CSE 101: Computer Science Principles (Fall 2019)

Homework #1

Assignment Due: Saturday, September 28, 2019, by 11:59 PM

Assignment Objectives

By the end of this assignment you should be able to describe and analyze simple algorithms, and develop original Python functions to solve simple programming problems involving variables, conditional statements, and loops.

Getting Started

Part of this assignment requires you to write Python code to solve several computational problems. To help you get started, we will give you a basic starter file for each problem. These files will contain *function stubs*[1] and a few tests you can try out to see if your code seems to be correct (**note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!**). You need to complete (fill in the bodies of) these functions for the assignments. **Do not, under any circumstances, change the names of the functions or their parameter lists.** The automated grading system will be looking for functions with those exact names and parameter lists; if you change any function headers, the grading program will reject your solution and mark it as incorrect.

Directions

Solve each of the following problems to the best of your ability. Each of the algorithm questions should be answered in a separate Word (DOCX) or PDF file. We have provided a code skeleton for each of the programming problems. The automated grading system will execute your solution to each programming problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points. This assignment contains 4 problems, and is worth a total of 50 points. **Note that not every problem may be worth the same number of points!**

- ⚠ Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work. **If you leave this information out, you may not receive credit for your work!**
- ⚠ Submit your work as a set of individual files (one per problem). **DO NOT** zip or otherwise compress your files, and **DO NOT** place all of your work in a single file. If you do so, we may not be able to grade your work and you will receive a failing grade for this assignment!
- ⚠ Each of your functions **MUST** use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).
- ⚠ Each of your functions must explicitly ***return*** its final answer; the grading program will ignore anything that your code prints out. Along those lines, do **NOT** use `input()` anywhere within your functions (or anywhere before the `if __name__ == "__main__":` statement); your functions should get all of their input from their parameters. Programs that crash will likely receive a failing grade, so test your code thoroughly **with Python 3.7.4 or later** before submitting it.
- ⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your completed work as directed by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or that is submitted before you have signed the course's Academic Honesty agreement.
- ⚠ **ALL** of the material that you submit (for each problem) **MUST** be your own work! You may not receive assistance from or share any materials with anyone else, except for the instructor and the (current) course TAs.

[1] Stubs are functions that have no bodies, or have very minimal, bodies

Part I: Sorting Marbles (10 points)

You have been given:

- A box filled with an unknown number of marbles; each marble is either red or blue
- Two plastic cups (one red cup and one blue cup)

In a Word or PDF file, create a detailed set of instructions that someone (e.g., a robot or one of your classmates) can follow to sort the marbles by color (by transferring each marble from the box to the cup of the same color). After all of the steps of the algorithm have been completed:

1. The red cup should hold **all** of the red marbles (and **only** the red marbles)
2. The blue cup should hold **all** of the blue marbles (and **only** the blue marbles)
3. The original box should be empty (in other words, no marbles should be left over)

You may assume that the person (or robot) carrying out the algorithm is only capable of performing very short, simple tasks like:

- determining whether a marble is red or blue
- determining whether or not a container (e.g., a box or a cup) is currently empty
- picking up a marble
- dropping/putting down a marble

In other words, your algorithm must break down the overall task into many small, simple operations. A statement like "Pick up a red marble and drop it into the red cup" is too complex to be a single instruction; that action must be restated as a sequence of two or more simpler steps.

Your solution must be in written format. By "written", we mean "capable of being recorded on paper" (so video or audio recordings don't count). Your instructions can take one of many possible forms: a sequence of directions in English, a flowchart, a set of illustrations, etc. **DO NOT EXPRESS YOUR SOLUTION USING PYTHON CODE; YOU WILL NOT RECEIVE ANY CREDIT FOR THIS PROBLEM IF YOU DO SO!**

Part II: Analyzing Algorithms (10 points)

Carefully examine the following algorithm description:

1. Get values for a and b
2. If (either a is 0 or b is 0), set the value of product to 0
3. Else:
   (a) Set the value of count to 0
   (b) Set the value of product to 0
   (c) While (count < b) do:
      i. Set the value of product to (product + a)
      ii. Set the value of count to (count + 1)
   (d) (End of loop)
4. Print the value of product

In a Word or PDF file, answer the following questions:

1. This algorithm is intended to multiple two integer values. Does it work? If so, show that it works by displaying, step by step, how it processes a pair of integers.
2. This algorithm works some (or most) of the time, but it contains a serious error. Explain what the error is, and show (step by step) how this error may cause the algorithm to produce the wrong answer in some circumstances.

Part III: Stardate Calculations (15 points)

(Place your answer to this problem in the "stardate.py" file)

Star Trek represents time as a numerical ***stardate***, instead of using month/day/year notation. Several people have developed algorithms to convert Gregorian dates into stardates; the formula below is from https://www.wikihow.com/Calculate-Stardates, simplified to ignore leap years. Note that this formula ***ONLY*** works correctly for years after 2323!

The Gregorian date-to-stardate conversion formula is

$$(1000 * (y - b)) + (\frac{1000}{n} * (m + d - 1))$$

where:

- *y* represents the 4-digit Gregorian year to be converted to a stardate (e.g., 2374)
- *b* represents the *standard base year*, which is always 2323 for our purposes
- *n* represents the number of days in the year. For now, we are ignoring leap years, so *n* will always be 365.
- *d* represents the day of the month.
- *m* is the *month number* (the number of days in the year prior to the start of that month). For example, January's month number is 0 (there are 0 days between January 1 and the start of the year). A standard 365-day year uses these values:

Month	Value of m	Month	Value of m	Month	Value of m
January (1)	0	May (5)	120	September (9)	243
February (2)	31	June (6)	151	October (10)	273
March (3)	59	July (7)	181	November (11)	304
April (4)	90	August (8)	212	December (12)	334

For example, suppose that we want to convert July 12, 2467 into a stardate using this formula. *y* is 2467 (the year to convert). The base year *b* is 2323. *n* is 365, the number of days in a year. July's month number (*m*) is 181 according to the table above. The day number *d* is 12. Plugging these values into the formula, we get $((1000 * (2467 - 2323)) + ((1000/365) * (181 + 12 - 1))$, or 144526.02739726 (**DO NOT** round your stardate value for this problem).

Complete the `stardate()` function, which takes three arguments in the following order:

1. an integer between 1 and 12 inclusive, representing the month (January is 1, December is 12).
2. an integer between 1 and 31 inclusive representing the day of the month. Assume that this value is always appropriate for the chosen month, e.g., if the month is 2 (February), this value will never exceed 28.
3. an integer representing a 4-digit year (2323 or greater).

Be sure to convert the month to the proper month number *m* before you perform your calculation. We recommend presetting *b* to 2323 and *n* to 365; those values will never change in our simplified formula.

Examples:

Gregorian Date	Function Call	Return Value
February 21, 2365	`stardate(2,21,2365)`	`42139.72602739726`
November 29, 2401	`stardate(11,29,2401)`	`78909.5890410959`
May 4, 2390	`stardate(5,4,2390)`	`67336.98630136986`

Part IV: Variable Drink Pricing (15 points)

(Place your answer to this problem in the "bill.py" file)

Variables is a popular new nightclub that has a very unusual pricing policy for the three drinks that it sells (called A, B, and C here for simplicity). At the start of each night, each drink costs $2. Each time a drink is purchased, its price goes up by $1 for the next patron (the prices change independently; if someone purchases drink A, its price goes up but the prices for B and C stay the same). In order to prevent prices from getting too high, a drink's price cannot exceed $5; after a particular drink is sold for $5, its price is reset to $2 and begins to rise again.

For example, suppose several drinks are purchased in the following order: A, A, C, B, B, A, A. The following would happen:

1. One serving of drink A is sold for $2. Its price rises to $3 for the next customer; our current sales total is $2.
2. Another serving of drink A is sold for $3. Its price rises to $4 for the next customer; our current sales total is $5.
3. One serving of drink C is sold for $2. Its price rises to $3 for the next customer; our current sales total is $7.
4. One serving of drink B is sold for $2. Its price rises to $3 for the next customer; our current sales total is $9.
5. Another serving of drink B is sold for $3. Its price rises to $4 for the next customer; our current sales total is $12.
6. Another serving of drink A is sold for $4. Its price rises to $5 for the next customer; our current sales total is $16.
7. Another serving of drink A is sold for $5. Its price is reset to $2 for the next customer; our current sales total is $21.

Complete the `bill()` function, which takes a string representing a sequence of drink orders from the user (a series of capital As, Bs, and Cs, with no spaces or other characters present). Your function should calculate and return an integer value corresponding to the total cost of all the drinks sold (in the example above, the total cost is $21, so your function would return the integer 21). You may assume that the input will only contain uppercase As, Bs, and Cs.

HINT: Use a loop to examine each letter in the string. Inside your loop, use an `if-elif-else` statement to process each letter appropriately.

Examples:

Function Call	Return Value
`bill("AACBBAA")`	`21`
`bill("BC")`	`4`
`bill("ABCABCAB")`	`23`
`bill("CCCC")`	`14`
`bill("BBBBBBBBBB")`	`33`