Introduction to Python

MATH 249

1. To start Python, first find and run `Anaconda Prompt`. Change to your H: drive by entering `H:` and hitting enter, then type `jupyter notebook` <enter>. This will begin a Jupyter notebook that can see (and save to) your H: drive instead of just your local hard drive. (If you're working on your own laptop and always will be, this matters less.) Note that if you have assigned a different letter to your network drive, you should use that instead of H.

2. To execute a command, use SHIFT+ENTER. ENTER alone will give you another line in the same command window (also useful).

3. Python has a number of useful libraries of routines that have already been written. We will take advantage of several of these. Enter

   `import numpy as np` (numerical functions)
   `import sympy as sp` (symbolic functions)
   `import matplotlib.pyplot as plt` (needed for graphing)
   `from mpl_toolkits.mplot3d import axes3d` (needed for 3D graphing)

   in the first cell and execute it **twice**. (I have no idea why it takes twice.)

4. Loops look like `for i in range(100):` (for example). Note that in Python, `range(100)` refers to the integers 0, 1, 2, ..., 99 (not 1 to 100).

5. Python requires that you declare variables. Use `symbols` for this; e.g., `x,y=sp.symbols("x,y")` (note the . notation - in this context, this is how we tell Python which library to look in for a particular command).

6. Exponentiation is performed with `**` instead of `^` – watch out for that! (E.g., `3**2` instead of `3^2`.)

7. You can solve equations (and systems of equations) with the solve command:
   `solve((x+y-2,x-y-4),(x,y))` (for example). Python's solve command assumes that the equation is of the form `your_expression=0` – note that neither "equation" in the command has an = in it. Also, be sure to declare your variables first. Finally, solve is in the sympy package, so you may need to use `sp.solve` instead of just `solve`.

8. If `solve` fails, you can try to solve numerically: `sp.nsolve([expr1, expr2],[x,y],[x-guess,y-guess])`.

9. Sometimes Python just does what you want but doesn't report out. If you want to be sure to see the output, you may have to "print" it: `print(whatever)`.

10. In Python, you can define objects using the equal sign, = . To define a to have the value 3.14, enter `a=3.14`

11. Here are some individual commands we will use. Note that *arg* refers to the argument of a function (the number you plug in). More will be added to this sheet as we find we need them.

    (a) For the number π: `np.pi`

    (b) For ∞: `sp.oo` (letter o repeated)

    (c) Common functions in `numpy`: `np.cos(arg)`, `np.sin(arg)`, `np.exp(arg)` (which is e^{arg}), `np.sqrt(arg)`, `np.arccos(arg)`, `np.arcsin(arg)`

    (d) For the same functions in sympy, use `sp` as a prefix instead of `np`.

    (e) Other common functions: `abs(arg)` (for $|arg|$),

12. Vector operations.

    (a) Vectors are defined as lists, as in `u=[2,4,-3]`

    (b) Dot product of u and v: `np.dot(u,v)`

(c) Cross product of u and v: `np.cross(u,v)`

(d)

Graphing

13. Setup:

    `a=np.arange(-5,5,0.1)` This creates a list of values from -5 to 5 separated by 0.1. Change this to your desired range.
    `b=np.arange(-5,5,0.1)` (Same.)
    `x, y = np.meshgrid(a, b)` Here, `np.meshgrid(a,b)` creates a list with two arrays in such a way that every possible pairing of a and b values occurs. Then we set x equal to the first array and y equal to the second. This is how we set up for 3D graphing.

    Note that for both 2D and 3D graphing, the input values are arrays: one-dimensional arrays for 2D graphing, and two-dimensional arrays for 3D graphing. This tells Python what specific inputs to use.

14. 2D graphing

    `plt.plot(a,a**2)` graphs $y = x^2$ (using the a defined above by `arange`). (For graphing functions).

    `plt.plot(a**3,a**2)` graphs the parametric curve $\vec{r}(t) =< t^3, t^2 >$. Note that the structure is the same as for graphing a function.

15. 3D graphing

    (a) Surfaces:
    `z = np.sqrt(x**2+y**2)` (Sample function.)
    `fig=plt.figure()` (Initiates a graph.)
    `ax = fig.add_subplot(111, projection='3d')` (Sets up the graph to be 3D. We are calling the graph "ax.")
    `ax.plot_surface(x,y,z)`

    You can plot multiple things together on the same set of axes by including them in the same cell before executing. For example, if you add the line
    `ax.plot_surface(x,y,np.sqrt(z))`

    to the above, it will graph both the original z and $\sqrt{z}$ on the same set of axes.

    (b) Parametric curves:
    ```
    fig=plt.figure()
    ax=fig.add_subplot(111,projection='3d')
    ax.plot(t+2,3*t-1,2*t+1)
    ```

16. To define a function:

    ```
    def F(x,y,z):
        return(x**2+y**2+z**2-4)
    ```

    gives $F(x, y) = x^2 + y^2 - 4$ (for example).

17. To evaluate a defined function, use natural notation: `F(1,2,3)`

18. Calculus (in the `sympy` package)

    (a) To differentiate: `sp.diff(x**2,x)` (declare x as a variable first, and substitute your function for x^2)

    (b) To integrate: `sp.integrate(x**2,x)`

    (c) For limits: `sp.limit(f(x), x,x0)` (where `x0` is the limit point)

19. Vector fields: these are unfortunately rather awkward to work with.

    (a) `from sympy.physics.vector import ReferenceFrame,gradient,curl,divergence` (The tools we will need)

    (b) `R=ReferenceFrame('R')`
    `F=R[0]**2+R[1]**2+R[2]**2-4`
    (This defines F as $x^2+y^2+z^2-4$. `R[0]` means x, the first coordinate in R.)
    `F=gradient(F,R)` (This computes the gradient of F in this coordinate system.)
    `v=R[0]*R.x+R[2]*R.y+2*R[1]*R.z` (Defines the vector field $\vec{v} =< x, z, 2y >$.)
    `curl(v,R)` (Computes curl($\vec{v}$))
    `divergence(v,R)` (Computes div(v))

    The notation `R.x` gives us the first vector in the `R` reference frame. If we're working in a Cartesian system, it means $\hat{i}$.

20. To plot vector fields

```
X = np.arange(-10, 10, 1)
Y = np.arange(-10, 10, 1)
U, V = np.meshgrid(X, Y) (Setting up the points to plot)
fig, ax = plt.subplots()
q = ax.quiver(X, Y, U**2, V**2)
```

    (This plots the vector field $< x^2, y^2 >$.)