ASSIGNMENT 1 – THUE-MORSE SEQUENCES

Overview
In this assignment you will have the opportunity to test your Python skills in generating and manipulating text.

Timelines and Expectations
Percentage Value of Task: 20%
Due: Fri, September 8,2023 17:00 (Week 7)
Minimum time expectation: 20 hours

Learning Outcomes Assessed
The following course learning outcomes are assessed by completing this assessment:

Knowledge:
K1. Identify and use the correct syntax of a common programming language.
K2. Recall and use typical programming constructs to design and implement simple software solutions.
K4. Explain the importance of programming style concepts (documentation, mnemonic names, indentation).

Skills:
S1. Utilise pseudocode and/or algorithms as a major program design technique.
S2. Write and implement a solution algorithm using basic programming constructs.
S4. Describe program functionality based on analysis of given program code.

Application of knowledge and skills:
A1. Develop self-reliance and judgement in adapting algorithms to diverse contexts.
A2. Design and write program solutions to identified problems using accepted design constructs.

Assessment Details

Task 1. Constructing a Cube-Free Word in the Alphabet of Two Symbols.

In this task you are required to develop a Python program that constructs an arbitrarily long (potentially infinite) cube-free word in the alphabet of two symbols, ‘0’, ‘1’. I.e., a word that does not contain “cubes” – three consecutive identical sub-words.
Such words (sequences) are called Thue–Morse sequences and have multiple applications ranging from Chess to Group Theory and Differential Geometry.

1.1. Construction of the Cube-Free Word: The Thue–Morse sequence is a (potentially) infinite word in the alphabet of two symbols, **‘0’** and **‘1’**, which can be constructed in the following way:

(0) $\mathbf{t_0}$ **= ‘0’**

(1) $\mathbf{t_1}$ **= ‘0’ + ‘1’ = ‘01’**

(2) $\mathbf{t_2}$ **= ‘01’ + ‘10’ = ‘0110’**

(3) $\mathbf{t_3}$ **= ‘0110’ + ‘1001’ = ‘01101001’**

...

(n) $\mathbf{t_n = t_{n-1} + \overline{t_{n-1}}}$, where $\overline{t_{n-1}}$ denotes the ‘inverse’ of $\mathbf{t_{n-1}}$,

i.e., all **‘0’**s in $\mathbf{t_{n-1}}$ are replaced by **‘1’**s and vice versa.

...

It is easy to see that each $\mathbf{t_i}$ is the first half of $\mathbf{t_{i+1}}$, and therefore $\mathbf{t_{i+1}}$ can be seen as an extension of $\mathbf{t_i}$.

This sequence of extensions can be continued indefinitely, thus constructing an infinite word, **T**, which contains all $\mathbf{t_i}$s as its prefixes (beginnings):

T = ‘01101001100101101001011001101001100101100110100 10...’

T has an important property: it does not contain “cubes”, i.e., three consecutive identical blocks (sub-words).

1.2. Programming Task. In this task you are required to write a Python function named

thue_morse(n),

that takes a positive integer parameter, **n**, and returns the string $\mathbf{t_n}$ (defined in the previous section).
In your program you may define other (auxiliary) functions with arbitrary names, however, the solution function of this task should be named **thue_morse(n)**.

In your report, you are **required to submit a brief explanation of your program in plain English.** It will be useful in confirming ownership of your work.

[5 marks]

Task 2. Constructing a square-free word in the alphabet of three symbols.

In this task you are required to write a function in Python that constructs an arbitrarily long (potentially infinite) **square-free** word in the alphabet of three symbols, **'1'**, **'2'**, **'3'**. I.e., a word that does not contain "squares" – two consecutive identical sub-words.
Although, this can be done by using the Thue-Morse sequence, in this task we will use another construction, suggested by S. Arshon.

2.1. Construction of the Square-Free Word: Again, as in Task 1, we will build a sequence of finite words, $\mathbf{a_i}$, that can be extended to an infinite word.
We start with $\mathbf{a_0}$ **= '1'**.
Each next word, $\mathbf{a_{k+1}}$, is constructed by replacing each occurrence of the symbols **'1'**, **'2'**, **'3'** in the previous word, $\mathbf{a_k}$, with 3-letter words according to the following rules:

(1) if symbol **'1'** is in an odd position in $\mathbf{a_k}$, then it is replaced with the word **'123'**, if **'1'** is in even position, it is replaced with **'321'**.

(2) if symbol **'2'** is in an odd position in $\mathbf{a_k}$, then it is replaced with the word **'231'**, if **'2'** is in even position, it is replaced with **'132'**.

(3) if symbol **'3'** is in an odd position in **a_k**, then it is replaced with the word **'312'**, if **'3'** is in even position, it is replaced with **'213'**.

Here's the table repeating the replacement rules in a tabular format:

Symbols in odd positions	Replacement string for odd positions	Symbols in even positions	Replacement string for even positions
1	123	1	321
2	231	2	132
3	312	3	213

Please note, that in this description the **first symbol is considered to be in position 1.** Therefore, you will need to make the necessary adjustments when using Python strings since they are zero-based.

Below, you can see first few steps of the construction process:

a_0 = '1'
a_1 = '123'
a_2 = '123132312'
a_3 = '123132312321312132312321231'
...
A = '12313231232131213231232123121323132131232123132131213 23...'

2.2. The Programming Tasks.

(a) You are required to write a Python function named **square_free(n)**, that takes a positive integer parameter **n** and returns the string **a_n** (defined in the previous section).
Again, as in Task 1, you may define other (auxiliary) functions with arbitrary names, however, the solution function of this task should be named **square_free(n)**.

In your report, you are **required to submit a brief explanation of your program in plain English.** It will be useful in confirming ownership of your work.

[9 marks]

(b) Write a Python function named **print3Blocks(s)** that takes a string, **s**, as a parameter and prints it in blocks of 3 symbols separated by white spaces. For example, **print3Blocks(a_3)** will print:

123 132 312 321 312 132 312 321 231

[1 mark]

Task 3. Counting the number of squares in a string.
In this task you are required to write a Python function named **count_squares(s)** that takes a string, **s**, as a parameter and returns the number of "squares" in **s**, i.e., the number of occurrences of two consecutive identical sub-words in **s**.

For example, **count_squares ('1231233')** should return **2** as there are two "squares" in its argument: '1231233' and '1231233'.

In your report, you are **required to submit a brief explanation of your program in plain English.** It will be useful in confirming ownership of your work.

[5 marks]

Allocated Marks: See Course Description
Due Date: See Course Description
Please refer to the Course Description for information relating to late assignments and special consideration.

Assignment Submission

You must supply your program source code files and your documentation as a single zip file named as follows:

<YOUR-NAME>_<YOUR-STUDENT-ID>.zip,

e.g., John_SMITH_30000000

Your documentation should be in PDF format.

Assignments will be marked on the basis of fulfilment of the requirements and the quality of the work.

In addition to the marking criteria, marks may be deducted for failure to comply with the assignment requirements, including (but not limited to):

- Incomplete implementation(s), and
- Incomplete submissions (e.g., missing files), and
- Poor spelling and grammar.

You might be asked to demonstrate and explain your work.

Marking Criteria/Rubric

Student ID: ____________ Student Name: ______________

Tasks	**Weight**	**Awarded**
Marks deducted for badly commented or badly written code	(-4)	
Task 1. thue_morse function • Report • Python code • Demonstration that code works correctly using representative samples	1 3 1	
Task 2. square_free function • Report • Python code • Demonstration that code works correctly using representative samples	2 6 1	
Task 2. print3Blocks function • Python code	1	
Task 3. count_squares function • Report • Python code • Demonstration that code works correctly using representative samples	1 3 1	
Total	20+(-4)	