# Assignment 1: Python practice

Re-submit Assignment

**Due** Jan 29 by 11:59pm **Points** 100 **Submitting** a file upload
**File Types** py and ipynb

For this assignment, you may use the file Assignment_1_Template.ipynb as a template. **When submitting the assignment, please change the file name to match the following format:**

**YOUR_NAME_Assignment_1.ipynb (or .py if you're not using Jupyter)**

**For example, if your name is Sarah Jane Smith, then please name the file Sarah_Jane_Smith_Assignment_1.ipynb**

## Problem 1: Message Coder

For parts 1 and 2 you will need to use the functions **ord** and **chr**.

**ord** takes a character as an argument and returns its ASCII integer equivalent. For example:

**ord('a')** returns 97

**ord('A')** returns 65

**chr** takes an integer as an argument and returns the ASCII character equivalent to it. For example:

**chr(97)** returns 'a'

**chr(65)** returns 'A'

Note also that the entire character set has an ordering defined on it. So, the characters a to z are in increasing order (a < b < … <y < z) and so are the characters A to Z (A < B < … < Y < Z).

### Part 1:

Write a function **preceding** that takes a single character as an input and returns a code for that character. The coding scheme is as follows:

1. if the character is ‘a’, then the code for the character is ‘z’
2. if the character is ‘A’, then the code for the character is ‘Z’
3. if the character is between ‘b’ and ‘z’, then the code for the character is the letter that precedes it. For example, the code for ‘b’ is ‘a’, the code for ‘c’ is ‘b’ and the code for ‘z’ is ‘y’
4. if the character is between ‘B’ and ‘Z’, then the code for the character is the letter that precedes it. For example, the code for ‘B’ is ‘A’, the code for ‘C’ is ‘B’ and the code for ‘Z’ is ‘Y’
5. if the character is anything else, then the code for the character is the character itself. For example, the code for ‘!’ is ‘!’, the code for ‘ ‘ is ‘ ‘ (space is space), etc.

**preceding(‘a’)** returns ‘z’

**x=‘!’**

**preceding(x)** returns ‘!’

## part 2:

Write a function **succeeding** that takes a single character as an input and returns a code for that character. The coding scheme is as follows:

1. if the character is ‘z’, then the code for the character is ‘a’
2. if the character is ‘Z’, then the code for the character is ‘A’
3. if the character is between ‘A’ and ‘Y’, then the code for the character is the letter that succeeds it. For example, the code for ‘A’ is ‘B’, the code for ‘B’ is ‘C’ and the code for ‘Y’ is ‘Z’
4. if the character is between ‘a’ and ‘y’, then the code for the character is the letter that succeeds it. For example, the code for ‘a’ is ‘b’, the code for ‘b’ is ‘c’ and the code for ‘y’ is ‘z’
5. if the character is anything else, then the code for the character is the character itself. For example, the code for ‘!’ is ‘!’, the code for ‘ ‘ is ‘ ‘ (space is space), etc.

**succeeding(‘a’)** returns ‘b’

**x=‘!’**

**succeeding(x)** returns ‘!’

## part 3:

Write a function **message_coder** that takes two arguments - a string and a function - and returns a coded message that uses the function for coding each character in the string.

For example:

message_coder('hello dolly zebra!’,preceding) returns 'gdkkn cnkkx ydaqz!’

message_coder('hello dolly zebra!’,succeeding) returns 'ifmmp epmmz afcsb!’

## Problem 2

Write a function **word_distribution** that takes as inputs a string and an optional word_list as arguments and returns a dictionary containing the frequency of each word in the text, or of each word in the word_list if it is included in the function call. For example if the argument to the function is:

text_string = “Hello. How are you? Please say hello if you don’t love me!”

print(word_distribution(text_string)) should print

{‘hello’: 2, ‘how’:1, ‘are’:1, ‘you’:2,’please’:1, “don’t”: 1 …}

If the word_list is provided then the function should return the count of words in the word_list and ignore all other words in the text. word_list is the set of words that should be counted. (The value for word_list should default to None.)

text_string = ‘I came, I saw, I conquered!’

word_distribution(text_string,word_list=[‘I’,’saw’,’Britain’]) returns

{‘i’:3,’saw’:1,’britain':0}

Make sure that you exclude all punctuation from your count and that your count ignores case. Thus, Hello and hello are the same word in the above example. The . ? and ! are all ignored. But the apostrophe in don’t doesn’t count and don’t is treated as a single word.

**Notes**:

1. word.split() splits a string on spaces and returns a list. Note that this will concatenate punctuation with words. For the above example, you will get:

[‘Hello.’,’How’,’are’,’you?’,…]

1. word = word.lower() converts the string into lower case.

1. You can assume that all punctuation occurs at the end of the word and, for our purposes, assume there is only one punctuation mark at the end of a word. (Try, optionally, extending your program to account for more than one punctuation mark at the end of a word. What would you do if x=‘“Go home!”, John shouted’?)
2. To eliminate punctuation, note that every word (given our assumption above) can potentially have a punctuation mark as the last character. The last character in a word is word[-1]. So, check if word[-1] is not between ‘A’ and ‘Z’ and not between ‘a’ and ‘z’. If it is not, then set word to word[:-1].
3. create an empty dictionary at the top of the function. Then, iterate through the list of words. Check if the word is in the dictionary (the_dictionary.get(word) will return None if it isn’t). Create a new entry if it isn’t and initialize that entry’s value to 1. If it is already in the dictionary, increment the value by 1
4. The dictionary returned by the function should contain all words in word_list whether they do or do not occur in the text_string argument (when word_list is not None).