Snow

COMP0015 Term 2 Coursework 6 – 20% of the module

This document explains the arrangements for the coursework. You will create an application to embed a secret message in a file and, conversely, to decode the secret message hidden in a file. You will use steganography to do this. It's possible to embed steganographic messages in pictures, audio clips or in text and steganography is the practice of embedding secret content in an artefact such that it cannot be detected by the reader, viewer or listener. But why is this coursework called 'Snow'? This is a reference to Matthew Kwan's program snow, an application that you can install on your computer and use to create encrypted steganographic messages in real life.

Deadline

4pm Tuesday 23rd March 2021.

Aim

At a high-level, the aim of this coursework is for you to demonstrate that you can use text files. You are given some starter code; you do not need to change this code. You are required to complete some of the functions as described in this document.

Starting Your Assignment

You are provided with some starter code and some text files which you must download and save before starting the assignment. Feel free to use CoCalc to develop your code.

Running the stegsnow program

There are two ways to run the program from the terminal depending on whether you want to encode or decode a secret message. The code in `main()` contains code to handle the options you type on the command line, you are not required to edit the code in `main()`.

Encoding a secret message

Note: for the purposes of this coursework, you will only be required to use secret messages that contain the lower case characters a-z and space, for example: "dinner at eight pm".

Type the following at the terminal:

```
python3 stegsnow.py -m “dinner at eight pm” okonomiyaki.txt secret.txt
```

`python3`	The python interpreter. On macos this will be python3 and on Windows, this will be py or python.
`stegsnow.py`	Name of the python program.
`-m`	Use this option when you want to encrypt a secret message.

`“dinner at eight pm”`	The secret message.
`okonomiyaki.txt`	The input file in which the secret message will be embedded. You are provided with the file okonomiyaki.txt
`secret.txt`	The output file containing the steganographic text.

Table 1 Running the program when encoding a secret message

Decoding a secret message

Type the following at the terminal:

```
python3 stegsnow.py secret.txt
```

`python3`	The python interpreter. On macos this will be `python3` and on Windows, this will be py or `python`.
`stegsnow.py`	Name of the python program.
`secret.txt`	The file containing the steganographic text.

Table 2 Running the program when decoding a secret message

Representing characters as binary

Creating the secret message relies on some facts about the way that characters are represented in the computer. Computers represent all information as a series of 0s and 1s or binary. Characters are represented by a set of commonly used binary numbers. The mapping of characters to numbers is known as ASCII which stands for American Standard Code for Information Interchange. Some characters and the mapping to binary numbers are shown in the table below.

Letter	**Binary representation**
a	1100001
b	1100010
c	1100011
d	1100100

Table 3 ASCII representation of characters

In stegsnow.py there is a dictionary, CHAR_TO_BINARY, that contains a mapping from a character to binary:

```
CHAR_TO_BINARY = { # binary representation of ASCII characters
    'a': '1100001',
    'b': '1100010',
    'c': '1100011',
    'd': '1100100',
    'e': '1100101',
    'f': '1100110',
    'g': '1100111', ….
```

and another dictionary BINARY_TO_CHAR, that contains a mapping from binary to character:

```
BINARY_TO_CHAR = { # binary string to ascii characters
    '1100001': 'a',
    '1100010': 'b',
    '1100011': 'c',
    '1100100': 'd',
    '1100101': 'e',
    '1100110': 'f',
    '1100111': 'g', ….
```

You will use both dictionaries in your solution.

Functionality

The skeleton code contains the empty functions: `encode_secret_message()` and `decode_secret_message()` you must complete them according to the instructions here and the specification in the docstrings (comments) at the start of the functions.

1) Function `encode_secret_message(message, infile, outfile, zero, one)`

In this function you will take the secret message, encode it as a series of 1s and 0s, combine it with the input file and write the result to the output file. Here is an example, let's say your secret message is "dinner at eight pm" and the input file is the file that is distributed with the coursework.

Input file – okonomiyaki.txt	`Output file
Okonomiyaki https://thejameskitchen.wordpress.com/2013/12/27/ okonomiyaki/ Serves 2 1 small Chinese cabbage A small bunch of choi sum or any other bitter dark greens like kale 2 large carrots 6 spring onions	Okonomiyaki1100100 d https://thejameskitchen.wordpress.com/2013/12/27/okono miyaki/1101001 i 1101110 n Serves 21101110 1 small Chinese cabbage1100101 e 1110010 r A small bunch of choi sum or any other bitter dark greens like kale1111011 space 2 large carrots1100001 a 6 spring onions1110100 t

In this function you will perform 3 sub-tasks:

a) Process the secret message character by character, producing a list of binary strings. Use the dictionary CHAR_TO_BINARY.
b) Read the input file, `infile`, line by line.
c) Combine each line of the input file with one of the binary strings representing the secret message and write the result to the output file.

You may need to decompose this function further, writing more functions. Make sure you document your functions with docstrings.

You are probably thinking that this secret message is not very secret. Don't worry! Once you have everything working, you will attend to this detail.

2) Function `decode_secret_message(infile, zero, one)`

In this function you will take the file with file name `infile`, decode the secret message inside and then print it. To do this, you will need to complete these sub-tasks:

a) For each line in the file, extract the last 7 characters. Check that the last 7 characters is composed of zeroes and ones. Use the values passed in as parameters to the function to do this. If the 7 characters is composed of zeroes and ones you have a binary string representing a character in your secret message.
b) Convert every binary string extracted to a character by using the dictionary BINARY_TO_CHAR. Add the characters to a list.
c) You have your secret message, print it.

You may need to decompose this function further, writing more functions. Make sure you document your functions with docstrings.

3) Making your message invisible

So far the message has been entirely visible, anyone intercepting the message will be able to see it and easily guess at its meaning. What if we replaced every occurrence of a '1' and a '0' with different invisible characters such as a space and a tab? Well then our message would disappear. There are two constants at the top of the program:

```
ZERO = '0'              # Character representing zero         ``
ONE = '1'               # Character representing one
```

These are used in `main()` and passed as parameters to `encode_secret_message()` and `decode_secret_message()`.

You can change the values for the constants so that:

```
ZERO = '\t'             # Character representing zero         ``
ONE = ' '               # Character representing one
```

There's a little more coding that we need to do before this will work.

a) Function `encode_secret_message()`

In this function you translated the letters in the message to binary strings. This won't change but now you must also translate each digit in the binary string using the parameters passed to the function `zero` and one. You have been provided with the function `encode(binary_letter, zero, one)` which you can use to do this. In this way, you can replace the pattern of zeroes and ones in the binary letter with symbols that you can't see such as a space and a tab.

`'1100001'` would be translated as `'  \t\t\t\t '`
`'1100010'` would be translated as `'  \t\t\t\ \t'`

b) Function `decode_secret_message()`

In this function you translated the binary strings in the file to ASCII characters. Once again, there's a little more that we need to do. When you process the last 7 characters on a line you must translate the characters from, say, spaces and tabs to zeroes and ones. Use the parameters zero and one which are passed as parameters to the function. You have been provided with the function `decode(binary_letter, zero_char, one_char)`, use it to translate what you get from the file to a string of binary digits. If a tab character represents '0' and a space represents '1' then:

`'  \t\t\t\t '` would be translated as `'1100001'`
`'  \t\t\t\ \t'` would be translated as `'1100010'`

That's it. The coursework is finished.

Testing:

You are responsible for testing your program carefully. Make sure that you have thought about all the things that can go wrong and test your program to ensure that you know it works correctly in all circumstances.

Submitting your assignment

At the submission link on moodle:

1. Make sure your student number (not your name) is included in comments at the top of your program.
2. Upload your program.
    a. You may upload multiple files but do not upload a folder containing your files because this can cause compatibility issues for the marking team.
3. You must ensure that your program works properly either on your own computer or on the CoCalc platform before you submit the code.

Assessment

You are expected to show that you can code competently using the programming concepts covered so far in the course including (but not limited to): use of files, strings, variables, conditions, loops and functions.

Marking criteria will include:

- Correctness – your code must perform as specified
- You must apply the Python concepts appropriately.
- Programming style – see section 'Style Guide' for more detail.
- Your assignment will be marked using the rubric at the end of this document. This is the standard rubric used in the Department of Computer Science. Marks for your project work will be awarded for the capabilities (i.e. functional requirements) your system achieves, and the quality of the code. Categories 5 and 6 will be used for coding assignments.

Additional Challenges

- Additional marks may also be gained by taking on extra challenges but you should only attempt an additional challenge if you have satisfied all requirements for the coursework. It's up to you what you choose to do - if anything.
- Note: You are strongly encouraged to follow the specification carefully and to use programming techniques as described in the course materials and textbooks. Poor quality code with additional functionality will not improve your marks.
- Options for this coursework might include:
    1. Write some unit tests for the functions you have written. You can write your own tests in a separate program using python or use doctest or unittest or pytest.
    2. Change the encryption / decryption algorithm so that the start and end of the encrypted message starts and ends with a special character that cannot appear in the text. For example: the binary sequence '0000000' could be used.
    3. The brief specifies that you encode one letter (ascii character) per line. Amend your application so that multiple ascii characters can be encoded, up to a specified maximum line length. The maximum line length should be a constant in your program.
    4. Take a look at the features of snow, can you add encryption / decryption using a Python cryptography library?

Plagiarism

Plagiarism will not be tolerated. Your code will be checked using a plagiarism detection tool.

Style Guide

You must adhere to the style guidelines in this section.

Formatting Style

1. Use Python style conventions for your variable names (snake case: lowercase letters with words separated by underscores (_) to improve readability).
2. Choose good names for your variables. For example, `num_bright_spots` is more helpful and readable than `nbs`.
3. Name constants properly using capital letters and put them at the top of your program.

4. Use a tab width of 4 or 8. The best way to make sure your program will be formatted correctly is never to mix spaces and tabs -- use only tabs, or only spaces.
5. Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

   ```
   b = 3 > x and 4 - 5 < 32
   b= 3>x and 4-5<32
   ```

6. Each line must be less than **80 characters** long *including tabs and spaces*. You should break up long lines using \. You don't need a continuation character if you are breaking up the parameters of a function.
7. Function names should also be in snake_case: encrypt_message(), print_introduction.
8. Functions should be no longer than about 12 lines in length. Longer functions should be decomposed into 2 or more smaller functions.

Docstrings

If you add your own functions you must comment them using docstrings. Take a look at the code you've been given for some examples. Your comments should:

1. Describe precisely *what* the function does.
2. Do not reveal *how* the function does it.
3. Make the purpose of every parameter clear.
4. Refer to every parameter by name.
5. Be clear about whether the function returns a value, and if so, what.
6. Explain any conditions that the function assumes are true. Examples: "n is an int", "n != 0", "the height and width of p are both even."
7. Be concise and grammatically correct.
8. Write the docstring as a command (e.g., "Return the first ...") rather than a statement (e.g., "Returns the first ...")

Note: you can use any style of docstring you like, your editor may have a default style – you are welcome to use this. Available styles include: NumPy, Google, ReStructuredText.

Here is a template for a NumPy-style docstring

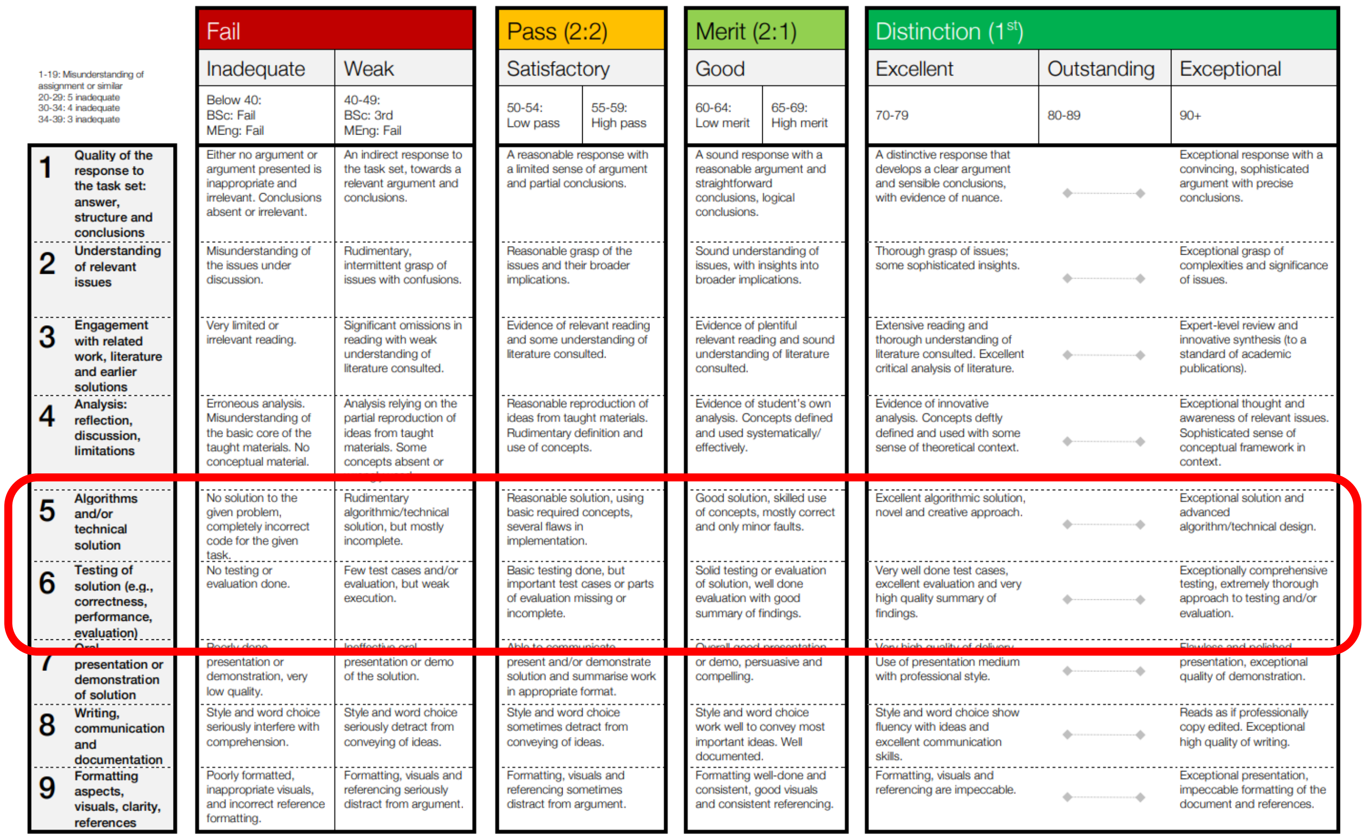
```python
def template(arg1, arg2):
    """
    Summary line.

    Extended description of function.

    Parameters
    ----------
    arg1 : int
        Description of arg1
    arg2 : str
        Description of arg2

    Returns
    -------
    int
        Description of return value

    """
    pass
```

Here is a completed example:

```python
def random_number_generator(start_range, end_range):
    """
    Returns a random number from the range specified. Includes the start and end of the range.


    Parameters
    ----------
    start_range : int
        Start of the range, inclusive
    end_range: int
        End of the range, inclusive

    Returns
    -------
    int
        Random number

    """
```

Computer Science Grading Criteria

Categories 5 and 6 will be used for assessing programs.



	Fail		Pass (2:2)	Merit (2:1)	Distinction (1st)		
	Inadequate	Weak	Satisfactory	Good	Excellent	Outstanding	Exceptional
1-19: Misunderstanding of assignment or similar 20-29: 5 inadequate 30-34: 4 inadequate 34-39: 3 inadequate	Below 40: BSc: Fail MEng: Fail	40-49: BSc: 3rd MEng: Fail	50-54: Low pass 55-59: High pass	60-64: Low merit 65-69: High merit	70-79	80-89	90+
1 Quality of the response to the task set: answer, structure and conclusions	Either no argument or argument presented is inappropriate and irrelevant. Conclusions absent or irrelevant.	An indirect response to the task set, towards a relevant argument and conclusions.	A reasonable response with a limited sense of argument and partial conclusions.	A sound response with a reasonable argument and straightforward conclusions, logical conclusions.	A distinctive response that develops a clear argument and sensible conclusions, with evidence of nuance.	◆┈┈◆	Exceptional response with a convincing, sophisticated argument with precise conclusions.
2 Understanding of relevant issues	Misunderstanding of the issues under discussion.	Rudimentary, intermittent grasp of issues with confusions.	Reasonable grasp of the issues and their broader implications.	Sound understanding of issues, with insights into broader implications.	Thorough grasp of issues; some sophisticated insights.	◆┈┈◆	Exceptional grasp of complexities and significance of issues.
3 Engagement with related work, literature and earlier solutions	Very limited or irrelevant reading.	Significant omissions in reading with weak understanding of literature consulted.	Evidence of relevant reading and some understanding of literature consulted.	Evidence of plentiful relevant reading and sound understanding of literature consulted.	Extensive reading and thorough understanding of literature consulted. Excellent critical analysis of literature.	◆┈┈◆	Expert-level review and innovative synthesis (to a standard of academic publications).
4 Analysis: reflection, discussion, limitations	Erroneous analysis. Misunderstanding of the basic core of the taught materials. No conceptual material.	Analysis relying on the partial reproduction of ideas from taught materials. Some concepts absent or [illegible]	Reasonable reproduction of ideas from taught materials. Rudimentary definition and use of concepts.	Evidence of student's own analysis. Concepts defined and used systematically/ effectively.	Evidence of innovative analysis. Concepts deftly defined and used with some sense of theoretical context.	◆┈┈◆	Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context.
5 Algorithms and/or technical solution	No solution to the given problem, completely incorrect code for the given task.	Rudimentary algorithmic/technical solution, but mostly incomplete.	Reasonable solution, using basic required concepts, several flaws in implementation.	Good solution, skilled use of concepts, mostly correct and only minor faults.	Excellent algorithmic solution, novel and creative approach.	◆┈┈◆	Exceptional solution and advanced algorithm/technical design.
6 Testing of solution (e.g., correctness, performance, evaluation)	No testing or evaluation done.	Few test cases and/or evaluation, but weak execution.	Basic testing done, but important test cases or parts of evaluation missing or incomplete.	Solid testing or evaluation of solution, well done evaluation with good summary of findings.	Very well done test cases, excellent evaluation and very high quality summary of findings.	◆┈┈◆	Exceptionally comprehensive testing, extremely thorough approach to testing and/or evaluation.
7 Oral presentation or demonstration of solution	Poorly done presentation or demonstration, very low quality.	Ineffective oral presentation or demo of the solution.	Able to communicate present and/or demonstrate solution and summarise work in appropriate format.	Overall good presentation or demo, persuasive and compelling.	Very high quality of delivery. Use of presentation medium with professional style.	◆┈┈◆	Flawless and polished presentation, exceptional quality of demonstration.
8 Writing, communication and documentation	Style and word choice seriously interfere with comprehension.	Style and word choice seriously detract from conveying of ideas.	Style and word choice sometimes detract from conveying of ideas.	Style and word choice work well to convey most important ideas. Well documented.	Style and word choice show fluency with ideas and excellent communication skills.	◆┈┈◆	Reads as if professionally copy edited. Exceptional high quality of writing.
9 Formatting aspects, visuals, clarity, references	Poorly formatted, inappropriate visuals, and incorrect reference formatting.	Formatting, visuals and referencing seriously distract from argument.	Formatting, visuals and referencing sometimes distract from argument.	Formatting well-done and consistent, good visuals and consistent referencing.	Formatting, visuals and referencing are impeccable.	◆┈┈◆	Exceptional presentation, impeccable formatting of the document and references.