Homework 1: Data Types, Functions and Conditionals

Due Sunday January 19 at 9:00 pm

24 points

Homework tips

1. **Start early!** If you run into trouble installing things or importing packages, it's best to find those problems well in advance, not the night before your assignment is due when we cannot help you!
2. **Make sure you back up your work!** At a minimum, do your work in a Dropbox folder. Better yet, use `git`, which is well worth your time and effort to learn.
3. **Be careful to follow directions!** Remember that Python is case sensitive. If you are ask you to define a function called `my_function` and you define a function called `My_Function`, you will not receive full credit. You may want to copy-paste the function names below to make sure that the functions in your notebook match.

Instructions for writing and submitting your homework

Your homework should be written in a jupyter notebook file. I have made a template available on Canvas, under "files/homework". You will submit, via Canvas, a zip file called `yourUniqueName_hwX.zip`, where `X` is the homework number. So, if I were to hand in a file for this assignment, homework 1, it would be called `regier_hw1.zip`. `zip` is the file extension for files compressed using the zip program; see `https://en.wikipedia.org/wiki/Zip_(file_format)`. You can make .zip files using the program `zip` or `gzip` in UNIX-style operating systems (e.g., Linux and Mac OS), and can also be run in the Windows program `cygwin`, which emulates a UNIX command line. Alternatively, many graphical user interfaces (GUIs) exist for creating zip archives.

When I extract your compressed file, the result should be a directory, also called `yourUniqueName_hwX`. In that directory, at a minimum, should be a jupyter notebook file, called `yourUniqueName.hwX.ipynb`, where again `X` is the number of the current homework. You should feel free to define supplementary functions in other Python scripts, which you should include in your compressed directory. So, for example, if the code in your notebook file imports a function from a Python file called `supplementary.py`, then the file `supplementary.py` should be included in your submission. In short, I should be able to extract your archived file and run your notebook file on my own machine by opening it in jupyter and clicking, for example, `Cells->Run all`. Importantly, please ensure that none of the code in your submitted notebook file results in errors. Errors in your code cause problems for our auto-grader. Thus, even though we frequently ask you to check for errors in your functions, you should not include in your submission any examples of your functions actually raising those errors. Please include your solutions for all problems in this homework in a single Python notebook.

Please include in your notebook file a list of any and all people with whom you discussed this homework assignment. Even if you discussed the assignment with others, your homework submission must not trigger the MOSS plagerism detector.

Please also include an estimate of how many hours you spent on each of the sections of this homework assignment.

1 Severance textbook exercises (8 points)

Chapter 1, Exercises 3, 5, 6, 8
Chapter 2, Exercises 3, 4
Chapter 3, Exercises 1, 2
Chapter 4, Exercise 6
Chapter 5, Exercise 1

2 Euclid's algorithm (3 points)

Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm`) is a method for finding the greatest common divisor (GCD) of two numbers. Recall that the GCD of two numbers m and n is the largest number that divides both m and n.

1. The Wikipedia page above includes several pseudocode implementations of Euclid's algorithm. Choose one of these, and use it to implement a function `gcd`, which takes two integers as its arguments and returns their GCD. You may assume that both inputs are integers, so there is no need to include any error checking in your function. **Note:** this is one of the rare occasions where you have my explicit permission to look up your answer. Unless otherwise stated (e.g., as in this problem), looking up solutions on Wikipedia or in any other non-class resource will be considered cheating!

2. Use your function to evaluate the GCDs of the following pairs of numbers:

   (a) 2019, 2020

   (b) 1600, 400

   (c) 5040, 60

3. What does your function do if one or both of its arguments are negative? Does this behavior make sense?

3 Approximating Euler's number e (8 points)

The base of the natural logarithm, e, is typically defined as the infinite sum

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \dots, \tag{1}$$

where $k!$ denotes the factorial of k,

$$k! = k \cdot (k-1) \cdot (k-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1,$$

where we define $0! = 1$ by convention. For more on Euler's number, see `https://en.wikipedia.org/wiki/E_(mathematical_constant)`. In this problem, we will explore different approaches to approximating this number.

1. An early characterization of Euler's number, due to Jacob Bernoulli, was as the limit of
$$\left(1 + \frac{1}{x}\right)^x \tag{2}$$
as $x \to \infty$. Define a function called `euler_limit` that takes as an argument an integer n, and returns a float that approximates e by taking $x = n$ in Equation (2). You may assume that the input to your function will be a positive integer (of course, your function will still run just fine if n is, say, a positive float, but we will only use integer values in what follows).

2. What happens when you call `euler_limit(n)` for really huge values of `n` (say, 10^{16} or 10^{18})? Why does this happen? **Hint:** the answer has to do with floating point arithmetic. Think about how $1/n$ is represented on your computer when n is big.

3. Define a function called `euler_sum` that takes a single non-negative integer argument `n`, and returns an approximation to e based on the first `n` terms of the sum in Equation (1). Your function should return a float. You may assume that the input will be a non-negative integer, so you do not need to include error checking in your function. As an example, `euler_sum(4)` should return the sum of the first four terms in Equation 1, so that `euler_sum(4)` returns $1 + 1 + 1/2 + 1/6 \approx 2.667$. **Note:** the sum in Equation 1 starts counting with $k = 0$ (i.e., it is "0-indexed"), while our function starts counting with $n = 1$ (i.e., it is "1-indexed"). `euler_sum(1)` should use *one* term from Equation (1), so that `euler_sum(1)` returns 1. Similarly, `euler_sum(0)` should return 0, since by convention an empty sum is equal to zero. **Note:** you may use the `math.factorial` function to compute $k!$, but I recommend, for the sake of practice, implementing the factorial function yourself, as it is a nice example of a problem that is easily solved with recursion.

4. Define a function called `euler_approx` that takes a single argument, a positive float `epsilon`, and uses the sum in (1) to obtain an approximation of e that is within `epsilon` of the true value of e. **Hint:** use a while-loop. **Note:** you can use the Python math module to get the true value of e (up to floating point accuracy): `math.exp(1)`.

5. Define functions called `print_euler_sum_table` and `print_euler_lim_table` that each takes a single positive integer `n` as an argument and prints the error between the approximation given by, respectively, `euler_sum(k)` or `euler_limit(k)` and the true value of e as `k` ranges from 1 to `n`, one per line. That is, the functions should print a table of the the approximation error for different choices of k.

6. Which of these two approximations is better?

4 Testing Properties of an Integer (5 points)

In this problem, you'll get a bit more practice working with conditionals, and a first exposure to the kind of thinking that is required in a typical "coding interview" question. A positive integer n is a *power of* 2 if $n = 2^p$ for some integer $p \geq 0$.

1. Write a function `is_power_of_2` that takes a positive integer as its only argument and returns a Boolean indicating whether or not the input is a power of 2. You may assume that the input is a positive integer. You **may not** use the built-in `math.sqrt` function in your solution. You should need only the division and modulus (`%`) operations. **Hint:** the simplest solution to this problem makes use of recursion, though recursion is not necessary.

2. Generalize your previous solution to a function `is_power` that takes two positive integers as its arguments, `b` and `n`, in that order, and returns a Boolean. `is_power(b,n)` should return `True` if `n` is a power of `b` and `False` otherwise.