The following problem has a template ready for you to get started on Github. Do not forget to adjust the README to indicate you have completed the assignment before your final commit! There are no written problems this week, so you won't need to scan anything to turn submit.

Get Assignment link: `https://classroom.github.com/a/EEuXq7E0`

1. Writing, reading, and receiving secret messages is a fond memory of youth for many, but encryption and secret messages are still very much at play in the modern world. To be clear about some of the jargon, here is some quick vocabulary:

   Encryption:
   The process of obscuring or encoding messages to make them unreadable.

   Decryption:
   The process of making encrypted messaged readable again by decoding or unobscuring them

   Cipher:
   An algorithm for performing encryption and decryption

   Plaintext:
   The unencrypted message

   Ciphertext:
   The encrypted message

   Here you will investigate and implement the classic Caesar Cipher, one of the earlier and simpler known ciphers.

   The idea behind the Caesar Cipher is to shift every letter a fixed integer. For instance, suppose we shift by an integer value k. Then what was formerly the ith letter of the alphabet gets substituted with the $i+k$th letter of the alphabet. Should $i+k$ be greater than 26, we should wrap back around and start at 0 again.

   Here we will treat lowercase and uppercase letters individually, so that if, for instance, "a" maps to "j", then so too does "A" map to "J". Punctuation and spaces will remain as they are, unchanged. Only actual letters get shifted! Some example are below:

plaintext	shift	ciphertext
`"abcdef"`	2	`"cdefgh"`
`'Hello, World!'`	4	`'Lipps, Asvph!'`

   To implement the Caesar Cipher we will be using 3 classes: a parent `Message` class and then two subclasses: `PlainMsg` and `CipherMsg`. The `Message` class will contain methods which can be used to apply a cipher to a string, either encrypting or decrypting the message. `PlainMsg` will have methods to encode the string using a specific shift value, and the class will always create an encoded version of the string with the latest values. `CipherMsg` will contain methods to be used to brute-force decode an encrypted string.

You are tasked with filling out the methods in the provided template. I'll go over the general objectives and requirements below, but you should consult the docstrings in the template for more information.

(a) Fill out the below methods in the `Message` class to obtain the desired functionality. The first few should be pretty straightforward initialization and getter/setter methods, while the last two are actually implementing the basics of the Caesar Cipher.

- `__init__(self, text)`
- `__repr__(self)`
- The getter method: `get_message_text(self)`
- A setter method: `change_message_text(self)`
- `build_shift_dict(self, shift)`
  - An easy way to setup the substitution is to construct a dictionary mapping each letter to another (shifted) letter. Looping through a string of letters may be useful in creating this, in which case you can type out your own or consider importing `string` and using that module's `ascii_lowercase` or `ascii_uppercase` constants. Remember that your dictionary should include both lower and uppercase mappings!
- `apply_shift(self, shift)`
  - This method would use the dictionary created with `build_shift_dict` to create and return a new string which has had all its letter characters shifted. Punctuation and spaces should stay as they are.

Some examples of creating and using the `Message` class are included in the docstring. Make sure your code is behaving properly!

(b) The `PlainMsg` class is a child of `Message` and thus everything you've written above will still be applicable. However, we will be tweaking a few things to make `PlainMsg` better suited toward taking an unencrypted message and easily and flexibly converting it to an encrypted message. Methods you will need to override/add are:

- `__init__(self, text, shift)`
  - You can initialize from the parent class here to save a little typing, but will still need to add a new `shift` data attribute. Two other data attributes should be added here: `self.shift_dict` and `self.encrypted_msg`, both of which can be calculated from just the `message_text` and the `shift`.
- `get_shift`
- `change_shift`
  - You should make sure that you update the content of any data attribute that depends on the value of `shift` after you change it!

- `get_encrypted_msg`
  - This is just a getter method to return the encrypted message data attribute.

Again, examples of using `PlainMsg` are included in the docstring. Ensure that things are working properly before going on!

(c) Encrypting messages is fun, but decrypting secret messages is where the real power of computers comes in to play. For the Caesar Cipher, there are only 26 possible shifts that can take place to try to obfuscate the original message. A computer can easily try decrypting a message with each of these possibilities, and then return the one that results in the "best" output. What is best in this case? Generally the output that results in the greatest number of known words in the decrypted text! Here you'll just need to fill out 2 methods.

- `__init__(self, text)`
  - You can use the parent class constructor again to simplify this, but you should also add a few other data attributes:
    - `self.is_decrypted` which would default to `False`
    - `self.best_shift` which initially is `None` and would get updated when you decrypt the message
    - `self.decoded_msg` which is also initially `None` and will get updated when the decryption takes place
- `decrypt_msg(self)`
  - Here you'll need to loop through all the possible shifts, decode the message in each, and keep track of which result in the best. To do so, you'll need some functions to convert the string of words to a list of words which you can loop over and check to see how many are in the list of valid words. I have provided the code to load in the list of valid words, but you'll need to flesh out the `get_words` and `count_valid_words` functions (not methods) if you want to use them.

(d) I have included a small snippet from one of my favorite childhood books which has been encoded. The function `read_story` will read in this text file as a string (which does have new-line characters in it, but these will not effect your decoding). Write a function `decode_story` which finds the best shift value to decode the story, and then writes the decoded story to the file `decoded_story.txt`.