Python software accessories

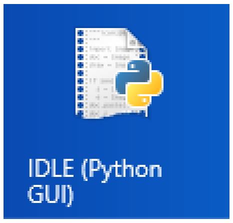
Programming for Biologists

January 13, 2015

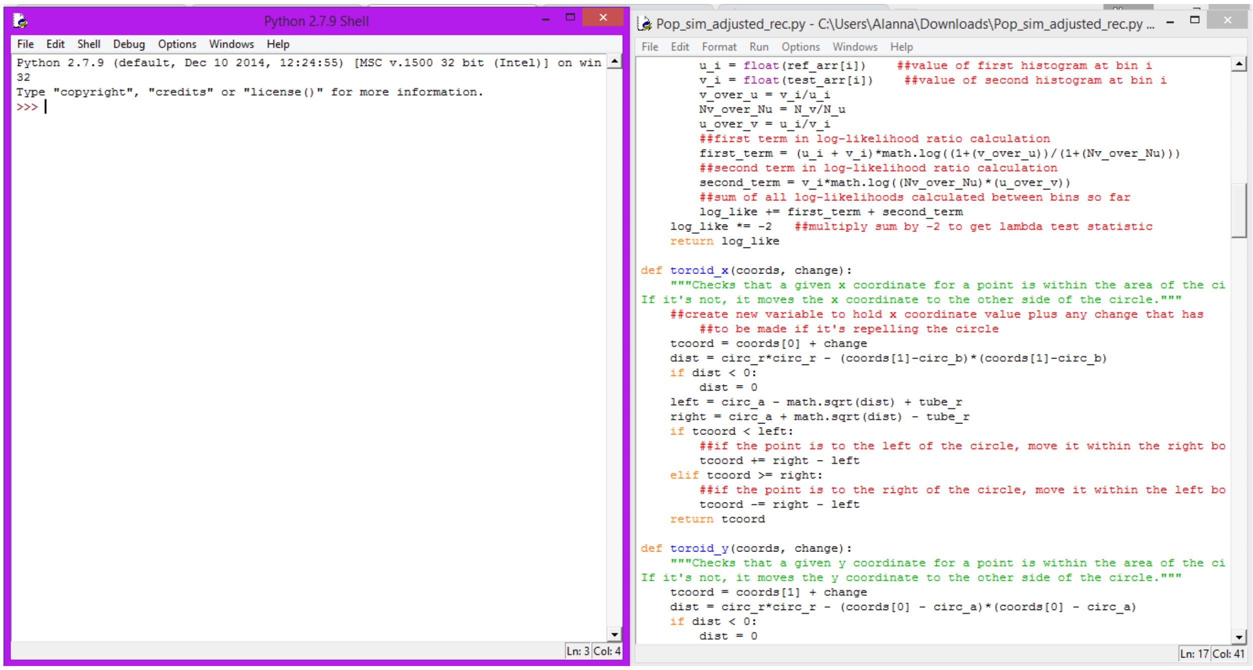
When starting out with the programming language Python, it's important to consider whether or not you want to install a separate integrated development environment (IDE) to construct and edit your scripts. These programs are accessories to Python and can streamline the development process by offering better looking interfaces, automatic code completion, and debugging tools. This document will serve as a starting guide to help you navigate the free options currently available for Python programming to help you decide what editing environment you'll use.

Install Python first

The first step before beginning this journey is to, if you haven't already, install Python 2.7.9. When you install Python, the program comes with it's own graphic user interface: IDLE. Upon installation, you should be able to find a program with this icon available in your programs and applications.



This basic program is enough to manage any coding you want to do. IDLE offers useful syntax coloring (as long as your script is saved as a .py file), basic debugging, and smart tab insertion that helps you manage whitespace in your code. Something that may be considered a drawback is that the interactive shell (where you can type commands after the >>>) opens in a separate window from any scripts you may open. Each script then also opens in its own window, which could create a mess if you're working with a lot of .py files at the same time. The following screenshot shows you an example of IDLE with the shell and one script open at the same time.


Python 2.7.9 Shell
File Edit Shell Debug Options Windows Help
Ln: 3 Col: 4
Pop_sim_adjusted_rec.py - C:\Users\Alanna\Downloads\Pop_sim_adjusted_rec.py ...
File Edit Format Run Options Windows Help
Ln: 17 Col: 41

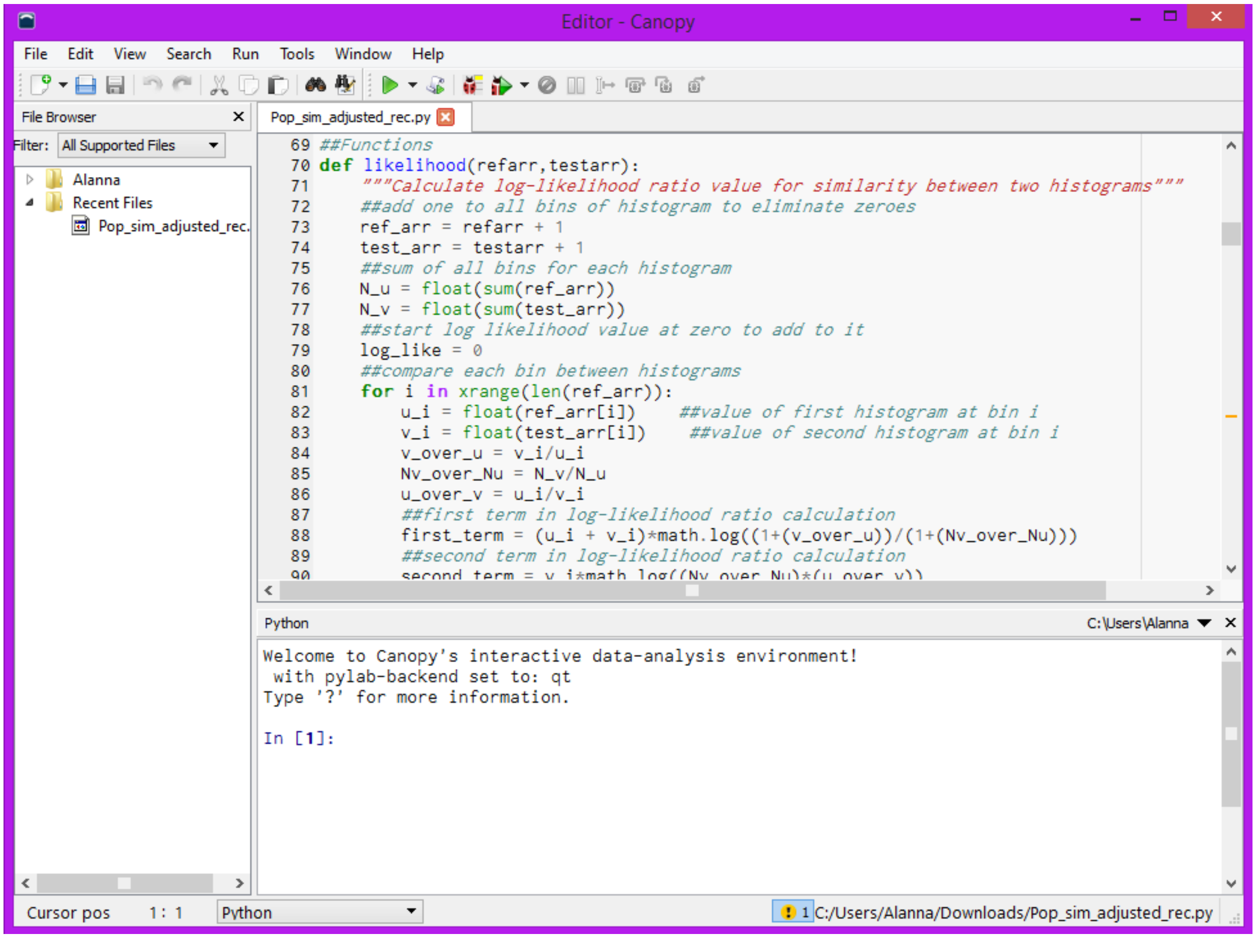
```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

```
        u_i = float(ref_arr[i])    ##value of first histogram at bin i
        v_i = float(test_arr[i])    ##value of second histogram at bin i
        v_over_u = v_i/u_i
        Nv_over_Nu = N_v/N_u
        u_over_v = u_i/v_i
        ##first term in log-likelihood ratio calculation
        first_term = (u_i + v_i)*math.log((1+(v_over_u))/(1+(Nv_over_Nu)))
        ##second term in log-likelihood ratio calculation
        second_term = v_i*math.log((Nv_over_Nu)*(u_over_v))
        ##sum of all log-likelihoods calculated between bins so far
        log_like += first_term + second_term
    log_like *= -2   ##multiply sum by -2 to get lambda test statistic
    return log_like

def toroid_x(coords, change):
    """Checks that a given x coordinate for a point is within the area of the ci
If it's not, it moves the x coordinate to the other side of the circle."""
    ##create new variable to hold x coordinate value plus any change that has
        ##to be made if it's repelling the circle
    tcoord = coords[0] + change
    dist = circ_r*circ_r - (coords[1]-circ_b)*(coords[1]-circ_b)
    if dist < 0:
        dist = 0
    left = circ_a - math.sqrt(dist) + tube_r
    right = circ_a + math.sqrt(dist) - tube_r
    if tcoord < left:
        ##if the point is to the left of the circle, move it within the right bo
        tcoord += right - left
    elif tcoord >= right:
        ##if the point is to the right of the circle, move it within the left bo
        tcoord -= right - left
    return tcoord

def toroid_y(coords, change):
    """Checks that a given y coordinate for a point is within the area of the ci
If it's not, it moves the y coordinate to the other side of the circle."""
    tcoord = coords[1] + change
    dist = circ_r*circ_r - (coords[0] - circ_a)*(coords[0] - circ_a)
    if dist < 0:
        dist = 0
```

Bundled IDEs with modules

If you are just starting out with Python and know you'll be using it for scientific analysis (like the assignments for this class), you may want to consider using software that is not just a programming GUI, but also has many of the most popular scientific Python modules pre-installed. Enthought Canopy and Anaconda are two environments that provide such a benefit.

Enthought Canopy, in addition to including pre-installed scientific modules, features a graphical package manager that simplifies the download and installation process for a new module into one click of the mouse. How many modules are included depends on whether you download the free version Canopy Express or Canopy Academic. The academic version includes almost twice as many modules and is still essentially free if you register for an Enthought account using your temple.edu email address. A perk included with academic license (which expires after a year) is access to Enthought's Training On Demand, currently a collection of 6 online courses in learning basic Python and its scientific applications through video lectures and exercises. This software works on Mac, Windows, and Linux. The following screenshot is an example of the Enthought Canopy editor.



```
##Functions
def likelihood(refarr,testarr):
    """Calculate log-likelihood ratio value for similarity between two histograms"""
    ##add one to all bins of histogram to eliminate zeroes
    ref_arr = refarr + 1
    test_arr = testarr + 1
    ##sum of all bins for each histogram
    N_u = float(sum(ref_arr))
    N_v = float(sum(test_arr))
    ##start log likelihood value at zero to add to it
    log_like = 0
    ##compare each bin between histograms
    for i in xrange(len(ref_arr)):
        u_i = float(ref_arr[i])    ##value of first histogram at bin i
        v_i = float(test_arr[i])    ##value of second histogram at bin i
        v_over_u = v_i/u_i
        Nv_over_Nu = N_v/N_u
        u_over_v = u_i/v_i
        ##first term in log-likelihood ratio calculation
        first_term = (u_i + v_i)*math.log((1+(v_over_u))/(1+(Nv_over_Nu)))
        ##second term in log-likelihood ratio calculation
        second_term = v_i*math.log((Nv_over_Nu)*(u_over_v))
```

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]:
```

Anaconda works with both Python 2 and Python 3 and includes more than 200 packages frequently used in science and data analysis. Among these modules are Biopython, matplotlib, and SciPy, which will be needed for class assignments. Similar to Enthought, Anaconda offers both a free trial version as well as a slightly more deluxe free academic version once you register with a .edu email. Anaconda is also compatible across all operating systems.

IDEs without modules

It's more common that an IDE developed for Python will not include any pre-installed modules, making for a lighter design more customizable by the user. There are so many options available that this guide will be limited to listing just a few common programs, not attempting a comprehensive review of everything out there.

Interactive Editor for Python (IEP) is an IDE that works across Windows, Mac, and Linux. With this editor, multiple shells can be run at once, and these shells can even be in different versions of Python. This can be a useful feature when trying to test or convert code between Python 2 and Python 3.

Spyder is another IDE that can be installed on any operating system. In this GUI you can select the colors for the background and syntax coloring to customize how you want your code environment to look.

PyScripter is the only editor listed in this guide that has operating system limitations. PyScripter can only be installed on a Windows computer. The light design and the fact that it's built in a compiled language can give it a speed advantage over some other Windows options.

As stated previously, this is not a comprehensive list, just a few links to programs if you want to try something and don't know where to start! Check out more at the Python Wiki page or these programming discussion threads if you want to explore all the options.

Multiple language IDEs

If you've already been programming in other languages and are just adding Python to your repertoire, you may want to consider downloading an IDE that can handle multiple languages for syntax and debugging. Two such examples are Eclipse and Aptana Studio. Both IDEs are downloadable across Windows, Mac, and Linux, and each allow you to manage projects across languages through a single computer workspace. You can switch easily from Ruby to Java to C++ to Python without needing space-consuming IDEs on your machine. It may also ease the transition to Python if you're already familiar with the IDEs functions and debugging.