Homework 1: Data Types, Functions and Conditionals

Due February 9, 11:59 pm

Worth 10 points

Instructions on writing and submitting your homework can be found on the course webpage at `http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2022/STAT679/hw_instructions.html`. *Failure to follow these instructions will result in lost points.* Please direct any questions the instructor.

Read this first. A few things to bring to your attention:

1. Start early! If you run into trouble or you have questions, it's best to find those problems well in advance, not in the hours before your assignment is due!
2. If you have clarifying questions or you run into issues, please do not email the instructor directly. Instead, post to the discussion board so that your classmates can benefit as well if they have the same question.
3. **Make sure you back up your work!** I recommend, at a minimum, doing your work in a Dropbox folder or, better yet, using `git`, which is well worth your time and effort to learn.

1 Warm up: Defining Simple Functions (2 points)

In this problem, you will get practice defining simple functions in Python.

1. Define a function called `kitten`, which takes no arguments and prints the string `meow` when called. **Note:** be careful about the difference between `print` and `return` in Python!
2. Define a function called `bird_pad`, which takes a string as its only argument, and prints that string, prepended and appended with the string `bird`. So, `bird_pad('goat')` should produce the output

   `birdgoatbird` ,

   `bird_pad('_')` should produce the output

   `bird_bird`

   and so on. You may assume that the input is a string, so there is no need to perform any error checking in your function.
3. Define a function called `print_n`, which takes two arguments, a string `s` and a non-negative integer `n` (in that order), and prints the string `n` times, each on a separate

line. You may assume that `s` is a string and that `n` is a non-negative integer. So, for example, `print_n( 'cat', 3 )` should produce the output

```
cat
cat
cat
```

Hint: be careful of the case where `n` is 0.

2 Euclid's algorithm (2 points)

Euclid's algorithm[1] 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! **Another note:** many Python novices confuse `print` and `return`. Be careful of the difference!
2. Use your function to evaluate the GCDs of the following pairs of numbers (the first three are simple enough that you should be able to compute the correct answers by hand, so it's an easy way to verify that your function is working correctly):
   (a) 1200, 300
   (b) 5040, 60
   (c) 29, 31
   (d) 2021, 2022
3. What does your function do if one or both of its arguments are negative? Does this behavior make sense? Try running `gcd` with one or both arguments negative and write a sentence or two in a text block discussing what you see.

3 Approximating Euler's number e (4 points)

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

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

where $k!$ denotes the factorial of k,

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

[1] `https://en.wikipedia.org/wiki/Euclidean_algorithm`

where we define $0! = 1$ by convention.[2] In this problem, we will explore different approaches to approximating this number. You may assume that all inputs are as specified, so there is no need for error checking.

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 its only argument an integer n, and returns a float that approximates e by taking $x = n$ in Equation (2).

2. To compute the sum in Equation (1), we need to be able to compute factorials. Recall that for non-negative integer k, we define
$$k! = \begin{cases} 1 & \text{if } k = 0 \\ k \cdot (k-1)! & \text{otherwise.} \end{cases}$$
Define a function `factorial` that takes a non-negative integer `k` as its only argument and returns the factorial of `k`. You may assume that `k` is a non-negative integer, so there is no need for error checking. Your function should use recursion– namely the fact that $k! = k \cdot (k-1)!$. You **may not** use the built-in `math.factorial` function in your function, but you are free to use it to check that your function is working correctly.

3. Define a function called `euler_infinite_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. As an example, `euler_infinite_sum(4)` should return the sum of the first four terms in Equation 1, $1 + 1 + 1/2 + 1/6 \approx 2.667$. **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_infinite_sum(1)` should use *one* term from Equation (1), so that `euler_infinite_sum(1)` returns 1. Similarly, `euler_infinite_sum(0)` should return 0, since by convention an empty sum is equal to zero. **Note:** if you did not complete the previous part of the problem, or if you are not sure your implementation is correct, you may use the `math.factorial` function to compute $k!$.

4. Define a function called `euler_approx` that takes a single argument, a positive float `epsilon`, and returns the smallest number of terms in the sum in (1) required 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`, each of which takes a single positive integer `n` as an argument and prints the successive values obtained from `euler_infinite_sum(k)` or `euler_limit(k)` as `k` ranges from 1 to `n`, one per line.

6. Compare these two approximations. Which one approaches the true value of e faster? A single sentence will do, here.

[2] For more on Euler's number, see `https://en.wikipedia.org/wiki/E_(mathematical_constant)`.

4 Testing Properties of an Integer (2 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. An integer n is a *power of* 2 if $n = 2^p$ for some integer p. There is no need for error checking in this problem.

1. Write a function `is_power_of_2` that takes an integer as its only argument and returns a Boolean indicating whether or not the input is a power of 2. That is, `is_power_of_2(n)` should return `True` if `n` is a power of 2 and `False` otherwise. You **may not** use the built-in `math.sqrt` function or any other functions from the `math` module in your solution. Indeed, you should need only the division and modulus (`%`) operations (and there is a solution that uses only division). **Hint:** the simplest solution to this problem makes use of recursion, though recursion is not strictly necessary.

2. Generalize your previous solution to a function `is_power` that takes two integers, b and n, as arguments, and returns a Boolean. `is_power(b,n)` should return `True` if n is a power of b (i.e., $n = b^p$ for some integer p) and `False` otherwise. You may assume that b is non-negative. **Hint:** Be careful of the edge case where `b=0`. By convention, $0^0 = 1$, and $0^k = 0 \cdot 0^{k-1}$ for $k \geq 1$.