PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 1: Python Introduction

Due Friday, Sept. 1st, 2017, at 11:59pm.

Please review the *Homework Submission Guidelines* and consult with the TA if needed to correctly format your homework files before submission. As this is the first homework, a summary of what is to be delivered is included for each problem; this will not generally be done in later problem sets where students will be expected to carefully follow what is asked for in each problem.

Problem 1 – Plotting in Python using Matplotlib (14 points)

a) Write a function, `main_a()`, that plots the hyperbolic tangent, tanh(x), between $-5 \le x \le 5$. The NumPy and Matplotlib functions `np.arange`, `plt.show`, `plt.xlim`, `plt.xlabel`, `plt.ylabel`, and `plt.plot,` and `plt.legend` may be helpful to you. Be sure to include an appropriate set of axis labels. The plot should display in the command window.

b) Write a function, `plotfunc(a)`, that accepts a parameter and plots the function tanh(a*x) in the domain $-5 \le x \le 5$. Then write a function, `main_b()`, that uses plotfunc to plot tanh(a*x) for a = 0.5, 1.3, and 2.2 all within the same plot. Label the x and y axes and add a legend.

c) What spacing between x values starts to look too coarse (*i.e.*, no longer faithfully reproduces the data curves) for the plots of parts (a) and (b)?

<u>Summary of what is to be delivered:</u> You will deliver a *HW1p1.py* file which should contain: function `main_a`, function `plotfunc`, and function `main_b` in that order. Any import statements needed should go at the beginning. The end of the file should execute `main_a()` and `main_b()`, so that when we run your file both parts (a) and (b) output to the command window. Your file should contain comments (using `#`), including your name at the top, and brief explanations for each part.

You will also deliver a *HW1.pdf* file that contains a written or typed response to part (c). This same file will also contain all of the other free responses requested in the problems below.

Problem 2 – Functions and Control Flow in Python (10 points)

Recall that the Taylor series expansion of sin(x) is given by

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \tag{1}$$

Write a Python function `taylor_sin(x0, n)` which returns the value of the n-th order term in the Taylor expansion of sin(x) around the point x= x0. For instance, taylor_sin(1.7, 3) should return the numerical floating-point value of $\frac{-(1.73)}{3!}$. Note that by our definition, since sin(x) is an odd function, all terms where n is even are zero. Your solution should work for any positive integer value of n (n=1,2,3,…) that is passed to the function.

<u>Summary of what is to be delivered:</u> Your *HW1p2.py* file should contain just the function `taylor_sin`, as well as any needed import commands and comments.

Problem 3 – Random Numbers, Lists, and Masking (10 points)
Write a function `maskn(lst, i)` which accepts a list of integers *lst*, and a single integer *i*. It returns a list of the same length as *lst*, but with zeros for each number not evenly divisible by *i*, and with ones for each number that is. For example, `maskn([2,3,4],2)` should return `[1,0,1]`. The function should work for a list of any length.

Summary of what is to be delivered: Your *HW1p3.py* file should contain just the function `maskn`, as well as any needed import commands and comments.

Problem 4 – Recursive Functions (16 points).
The *n*-th Fibonacci number is generated by the sequence beginning with zero, where the *n*-th value is the sum of the previous two elements at $n-1$ and $n-2$. Therefore, the first few elements are given by

$$0,1,1,2,3,5,8,13,.... \tag{2}$$

where we define the sequence to begin with $n = 0$ (*i.e.*, F_n such that F_0=0, F_1=1, F_2=1, *etc*). One of the most natural ways to compute the *n*-th Fibonacci number is in terms of a recursive function, or a function that calls itself. In order to prevent such a function from recursively calling itself an infinite number of times, it is important to identify a base case, or a condition that will cause the function to immediately return for a particular input.

a) What is an appropriate base case to use when writing an algorithm to compute the *n*-th Fibonacci number, where the only input to the algorithm is a finite integer $n \geq 0$?

b) Implement the function `fib_loop(n)` using a for or while loop which calculates and returns the *n*-th Fibonacci number. Assume that *n* is an integer >=0.

c) Implement the function `fib_recur(n)` using recursion (and no loops) which calculates and returns the *n*-th Fibonacci number.

Summary of what is to be delivered: Your *HW1p4.py* file should contain just the functions `fib_loop` and `fib_recur` in order, as well as any needed import commands and comments.
You will also add a response to part (a) in your *HW1.pdf* file.

Summary of the summaries: You will be delivering one .zip file that contains *HW1p1.py*, *HW1p2.py*, *HW1p3.py*, *HW1p4.py*, and *HW1.pdf*. The .zip file should have the naming convention *youronyen_HW01.zip*. It needs to be uploaded to Sakai by the posted deadline. **As a reminder, make sure each .py file runs from a fresh kernel, *i.e.*, restart the kernel and run them.** The grader will be spot-checking your work and will randomly choose one or more problems to see if they execute as a stand-alone. One common mistake would be that if you imported a module in one problem but didn't in a later problem, and didn't refresh your kernel, you might not have noticed the missing import statements.