INTRODUCTION TO PYTHON IN MATH 152

Welcome to your Computer Lab for MATH 152!

Those who did not take 151 here may ask, "Why a computer lab in a Calculus course?" There are several reasons discussed below. One major reason is that many of you are students in the College of Engineering and are learning (or will learn) Python in your ENGR 102 course. This lab gives you an integrated curriculum; one whereby you apply techniques of Python programming to help solve Calculus problems. Generally, these computer labs will contain three types of problems:

1) Problems which allow you to review the general algorithmic processes to solve by hand while eliminating the tedious algebraic and computational aspects.
2) Problems which allow you to visualize some of the important principles of Calculus to help improve conceptual understanding.
3) Problems (often of a practical nature) which are too algebraically and/or computationally tedious to solve by hand.

We will spend the first lab reviewing the basic commands of Python that were used in 151 and will be used in 152 (numerical calculations, substituting, symbolic manipulation, solving equations, and plotting graphs). Since labs will be done in teams, we will wait to start until add/drop is finished. For now, some important information about the Python interface we will be using, Jupyter, and basics about the Python program itself.

HOW TO GET JUPYTER AND PYTHON ON YOUR DEVICE

Obviously, since all Python labs are remote, you will need to have the software on your device. The Jupyter interface we use in 152 may be different from the interface used in ENGR 102 (such as PyCharm or Spyder), but ALL these interfaces run Python! The Jupyter notebook makes it easier to collect your input, Python's output, and graphs all in one "Notebook", which your team will use to produce one PDF file to upload when you are done. **NOTE**: if you prefer to use another interface, you may do so, but it is YOUR team's responsibility to figure out how to put their code, Python's output, and all graphs in ONE easy-to-follow PDF file. If you need to download Jupyter, it is part of the Anaconda package (which also include Python). Here are the steps to download it:

1) Go to www.anaconda.com/products/individual
2) Near the bottom of the page, you will find "Anaconda Installers". Click on the one appropriate to your operating system (Windows, Mac, or Linux). If the link to the top version doesn't work, try the bottom version.
3) Once the installer is downloaded, click on it to install the software (as you would any other app)
4) After the Anaconda software is installed (it may take a while!), find Jupyter in your usual start menu (probably under "Anaconda"). Alternatively, open a command window and type "jupyter notebook".
5) When you run Jupyter, you may be asked which app to run it in. Jupyter opens in a browser window, so choose your favorite browser.

IMPORTANT THINGS TO KNOW ABOUT PYTHON

Packages Packages in Python are "libraries" of commands. We will mostly use the SymPy ("Symbolic Python") package in this course, so we will start all of our labs with the next set of commands. The commands allow us to import all of the commands in sympy (i.e., "check out all the library commands")

[2]:
```
from sympy import *
from sympy.plotting import (plot, plot_parametric)
```

Variable types Like most other computer programs, Python stores values in variables. However, these variables can take on different forms, or types. The biggest difference is with type "integer" and type "float" as shown below:

[3]:
```
a=5  # This is type integer.  Also, the hashtag indicates these are comments␣
 ↪which Python ignores
b=5.0  # This is type float
print('Is the square root of 5 equal to',sqrt(a),'or is it equal to',sqrt(b),'?')
# The "print" command produces output, which can (and usually should) contain␣
 ↪explanatory text as demonstrated here
```

```
Is the square root of 5 equal to sqrt(5) or is it equal to 2.23606797749979 ?
```

Notice that the first answer was left as sqrt(5), but the second answer was given as a decimal approximation. In SymPy, outputs are exact values UNLESS inputs are given as floating-point decimals (type "float").

Other types of variables are shown below.

[4]:
```
c=[1,2,3,4]   #List
d=(1,2,3,4)   #Tuple-we will use these for domains of graphs
x=symbols('x') #Symbolic variable
f={x:2} #Dictionary-can be used to make multiple substitutions into an expression
```

Executing commands on variables

In most cases, when you want to perform a command on a variable, the correct Python syntax is

variable.command

instead of "command(variable)", as demonstrated in the example below (though sometimes either format works-but not always!)

```
[7]: f=x**2*exp(x)  #Recall we defined x as a symbolic variable in the previous block␣
      ↪of commands. So f is a symbolic expression
                    #Also notice that "x squared" is NOT entered as x^2, but as x**2!␣
      ↪ exp(x) can also be entered as E**x
     df=f.diff(x,3)  #Taking the third derivative of f.  diff(f,x,3) also works
     print('The third derivative of f is',df)
     # Now suppose we wanted to substitute x=2 into the third derivative.
     print('The third derivative of f at x=2 is',df.subs(x,2))
     # Notice the answer is exact as stated above.  If we wanted to get a decimal␣
      ↪approximation, we use the 'evalf' command
     print('which is approximately',df.subs(x,2).evalf())
```

```
The third derivative of f is (x**2 + 6*x + 6)*exp(x)
The third derivative of f at x=2 is 22*exp(2)
which is approximately 162.559234176474
```

Don't worry if you did not do Python in 151 and didn't get all of this right away. We will spend the first lab reviewing these data types and commands in greater detail! To help you catch up, I recommend searching the online documentation (docs.sympy.org) for the following commands: **symbols**, **simplify**, **evalf**, **subs**, **solve**, and **plot**, as well as watching Python videos 0-8 at the Math Learning Center website: http://mlc.tamu.edu/Supplemental-Material/Python-mini-course