Fall 2020
Due Sept 25th

Computer Science 1001
Problem Set #1

Important Notes:

- Even though this, or any other subsequent, Problem Set is only graded based on whether or not a reasonable attempt was made for each problem, it is a good idea to complete each Problem Set by the due date and upload it to the appropriate dropbox in the course Brightspace shell to get credit for completing the Problem Set and see if you have understood the required concepts.

- While working on your Problem Set you can get help by posting your questions in the appropriate folder in the discussion forum. Note that sometimes, even though your code can generate the expected output, it may still not be correct as it may work for a specific data set and not for all valid data sets. Also, it may not use the programming concepts and best practices that we have emphasized in the course.

- Completing the Problem Sets will help you to gain hands-on experience with coding in Python and understand the learnt concepts well enough so that you can apply the concepts to solve and code the solution for the given problems. All of this will help you in doing well on your tests and exam.

- The files that you upload to the dropbox should be your source code (`.py`) files as practiced in Lab 0, and any other requested solution files.

- While coding solutions for the problems given below, keep in mind that on the test/exam you will also be marked on the following:

  - Efficient solution of the problem. A given problem can be solved in a number of different ways, but the best solution is the one that is efficient; ie., the one that uses the right concepts in a very productive way.
  - Including sufficient descriptive comments in your program. The person looking at your code should be able to understand how your code is solving the given problem even if the person reading your Python program does not know the Python language. In addition, the reader of your program should be able to understand what each variable represents.
  - Labelling of input and output. All input and output should have a descriptive label so that the reader of your program understands what input is expected and what output the program has generated.
  - Program style - consistent formatting and indentation of program statements, meaningful variable names (identifiers) and the use of constants (constant identifiers), where appropriate.

  Practicing these rules will build a good foundation for programming.

- This Problem Set is based on Chapters 1, 2, and 3 of the textbook, without the graphics components from each chapter. Please use only the following:

  - Simple assignment statement with expressions and arithmetic and string operators and escape sequences.
  - Do not use magic numbers (see textbook pages 30–31).
  - Functions such as `input()`, `print()`, `int()`, `float()`, `str()`, string functions, and, from the math module, the `pow()`, `sqrt()`, etc. functions.
  - String methods such as: `s.lower()`, `s.upper()`
  - Format specifiers (see textbook pages 50–51).
  - Conditional statements where appropriate.
- Rubrics/solution outlines for each Problem Set will be provided after the grades for the Problem Set have been released.

Solutions to these problems will not be posted, but you can get as much help as you need to complete your assignment.

With this understanding please complete the following questions:

---

1. (a) Trace (by hand) the execution of the following Python program by completing the trace table below. For Each line of code (as labelled with line numbers), you would enter the line number in the trace table, the state/value of the variable(s) that is/are affected within that line, and any output displayed by that line.

   Note: Tracing a program with test data is a good mechanism to see if your code is working.

   Python program:

```
from math import *
x = 13                                              #LINE 1
y = 2                                               #LINE 2
v = 120                                             #LINE 3
z = 9.0                                             #LINE 4
w = 1.15                                            #LINE 5
w = (x//y*100)                                      #LINE 6
v = x/y*100                                         #LINE 7
w = w/1000                                          #LINE 8
v = int(w+999/1000)                                 #LINE 9
w = 2.5+4*-1.5-(2.5+4)*-1.5                         #LINE 10
x += 1                                              #LINE 11
w = x%y * pow(x*y/4,y)                              #LINE 12
v = int((6+int(w))/x )                              #LINE 13
w = x*float(20/8)                                   #LINE 14
a =  y%x                                            #LINE 15
print("Sum of w,x,y is: " + 'w+x+y')                #LINE 16
print("Price per litre is $%.2f" %((w+x+y+z)% 59))  #LINE 17
b = "Supercalifragilisticexpialidocious"            #LINE 18
c = len(b)                                          #LINE 19
d = b[0:5]+" "+b[27].upper()+b[3]+b[7]+b[8]+b[-5:]  #LINE 20
```

Trace table:

Line number	x	y	v	z	w	Output
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Note: to submit your solution for this question you can create a new file in idle and save it with a `.txt` extension (rather than a `.py` extension).

(b) i. Given `x=45` and `y=4`, write a division or modulus expression that evaluates to `11`.

ii. Given `x=59` and `y=6`, write a division or modulus expression that evaluates to `5`.

iii. Given `i=23` and `me=4.0`, write a division or modulus expression that evaluates to `5.75`.

iv. What value is assigned to the variable `c` in the following statement?

```
c = "Hi" + " "+ "Ho" * 5
```

v. Rewrite the following mathematical expression as a Python expression. Use Python operators and/or math module functions as necessary. You may assume that the division is real division.

$$(x \bmod y)\left(1 + \frac{xy}{4}\right)^{y}$$

Note: to submit your solution for this question, include your Python expressions in a file called `operators.py`.

2. Write, test, and document a Python program, in a file called `MarsCoins.py`, to solve the following problem.

Marvin from Mars has a jar full of martian currency (Maruvians, Caruvians, Taruvians and Paruvians). The breakdown of the currency units on Mars is as follows:

- The single smallest unit of currency is a Paruvian.
- 6 Paruvians = 1 Taruvian
- 12 Paruvians = 1 Caruvian
- 24 Paruvians = 1 Maruvian

Marvin wants to share his money with his friends but he doesn't want to carry around all those marscoins. The marscoin that weighs the least is the Maruvian, so Marvin plans to go to the Martian Bank and exchange all his other marscoins for Maruvians, then share them evenly with his friends. Any leftover Paruvians and/or Maruvians will be put back in the jar.

Given the number of Maruvians, Caruvians, Taruvians and Paruvians originally in the jar, and the number of friends that Marvin wants to share with, write a Python program to compute the total number of Maruvians that Marvin will share equally between himself and his friends before placing the leftover Paruvians and Maruvians back in the jar. Your algorithm should output the total number of Maruvians that Marvin has after going to the bank (including the ones already in the jar), the number of Maruvians that each person gets, and the number of Paruvians and Maruvians that go back in the jar (if any). All output should be printed with appropriate labels.

Sample input/output:

```
Please enter the number of Maruvians: 10
Please enter the number of Caruvians: 6
Please enter the number of Taruvians: 12
Please enter the number of Paruvians: 20
Please enter the number of friends: 2
Marvin has 16 Maruvian(s) in total.
He gives 5 Maruvian(s) to himself and to each of his 2 friends.
Marvin puts 1 Maruvian(s) and 20 Paruvian(s) back in the jar.
```

3. *House of Ice Screams* is getting ready for summer and they plan to place three large wholesale orders for chocolate ice cream bars, one in June, one in July, and one in August. Note that the cost per bar is different for each order month as demand changes, hence the individual bar cost, based on the order month, is as follows:

Order Month	Cost Per Bar
June	\$0.50
July	\$1.00
August	\$0.75

When *House of Ice Screams* has received all the bars, they package them into two different size boxes for resale: small boxes (which contain 5 bars per box) and large boxes (which contain 20 bars per box). Half of the total bars ordered are packed in small boxes and half are packed in large boxes. You may ignore any leftover ice cream bars from either order.

Write a Python program to input the number of bars ordered for each of the three months and compute and output the following:

- The total number of small boxes packed (for all orders combined).
- The total number of large boxes packed (for all orders combined).
- The selling price of each box, which is computed as follows:

  Small Box Selling Price = average cost per bar (calculated by taking the total amount paid for all three orders, divided by the total number of bars ordered), multiplied by the number of bars per box, with a 90% markup on each bar in the box.

  Large Box Selling Price = average cost per bar (calculated by taking the total amount paid for all three orders, divided by the total number of bars ordered), multiplied by the number of bars per box, with an 80% markup on each bar in the box.

Sample input/output:

```
Enter the number of bars ordered in June: 400
Enter the number of bars ordered in July: 200
Enter the number of bars ordered in August: 120
Number of Small Boxes packed:  72
Number of Large Boxes packed:  18
Selling Price Per Small Box: $6.47
Selling Price Per Large Box: $24.50
```

Call your program `IceScream.py`.

4. The monthly payment to purchase insurance from Purple Cross Insurance is based on the insurance level of the plan (high, regular and low), the type of coverage (family or single) and the status (children or no children), as given in the table below:

Benefit Level	Persons covered		
	Single	Family	
		No children	Children
Low	$36.25	$56.50	$98.35
Regular	$48.90	$74.70	$136.75
High	$69.80	$99.45	$174.55

Write a Python program in a file called `insurance.py` to request input for the level of the plan, type of coverage, and status, and output the monthly payment.