CSC336: Homework 1

Due May 12 2020 before 11:59pm

General instructions

Please read the following instructions carefully before starting. They contain important information about general expectations, submission instructions, and reminders of course policies.

- Your work is graded on both correctness and clarity of communication. Solutions that are technically correct but poorly written will not receive full marks. Please read over your solutions carefully before submitting them.
- Solutions may be handwritten or typeset as appropriate, but must be submitted as PDFs and must be legible. Any code should be submitted as specified.

  The required filenames for this problem set are listed under additional instructions.
- Your work must be submitted online through MarkUs.
- Your submitted file(s) should not be larger than 9MB. You might exceed this limit if you use a word processor like Microsoft Word to create a PDF; if it does, you should look into PDF compression tools to make your PDF smaller, although please make sure that your PDF is still legible before submitting!
- Submissions must be made *before* the due date on MarkUs.
- The work you submit must be your own.

Additional instructions

- The required files are hw1.py (the provided starter code) and hw1.pdf (for any written answers).
- Questions marked with [autotested] will be autograded on MarkUs. When you submit on MarkUs you will be able to run the tests (once they are setup).
- If anything is unclear, you suspect an error, or you encounter issues with the autotesting, please post on Piazza.

1. Converting Decimal to Binary (Programming Question) [autotested]

   (a) Implement the algorithm from this week's worksheet for converting a decimal number into its binary representation (repeated division by 2). Your function should be called int2bin. (see the provided starter code for examples of the expected output) (You may use Python's builtin bin function if you want.)

   (b) Now implement the algorithm from this week's worksheet for converting a fractional number into its binary representation (repeated multiplication by 2). Note that since x stored in Python has been rounded and stored in a finite sequence of bits, the algorithm will eventually result in a zero. You should not try to identify a repeating pattern to terminate the iteration, but wait until you get a zero.

   You may assume the input x is positive and $x \in [0, 1)$. Your function should be called frac2bin. (see the provided starter code for examples of the expected output)

   We'll use this function again in next week's homework.

2. Condition Number and Relative Error

(a)-(d) Do exercise 1.4 from Heath

   (e) (Problem from Cheney and Kincaid textbook) For small x, the approximation $\sin x \approx x$ is often used. For what range of x is this good to a relative accuracy of δ?

3. Numbers Between 0 and 1 in Python

   Your friend thinks that Python stores real numbers using the fractional binary representation (fixed point with $0.\ldots$) with n bits that we discussed on this week's first worksheet. For argument's sake, assume that you don't know much about floating point numbers yet, so you aren't fully equipped to immediately prove your friend wrong.

   Pressing your friend for what value of n the computer supposedly uses, you are told that $n = 64$ (8 bytes, seems reasonable).

   If this were the case, what value could you type into python to prove your friend wrong? That is, a value that would clearly contradict that $n = 64$ (but isn't so small that it rounds to zero). **Assign this value to x1 in hw1.py** [autotested]

   Still not convinced, your friend decides to check the size of the Python object representing the number you just typed in and sys.getsizeof(x1) returns that the size is 24 bytes. Your friend has new hope that n was just much larger than originally thought.

   What number can you type into python to dash your friend's hopes and completely disprove the hypothesis? **Assign this value to x2 in hw1.py** [autotested]

   Include a 1-2 sentence explanation in hw1.pdf of why your number, x2, in the last part is correct

4. Floating Point Numbers in Numpy (Programming Question) [autotested]

   Over the next few weeks, we will start writing code making use of the numpy package in Python, so we just want to get a bit of practice sooner than later. Next week we will learn about floating point number systems in more detail, but the key here is that numpy provides datatypes with different precisions. This will be useful when we want to observe certain behaviours discussed in the textbook. Typically, we use 8 byte floats (i.e double precision - the default in Python), but sometimes we use 4 byte floats (i.e single precision). Numpy provides float16, float32, float64, and float128. (You need to be careful initializing a float128, since any non-machine number will get rounded to an 8 byte float (e.g. np.float128(0.1) != np.float128(1)/np.float128(10))

In this question, refer to the function eval_with_precision provided in the starter code.

Make sure that it returns the same data type as provided as the third argument and that you didn't accidentally use the wrong data type at some point during the calculation! Hint: make sure you use the numpy versions of any special functions you use - otherwise you may get the wrong data type returned.

It might be useful to type a few simple statements in the python shell to make sure they return the type you think they do.