |
| 1 | '1' |
| 2 | '1, 2' |
| 5 | '1, 2, 3, 4, 5' |

**Hints:**  
The first part of the problem is to generate the sequence from 1 to an\_int inclusive. There many ways to do this but two of the most straightforward are a while loop:

# a template while loop  
an\_int = 5 # this value will come from parameter  
count = 0  
while count < an\_int: # check this produces the correct sequence  
 print(count, an\_int)

Alternatively, a for loop using range()

# a template for loop  
an\_int = 5 # this value will come from parameter  
for i in range(an\_int): # check this produces the correct sequence  
 print(i, an\_int)

Choose whichever you prefer, or even something else. Once you can generate the sequence of numbers, work out how to assemble the string this function must return.  
Two useful facts: you can turn anything into a string with str(anything) and you can add strings together with +.

## num\_seq\_list(an\_int)

The parameter is guaranteed to be a positive integer greater than zero. Return a list containing the integers 1 to an\_int inclusive.

| parameter value | return value |
| --- | --- |
| 1 | [1] |
| 2 | [1, 2] |
| 5 | [1, 2, 3, 4, 5] |
| 11 | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] |

**Hints:**  
Same problem as the previous except this time you need to return a list rather than a string.  
Perhaps start by creating an empty list with the statement [] and append items to the end with the list.append().  
There are many other ways to write this function.

## num\_seq\_set(an\_int)

The parameter is guaranteed to be a positive integer greater than zero. Return a set containing the integers 1 to an\_int.  
For example:

| parameter value | return value |
| --- | --- |
| 1 | {1} |
| 2 | {1' 2} |
| 5 | {1, 2, 3, 4, 5} |

**Hints:**  
Same problem as the previous function except this time you need to return a set().

## num\_seq\_dict\_(an\_int)

The parameter is guaranteed to be a positive integer greater than zero. Return a dictionary containing the integers 1 to an\_int as the keys of the dictionary. The value associated with each key should be the string returned by fizzbuzz(key).  
For example:

| parameter value | return value |
| --- | --- |
| 1 | {1: '1'} |
| 2 | {1: '1', 2: '2'} |
| 3 | {1: '1', 2: '2', 3: 'Fizz'} |
| 5 | {1: '1', 2: '2', 3: 'Fizz', 4: '4', 5: 'Buzz'} |

**Hints:**  
Same problem as the previous function except this time you need to return a dictionary rather than a string. This has the added complexity that every entry in a dictionary has two parts, a ‘key’ and an associated ‘value’.

One approach is to start by creating the dictionary with the correct keys but all with just a single default value. Then iterate through the dictionary, key-by-key, setting the ‘value’ part to the correct value.

You will find the dictionary methods you need in the [Python dictionary documentation](https://docs.python.org/3/library/stdtypes.html#typesmapping)

## get\_chars\_set(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return the set of all characters present in the file.  
For example:

| file content | return value |
| --- | --- |
| '' | {} |
| 'abc 123' | {'a', 'b', 'c', '1', '2', ' ', '3'} |
| '~abc+cab~' | {'a', 'b', 'c', '~', '+'} |

**Hints:**  
Read the file into a single string then convert that to a set.

filename = 'the name of the file' # this is the parameter value  
with open(filename, 'r', encoding='utf-8') as f: # open the file for read  
 content = f.read() # read the file  
 print(content) # your code replaces this

## count\_chars(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return number of characters in the file. There is no need to be concerned as to what constitutes a ‘character’. Count everything.  
For example:

| file content | return value |
| --- | --- |
| '' | 0 |
| 'abc 123' | 7 |
| '-abc-123-' | 9 |

**Hints:**  
Similar to the previous function. Read the file and count the characters in that.

## count\_Q(filename);

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and count the number of occurrences of the character ‘?’ (a question mark, unicode 63 decimal or 3F hexadecimal).

| file content | return value |
| --- | --- |
| question | 0 |
| ?-? | 2 |
| any???thing | 3 |

**Hints:**  
Open the file with a with open(): statement making sure to specify the encoding as encoding='utf-8'. Read the whole file into a text string and then use [string methods](https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str) to get the counts.

## count\_and(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and count the number of occurrences of each of the character strings 'and', ' and ', and ' and, '(the string ‘and’; same but with spaces both sides; same again but with a space before and a comma and a space after). Ignore the case of the text. Return the respective counts as integers.

| file content | return value |
| --- | --- |
| '' | 0, 0, 0 |
| 'and and and, ' | 3, 1, 1 |
| 'Andover and shandy' | 3, 1, 0 |

**Hints:**  
Much the same as count\_Q(). To return more than one value at the same time just separate them with commas in the return statement like this: return a,b,c,d

## count\_words(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return number of words in the file.  
For example:

| file content | return value |
| --- | --- |
| '' | 0 |
| 'abc 123' | 2 |
| 'abc 123 \n\t xyz' | 3 |

**Hints:**  
Do not make the assumption that words are separated only by spaces as other types of ‘whitespace’ characters are often used to separate words. These other ‘whitespace’ characters include: tab '\t', newline '\n', carriage return '\cr', and more.  
There is a ready-made string method that identifies whitespace characters, find it and use it.

## char\_frequency(filename)

Return a dictionary with every character in filename as a key with a value equal to the number of its occurrences with filename.

**Hints:**  
Use the same reasoning as for your function count\_chars(filename).

## word\_frequency(filename)

Return a dictionary with every word in filename as a key and the associated value the number of its occurrences with filename.

**Hints:**  
Use your function count\_words(filename) for each word you put in the dictionary.

## average\_word\_length(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return the average length of the words in the file. The average length is calculated as the (sum of the lengths of every word) / (number of words).  
For example:

| file content | return value |
| --- | --- |
| '' | 0 |
| 'abd def xyz' | 3 |
| 'a bc def ' | 2 |
| 'Andover and shandy' | 5.333333333333333 |

## text\_analysis\_01(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return a tuple containing:

* Total number of sentences
* Total number of words
* Total number of non-whitespace characters
* Total number of whitespace characters

Assume that:

* All sentences are terminated by a period (full stop) followed by a space except for the last sentence in the file which might be terminated by the end of the file or just a period mark.
* The file contents are entirely prose-like text. There are no tables, diagrams or any other complexities of any kind.

**Hints:**  
This is just an extension of the counting characters and counting words problems.

## write\_Q(an\_int, filename)

The parameters are guaranteed to be a positive integer **greater than zero** and a legal filename. Write the sequence of integers from 1 to an\_int inclusive to a text file, with each integer separated from the next by the string ’ ? ’ (space, question mark, space). The return value should be the count of the total number of characters written to the file.  
For example:

| parameter values | file content | return value |
| --- | --- | --- |
| 1, 'file.txt' | 1 | 1 |
| 3, 'file.txt' | 1 ? 2 ? 3 | 9 |
| 5, 'file.txt' | 1 ? 2 ? 3 ? 4 ? 5 | 17 |

**Hints:**  
A for or while loop to generate the sequence of integers, and add the ‘space-?-space’ while iterating.

## write\_ints(an\_int, filename)

The parameters are guaranteed to be a positive integer **or zero** and a legal filename.

Create the file if it does not already exist and write an\_int lines to the file.  
The successive lines of the file each start with the next integer in the ascending sequence from 1 to an\_int followed by the sequence of integer multiples of next\_integer up to and including next\_integer \* next\_integer.  
All the integers in the same line are separated by a single space. No space is added at the beginning or the end of the line.  
For example:

| parameter value | file content |
| --- | --- |
| 0 | empty |
| 1 | 1 |
| 2 | 12 4 |
| 5 | 12 43 6 94 6 12 165 10 15 20 25 |

**Hints:**  
This feels like a loop within a loop.  
A for or while loop to generate the sequence of integers from 1 to an\_int, and then another for or while loop within that to generate the ascending sequence from next\_integer to next\_integer \* next\_integer.

# Section 2 - Extra Credit

These last questions are more difficult and are not part of the assignment.  
If you attempt them I will mark your work and, providing you have completed section 1 to a good standard, I will award extra credit

## adjacency(filename)

The parameter is guaranteed to be the name of a text file encoded as utf-8. Read the file and return a dictionary containing every word as a key.  
Associated with each key are two lists:

* The words that have occurred immediately **before** the key word in filename
* The words that have occurred immediately **after** the key word in filename

## predict\_next(word, filename)

The parameter word is a string with no punctuation or spaces. The filename is guaranteed to be the name of a text file encoded as utf-8. Read the file and provided word occurs within it, return the word most likely to follow word based upon the corpus of text within filename. If word is not present in filename return None.

In this simple case you could just read the next word in filename to ‘predict’ it with 100% accuracy. The purpose of this exercise is of course more general than that. Imagine that filename is a collection of all the known writings of Arthur Conan Doyle. If we are presented with a new fragment of text purported to have been written by ACD we could use this function to gain a primitive measure of how likely it is that it really was written by ACD.

## pluralizer(a\_str)

The parameter is guaranteed to be a word. Return the plural of that word.

**Test Data:** Banana, Leaf, Trolley, Lorry, Sheep, Church, Sausage, Monkey, Knife, Child, Match, Poppy **Hints:** Given the variability and irregular structure of natural language there is no perfect version of this function and a balance needs to be struck between coverage and the ever-increasing complexity of the code.