

CS Scholars - Programming Hw1 - Written

Due Date: Friday 07/08 EOD

Name:

AndrewID:

For full credit on the assignment, complete either all Review + Core problems (#1-#10) or all Core + Spicy problems (#3-#12).

Bonus problems are related to the Advanced Track content, and are optional.

Review Problems - Written

[#1 - Running Code](#)

[#2 - Function Calls](#)

Core Problems - Written

[#3 - Algorithms and Abstraction](#)

[#4 - Writing Algorithms](#)

[#5 - Reading Code](#)

[#6 - Function Components](#)

Core Problems - Programming

[#7 - Data Types](#)

[#8 - Printing](#)

[#9 - Using Function Calls](#)

[#10 - Graphics](#)

Spicy Problems - Programming

[#11 - Advanced Math](#)

[#12 - Advanced Graphics](#)

Bonus Problems

[Advanced Programming - External Libraries](#)

[Advanced Computer Science - Data Representation](#)

Review Problems - Written

#1 - Running Code

Can attempt after Programming Basics lecture

The following question is intended to make you feel more comfortable with running code and encountering errors. In each of the following examples, copy the line of code into the interpreter (next to `>>>`) and press Enter to run it. Then copy the output in the interpreter into the space below the code, and check a box below that to indicate whether the code ran successfully or raised an error.

5 / (4 - 2)

Ran Successfully Raised an Error

"Hello World"

Ran Successfully Raised an Error

(8 + 3) < (5 * 2)

Ran Successfully Raised an Error

8 + "two"

Ran Successfully Raised an Error

#2 - Function Calls

Can attempt after Function Calls lecture

What does each of the following expressions **evaluate** to?

Hint: what is the *returned value* of each function call?

Don't just copy and run this code to find the answer- try to determine what the answer is yourself!

```
int(15.11)
```

```
print(5 + (3 * 2))
```

```
round(abs(-19.86), 0)
```

```
math.ceil(6.2) # assume we already imported math
```

Core Problems - Written

#3 - Algorithms and Abstraction

Can attempt after Introductions and Algorithms lecture

In lecture we defined two central concepts of computing: **abstraction** and **algorithms**.

First, in your own words, briefly provide your own example of a non-computer **algorithm** that could be used in the real world.

Example: the peanut-butter-and-jelly sandwich recipes we came up with in class are real-world algorithms.

Second, still in your own words, provide your own brief example of non-computational **abstraction** that can be found in the real world.

Example: when thinking of animals, you could refer to a creature as a mammal (high abstraction), a dog (medium abstraction), or a Siberian husky (low abstraction).


#4 - Writing Algorithms

Can attempt after Introductions and Algorithms lecture

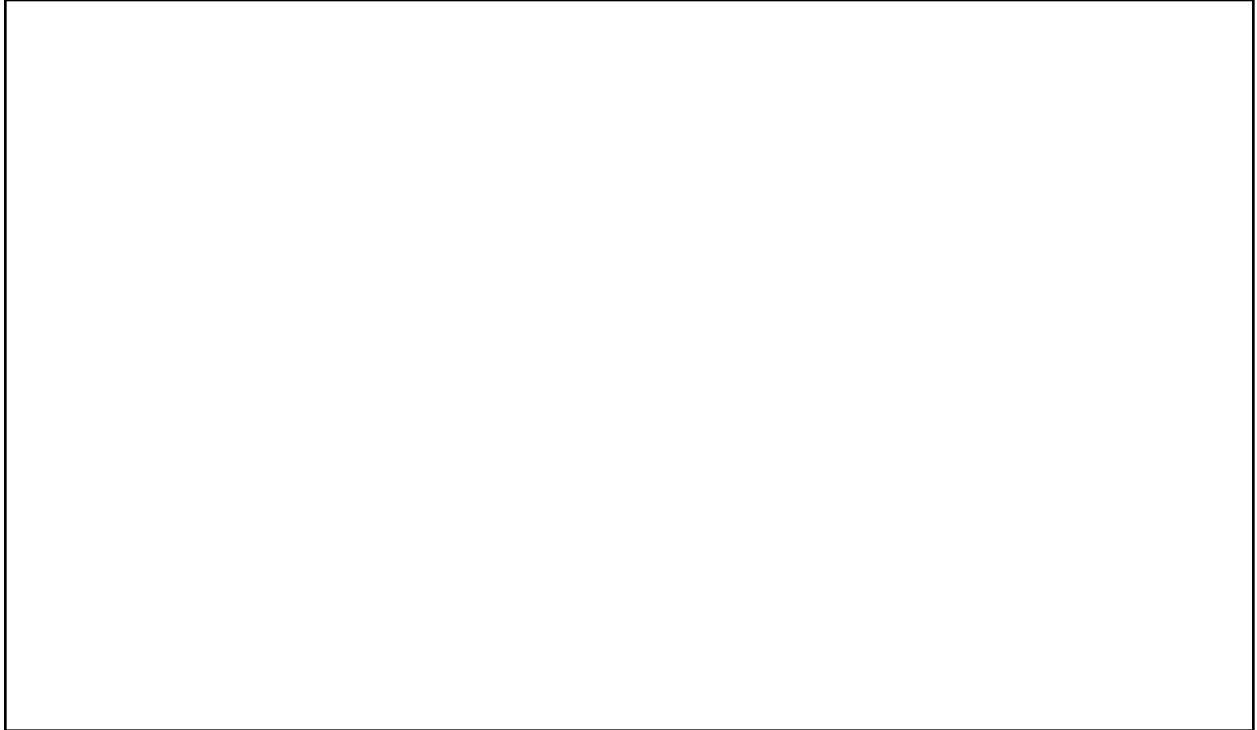
In this problem, you will write plain-language algorithms (not code!) at three levels of abstraction. Assume all of your instructions will be provided verbally (no pictures).

Do not write more than 100 words per question (and can write much less!).

First, write an algorithm at a **low** level of abstraction that instructs a person on how to write the capital letter L. Assume the person you are instructing has almost no prior knowledge- they know directions (up/down/left/right/etc) but nothing else about writing.

A large empty rectangular box with a thin black border, intended for the student to write their algorithm for writing the capital letter L. The box is currently blank.

Second, write an algorithm at a **medium** level of abstraction that instructs a person on how to write the word 'ALL' in English, in all capital letters. This time you can assume the person you're instructing has a little more prior knowledge- what paper and a pen/pencil are, how to draw straight lines, etc.



Finally, if you wanted to provide an algorithm on how to write the word 'ALL' in English at a **high** level of abstraction, what additional starting knowledge would you give the person being instructed?



#5 - Reading Code

Can attempt after Programming Basics lecture

Assume the following lines of code are written in the editor and run as a script. Your job is to figure out what will be shown in the interpreter as a result. Don't just copy and run this code to find the answer- try to determine what the answer is yourself!

Next to each line in the table, write what will appear in the interpreter based on that line of code running from the editor. If nothing should appear, leave the box blank. We've filled in the first two entries for you as an example.

Code Line	Interpreter Output
age = 18	
print("test")	test
print("age")	
print(age)	
age + 5	
print(age / 3)	
print(age // 3)	
print(age < 18)	
# print(age * 3)	
print(age - 10 * 2)	
age = age + 1	
print("Age:", age)	
age -= 2	
print(age)	

#6 - Function Components

Can attempt after Function Calls lecture

Each of the following code snippets contains a function call. Identify and describe the **function name**, **argument value(s)** and **returned value** of each call. If there is no name / argument / returned value, leave the space blank.

```
print("Result:", 10 ** 2)
```

Name	Argument Value(s)	Returned Value

```
import random  
num = random.random()
```

Name	Argument Value(s)	Returned Value

```
x = 3  
y = 2  
z = pow(x, y)
```

Name	Argument Value(s)	Returned Value

Programming Problems

Each of these problems should be solved in the starter file available on the course website. They should be submitted to the Gradescope assignment Hw1 - Programming to be autograded. Make sure to check the autograder feedback after you submit!

For each of these problems (unless otherwise specified), write the needed code directly in the Python file **under** the comment and print statement that correspond to the problem. Do not delete the provided print statements- we're using them to autograde.

Core Problems - Programming

#7 - Data Types

Can attempt after Programming Basics lecture

Write Python code at the top level of the file to do the following:

1. Assign the integer 15 to the variable **a**.
2. Assign the float 3.14 to the variable **b**.
3. Assign the string "20" to the variable **c**.
4. Assign the boolean True to the variable **d**.
5. Evaluate 5 minus 1.7 and assign that expression to the variable **e**.
6. Check whether 8 is less than 5 and assign that expression to the variable **f**.
7. Reassign the variable **a** to hold the value 45.
8. Concatenate **c** and "21" and assign the result to variable **g**. Don't change the value in **c**.

Feel free to print any of these variables to check your work.

#8 - Printing

Can attempt after Programming Basics lecture

Write code at the top level of the file to match the following algorithm.

1. Assign the string `Kelly` to **prof**.
2. Assign a string holding another CS Scholars student's name to **student**.
3. Write a single print statement that greets both Prof. Kelly and the student by name. The statement must use the variables **prof** and **student**, as well as at least one additional string.

#9 - Using Function Calls

Can attempt after Function Calls lecture

Write code at the top level of the file to match the following algorithm:

1. Import the **random** library and the **math** library
2. Generate a random integer between `[1, 360]` and store it in the variable **x**.
3. Convert the integer value in **x** to a radian number with a function in the **math** library and store the result in the variable **r**. *Hint:* use the **radians** function.
4. Write a single print statement that describes the relationship between the values in **x** and **r**. The print statement must use both variables, as well as at least one additional string.

#10 - Graphics

Can attempt after Function Calls lecture

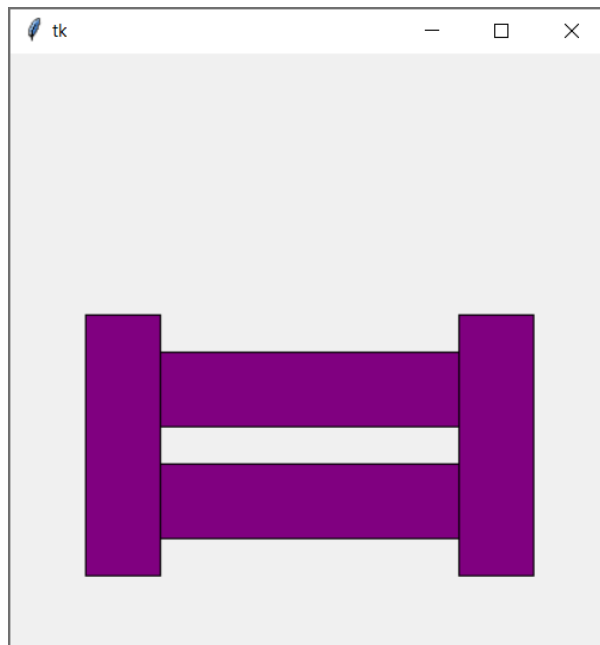
Add code to the location designated by `# write your code here #` that draws a simple version of the CMU Fence on the canvas. If you have not yet heard of the Fence, learn more here:

www.amusingplanet.com/2014/09/the-fence-of-carnegie-mellon-university.html

Your Fence must meet the following basic requirements, but otherwise you may customize it as much as you like.

- The Fence must have at least two columns, with one column ending the left side of the Fence and one ending the right
- The Fence must have two cross-boards that connect all the columns
- There must be a gap above the top cross-board, between the cross-boards, and below the bottom cross-board
- The Fence must be painted at least one color

Here's an example of a simple Fence that meets the requirements:



Spicy Problems - Programming

#11 - Advanced Math

Can attempt after Programming Basics lecture

Set up two variables, **n** and **k**, so that **n** holds some integer with at least four digits, and **k** is some integer between 0-3. Then write code that finds the **k**'th digit of **n** from the right. You should count from 0, so **k=0** refers to the ones-digit, **k=1** refers to the tens digit, etc.

For example, with **n=9876** and **k=2**, the **k**'th digit from the right is 8 (the hundreds digit).

Once you've found the **k**'th digit, store it in a variable called **result**, then print out "kth digit:", then the result, all on one line.

Note: you only need to run this code on one example, but it should work if you change **n** and **k** to other numbers. However, you can assume that both numbers are integers, and that both will not be negative. Try testing your code on **n = 789** with **k** equal to 0, 1, 2, and 3 to see if you get the right digits (9, 8, 7, and 0).

Hint: you'll want to use the **div** and **mod** operators to modify the number.

#12 - Advanced Graphics

Can attempt after Function Calls lecture

Add code to the location designated by `# write your code here - spicy #` to draw a robot of your own design on the Tkinter canvas.

Your robot can look like whatever you want, but for full credit it should use:

- At least 10 shapes total, including at least one oval, one rectangle, one non-rectangular polygon, and one line.
- At least 2 different optional parameters (like fill or width).

Important note: the last line of the Tkinter code, `root.mainloop()`, has been commented out since not all students will attempt this problem. Remove the comment to see your robot.

Bonus Problems

Advanced Programming - External Libraries

Choose an external library from the advanced programming slides. Install this library on your machine and write a simple program (at least 5-10 lines of code) to do something interesting with that library.

Your options for libraries might be limited based on your current programming knowledge. Here's a quick breakdown of which libraries you should be able to do something interesting with based on different levels of knowledge.

Just data, variables, and function calls:

- OpenCV
- Pillow

Also function definitions, conditionals, and loops:

- Pygame
- Vpython

Also lists:

- NumPy
- SciPy
- Matplotlib
- Pandas
- Pydub

Also dictionaries:

- nltk
- BeautifulSoup

Also objects:

- Django
- Flask

You should complete this work in a separate file from hw1.py, and upload that file to Hw1 - Bonus, so that you don't break the autograder.

Advanced Computer Science - Data Representation

For each of the following problems, you must **show your work** to receive full credit. For example, to convert 0101 to decimal, you could show $0*8 + 1*4 + 0*2 + 1*1 = 4 + 1 = 5$.

Convert 29 from decimal to binary.

Work:	
Answer:	

Convert 1010101 from binary to decimal.

Work:	
Answer:	

Convert the following number from binary to ascii. You may wish to refer to this chart:
www.asciitable.com

01001111 01001011

Work:	
Answer:	

Convert the following three numbers from binary to decimal. Then enter the decimal numbers into the respective R, G, and B values here:

https://www.w3schools.com/colors/colors_rgb.asp .

What color does this binary represent?

R: 11100110 **G:** 11110000 **B:** 00111100

Work:	
Answer:	