Assignment 1

Instructions

- Unless specified otherwise,
  - you can use anything in the Python Standard Library;
  - don't use any third-party library in your code for this assignment;
  - you can assume all user inputs are always valid and require no validation.
- The underlined blue texts in the Sample Runs section of each question refers to the user inputs. It does NOT mean that your program needs to underline them.
- Please follow closely the format of the sample output for each question. Your program should produce **exactly** the same output as the sample (same text, symbols, letter case, spacing, etc.). The output for each question should end with a single newline character, which has provided by the `print()` function by default.
- If you see fit to define your own functions for specific tasks in any question, you may do so, but this is not required.
- Name your script files as q1.py for Question 1, q2.py for Question 2, …, q5.py for Question 5 (case-sensitive). Using other names may affect our marking and may result in deduction.
- Your source files will be tested in script mode rather than interactive mode.

Question 1 (20%)

Write a program to perform the following tasks **in sequence**. The data values involved in each task are to be hard coded in your script. You need not use the `input()` function.

(a) Task 1:

Print the following lyrics using a single `print()` call. Every line has a single line break.

```
Hush little baby, don't say a word,
Papa's gonna buy you a mockingbird.
And if that mockingbird won't sing,
Papa's gonna buy you a diamond ring.
```

(b) Task 2:

Print the following string in a literal sense (including every special symbol) to the console:

```
\\tserver:\backup\version\'my doc'\"aist class_list"
```

Note: you are required to use the escape character instead of the r prefix (raw string) in this task.

(c) Task 3:

Define the following variables:

```
a = 5
b = 1.6180339
c = 100000000
```

and use f-string or the `format()` function of the `str` class to print the following string to the console using the variables:

```
0005, 5**, +1.618, 1.6180339000e+00, 100,000,000, 5f5e100
```

(d) Task 4:

Define the following dictionary (refer to the end of Lecture 1):

```
toys = {"Lego": 82.5, "Peppa Pig": 29.9, "Transformer": 120 }
```

Use f-string or the `format()` function of the `str` class to print the following output using the `toys` dictionary:

```
        Lego: $  82.5
   Peppa Pig: $  29.9
 Transformer: $ 120.0
```

To help you better visualize the spacing requirements, please refer to the following output augmented with a ruler and gridlines:

```
01234567890123456789
--------------------
        Lego: $  82.5
   Peppa Pig: $  29.9
 Transformer: $ 120.0
```

Question 2 (20%)

Write a Python program to accept an integer *h* from the user and print a triangular pattern of height *h* (i.e., number of lines printed), as illustrated in the following sample runs.

If the user input is not in this specified range: $2 < h \leq 30$, keep prompting the user to input again until a valid value is received.

For better visual effect, the line in the middle is formed by hyphens (-) when *h* is odd; it is formed by underscores (_) when *h* is even. The line at the bottom is always formed by underscores.

The following table may help you understand the pattern better.

h	# lines to print	# hyphens (or _) for the middle line	# underscores for the bottom line
3	3	2	4
4	4	2	6
5	5	4	8
6	6	4	10
7	7	6	12
8	8	6	14

Hint: to get started, think about how many spaces to print before printing the / symbol for each line.

Sample Runs:

```
Enter h: 3
  /\
 /--\
/____\
```

```
Enter h: 4
   /\
  /__\
 /    \
/______\
```

```
Enter h: 5
    /\
   /  \
  /----\
 /      \
/________\
```

```
Enter h: 6
     /\
    /  \
   /____\
  /      \
 /        \
/__________\
```

```
Enter h: 7
      /\
     /  \
    /    \
   /------\
  /        \
 /          \
/____________\
```

```
Enter h: 8
       /\
      /  \
     /    \
    /______\
   /        \
  /          \
 /            \
/______________\
```

Question 3 (20%)

Consider a sequence x_1, x_2, ... generated by the following formula:

$$x_{i+1} = \begin{cases} \frac{x_i}{2} & \text{if } x_i \text{ is even} \\ 3x_i + 1 & \text{if } x_i \text{ is odd} \end{cases}$$

That is, the next term in the sequence is derived from the last term based on the above formula. Interestingly, it is conjectured that for any positive integer x_1, the sequence will always end with the value 1. This is known as *Collatz conjecture*.

Write a program to accept a positive integer from the user; then print the sequence generated by the above formula until the term equals 1.

You may assume that the user always enters an integer value, but you need to make sure it is positive before generating the sequence.

Sample Runs:

```
Enter a positive integer: -10
Only positive integer is acceptable.
```

```
Enter a positive integer: 20
20, 10, 5, 16, 8, 4, 2, 1
```

```
Enter a positive integer: 30
30, 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

```
Enter a positive integer: 100
100, 50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10,
5, 16, 8, 4, 2, 1
```

```
Enter a positive integer: 54321
54321, 162964, 81482, 40741, 122224, 61112, 30556, 15278, 7639, 22918, 11459,
34378, 17189, 51568, 25784, 12892, 6446, 3223, 9670, 4835, 14506, 7253, 21760,
10880, 5440, 2720, 1360, 680, 340, 170, 85, 256, 128, 64, 32, 16, 8, 4, 2, 1
```

(For the last two sample runs, the sequences are too long to be shown on a single line here, but they do not consist of any newline characters. You may print a line break at the end of the sequence but no breaks between sequence items.)

Question 4 (20%)

Write a Python program that solves the following problem by a brute-force search approach (i.e., try all possible candidates for the solution):

How many ways are there to change an amount $A into coins of $1, $2, $5?

Suppose that $*A* is an integer between 1 and 100, inclusive, entered by the user.

Sample Runs:

```
Enter an integer amount between 1 and 100: 8
7 ways
```

```
Enter an integer amount between 1 and 100: 50
146 ways
```

```
Enter an integer amount between 1 and 100: 100
541 ways
```

Question 5 (20%)

Vending machines often need to return the change in coins (and notes in newer models) to the buyer. They embed a small program to control this operation. In this question, we are going to simulate that program which calculates the number of notes and coins for a specific amount of change to return when the machine receives an amount of money inserted and the price of a chosen item. The input and output will be performed via the keyboard and screen.

Write a program to accept two floating point numbers, which represent the amount of money inserted and the price of an item chosen by the user. The program responds by telling the user one combination of notes and coins that equals that amount of change. We adopt the US currency system which has the following 10 denominations:

Notes: $100, $50, $20, $10, $5, $1

Coins:

Face value	Coin's name
1¢	penny
5¢	nickel
10¢	dime
25¢	quarter

Sample Runs:

```
Insert money ($): 100
Item price ($): 5.51
Change: $94.49
 1 x $50
 2 x $20's
 4 x $1's
 1 x quarter
 2 x dimes
 4 x pennies
```

```
Insert money ($): 55.55
Item price ($): 44.44
Change: $11.11
 1 x $10
 1 x $1
 1 x dime
 1 x penny
```

```
Insert money ($): 10000
Item price ($): 87.5
Change: $9912.5
99 x $100's
 1 x $10
 2 x $1's
 2 x quarters
```

```
Insert money ($): 100
Item price ($): 9.781
No change combination is found!
```

```
Insert money ($): 99.999
Item price ($): 50.1
No change combination is found!
```

```
Insert money ($): 80.5
Item price ($): 20.51
Change: $59.99
 1 x $50
 1 x $5
 4 x $1's
 3 x quarters
 2 x dimes
 4 x pennies
```

```
Insert money ($): 1024.38
Item price ($): 88.81
Change: $935.57
 9 x $100's
 1 x $20
 1 x $10
 1 x $5
 2 x quarters
 1 x nickel
 2 x pennies
```

Notes:

- There are other less common denominations in US which are ignored here.
- Print the plural form of each denomination conditionally, i.e., when the count is greater than 1, print "quarters", "dimes", "nickels", and "pennies"; otherwise, print "quarter", "dime", "nickel", and "penny".
- Whenever a larger denomination can form the required amount, don't use a smaller one. For example, for amount 20¢, always use 2 dimes instead of 1 dime plus 2 nickels nor 4 nickels. With this rule, you should never hit the case printing the plural form "nickels".
- If a value of more than 2 decimal places is entered, it can’t be represented exactly by cents. Then the program should simply report that no change combination is found.
- There are some subtleties around the floating-point number precision issues when representing the change amount. Watch out for the last two sample runs; and you may make reasonable assumptions around the accuracy required in your handling.