HW 1: Data types and arithmetic operations

Due Sep 22nd, 6pm, codePost.io
Please submit all files created for this assignment to codePost, including the text file. You can submit as many times as you like up until the deadline. When you are finished with the homework, please finalize your submission (this a toggle you'll find in codePost after you upload files). Otherwise we will assume you are using your late days.

Familiarize yourself with how to approach a programming assignment and review the grading rubric on this assignment's page in Canvas before getting started.

Important: codePost is buggy!
In our first lab we encountered several bugs when using codePost. Luckily, workarounds were found for all bugs so far. This is my first semester using codePost for auto-grading and I expect we may encounter more problems. If you have any issues with codePost, please check the list of known bugs and solutions, which you can find by going to Canvas and clicking on the Resources link in the course navigation. This page will be updated as issues arise. If your bug is not on the list, reach out to an instructor right away.

Code style and documentation
Make sure you review the PEP 8 Style Guide. There's a lot in there, so for this homework focus on using the codePost autograder to identify and fix style violations. For this homework, we won't deduct points for any style violations but the style checker will still run. If codePost style tests fail, try to fix them—on future assignments, a failed style test will result in points deducted.

The formatting guidelines introduced in Lab 1 are ***requirements*** for all assignments: every file must have a file comment and your programs must start with `main()`.

Written Component (10 pts)

Save your answers in a plain text file called written.txt.

Question 1: What is the data type of each of the following values? (0.5pts each)

a) 48.25
b) 48
c) -1
d) True
e) "False"
f) -5.0
g) 'hello'
h) 0

Question 2: What do the following expressions evaluate to? (0.5pts each)

a) 7 / 5
b) 7.0 / 5.0
c) 14 // 10
d) 14 % 10
e) 6 ** 2
f) 6 % 2
g) “red” + “1234”
h) “yellow” * 2

Question 3: If you were to enter the following lines into the Python terminal, the third line would result in an error:

```
print(“hello”)
print(“hello”, “world”)
print(hello)
```

Explain the cause of the error. (1pt)

Question 4: What does the following code print to the console if the user enters 10? (1pt)

```
user_number = int(input(“Enter a whole number greater than 0”))
x = user_number // 3
x = x ** 2
print(x)
```

Programming Component (50 pts)

Problem 1: racepace.py

Write a program that runners can use to calculate statistics about a race time. Open up VS Code and create a new file called racepace.py.

Your program should prompt the user to enter the following information, in this order:

- The distance they ran in kilometers (float). Use this prompt: `“How many kilometers did you run? “`
- The time it took them. You’ll have to prompt the user to enter each piece separately:
    - The number of hours (int - if it took less than an hour, the user can enter 0). Use this prompt: `“What was your finish time? Enter hours: “`
    - The number of minutes (int). Use this prompt: `“Enter minutes: “`

You can assume the user will only enter valid inputs: non-negative numbers, either float or int as specified.

Based on the user's inputs, your program should report the following information back to the user:

- The number of miles they ran, rounded to 2 decimal places.
- Their average pace per mile in minutes and seconds.
- Their average speed in miles per hour, rounded to 2 decimal places.

Additional things you need to know:

- There are 1.61 kilometers in a mile.
- Use Python's built-in round function to round a float to a given number decimal places. E.g. `x = round(9.185, 2)` will round 9.185 to 2 decimal places and assign the result to x. To round a float to 0 decimal places, use `round(number_to_round)` e.g. `round(9.185)` will return 9.
- When concatenating strings and numbers together, you will need to change the type of numeric variables to string using Python's built-in str function. This function works just like the type conversion functions you saw in lecture, `int()` and `float()`. For example, if I have a variable called `average_slices` that stores the average number of pizza slices eaten by guests at a pizza party. To create a string that includes some text and the value of that variable, I would use something like the following statement:
  `msg = "Party guests ate " + str(average_slices) + " slices on average"`

Required output format: Bolded text between <> should be replaced with the calculated values

<miles run to 2 decimal places> `miles,` **<minutes:seconds>** `pace,` **<speed to 2 decimal places>** `MPH`

Note: Your input prompts and the format of your program's output must match the format given exactly otherwise tests will fail even if your program calculates the correct values.

Example input / output (user input is shown in blue)

```
How many kilometers did you run? 5
What was your finish time? Enter hours: 0
Enter minutes: 30
3.11 miles, 9:40 pace, 6.21 MPH
```

Include 3 test cases in your file comment. A test case should show the expected output for example input. The exact format doesn't matter as long as we can tell what you expect your program to output based on certain input. E.g., the test case for the example above might look like this:

```
5km, 0 hours, 30 minutes => 3.11 miles, 9:40 pace, 6.21 MPH
```

Feel free to use this test case in your file comment but make sure to include 2 more of your own.

Problem 2: tables.py

A table-making robot assembles tables from the parts it's given. The robot requires the following ingredients to make a table:

- 1 table top
- 4 legs
- 8 screws

The robot needs *all* those parts to make a table. For example, if you give it 99 table tops, 400 legs, and 405 screws, it will make 50 tables. Leftover there will be 49 table tops, 200 legs, and 5 screws.

Write a program to run the robot. Open up VS Code and create a new file called tables.py.

Your program should:

- Prompt the user to enter the number of table tops. Use this prompt: `"Number of tops: "`
- Prompt the user to enter the number of legs they have. Use this prompt: `"Number of legs: "`
- Prompt the user to enter the number of screws they have. Use this prompt: `"Number of screws: "`
- Report the total number of tables that can be made with all the ingredients available. For this step, you might find Python's built-in `min()` function helpful. `min()` returns the minimum of several numbers. E.g. `min(4, 10, 2)` would return 2.
- Report the amount leftover of each ingredient after the sandwiches are made.

You can assume the user will only enter valid inputs: non-negative integers.

Required output format: Bolded text between <> should be replaced with the calculated values

`<number of tables>` `tables assembled. Leftover parts:` **`<remaining table tops>`** `table tops,` **`<remaining legs>`** `legs,` **`<remaining screws>`** `screws.`

Example input / output (user input is shown in blue)

```
Number of tops: 4
Number of legs: 20
Number of screws: 32
4 tables assembled. Leftover parts: 0 table tops, 4 legs, 0 screws.
```

Include 3 test cases in your file comment. A test case should show the expected output for given input. The exact format doesn't matter as long as we can tell what you expect your program to output based on example input. E.g., the test case for the example above might look like this:

```
4 tops, 20 legs, 32 screws => 4 tables assembled. Leftover parts: 0
table tops, 4 legs, 0 screws.
```

Feel free to use this test case in your file comment but make sure to include 2 more of your own.