CSCI 141
Fall 2019
Assignment 1

Topics: Variables, `print`, `input`, operators

Introduction

For this first assignment, you will write three small Python programs. The following general guidelines and tips apply to all three programs.

Getting Started

Refer to lab 1, as well as the lecture slides, to review what you've learned so far. In this and future assignments, you may not have seen all the topics in lecture before the assignment is released, but they will be covered well before the deadline. As usual, seek help early if you get stuck: come talk to me or the TAs during office hours, or visit the CS mentors for help. Please keep track of approximately how much time you spend on both portions of this assignment. You will be asked to report your estimate on Canvas after you submit.

Collaboration and Academic Honesty

The programs you write solution MUST be authored solely by you. You can discuss the problems with your peers, but these discussions should happen away from computers and you should take a break before returning to write code to help ensure that you truly understand the answers. You may not copy another person's code, or have another person tell you what code to type. If you have any questions, or are unsure about whether a specific sort of collaboration violates academic honesty, please come talk to me.

Coding Style

All your programs must have a comment at the top stating the author, date, and a short description of your program. You should also include comments elsewhere in your code anytime you think your code is doing something non-obvious and a reader of your code would benefit from some explanation.

Your code should be written as clearly as possible (i.e., try to avoid the need for explanatory comments). Variable names should follow the guidelines discussed in lecture for sensible naming: neither too verbose nor too terse.

Valid Input and Error Checking

You may assume that the user is well-behaved and enters data as prompted. Your program is *not* required to check the user's input to make sure it's well-formed. Your program is allowed to throw an error if the input is bad (i.e., the user enters a string instead of a number).

Testing

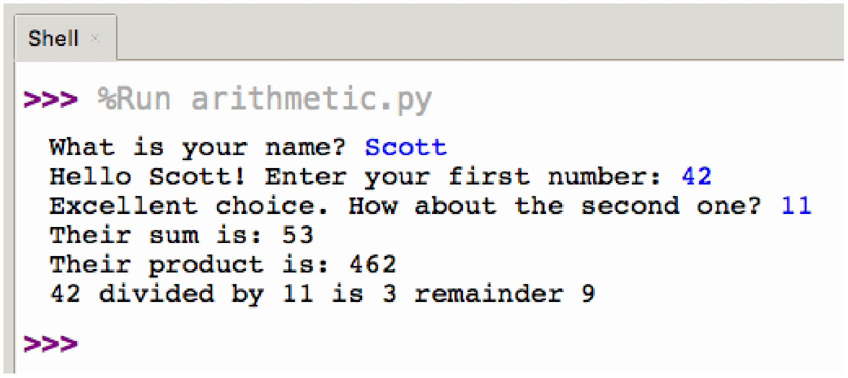
Testing is a major component in the process of writing software. Often, testing (detecting errors) and debugging (locating and fixing errors) takes way more effort than writing the code did in the first place. We'll talk more about testing as the quarter progresses; in the meantime, for each problem below we provide a table with some helpful test cases that you can use to see if your program is working correctly. Try your code out with the given inputs and make sure your output matches the program output specified in the table.

1 Arithmetic Homework Helper

Congratulations! You've just been hired as a Python programmer at an education start-up company. Your first task is to develop a prototype of a program that kindergarten students will use to check their homework assignments which involve addition, multiplication, and division problems. The program begins with a series of prompts, then prints a few lines to the screen in response. In total there are 6 lines that are printed each time the program is run:

1. Prompt the user for their name
2. Greet the user and ask them to supply the first positive integer
3. Prompt the user for a second positive integer
4. Output the sum of the two numbers
5. Output the product of the two numbers
6. Rephrase the division question, and output the whole number and remainder. All numerical outputs on the 6th line of output must be integers (whole numbers, without decimals).

A sample invocation of the program is shown in Figure 1:



```
>>> %Run arithmetic.py
  What is your name? Scott
  Hello Scott! Enter your first number: 42
  Excellent choice. How about the second one? 11
  Their sum is: 53
  Their product is: 462
  42 divided by 11 is 3 remainder 9
>>>
```

Figure 1: Sample Output

Although this is a simple set of steps, there are many, many different Python programs that can achieve it. The text of your prompts does not need to match the example exactly. However, your solution **must** follow the the instructions above exactly as specified. For example:

- Both the greeting and the prompt for the first number must be printed on the second line of output.
- The last (6th) line of output must rephrase the division question and output the whole number and remainder portions of the calculation on a single line.

Test Cases

First Integer	Second Integer	Sum	Product	Division
7	5	12	35	1 remainder 2
5	7	12	35	0 remainder 5
3	3	6	9	1 remainder 0
1	678	679	678	0 remainder 1
8364724	9738	8374462	81455682312	858 remainder 9520

2 Mortgage Calculator

Many online real estate websites have mortgage calculator features[1]. These calculators ask for some information, such as the price of a home, the down payment (amount of the home price you'd pay up front), and the interest rate, then calculate the amount you'd have to pay monthly on a loan for the home.

According to NerdWallet[2], the formula used to calculate the monthly payment based on these inputs is as follows:

$$M = (P - D)\frac{r(1+r)^N}{(1+r)^N - 1}$$

Where:

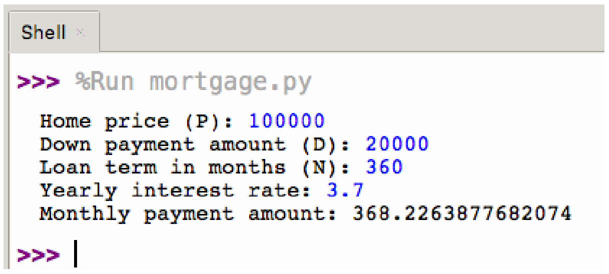
$M =$ The monthly payment
$P =$ The price of the home
$D =$ The down payment amount
$N =$ The number of months over which the loan will be paid off
$r = R * .01/12$, the monthly interest rate, which is the yearly percentage converted to a decimal and divided by

Write a program called `mortgage.py` that prompts the user to enter (one at a time, and in exactly the given order order) values for P, D, N, and R, and outputs the monthly payment amount M. Notice that you are asked to prompt the user for R, the annual interest rate as a percentage (e.g., 3.7), but the formula uses r, the monthly interest rate.

A sample invocation of the program is shown in Figure 2.

[1]See https://www.zillow.com/mortgage-calculator/ for an example

[2]Go to https://www.nerdwallet.com/mortgages/mortgage-calculator/calculate-mortgage-payment and click "How to calculate your mortgage payment" for the source of the formula


Shell

```
>>> %Run mortgage.py
  Home price (P): 100000
  Down payment amount (D): 20000
  Loan term in months (N): 360
  Yearly interest rate: 3.7
  Monthly payment amount: 368.2263877682074
>>> |
```

Figure 2: Sample Output

Test Cases

For brevity, the output is truncated after 3 decimal places in the table below; your program will output more decimal places (as in the example invocation).

P	D	N	R	Output (M)
100000	20000	360	3.7	368.226
1000000	10	180	3.4	7099.747
549050	103200	800	5.1	1960.773

3 Making Change

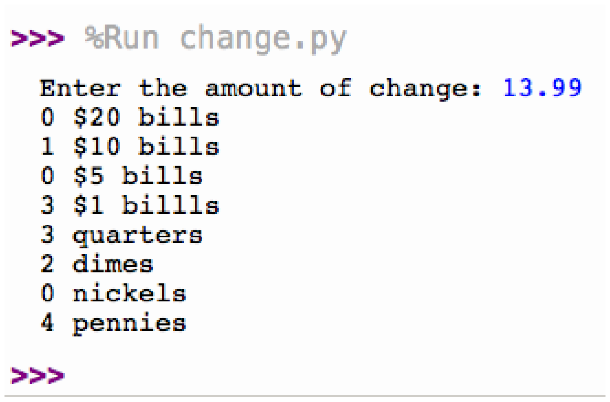
You are tasked with writing a software component for a self-checkout machine to be deployed in grocery stores. In particular, you will write a program that calculates how to give change to customers who paid with cash.

The program should begin by prompting the user to input the dollar amount of change the machine needs to dispense. Then, your program should calculate the most efficient way to give change and print the amount of each denomination required. The highest-value currency the machine stocks is $20 bills, and it has all the standard bills and coins all the way down to pennies.

If you've ever worked in retail, you know that the algorithm for giving change with the smallest number of bills and coins is quite simple: starting with the highest-valued bill or coin, use the largest denomination that's less than the remaining amount left. For example, to give change for $.60, I'd notice that a quarter ($.25) is the highest-valued denomination less than $.60, and I can use two of before going over $.60. I now have $.10 left, and I'd repeat this process until the total remaining is zero.

The number of each denomination should be printed as an integer (no decimals).

An example invocation of my solution program is shown in Figure 3.



```
>>> %Run change.py

  Enter the amount of change: 13.99
  0 $20 bills
  1 $10 bills
  0 $5 bills
  3 $1 billls
  3 quarters
  2 dimes
  0 nickels
  4 pennies

>>>
```

Figure 3: Sample Output

Test Cases

Input	$20s	$10s	$5s	$1s	Quarters	Dimes	Nickels	Pennies
13.99	0	1	0	3	3	2	0	4
209.14	10	0	1	4	0	1	0	4
36.41	1	1	1	1	1	1	1	1
0.32	0	0	0	0	1	0	1	2

Submission

Double check that your programs work according to the specification and produce the output given in the test cases. Take a look through the rubric below and make sure you won't lose points for reasons that could easily be foreseen and fixed. When you're finished, submit each of your programs as a `.py` file named `arithmetic.py`, `mortgage.py`, and `change.py`, respectively. Finally, fill out the A1 Hours quiz with an estimate of the number of hours you spent on A1.

Rubric

`arithmetic.py` (15 points)

Author, date, and program description given in a comment at the top of the file	1 point
Code is commented adequately and variables are appropriately named	1 points
Program prompts for user's name on the first line	3 points
Greeting on second line includes user's name	3 points
First integer prompt also appears on second line	2 points
Correct sum output on fourth line	1 points
Correct product output on fifth line	1 points
Division question is rephrased, quotient and remainder are printed on sixth line	3 points

`mortgage.py` (15 points)

Author, date, and program description given in a comment at the top of the file	1 point
Code is commented adequately and variables are appropriately named	1 points
Prompts for the correct values	4 points
Prompts for the values in the correct order	4 points
Produces the correct output	5 points

`change.py` (15 points)

Author, date, and program description given in a comment at the top of the file	1 point
Code is commented adequately and variables are appropriately named	1 points
Prompts the user for an amount of money	3 points
Calculates a correct way to give change	4 points
Calculates the most efficient way to give change	4 points
Outputs the number of each denomination as an integer	2 points
Total	45 points