1 Python in its Natural Habitat

"Crikey!" - The Crocodile Hunter

In which we track the beast to its lair and examine its native lifestyle.

For the purposes of this course, we'll be interacting with the Python programming language using three main tools:

1. `python`, the Python interpreter.
2. **jEdit**, a Python (and general text) editor.
3. The **Command Prompt**, **Terminal**, **Console**, **Command Line**, **Shell**, or whatever other name you choose to apply to the textual environment in which you'll manipulate files and programs. Yes, all of those words really do mean essentially the same thing.

All of these things are independent of the Python programming language, which is an abstract entity not unlike the English language (except less confusing, hard as that may be to believe for those of you who have always known English but never known Python). And for that matter, learning Python is not all that different from learning any programming language; in some sense, if you've seen one, you've seen them all. That's one of the reasons why this portion of the course is important - once you tackle Python, you can go out and use the programming environment of your choice to solve the problems of your choice.

1.1 The Command Prompt

"In the Beginning was the Command Line" - Neal Stephenson, "In the Beginning was the Command Line"

First of all, you've all already encountered the relic that is the command prompt - and, as Neal Stephenson discusses in his essay of the same name, it's been around since the start. To provide some context, everybody nowadays interacts with computers using a graphical user interface, or GUI. This provides a means by which all of the virtual things you can't see (bits and bytes representing data on your hard disk) are represented by symbolic real-world objects (files and folders). If you go to My Computer (or, for you Mac folks, your hard disk icon), you don't see a bunch of ones and zeros - you see a virtual space containing icons, each of which represents an individual document file, application program, or directory folder.

That being said, the command prompt is no different. Windows Explorer (the thing you get by clicking on My Computer) or the Macintosh Finder (the thing you get by clicking on your hard disk) allows you to navigate through folders, launch programs, and examine files by clicking on them; information is provided in the form of lists containing different icons. The command prompt allows you to navigate through folders, launch programs, and examine files by typing in specific commands. Now, though, information is provided in the form of textual descriptions rather than pretty pictures.

I won't bore you with the details here - if you're interested, go read the aforementioned Neal Stephenson work (it's freely available online at `http://www.cryptonomicon.com/beginning.html`). But I will provide a list of analogous tasks between the usual graphical Explorer/Finder interface and their textual command line counterparts. And, because I'm just that nice, I'll include a list of handy-dandy command line shortcuts that will make your life in its walls of black-and-white text a little less constraining. I'll even break them down into Windows and Macintosh sections, although you should note that almost everything is the same or nearly so (determining the precise history and reasons for this fascinating piece of divergent/convergent evolution is left as an exercise for the reader). But first...

A Note on Paths

You'll see me refer to something called a path here, and you'll get to know the suspiciously similar `PATH` variable intimately below when you set up Python to run on your computer. When dealing with the command prompt, a path is a complete list of directory names, separated by a backslash \ (on Windows) or a forward slash / (on a Macintosh), specifying where a particular directory or file lives. So if you normally get to a file by clicking on the

C: drive in Windows Explorer, then on Python26, then on Java, then on python.exe, the full path to that file is C:\Python26\python.exe. Note that paths can refer to directories as well; you could say that the file python.exe resides in the directory C:\Python26\. And, for the curious, the backslash (or slash) at the very end is optional. A couple points of interest:

- Spaces in paths can be problematic. I won't type them here, but you'll often need quotes around paths with spaces (e.g. "C:\Program Files\jEdit"). Quotes will work on either Windows or Macintosh machines; on a Mac, you also have the option of simply preceding each space with a backslash (e.g. /Program\ Files/jEdit, although you'll be hard pressed to find such a directory on a Mac).
- Almost everything we say about the Command Prompt on a Windows machine applies to the Terminal on a Macintosh machine. You'll always have to turn backslashes into forward slashes, and you can omit the C: business (the root directory on a Macintosh is just /, not C:\).
- Finally, Macs also have the concept of a home directory, which is often represented by a single tilde ~. You can think of this sort of like your Documents or user directory on Windows. So on a Windows machine, if you created the path C:\Users\chuttenh\Destktop\problems02 for the first assignment, on a Mac you might create ~/Desktop/problems02.

Also note that paths can be either absolute or relative. A path is absolute if it begins with C:\ (on Windows) or just a slash / (on a Macintosh). This is also known as a full path, since you can use an absolute path successfully no matter what your current directory happens to be. "What," you say, "is a current directory?" Just like you generally have one topmost window in Explorer or the Finder, representing the directory full of files that you're currently inspecting, the command prompt always has a single unique path called your current or working directory. You can think of it like a cursor inside of your file system; if you ask the computer to do something, it'll assume (given no additional information) that you want to do it there. So if you just tell it to, say, python problems02.py, it'll assume you mean the problems02.py file that's in the current directory. The current directory is always displayed at the bottom of the command prompt.

We still haven't gotten to relative paths, though. Suppose we call the current directory . That's right, a period - the current directory is always represented by a single period, and its parent directory is always represented by two periods .. A relative path is assumed to start from the current directory, and you can include references to the parent directory (..) or to child directories in the text of the path. This doesn't make any sense at all when I try to say it in words, so let's look at some examples. If my current directory is:

```
C:\Program Files\jEdit
```

Then the following paths all refer to the same file:

```
C:\Program Files\jEdit\doc\users-guide\index.html
           doc\users-guide\index.html
          .\doc\users-guide\index.html
        ..\jEdit\doc\users-guide\index.html
..\..\Program Files\jEdit\..\jEdit\doc\.\.\.\users-guide\index.html
```

The last few get a little esoteric (i.e. don't ever let me catch you actually using something like that!), but they're still technically valid. If you change your current directory to, say, C:\Program Files, only the first path (the absolute one) will still work. Now on to the good stuff...

I'm using a Macintosh. I want to...

- Get things started.
  - *Finder*: Double click on your hard disk icon to open a Finder window.
  - *Terminal*: Navigate to Applications/Utilities and launch Terminal to open a new command line.
- Change my current directory.

  - *Finder*: Double click on a folder to activate it in the current window.
  - *Terminal*: Type "`cd <path>`" to change to the given (absolute or relative) location. "`cd`" stands for "change directory."
  - *Example*: `cd ~/Desktop/problems02`

- See the files in my current directory.
  - *Finder*: Um... they're just sort of there, listed as icons, in the window.
  - *Terminal*: Type "`ls`" to show the files in the current directory, or "`ls <path>`" to show the files in the given location. "`ls`" stands for "list."
  - *Example*: `ls /Applications`

- Launch an application or open a file.
  - *Finder*: Double click on a program or document icon.
  - *Terminal*: Type in the program or file's path. Note that MacOS is a little strange about this; most "normal" programs will show up as weird things that end in .app. Caveat typer.
  - *Example*: `python problems02.pya`

- Move a file.
  - *Finder*: Click and drag.
  - *Terminal*: Type "`mv <file> <path>`" to move the given file to the given location. Be careful - if `<path>` is a path to a file (rather than a directory), you'll overwrite it (i.e. you'll rename `<file>` instead of or in addition to moving it).
  - *Example*: `mv problems02.py ~/Desktop/problems02`

- Delete a file.
  - *Finder*: Command-Delete or click and drag to the trash bin.
  - *Terminal*: Type "`rm <file>`" to completely delete the given file. Note that this does not move it to the trash, it just goes byebye.
  - *Example*: `rm problems02.py`

- Rename a file.
  - *Finder*: Enter or click and wait.
  - *Terminal*: See the bit above about `mv`.
  - *Example*: `mv problems02.py problems42.py`

- Create a directory.
  - *Finder*: Command-Shift-N or Control-click.
  - *Terminal*: Type "`mkdir <path>`" just like in the assignment. If the given path already exists, it'll give you a little error message without hurting anything.
  - *Example*: `mkdir ../problems03`

I'm using Windows. I want to...

- Get things started.
  - *Explorer*: Double click on My Computer on the desktop, or single click on it in the Start menu.
  - *Command Prompt*: Either select Run from the Start menu and type `cmd`, or select Start/Programs/Accessories/Commar Prompt.
- Change my current directory.
  - *Explorer*: Double click on a folder to activate it in the current window (or, depending on your settings, a new window).
  - *Command Prompt*: Type "`cd <path>`" to change to the given (absolute or relative) location. "`cd`" stands for "change directory."
  - *Example*: `cd c:\Users\chuttenh\Desktop\problems02`
- See the files in my current directory.
  - *Explorer*: Still just sort of there, listed as icons.
  - *Command Prompt*: Type "`dir`" to show the files in the current directory, or "`dir <path>`" to show the files in the given location. "`dir`" stands for "directory."
  - *Example*: `dir \Users\chuttenh`
- Launch an application or open a file.
  - *Explorer*: Double click on a program or document icon.
  - *Command Prompt*: Type in the program or file's path. This will actually work with both applications and documents; the former will launch the program, and the latter will open the document just like clicking on it in Explorer.
  - *Example*: `python problems02.py`
- Move a file.
  - *Explorer*: Click and drag.
  - *Command Prompt*: Type "`move <file> <path>`" to move the given file to the given location. Be careful - if `<path>` is a path to a file (rather than a directory), you'll overwrite it (i.e. you'll rename `<file>` instead of or in addition to moving it).
  - *Example*: `move problems02.py c:\Users\chuttenh\Desktop`
- Delete a file.
  - *Explorer*: Delete or click and drag to the recycling bin.
  - *Command Prompt*: Type "`del <file>`" to completely delete the given file. Note that this does not move it to the recycling bin, it just goes byebye.
  - *Example*: `del problems02.py`
- Rename a file.
  - *Explorer*: Enter or click and wait.
  - *Command Prompt*: Type "`ren <old> <new>`", where `<old>` and `<new>` are both paths to files.
  - *Example*: `ren problems02.py ILikeToast.py`
- Create a directory.
  - *Explorer*: File/New/Folder or right-click.
  - *Command Prompt*: Type "`mkdir <path>`" just like in the assignment. If the given path already exists, it'll give you a little error message without hurting anything.
  - *Example*: `mkdir ..\problems03`

1.2 Command Line Quick Reference

Below, `<file>` represents a path to a document file, `<dir>` a path to a directory, and `<path>` a path to either one.

Macintosh

Action	Command	Example
Repeat last command	Up arrow	
Complete a partial filename	Tab	`cd prob<press Tab here>`
Move to start of line	Control-A	
Move to end of line	Control-E	
Change current directory	`cd <dir>`	`cd ~/Desktop/problems02`
Launch a program/file	`<file>`	`python problems02.py`
List files	`ls` or `ls <dir>`	`ls ~/Desktop`
Move a file	`mv <file> <path>`	`mv problems02.py ../`
Rename a file	`mv <file old> <file new>`	`mv problems02.py problems42.py`
Delete a file	`rm <file>`	`rm problems02.py`
Create a directory	`mkdir <dir>`	`mkdir problems03`

Windows

Action	Command	Example
Repeat last command	Up arrow	
Complete a partial filename	Tab	`cd prob<press Tab here>`
Move to start of line	Home	
Move to end of line	End	
Change current directory	`cd <dir>`	`cd c:\Users\chuttenh\Desktop\problems02`
Launch a program/file	`<file>`	`python problems02.py`
List files	`dir` or `dir <dir>`	`dir \Users`
Move a file	`move <file> <path>`	`move problems02.py ..\`
Rename a file	`ren <file old> <file new>`	`ren problems02.py problems42.py`
Delete a file	`del <file>`	`del problems02.py`
Create a directory	`mkdir <dir>`	`mkdir problems03`

1.3 All Those .Exes

"Live in Texas" - George Strait, "All My Ex's Live In Texas"

That's probably more than you ever wanted to know about about the command line - and after all that, it could be argued that the command prompt by itself is fairly useless. It exists as a tool for interacting with files and programs; it's not much fun if you don't use it to run other things! And of course, what we've been using it for so far (and will continue to use it for) is to run Python. When we say "run Python," though, we're really talking about a suite of interacting programs: an interpreter that makes your source files do cool stuff, and an editor (jEdit) that creates .py source files so that you don't have to enter every command individually. Oh, and so the Mac users don't feel bad, note that I don't intend any Windows bias by referring to these as .exe files - it's just a convenient way of specifying "Python the program on your computer" versus "Python the abstract language" or "Python the thing that eats rodents."

1.3.1 `python.exe`

`python.exe`, pronounced "python-dot-ee-ex-ee," refers to the Python interpreter, as we've said a few times now. But what does that mean? Well, think about how your programs start out. A .py file is just plain text, and given the current state of natural language processing (the field of making computers understand normal human languages like English), a computer is no more able to understand that text than it is the text of A Clockwork Orange (then again, neither is the average human being). An interpreter is any program that turns a human-readable source file (stored as text) into series of machine-executable commands (stored in memory as binary gibberish) and executes them to make the computer do something. As with most commands that you can execute from the command prompt, `python.exe` accepts command line arguments, one or more words, options, or filenames after the command itself that modify how it executes. If you run Python with no arguments, you'll get a bit of information followed by a new prompt that looks something like this:

```
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

As discussed in the notes, this is an interactive Python session, where you can type Python statements into the interpreter by hand one-by-one. That gets old real fast. Trust me. Instead, most programs are stored in source files that are just plain text, with a special .py extension, and run by providing that file name to the Python interpreter on the command line:

```
python problems02.py
```

This will run the entire contents of `problems02.py` as a single Python program all in one go. Note that you do not need to specify the `.exe` part of the program name, even on Windows (although you can).

However, in order for Windows or MacOS to know what you mean by `python`, the operating system must know how to locate that program from the command line. In order to make sure everyone's on the same page with respect to Python, the following instructions provide everything you need to get it up and running:

1. Go to `http://www.python.org/download/`, download the appropriate version of Python 2 for your operating system, and install it. As of this writing, the following caveats hold:
   (a) Many 64-bit extensions don't work very well, so even if you're running 64-bit Windows, install 32-bit Python.
   (b) The latest version is 2.7; these notes were written for 2.6, and the two are essentially identical. Python 3 has been released, but is not yet compatible with many common add-ons and modules, so we will not be using it just yet (at least not this year!), and you should only install it if you want to get to know it separately on your own.
2. After Python has fully installed, add it to your path:

(a) On Windows:

i. Go to the Control Panel, System, Advanced System Settings. This should bring you to the Advanced tab of a small dialog box.
ii. Click the Environment Variables button in the lower right.
iii. In the bottom half of this window, labeled System Variables, find the variable named `PATH` or `Path`.
iv. Click to highlight it, then click the Edit button.
v. Make sure your cursor's at the rightmost edge of the Variable Value text box, then type "`;C:\Python26`" without the quotes.
vi. Click Ok, then Ok, then Ok again. Logout and back in, or restart your computer.

(b) On MacOS:

i. Congratulations! The Python installer should automatically include `python` in your `PATH` so it's accessible from the default Terminal application.

1.3.2 jEdit

The yin to Python's yang, though, is jEdit, a "Programmer's Text Editor," to quote its web site. And if you're wondering why I'm not referring to it as `jedit.exe`, check out what lives in `C:\Program Files\jEdit` (if you're a Mac user, look over somebody's shoulder). It's not an executable file! jEdit is actually itself written in Java, so what ends up running is a Jar file, which is kind of like an .exe file except not.

You can think of jEdit as Notepad on frighteningly scary performance-enhancing drugs. What Word is to Wordpad, jEdit is to Notepad. It only knows how to edit plain text files - it doesn't do any sort of fancy formatting, underlining, tables, all that sort of thing - but it knows all sorts of crazy ways to edit them. At its core, though, all jEdit is doing is manipulating plain old textual characters in a plain old file, so don't be afraid of it! Don't forget that you can browse through its help files by going to the Help menu, and there's all sorts of introductory information linked from its web page as well. The five features that make jEdit a suitable choice for our purposes (or for any scripting language) are:

- Syntax highlighting. jEdit knows how to paint different parts of a Python program different colors. No explanation needed - as soon as you start your first program, you'll see what I mean.
- Line numbering. Python errors tell you on which line the error occurred, and jEdit tells you which line you're currently editing. When you need to fix a bug, making those two numbers match is key.
- Parenthesis and bracket matching. When you close a) or], jEdit will highlight the corresponding [or (. In complex statements, this is a lifesaver.
- Non-lobotomized search-and-replace. Try finding a text pattern containing a wildcard using Notepad.
- Automatic indentation and tab/space conversion. I don't use this a lot myself (I'm a hard-tabs kind of guy), but it can be a boon for Python in particular, which is a bit OCD about whitespace.

What do we want you to know to get started? Well, the easy parts are all of the usual file operations: New, Open, Close, Save, and Save As are all available on the File menu. Similarly, it has all of the standard editing operations under, well, the Edit menu: Undo, Redo, Cut, Copy, Paste, and Select All. Almost all of these (and almost every other operation in jEdit) has a corresponding keyboard shortcut. Open is Control-O (or Command-O on a Mac), Copy is Control-C (you've probably seen that one before), and so forth. But because jEdit is a "programmer's" text editor, it provides a whole slew of useful shortcuts that Notepad (and even Word) excludes. The table below contains only a few of the ones I use most; don't be afraid to dig through jEdit's menus and try things out. As long as you don't save over any important files, you can't do any damage - really!

jEdit Quick Reference

Task	Shortcut	Menu	Effect
New	Ctrl-N	File	Create new empty file
Open	Ctrl-O	File	Open existing file
Close	Ctrl-W	File	Close current file
Save	Ctrl-S	File	Save current file
Save As		File	Save current file with new name
Undo	Ctrl-Z	Edit	Undo previous action
Redo	Ctrl-E Ctrl-Z	Edit	Redo previous undone action
Cut	Ctrl-X	Edit	Remove selection, place in clipboard
Copy	Ctrl-C	Edit	Copy selection to clipboard
Paste	Ctrl-V	Edit	Insert contents of clipboard
Select All	Ctrl-A	Edit	Select entire current file
Go to Line	Ctrl-L	Edit	Move cursor to line number
Find/Replace	Ctrl-F	Search	Find/replace given text
Next Editor	Ctrl-Page Up	View	Activate next open file
Previous Editor	Ctrl-Page Down	View	Activate previous open file
Indent Selection	Alt-Right Arrow	Edit/Indent	Move selected lines one tab to the right
Unindent Selection	Alt-Left Arrow	Edit/Indent	Move selected lines one tab to the left
Select Text	Shift-Left/Right Arrow Shift-Home/End Shift-Page Up/Down		Select text while moving cursor
Move by Word	Control-Left/Right Arrow		Move cursor by word rather than character
View Line Number			Watch the lower left corner of your window

To download jEdit, visit `http://www.jeditor.org`

To make jEdit the default editor for .py files on Windows:

1. Start up a command prompt.
2. Type the following, including quotes, and modifying the paths as necessary to reflect what's actually on your computer:
   ```
   ftype Python.Source="C:\Windows\System32\javaw.exe" -jar "C:\Program Files\jEdit\jedit.jar" -reuseview
   "%1"
   ```
3. This command should silently complete. If you don't get any weird errors, type the following two lines:
   ```
   assoc .py=Python.Source
   assoc .pm=Python.Source
   ```

To make jEdit look less ugly on Windows:

1. Go to the Uilities menu and select Global Options.
2. Select Text Area, change Anti Alias Smooth Text to "subpixel", and check the Fractional Font Metrics box.

1.4 An Alternative

"You oughta know." - Alanis Morissette, "You Oughta Know"

Truth be told, I really dislike jEdit. It's better than Notepad, but if you're looking for a text editor, there are better options (UltraEdit or Notepad++ on Windows, BBEdit or TextWrangler on a Mac). For that matter, we're not even interested in it as a text editor - we want a Python editor. And if you're after a Python editor, there are way better options. Allow me to suggest that everyone should, instead of jEdit, at least consider using **Eclipse**. Eclipse was originally developed as, among other things, an integrated Java programming environment (referred to as an Integrated Development Environment or IDE). It can be supplemented with the freely available **PyDev** tool, however, to become an equally powerful Python IDE. This integration means that a single tool (Eclipse) replaces the whole bundle (jEdit, the command prompt, and `python.exe`). This, of course, is a Good Thing.

"What!?!" you say? "Someone set up us the bomb? We went through all of that nonsense with PATHs and the command prompt and Python and now you tell us we didn't have to? Ahh!" But wait! Before you defenestrate me (I score points for using that in casual conversation, right?), realize that this fruit of knowledge comes with the usual disadvantages: Eclipse is itself complicated, arguably more so than jEdit, the command prompt, and Python separately. However, by the same token, it's also massively more powerful. Its syntax highlighting is better, its error checking is better, it autodetects and autogenerates all sorts of things (yes, it autogenerates certain pieces of code), it allows integrated debugging, in will automatically organize your source files... the list goes on and on.

So if I've managed to tempt you into trying this daunting morsel, begin with the following instructions:

1. Go to `http://www.eclipse.org` and click on the Download Eclipse button on the upper right.
2. Now we have a platform split (and yes, "for Java Developers" is right... for now):
   (a) If you're using Windows...
      i. Choose "Eclipse IDE for Java Developers" and the 32-bit version (this is true, for complicated reasons, even if you're using a 64-bit OS), then on any of the mirror locations, then save the .zip file somewhere.
      ii. Expand the .zip file (depending on your system, you probably just need to double click it), which should result in a folder named `eclipse`.
      iii. Using Windows Explorer, drag the `eclipse` folder into your `C:\Program Files` folder.
      iv. Then, inside of the `C:\Program Files\eclipse` folder, you should see a file named `eclipse.exe`. Drag this file into your Start menu, then into the Programs menu, and then release it. This should create an `eclipse.exe` entry in your Programs list, which you can then use to launch Eclipse.
      v. Alternatively, you can right click and drag the `eclipse.exe` file to your desktop and choose "Create shortcut here" to create an icon on your desktop instead of in your Start menu.
   (b) If you're using a Macintosh...
      i. The web site should autodetect this fact, and the "Eclipse IDE for Java Developers" link will say "Mac OS X" beside it. If so, you can download the file as per the first Windows instruction. If not, click on "Packages for," then on "Mac OSX." If you get it, click through the warning screen about requiring 10.3 or higher, pick a mirror, and save the .tar.gz file somewhere.
      ii. Expand the .tar.gz file (by double clicking it), which should result in a folder named `eclipse`.
      iii. Using the Finder, drag the `eclipse` folder into your Applications directory.
      iv. To launch Eclipse, double click on the Eclipse icon in the `eclipse` folder; it'll look like a pretty blue stripey ball. This is one case in which the whole Macintosh easy-to-use thing really shines!
3. When you start Eclipse for the first time, it'll ask you to "Select a workspace." The default is fine, so check the "Use this as the default and do not ask again" box and press ok.
   (a) Do, however, notice the path that it uses! Just as you saved your jEdit .py files under `C:\Users\chuttenh\Desktop\pr` or `~/Desktop/problems02`, Eclipse will default to saving your .py files to something like `C:\Users\chuttenh\workspa` or `~/Documents/workspace/ProjectName`. So that's the directory you'll have to browse to when you submit them online.

4. Then (this is still the first time only) you'll see a mostly blue screen with some funky icons in odd places. Ignore them. The only one you want is the little curvy arrow in the upper right, which will get rid of this welcome business and get us to the real stuff. When you mouse over, it'll say Workbench; click on it, and your world will change.

5. You're now looking at the Eclipse main view, also known as the Resource perspective. It consists of four main parts:

   (a) The Package Explorer on the left, which lists projects (collections of related files), files, and folders.

   (b) The Outline on the right, which provides some random Python information (I don't really use it much; think of it as a bird's-eye view of your program).

   (c) The Editors in the upper middle, which are just like text editing windows in jEdit (but way better!)

   (d) Various tabbed windows along the bottom, primarily Tasks, the Console, and Problems. If these aren't all present, you can get to them by going to the Window/Show View menu and selecting the missing window (don't forget to select Other... and browse through your options if something you want isn't in the default list).

6. Now let's install PyDev, since Eclipse only knows about the Java programming language by default. We'll follow the instructions at: `http://pydev.org`

   (a) From the Help menu, choose Install New Software.

   (b) Click the Add button, type PyDev in as the Name, and the URL `http://pydev.org/updates` for the Location.

   (c) Click Ok, then select your new PyDev repository from the Work With dropdown menu (if it doesn't auto-activate).

   (d) Check the box beside PyDev, then click Next.

   (e) Click Next again for no real reason, then accept the license and click Finish.

   (f) While PyDev is installing, it will ask you to check a box accepting the Aptana certificate; do so, then click Ok.

   (g) Restart Eclipse when prompted at the completion of PyDev's install.

7. The first thing you must do after PyDev installs (as described at `http://pydev.org/manual_101_root.html`) is configure the Python interpreter. This is a one-time deal, so it only happens the first time you start Eclipse after installing PyDev:

   (a) Go to Windows/Preferences and expand the Pydev item in the left pane of the resulting dialog box.

   (b) Select Interpreter - Python, and on the upper part of the resulting right pane, click New.

   (c) Type "Python 2.6" for the Interpreter Name (without quotes), click Browse, and find your Python binary (e.g. `C:\Python26\python.exe` on Windows).

   (d) Click Ok, then Ok, then Ok, then Ok.

8. Finally, let's create a new Python project - say, for your first assignment.

   (a) First, activate the Python perspective by going to Window/Open Perspective/Other and select Pydev.

   (b) Right click anywhere in the blank area of the Package Explorer pane and select New/Project...

   (c) Expand the Pydev folder in the window that appears, click on the Pydev Project line, and hit Next.

9. In the Project name line, type something like "Problems 02" (without the quotes, of course), and then press Finish.

10. Let's try creating and running a `problems02.py` file the Eclipse way.

    (a) Expand the Problems 02 project, right click the src directory, and select New/Pydev Module.

    (b) Ignore the Module line. In the Name line, type "problems02" (note the lack of the .py extension!)

    (c) Press Finish.

11. Whoah! Eclipse just magically generated your `problems02.py` file, including a proto-docstring that you can reformat to suit the course requirements. Let's ignore that for now, though, and demonstrate how Eclipse can run Python files by leaving the docstring as-is and adding the following line to the bottom of the file:

```
print( "Hello, world!" )
```

13. Let's run this thing! From the Run menu, select Run, or just hit Control-F11. When Eclipse asks you how to run the file (which it should only do the first time), select Python Run. If all goes as planned, the string "Hello, world!" should appear in black text in the Console window at the bottom of the screen. This Console takes the place of output that would normally go to the command prompt: anything you print appears in black text, and errors appear in red. Congratulations - you've written your first Python program the cool way!

This is not, of course, to disparage jEdit; I'm just personally a big fan of Eclipse, and goodness knows there are professional programmers who use jEdit and the command prompt instead. For that matter, development environments are something of a religious matter among programmers (Google "vi vs. emacs" sometime when you're bored), and there are nearly as many cool ways to develop as there are cool developers. So consider this little tutorial an option (well, a suggestion, really) rather than a requirement.

To offer some parting words about Eclipse, keep in mind that we've barely scratched the surface. If you thought jEdit had a lot of features, Eclipse blows it away. You can configure almost everything that it does, and it provides several additional perspectives and view windows aside from the ones that we've explored. If you're curious, poke about in the Window menu Preferences dialog, and you can see the veritable plethora of settings and options made available to you whether you want them or not. Finally, feel free to ask me more questions about Eclipse! I do strongly encourage you to try it out, and we're more than willing to spend time during office hours helping you to set it up.