Chapter 2

A short introduction to Python

One goal of this class is that you become proficient with Python in a scientific setting. This will not happen all at once, but gradually over the course of the entire semester. There are several key programming ideas that will be used repeatedly throughout the semester. It is critical that you grasp the concepts and are able to implement them in python. What follows is a short description of those concepts with examples to illustrate.

Data Types

The most commonly used types of data that will be used for this class are: integers, floats, strings, lists, and arrays.

P2.1 Read sections 3.1 – 3.6 in the Python manual to learn about the first four data types and sections 5.1–5.3 to learn about arrays. Execute at least 25% of the text `in this font`.

Loops and Logic

A loop is a set of instructions to be performed until some pre-determined criteria is satisfied. Loops are used heavily in this class and you must grasp what the are as soon as possible. Logic refers to the process of checking whether the pre-detemined criteria has been met and is thus very closely linked to the topic of loops.

P2.2 Read chapter 6 in the Python manual to get a better feel for what we mean. Execute at least 25% of the text `in this font`.

Functions

A function is like a black box. The user passes some needed information into the box and the box uses that information to perform some useful calculation and spits the result out.

P2.3 Read chapter 4 in the Python manual to see how to build functions in Python. Execute at least 25% of the text `in this font`.

Plotting

Plotting in Python is slightly different than what you may be used to because Python cannot plot analytic functions. In other words, you can't ask Python to plot $sin(x)$. It can only plot data points. If you'd like to plot a function, you must construct a list of function values and then plot them. This way of thinking will be increasingly important as the semester progresses.

P2.4 Read chapter 7 in the Python manual to get a better idea of how to construct plot in Python. Execute at least 25% of the text `in this font`.

Classes

You can think of a class as an object that contains both data and functions. Since you are already familiar with variables and functions, the best way to learn about classes is to see one and then ask questions. Here is a simple class used to analyze ideal gas processes:

```
class idealGas():
    def __init__(self,R=8.314,kB=1.38e-23):
        self.R = R
        self.kB = kB

    #Define what kind of ideal gas process we are dealing with
    def setProcessType(self,process, n,cV):
        self.n = n  #Set the number of moles of the gas
        self.process = process  #Indicate what kind of process it is:
                                # Isothermal, Isobaric, Adiabatic, Isochoric
        self.cV = cV
        self.cP = self.cV + self.R
        # If the process is adiabatic, we might need to know what gamma is.
        if process == 'Adiabatic':
             self.gamma = self.cP/self.cV

    # Set the initial and final conditions for the process.
    def setConditions(self,pi=None,Vi=None,Ti=None,pf=None,Vf=None,Tf=None):
        self.pi = pi
        self.Vi = Vi
        self.Ti = Ti
        self.pf = pf
        self.Vf = Vf
        self.Tf = Tf

    def isothermal(self):
        from math import log
        self.work = -self.n * self.R * log(self.Vf/self.Vi)
        self.deltaE = 0
        self.heat = - self.work

    def isochoric(self):
        self.work = 0
```

```
            self.heat = self.n * self.cV * (self.Tf - self.Ti)
            self.deltaE = heat

        def isobaric(self):
            self.work = - self.pI * (self.Vf - self.Vi)
            self.heat = self.n * self.cP * (self.Tf - self.Ti)
            self.deltaE = self.n * self.cV * (self.Tf - self.Ti)

        def adiabatic(self):
            self.heat = 0
            self.deltaE = self.n * self.cV * (self.Tf - self.Ti)
            self.work = self.deltaE

        # Depending on what type of process it is, calculate the work
        # done, heat absorbed, and change in internal energy for the process.
        def calculate(self):


            # These three lines can replace the lines below and its a really
            # cool use of dictionaries. Try it.
            #         processes = {'Isothermal':self.isothermal, 'Adiabatic':
            #   self.adiabatic, 'Isochoric':self.isochoric, 'Isobaric':self.isobaric}
            # processes[self.process]()

             if self.process == 'Isothermal':
                 self.isothermal()
             elif self.process == 'Isochoric':
                 self.isochoric()
             elif self.process == 'Isobaric':
                 self.isobaric()
             else:  #Adiabatic
                 self.adiabatic()

    # Class definition done. What follows below illustrates how to use the class.
    gasOne = idealGas()  #Initialize the class.
    gasOne.setProcessType('Isothermal',3,20.8)  # Set the type of process and number of moles
    gasOne.setConditions(pi = 2.0e5, pf = 1.0e5,Ti = 200, Tf = 200, Vi = 2, Vf = 5 )  # Set initial and final co
    gasOne.calculate()  # Calculate thermodynamic properties.
    print(gasOne.work)  # Print results
    print(gasOne.deltaE)
```

Activities

H2.5 Write a `for` loop that counts by threes starting at 2 and ending at 101. Along the way, every time you encounter a multiple of 5 print a line that looks like this (in the printed line below it encountered the number 20.)

```
fiver: 20
```

You will need to use the modulo opertor(%) to check for multiples of 5.

H2.6 Build a function that contains a loop that sums the integers from 1 to N, where N is an integer value that is passed into the function. By calling the function at least 5 times, verify that the formula

$$\sum_{n=1}^{N} n = \frac{N(N+1)}{2}$$

is correct.

H2.7 Build a function that takes argument x and performs the sum

$$\sum_{n=1}^{1000} nx^n$$

using a `for` loop. By calling the function with different values of x, verify that the sum only converges for $|x| < 1$ and that when it does converge, it converges to $x/(1-x)^2$.

H2.8 Redo the previous exercise using a `while` loop instead of a `for` loop. Make your own counter for n by using $n = 0$ outside the loop and $n = n + 1$ inside the loop. Have the loop execute until the current term in the sum, nx^n has dropped below 10^{-8}. Verify that this way of doing it agrees with what you found in the previous exercise.

H2.9 Verify, by numerically experimenting with a loop that uses the `break` command to exit the loop at the appropriate time, that the following infinite-product relation is true:

$$\prod_{n=1}^{\infty} \left(1 + \frac{1}{n^2}\right) = \frac{\sinh \pi}{\pi}$$

H2.10 (a) Build an array containing all multiples of 3 starting at 3 and ending at 2000.

(b) Calculate $3x^2 \sin(x) + \ln(x)$, where x is the list you just created. In other words, for every number in the array you just created, calculate the quantity above. This will yield an array of calculated values.

(c) Sum the elements of the list to get a single final answer.

Ans: -736905.545292
Hint: `arange` is a good choice for building the initial list. The native `sum` function will help you sum all of the elements of the list.

H2.11 Use a `while` loop to verify that the following three iteration processes converge. (Note that this kind of iteration is often called successive substitution.) Execute the loops until convergence at the 10^{-8} level is achieved.

$$x_{n+1} = e^{-x_n} \quad ; \quad x_{n+1} = \cos x_n \quad ; \quad x_{n+1} = \sin 2x_n$$

Note: iteration loops are easy to write. Just give x an initial value and then inside the loop replace x by the formula on the right-hand side of each of the equations above. To watch the process converge you will need to call the new value of x something like `xnew` so you can compare it to the previous x.

Finally, try iteration again on this problem:

$$x_{n+1} = \sin 3x_n$$

Convince yourself that this process isn't converging to anything. We will see in Lab **??** what makes the difference between an iteration process that converges and one that doesn't.

H2.12 Write a class to calculate the trajectory of a projectile. The initialization routine should define g. Additionally, you should have member functions to: i) set the launch conditions (initial velocity, position, etc. ii) calculate the landing location and iii) plot the trajectory.