

Software Project 1 Specifications

Topic Coverage: Weeks 1 to 5

Guidelines

- This project is meant to be done **individually**. You are expected to develop a solution to the problem independently. Your code will be run through a software similarity test and a plagiarism checker. Students who submit substantially similar work, copied code snippets from external sources, or otherwise receive outside assistance will be subject to investigation for intellectual dishonesty (refer to the course guide).
- Properly **INDENT** and **LABEL** your code. Points will be deducted for improper indentation and labelling.
- **DO NOT** use external libraries, unless specifically required in the following specifications.
- **DO NOT** use global variables, all variables must be properly scoped within their own functions.
- Define a main function as the entrypoint of your code. ([Defining Main Functions in Python – Real Python](#)).
- Submit your .py files in the provided UVLe submission bins on or before 11:55 pm of Nov 15, 2021.
 - Your final submission should be contained in a single .py file with the filename format: “main_<student_no>.py”. For example: “main_202012345.py”
 - Note that while this is the deadline for the submission of your final code, you will still have to schedule a live demo with your instructor. (See Part 5b below.)

A live class session will be held on the week of the release to explain the requirements and demonstrate a working version of this Software Project.

Super Secure Steganography

“Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program, or protocol. Media files are ideal for steganographic transmission because of their large size.”

(Wikipedia, <https://en.wikipedia.org/wiki/Steganography>)

Take a look at the images presented in Figure 1. From visual inspection, do you notice any difference between the two photos? Probably not. You shouldn't notice any distinct difference visible between the two images, well, not unless you have super vision capable of isolating each pixel and distinguishing minute color differences (which you probably don't). But even though we don't see it, the two images are actually almost completely different! If you don't believe that the two images are different, you can try to run the [two images](#) through this [image comparison website](#).



(a) Original image



(b) Modified Image

Figure 1. Example of Image Steganography

You should find that the two images, while not having any visually discernible difference, are actually vastly different from each other. With a Fuzzy setting of 0% (all differences highlighted in red), the image will actually look like the image in Figure 2. Isn't that amazing? You can be caught completely unaware that there was anything wrong with the image if you didn't already know that it had been modified beforehand. In fact, this is something that is sometimes used by hackers with ill-intent as a way to attach extra payloads (ransomware, viruses, etc.) into files or documents that users will then download onto their device (This is why you should be very careful when downloading files from the internet!). By using something as simple as image steganography, they are able to introduce files that contain possibly malicious data into unsuspecting computer systems, unleashing utter chaos!

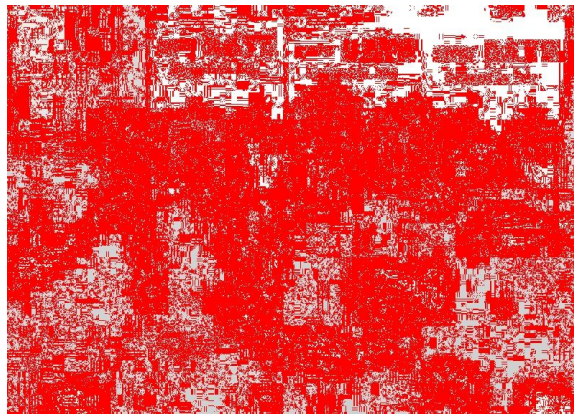


Figure 2. Pixel difference between Original and Modified Image

Now, while you won't be learning how to take over computer systems (that's a very advanced topic), you will be introduced to some basic concepts in computer security. For this software project, you will learn how information can be protected through encryption and steganography. You will be developing a program capable of hiding and finding a short encrypted message inside a given image. The following sections will describe the specific requirements of this software project.

Part 1: Program Mode

The first thing you will need to do when creating any program is to provide the user with an interface to guide them in using the program. For this project, your program should show a menu containing the following items upon execution: (1) encrypt, (2) decrypt, and (3) exit. If the user enters an option aside from the three menu items above, the program should show the message 'Invalid input, choose a different item!' and prompt the user for a new response.

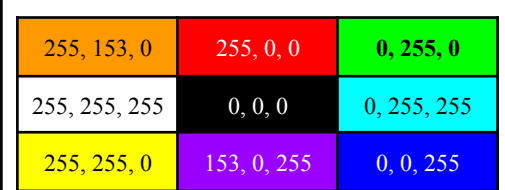
If the user enters 'encrypt' or 'decrypt', the program should proceed with asking the user to type in the filename of the image to be used. If the filename points to a file that either doesn't exist or isn't the right format (not a JPEG file: .jpg or .jpeg), the program shows the message 'Invalid image file' and will repeatedly prompt the user to enter a new filename until a valid file is found. Afterwards, the program will fetch the image data, as detailed in **Part 2**, and will then proceed with performing the necessary actions depending on the selected menu item. If the user selects 'encrypt', the program will execute **Part 3**, embedding an encrypted message into the given image data to create a new image file. If the user selects 'decrypt', the program will execute **Part 4**, processing the given image data and looking for possible embedded messages.

After performing the tasks for the selected menu items, your program should prompt the user to choose from the menu again to perform another action. The program will only stop showing the menu once the user types in 'exit'

as the menu input, prompting the program to show the message ‘Thank you for using this program!’ and exit altogether.

Part 2: Loading and Saving Image Data

After receiving a valid JPG filename, you will have to open the image file and load its contents for use. To do so, create a function `load_image_data()` that requires the string *filename* as an argument and returns the image’s pixel data. The function returns a tuple containing (1) the image dimensions in pixels: (w, h) and (2) the image data in the form of a list of tuples with RGB (red, green, blue) values. Refer to the example below:

 <p style="text-align: center;">image_3x3_pixels.jpg</p>	<p>To fetch the data from image_3x3_pixels.jpg: <code>size, image_data = load_image_data("image_3x3_pixels.jpg")</code></p> <p>The values of <code>size</code> and <code>image_data</code>: <code>size = (3, 3)</code> <code>image_data=[(255, 153, 0), (255, 0, 0), (0, 255, 0),</code> <code>(255, 255, 255), (0, 0, 0), (0, 255, 255),</code> <code>(255, 255, 0), (153, 0, 255), (0, 0, 255)]</code></p>
<p style="text-align: center;">*you can assume that the test images will not have an alpha layer.</p>	

While both the encryption and decryption modes of the program require access to the original image data to work, only the encryption mode will need to save a new image (since the encryption mode creates an image with a hidden message). To help with this action, create a function `save_image_to_file()` that accepts *filename*, *image_dimension*, and *image_data* as arguments. The argument *filename* is of type string, the argument *image_dimension* should be a tuple containing the width and height of the image in pixels: (w, h), and the argument *image_data* should contain a list of tuples with RGB values, similar to the data returned by the `load_image_data()` function.

Note: if you created the Software Project Building Block for Week 3 Part 2, you can utilize your code for that to satisfy the requirements of this part of the Software Project.

Part 3: Creating a Steganographic Image

If the user selects ‘*encrypt*’ mode, the program will prompt the user to enter the *secret_key* (‘Enter Key:’) and the *message* (‘Enter Message:’) to be encrypted. To do so, create the function `get_data_to_encrypt()` that accepts the argument *image_size* and returns the tuple: (*secret_key*, *message*) once the function determines that the following constraints for the variables have been met:

<i>secret_key</i>	Contains only ‘u’ or ‘d’ characters. Length should be at least 3 and at most 20 characters.
<i>message</i>	Contain characters from the ASCII table with values between 32 and 126. Length should be at least 10 and at most 1000 characters.

If the above constraints are not met, the program should show the message ‘Invalid Key/Message. Please Try again.’ and prompt the user to re-enter both *secret_key* and *message*, this will repeat until a valid *secret_key* and *message* is entered. Aside from this, the function should also check whether the *secret_key* and *message* will fit in the given image. To ensure this, you should make sure that the total pixel count if the image is at least six plus one-third the number of bits needed to represent the key and message. This will further be expounded upon when the key and message are encoded into the image. If the image cannot accommodate the entire *secret_key* and *message*, the program

should show the message ‘Message and Key cannot fit in the image.’ and prompt the user to re-enter both *secret_key* and *message*, this will repeat until a valid *secret_key* and *message* is entered. For example, the *secret_key* ‘udd’ and the *message* ‘Hello World’, there are a total of 14 characters or 112 bits ($14 \text{ char} * 8 \text{ bits/char}$), so to accommodate the *secret_key* and *message*, the image should have at least $6 \text{ pixels} + (112 \text{ bits}/(3 \text{ bits/pixel})) = 43.33 \approx 44 \text{ pixels}$ (always round up).

3.1 Encrypting the Message using a given Key

Once a valid *secret_key* and *message* is entered, the next step will be to encrypt the message using the given key. To do so, create the function `encrypt_text()` that accepts the arguments *text* and *key* and applies a modified version of Caesar’s Cipher and returns the encrypted text with the following changes: The cipher will use the *key* to determine the direction of the shift, with a ‘u’ indicating that the character must be shifted upwards while a ‘d’ indicating a downward shift. The number of characters in the *key* will determine the number of positions to shift. If the *text* has more characters than the *key*, the *key* will loop through again to accommodate the additional *text* characters. For example, with the *key* ‘udd’ and *text* ‘Hello World’, the *key* indicates that the *text* needs to be shifted three places, since it has three characters. Since the *text* has more than 3 characters, the *key* will loop through again to accommodate the additional characters. Performing the shifts accordingly will result in the encrypted word ‘Kbiol|Zlooa’, as seen below.

<i>text</i>	H	e	l	l	o		W	o	r	l	d
<i>key</i>	u	d	d	u	d	d	u	d	d	u	d
<i>encrypted</i>	K	b	i	o	l		Z	l	o	o	a

Note: If this looks familiar to you, it’s because it is! This is the building block that was released on Week 4, so if you made a submission for that, you can reuse your code for this part of the software project! **For the Software Project 1 version of this function, you can safely assume that the arguments are compliant with the given constraints.**

3.2 Translate Key and Message from String to Binary

To prepare the *secret_key* and the encrypted *message* for encoding into image data, we convert both into their binary form. To do so, create the function `char_to_ascii()` that accepts the argument *word* and converts each character in the *word* into its integer counterpart (ASCII decimal value) and returns them in the list *ascii_values*. Afterwards, create the function `ascii_to_binary()` that accepts the argument *ascii_values* and converts the given list into strings containing their binary values and returns the converted values in the list *binary_values*. As an example, the key ‘udd’ and the encrypted message ‘Kbiol|Zlooa’, when run through the two functions, will result in the following:

word	ascii_values	binary_values
‘udd’	[117, 100, 100]	[‘01110101’, ‘01100100’, ‘01100100’]
‘Kbiol Zlooa’	[75, 98, 105, 111, 108, 124, 90, 108, 111, 111, 97]	[‘01001011’, ‘01100010’, ‘01101001’, ‘01101111’, ‘01101100’, ‘01111100’, ‘01011010’, ‘01101100’, ‘01101111’, ‘01101111’, ‘01100001’]

Note: If this looks familiar to you, it’s because it is! This is the building block that was released on Week 2, so if you made a submission for that, you can reuse your code for this part of the software project! **Make sure that the resulting binary strings has 8 characters, as this was not required during the weekly activities and the leading zeroes will be important later on.**

3.3 Adding the Key and Message Binary to the Image

To encode the key and encrypted message into the image data, we will use the Least Significant Bit (LSB) Steganography. To do so, create a function `encode_message()` that accepts three arguments: (1) *image_data*, (2) *binary_key*, and (3) *binary_encrypted_message*. The function will then replace the least significant bit of each pixel's R, G, and B values with the key and encrypted message binaries. The order of encoding will be as follows:

<i>binary_key</i>	Delimiter	<i>binary_encrypted_message</i>	Delimiter
'udd'	0xFF	'Kbiol Zlooa'	0xFF
['01110101', '01100100', '01100100']	'11111111'	['01001011', '01100010', '01101001', '01101111', '01101100', '01111100', '01011010', '01101100', '01101111', '01101111', '01100001']	'11111111'

The function first encodes the *binary_key* before moving onto the *encrypted_message*. You should notice that a full byte (0b11111111 or 0xFF) is used to punctuate each part of the encoded data. The full byte serves as an indicator so the program knows where each part of the encoded data ends. This is also the reason why during your computation for the total pixels needed to accommodate the key and message, 6 pixels were added to the minimum required pixels (6 pixels x 3 bits/pixel = 18 available bits for the delimiter, which takes up 2 delimiter * 8 bits/delimiter = 16 bits). After encoding the data into the image, the function will return the modified *image_data*.

As an example, let's say we have the key: 'u' and the encrypted message 'K'. The following table shows how the data is encoded into the given 3x4 pixels image data. You will notice that the resulting image data pixel values have only been modified by either +1 or -1, depending on the value of the bit being encoded. While there are changes in the values, the difference is small enough to be negligible when presented as pixels with color to the human eye (This is why you couldn't find a difference in the two images presented in Fig. 1).

[(225, 12, 99), (155, 2, 50), (99, 51, 15), (15, 55, 22), (155, 61, 87), (63, 30, 17), (1, 55, 19), (99, 81, 66), (219, 77, 91), (69, 39, 50), (18, 200, 33), (25, 54, 190)]	+	'u' 0xFF 'K' 0xFF	['01110101'] '11111111' ['01001011'] '11111111'	=	[(224, 13, 99), (155, 2, 51), (98, 51, 15), (15, 55, 23), (155, 61, 87), (63, 30, 17), (0, 54, 19), (98, 81, 67), (219, 77, 91), (69, 39, 51), (19, 201, 33), (25, 54, 190)]
Original Image Data		Data to Encode			Modified Image Data

Once the *modified_image_data* is obtained, the program will call the `save_image_to_file()` to save the image with the original filename prepended by 'modified_'. For example, the file 'image_3x3_pixels.jpg', once encoded with data, will be saved as 'modified_image_3x3_pixels.jpg'. The file should be saved in the folder "output"; you may assume that the folder already exists in the present working directory.

Part 4: Getting the Message from Steganographic Images

If the user selects 'decrypt' mode, the program will proceed with processing the image data loaded from the provided *filename* in part 2 above. The following sections detail the steps you need to take to create the image decryption part of your program. If at any point during the decoding process the program finds that the image cannot be decoded, the program should show the message 'Error: cannot decode message!' and go back to the program menu.

4.1 Extract Key and Message Binary from Image Data

To decode the key and encrypted message from the image data, we will reverse the encoding performed in Part 3, that is, we will extract the Least Significant Bit (LSB) of each pixel in the image data. To do so, create a

function `decode_message()` that accepts the *image_data* as argument and returns the tuple: *binary_key*, and *binary_encrypted_message*. If the image data does not contain a key and/or message (i.e. the delimiters weren't found), the function returns the tuple (*None*, *None*). As an example, with the following image data, the function will return the tuple: (`['01110101']`, `['01001011', '01100010']`).

[(224, 13, 99), (155, 2, 51), (98, 51, 15), (15, 55, 23), (155, 61, 87), (63, 30, 17), (0, 54, 19), (98, 81, 67), (218, 77, 91), (68, 38, 50), (19, 200, 33), (25, 55, 191), (21, 31, 201), (33, 32, 32), (32, 32, 32)]	=	key 0xFF encrypted _message 0xFF	['01110101'] ['1111111'] ['01001011'] ['01100010'] ['1111111']
Image Data		Extracted Binary Data	

4.2 Translate Key and Message from Binary to String

To translate the binary data extracted from the image, we have to convert the binary information into ASCII characters. To do so, first you need to create the function `binary_to_ascii_string()` that accepts a list of binaries, *binary_values*, and returns them as a string. As an example, the list `['01001011', '01100010', '01101001', '01101111', '01101100', '01111100', '01011010', '01101100', '01101111', '01101111', '01100001']` will return the string 'KbiolZlooa' and the list `['01110101', '01100100', '01100100']` will return the string 'udd'.

4.3 Decrypt the Message using a Given Key

Once key and message strings have been extracted from the image, the next step is to decrypt the message using the key. To do so, create the function `decrypt_text()` that accepts the arguments *encrypted_text* and *key* and returns the *decrypted_text* by applying the inverse of the modified Caesar cipher described in Part 3 onto the *encrypted_text*. As an example, if we decrypt the *encrypted_text* 'KbiolZlooa' with the *key* 'udd', we will get the decrypted message 'Hello World'.

Note: If this looks familiar to you, it's because it is! This is the building block that was released on Week 5, so if you made a submission for that, you can reuse your code for this part of the software project!

4.4 Save Decrypted Message to Text File

Finally, once you've decoded the message from the image, the final step is to save the message. To do so, create a function `save_file()` that accepts the arguments *filename* and *text* and saves the *text* into the file *filename*. The *filename* should use the name of the image you decoded with '_decoded_message' appended to it (i.e. for the image *text.jpg*, the save file should have the name *text_decoded_message.txt*). The file should be saved in the folder "output"; you may assume that the folder already exists in the present working directory.

Note: if you created the Software Project Building Block for Week 3 Part 1, you can utilize your code for that to satisfy the requirements of this part of the Software Project.

Part 5: Live Demonstration

Aside from the challenge submissions made available through *HackerRank*, you will be required to attend a **live demonstration** session with your instructor (scheduling will be set by your instructor), which will also serve as a test of your understanding of the problem and the solution you submitted. Expect that new test cases will be given by the instructor on the day of the live demonstration for testing and validation of your work.

Grading Scheme

The breakdown of grades for this software Project can be found in the table below. Note that while we have separated the software project into numbered parts, grading for each part will still be independent of each other, meaning, you will still be able to get the full points for a later part of the project even if you fail to complete an earlier part as long as you are able to successfully meet the requirements for the part.

***Note:** If you created the building blocks from the weekly optional activities, the grades you received for the building block activities will be counted towards the corresponding part of your final grade.*

Rubrik		Building Block	Total
Part 1	Program Menu	-	15%
Part 2	Loading and Saving Image Data	Week 3 Part 2	5%
Part 3	Getting the Key and Message from the User	-	5%
	Encrypt the Message using a Given Key	Week 4	10%
	Translate Key and Message from String to Binary	Week 2	5%
	Adding the Key and Message Binary to the Image	-	15%
Part 4	Extract Key and Message Binary from Image Data	-	15%
	Translate Key and Message from Binary to String	-	10%
	Decrypt the Message using a Given Key	Week 5	5%
	Save Decrypted Message to Text File	Week 3 Part 1	5%
Part 5	Live Demonstration	-	10%
TOTAL			100%

Good Luck and Have Fun with the Software Project! :)

Weekly Activities

</>	40 min.	[Optional] Software Project 1 Familiarization (Week 1)	
		To help you prepare for the upcoming Software Project 1, the weeks preceding the release of the official Software Project 1 Specifications will include links to websites that contain important concepts that you will need. Additionally, short optional activities have also been prepared to help you develop snippets of code that will be beneficial to you when the Software Project comes.	
		For this week, our focus will be an overview of what the Software Project will be about: Steganography . The included readings provide a conceptual discussion on the subject matter, making it easy to understand what Steganography is. While this does not discuss the specifics of what you will be doing, it should provide you with enough context to understand what the weekly optional tasks are building towards.	
		Steganography: All You Need To Know In 5 Easy Points	[Website]
		What Is Steganography?	[Website]
</>	40 min.	[Optional] Software Project 1 Building Block (Week 2)	
		This week, we will be focusing on an important concept in computer programming: Data Representation. The included reading provides an overview of how data is represented in computers and will provide you with the knowledge needed to apply this in your programming activities.	
		Activity: ASCII Values This is an optional activity that you can do to create a piece of code that will benefit you when you develop your software project. <i>If you choose to submit this activity, it will be worth 5% of your Software Project 1 grade</i> ; not submitting this will not affect your SP1 grade, as you can still achieve full functionality even if you don't do this now (this helps prepare you for your SP).	
		Complete the given function, <code>char_to_ascii(word)</code> , that converts each character in the given string <code>word</code> into its integer counterpart (ASCII decimal value) and returns them in the list <code>ascii_values</code> .	
		Complete the given function, <code>ascii_to_binary(ascii_values)</code> , that converts the given list, <code>ascii_values</code> , into strings containing their binary values and returns the converted values in the list <code>binary_values</code> . e.g. the list <code>[4, 3, 1]</code> will be returned as <code>['100', '11', '1']</code>	
		Hint: you can solve each of these using Python's in-built functions.	
		Data In The Computer	[Website]
		ASCII Table - ASCII codes,hex,decimal,binary.html	[Website]
</>	40 min.	EEE 111 SP1 - ASCII Values (Week 2)	[HackerRank]
		Week 2 - ASCII Values Submission Bin	[UVLe]
		[Ungraded] ASCII Worksheet <i>Try to answer this on your own then ask for help if needed.</i>	[Website]
		[Optional] Software Project 1 Building Block (Week 3)	

	<p>This week, we will be working with files. Understanding File input/output (File I/O) is an important part of working with computers as most tasks involving programming</p> <p>Activity: File I/O</p> <p>This is an optional activity that you can do to create a piece of code that will benefit you when you develop your software project. <i>If you choose to submit this activity, it will be worth 10% of your Software Project 1 grade</i>; not submitting this will not affect your SP1 grade, as you can still achieve full functionality even if you don't do this now (this helps prepare you for your SP).</p> <p>Part I: Reading and Writing to Text Files (5%)</p> <p>Create a function <code>access_file(filename)</code> that copies the contents of the file <code>filename</code> and appends the data to the file 'file_logs.txt'.</p> <p><i>Note: You won't be able to check the functionality of your code through Hackerrank, as the platform does not support the use of files. However, you should be able to check whether your program works by creating simple text files and checking the result after running the program.</i></p> <table><tr><td>Python File I/O: Read and Write Files in Python</td><td>[Website]</td></tr><tr><td>Week 3 - File I/O Part I Submission Bin</td><td>[UVLe]</td></tr></table> <p>For the purpose of your Software Project, we will be dealing with image files (particularly JPEG/JPG files), and while it is possible to use the generic File I/O functions to open these images, it is quite difficult to understand the underlying format of the document, and it would be simpler for you to just learn how to use a library for handling image files. In this regard, we recommend that you use the Pillow library to open, manipulate, and save image files, and to learn how images are stored on computers and how you can access image data.</p> <p>Part II: Loading, Manipulating, and Saving Images (5%)</p> <p>Create a function <code>manipulate_image(filename)</code> that opens the image <code>filename</code>, prints the RGB values of each pixel, and saves the image with the blue component of all pixels halved. The image must have the original filename prepended by 'filtered_'. Assume that all images will be in the RGB encoding.</p> <p><i>Hint: check out the getdata(), putdata() methods of the Pillow library's Image module.</i></p> <table><tr><td>How are Images Stored on a Computer Grayscale & RGB Image Formats</td><td>[Website]</td></tr><tr><td>Tutorial — Pillow (PIL Fork) 8.3.2 documentation</td><td>[Website]</td></tr><tr><td>Week 3 - File I/O Part II Submission Bin</td><td>[UVLe]</td></tr></table>	Python File I/O: Read and Write Files in Python	[Website]	Week 3 - File I/O Part I Submission Bin	[UVLe]	How are Images Stored on a Computer Grayscale & RGB Image Formats	[Website]	Tutorial — Pillow (PIL Fork) 8.3.2 documentation	[Website]	Week 3 - File I/O Part II Submission Bin	[UVLe]
Python File I/O: Read and Write Files in Python	[Website]										
Week 3 - File I/O Part I Submission Bin	[UVLe]										
How are Images Stored on a Computer Grayscale & RGB Image Formats	[Website]										
Tutorial — Pillow (PIL Fork) 8.3.2 documentation	[Website]										
Week 3 - File I/O Part II Submission Bin	[UVLe]										
40 min.	<p>[Optional] Software Project 1 Building Block (Week 4)</p> <p>For this week, you will be focusing on implementing an algorithm for the software project using the concepts you have learned.</p> <p>Activity: Caesar's Magical Key Part I</p> <p>This is an optional activity that you can do to create a piece of code that will benefit you when you develop your software project. <i>If you choose to submit this activity, it will be worth 10% of your Software Project 1 grade</i>; not submitting this will not affect your SP1 grade, as you can still achieve full functionality even if you don't do this now (this helps prepare you for your SP).</p> <p>Complete the given function, <code>encrypt_text</code> that encrypts the variable <code>text</code> using the provided <code>key</code> by applying a modified version of Caesar's Cipher with the following rules:</p> <ul style="list-style-type: none">• The <code>key</code> will contain a word made up of the letters 'u' and 'd'. The cipher will use the <code>key</code> to										

		<p>determine the direction of the shift, with a ‘u’ indicating that the character must be shifted upwards while a ‘d’ indicating a downward shift. The number of characters in the <i>key</i> will determine the number of positions to shift.</p> <ul style="list-style-type: none">• The given <i>text</i> variable will only contain characters from the ASCII table with values between 32 and 126.• If the <i>key</i> is invalid (contains characters other than <i>u</i> and <i>d</i>) or the <i>text</i> is invalid (empty string), no encryption occurs and the function returns the original <i>text</i>.• If the <i>text</i> has more characters than the <i>key</i>, the <i>key</i> will loop through again to accommodate the additional <i>text</i> characters.• An iterative approach should be used to solve this problem. Brute force solutions will not be accepted. <p>As an example, for the <i>key</i> ‘udd’ and <i>text</i> ‘Hello World’, the encryption will go as follows:</p> <ul style="list-style-type: none">- The <i>key</i> indicates that the text needs to be shifted three places, since it has three characters.- Since the <i>text</i> has more than 3 characters, the <i>key</i> will loop through again to accommodate the additional characters. <table><tr><td><i>text</i></td><td>H</td><td>e</td><td>l</td><td>l</td><td>o</td><td></td><td>W</td><td>o</td><td>r</td><td>l</td><td>d</td></tr><tr><td><i>key</i></td><td>u</td><td>d</td><td>d</td><td>u</td><td>d</td><td>d</td><td>u</td><td>d</td><td>d</td><td>u</td><td>d</td></tr><tr><td><i>encrypted</i></td><td>K</td><td>b</td><td>i</td><td>o</td><td>l</td><td> </td><td>Z</td><td>l</td><td>o</td><td>o</td><td>a</td></tr></table> <ul style="list-style-type: none">- The encrypted word is: Kbiol Zlooa <table><tr><td>Crack the Code! Make a Caesar Cipher</td><td>[Website]</td></tr><tr><td>EEE 111 SP1 - Caesar’s Magical Key Part I (Week 4)</td><td>[HackerRank]</td></tr><tr><td>Week 4 - Caesar’s Magical Key Part I Submission Bin</td><td>[UVLe]</td></tr></table>	<i>text</i>	H	e	l	l	o		W	o	r	l	d	<i>key</i>	u	d	d	u	d	d	u	d	d	u	d	<i>encrypted</i>	K	b	i	o	l		Z	l	o	o	a	Crack the Code! Make a Caesar Cipher	[Website]	EEE 111 SP1 - Caesar’s Magical Key Part I (Week 4)	[HackerRank]	Week 4 - Caesar’s Magical Key Part I Submission Bin	[UVLe]
<i>text</i>	H	e	l	l	o		W	o	r	l	d																																	
<i>key</i>	u	d	d	u	d	d	u	d	d	u	d																																	
<i>encrypted</i>	K	b	i	o	l		Z	l	o	o	a																																	
Crack the Code! Make a Caesar Cipher	[Website]																																											
EEE 111 SP1 - Caesar’s Magical Key Part I (Week 4)	[HackerRank]																																											
Week 4 - Caesar’s Magical Key Part I Submission Bin	[UVLe]																																											
		<p>[Optional] Software Project 1 Building Block (Week 5)</p> <p>Following last week’s activity, you will be implementing an algorithm that reverses the encryption algorithm you created in the past week.</p> <p>Activity: Caesar’s Magical Key Part II</p> <p>This is an optional activity that you can do to create a piece of code that will benefit you when you develop your software project. <i>If you choose to submit this activity, it will be worth 5% of your Software Project 1 grade</i>; not submitting this will not affect your SP1 grade, as you can still achieve full functionality even if you don’t do this now (this helps prepare you for your SP).</p> <p>Complete the given function, <i>decrypt_text</i> that decrypts the variable <i>encrypted_text</i> using the provided <i>key</i> by applying a modified version of Caesar’s Cipher with the following rules:</p> <ul style="list-style-type: none">• The <i>key</i> will contain a word made up of the letters ‘u’ and ‘d’. The cipher will use the <i>key</i> to determine the direction of the shift, with a ‘u’ indicating that the character must be shifted upwards while a ‘d’ indicating a downward shift. The number of characters in the <i>key</i> will determine the number of positions to shift.• The given <i>text</i> variable will only contain characters from the ASCII table with values between 32 and 126.• If the <i>key</i> is invalid (contains characters other than <i>u</i> and <i>d</i>) or the <i>text</i> is invalid (empty string), no encryption occurs and the function returns the original <i>text</i>.• If the <i>text</i> has more characters than the <i>key</i>, the <i>key</i> will loop through again to accommodate the additional <i>text</i> characters.• An iterative approach should be used to solve this problem. Brute force solutions will not be accepted.																																										
</>	40 min.																																											

		Example, the text ‘Kbiol Zlooa’ with a key ‘udd’ should return the message ‘Hello World’.	
		EEE 111 SP1 - Caesar’s Magical Key Part II (Week 5)	[HackerRank]
		Week 5 - Caesar’s Magical Key Part II Submission Bin	[UVLe]
</>	- min.	Software Project 1 Specifications (Week 6)	
		The linked document provides a detailed explanation on the specifications of the Software Project 1.	
		“Software Project 1 Specifications.pdf”	[UVLe]