Simon Fraser University
School of Computing Science

CMPT 120, Summer 2018
Assignment 1

(Read the entire assignment, first!)

Objective

In Assignment 1, our objective is to practise using what we have learnt so far:

1. Values, types and variables
2. Python statements
3. User interaction
4. Comments

Create a group of 2 on CourSys

1. We shall do Assignment 1 in pairs. So, let's find ourselves a partner.
   You can choose a partner from any lab section.
2. Once you have found a partner, start working on the assignment.
3. In the week 2 of your assignment once you are sure about your group member, go to CourSys and create a **"group" of 2** for assignment 1 submission. Name the group as follows: a1_firstname_firstname. For example, group name for Joe Smith and Andrew Paul should be: **a1_joe_andrew**.

   Note: If you do not want to do the assignment in pair (i.e., you would like to work on our own), you still **must** create a **"group" of 1** on CourSys for Assignment 1.

Gold Exchange

In the year 3046, paper currency is no longer used. It has been replaced by coins made of gemstones. The government however, is obsessed with gold. Therefore, most humans find work mining gold, and trading it to the government for Gemstone coin.

The gold is exchanged by the miners for:

- Emerald coin, worth 80 ounces of gold,
- Sapphire coin, worth 35 ounces of gold,
- Ruby coin, worth 10 ounces of gold, and
- Amber coin, worth 1 ounce of gold.

However, to maintain consistency, exchange stations always trade the gold in a specific order.

Exchange stations will always give back the highest value gemstone coin first. Then the remainder will be given by the second highest value gemstone, and so on. Therefore, Emerald coin would be given first, then Sapphire, and then Ruby. The remainder will then be filled with Amber coin.

(Gemstone coin cannot be plural - i.e., 7 Emerald coin is correct, not 7 Emerald coins)

Problem Statement and Requirements

In this assignment, we are asked to write a Python program called **GoldExchange.py**, which computes the quantity of each gemstone coin that must be returned to the miner.

Here is a suggested algorithm for our program. You are free to use it or design our own.

- Identify the application (welcome the user).
- Ask the miner how much gold (in ounces) they have mined.
- Check that the value entered is greater than zero.
- Display how many Emerald, Sapphire, Ruby and Amber coin the exchange station must give back to the miner

The above algorithm is not very detailed. We may wish to decompose each of its steps into a series of more detailed sub-steps.

Sample Runs and Testing

Our Python program must display the user interaction exactly as it is in the sample runs illustrated below. Note that in these sample runs, the output produced by our program is in blue and the user input is in black.

Let's test our program with the test cases used in the three sample runs below as well as with other test cases, which we must we create.

Note: Right now, we do not want to thoroughly validate the user input. This signifies that we should stick to whole number integers only for the input, when testing our program. So, no string nor float values.

We can assume that the users of our program (i.e., the TA's marking our program) will never enter anything else than integers.

Sample Run 1

```
Welcome to the Gold Exchange Station!

-------------
Program Input
-------------

Please, enter the amount of gold mined in ounces (integer): 239

--------------
Program Output
--------------

The Gemstone coin equivalent to 239 ounces of gold are:

      2 Emerald Coin,
      2 Sapphire Coin,
      0 Ruby Coin,
      9 Amber Coin.

--- See you again! ---

>>>
```

Sample Run 2

```
Welcome to the Gold Exchange Station!

-------------
Program Input
-------------

Please, enter the amount of gold mined in ounces (integer): 389

--------------
Program Output
--------------

The Gemstone coin equivalent to 389 ounces of gold are:

      4 Emerald Coin,
      1 Sapphire Coin,
      3 Ruby Coin,
      4 Amber Coin.

--- See you again! ---

>>>
```

Sample Run 3

```
Welcome to the Gold Exchange Station!

-------------
Program Input
-------------

Please, enter the amount of gold mined in ounces (integer): 0

--------------
Program Output
--------------

   Sorry, Dude. Work harder next time.

--- Good Luck! ---

>>>
```

Sample Run 4

```
Welcome to the Gold Exchange Station!

-------------
Program Input
-------------

Please, enter the amount of gold mined in ounces (integer): -100

--------------
Program Output
--------------

   Come on...

--- Be positive! ---

>>>
```

Hints

- Let's develop our solution **incrementally**: write a few statements and execute them. If they work, write a few more and execute them, etc… This way, we avoid spending enormous amount of time debugging it.

- Feel free to use the print( ) built-in function when debugging. Then comment out these print statements as they are for testing purposes only.
- It is possible to solve this problem with a program of about 20 lines (not counting comments and header comment block).
- Please, refer to the Marking Scheme section below for more criteria our program must satisfied.

Submission

- Submit our file: **GoldExchange.py** on CourSys by Friday, June 15 no later than 4:30 pm .
- We can submit our work to CourSys as many times as we wish, but it is the last submission that counts. So, let's make sure that our last submission is done before the deadline.
- No late submission will be accepted: any submission made after the due date and time stated above will not be marked for grades. However, they will be marked to provide feedback to students.
    - The only exception is for medical reasons. If we are not able to submit our assignment on time due to a medical reason, we must notify the instructor by email right away and provide her with a doctor's note as soon as possible.

Marking Scheme

When marking Assignment 1, we will look at some or all of the following criteria:

- Are the requirements listed above satisfied? A requirement is a rule to which our Python program must abide. For example:
  - Are the Gemstone amounts in our program represented as integers?
  - Does the program always display the correct number of Gemstones according to the given rules?
  - etc…
- Have we respected the Good Programming Style (GPS) described on our course web site? For example:
  - Does our header comment block contain the name of our file (program), the description of our program, the name of the author (us!) and the creation date of our program?
  - Is our program easy to read (we used empty lines to create space)?
  - Is our code commented? Are our variables descriptively named?
- Is our program producing exactly the same user interactions illustrated in the sample runs (when the same test data is used)?
- Are we using appropriate Python building blocks in our program and are we using them well?
- Is our Python program a “real” program and not a screenshot of the Python Interpreter Shell window into which we have entered each of our Python statements one at a time?
- Is our program executing? Are there any syntax and/or runtime errors?
- Is our program solving the problem? Are there any semantic errors?

Have fun!
