Introduction to Programming, Part 1

Introduction

Goals

Welcome to Introduction to Programming, Part 1. In the Library field an increasing portion of any information organization's holdings consist of Born Digital objects; objects which are pure information and which have never had a particular physical form. These can be audio and visual materials stored in digital formats. They might be scanned text, or text that has never been bound up on paper; ebooks, blog postings, web sites to name a few. Many objects may be pure data; collections which were never meant to be processed by a human mind. Scientific data falls into this category and often consists of huge sets of numbers to be processed by software. Increasingly databases are important and valuable works for reference use. So Librarians are now required to manage exactly the sorts of things that Computer Programmers generate and manipulate.

So, where might Librarians have to program? I'll throw out some examples from my own career. Some of you might have read briefly about these in my LJA interview. I am a recovering IT Professional who was introduced to the Library field through work at a major university's digital library group. So my career has not been all that traditional. However, it has been a harbinger of the changes taking place in the field. During and after my education, I held down a job curating the holdings of a publisher of data. My organization published all manner of linguistic data for use in research on Computational Linguistics, Natural Language Processing, Artificial Intelligence, even Ethnolinguistic research and Sociolinguistics. Our holdings included formatted text (SGML, XML and so on), recorded speech, video, and derived data from any or several of the other types. A large part of my job involved Digital Preservation. Just as with physical works, digital objects wear out over time. In the case of digital objects, they wear out by becoming unreadable to current software. They are subject to physical degradation, but that's a whole 'nuther story. Generally, the data is still there, but it's inaccessible. In almost every case, the data can be rescued by transforming it. For instance, no one uses the Entropic audio format, but at one time, it was a player in the digital audio arena. Likewise, very few audio players recognize the NIST Sphere format. And, my favorite, back in the early days of computer audio, '.wav' on the end of a file meant any audio format. So, you have a collection of audio files from the early 1990s and nothing reads them. Knowing some basic information about the recording and how it was digitized means you can remove the formatting header and replace it with another. How do you do that? You write a program or script that performs the operations needed to accomplish this task.

Even if you are a Reference Librarian you are not immune from the technologization of knowledge. We all work with citation databases. For the most part, you're compiling resources and

bibliographies for individual users. However, if you work at a university library, you often wind up working with entire departments. They will sometimes come up with industrial scale requests for citations. They may be interested in doing some sort of bibliometric analysis for academic advancement. They might want to compile a list of publications to present to funding agencies. These requests might encompass twenty or more professors and span an equivalent number of years. Doing this work by hand through the web interface is prohibitive in terms of time and in terms of the capability of that interface. Most of the big databases have programming APIs which let you automate large requests. API stands for Application Programming Interface. An API allows one piece of software to access the internal information of another. They allow this access without exposing any code or methods. The system you access through an API remains a Black Box.

This course will not turn you into a seasoned programmer, but it will do two things: Introduce you to the fundamental data and control structures of any programming language, and introduce you to some Abstract Data Types which comprise solutions to the vast majority of computing tasks. I am not going to teach you how to code a Mobile App, nor will I show you how to do Web Programming, nor any sort of applied programming. My goal is to provide you with the computational foundations used in all programming solutions. To that end, I'm covering program structure, data and control structures. What I teach here is common to all programming languages though the syntax will differ greatly between languages. Nonetheless, when you move from Python to Java, Perl, PHP or Ruby, you'll find the same objects, for loops, and data structures merely cast into another form.

The computer language used in the class will be Python. Python is a very popular programming language in the sciences and is gaining ground in other areas. I am using Python for this course because it is a fully featured high level programming language. Python, for many reasons, is very easy to learn. I have personally used Python in several capacities. On a number of occasions, I have used the language to automate tasks in System Administration, audio and video editing, pattern matching, and Natural Language Processing. Python, in many ways, is this generation's Pascal, a solid beginner's language which possesses all the features of more serious programming languages like C or Java. You may never need to make the jump to these languages, though learning Python will set you up to rapidly learn the others.

Course Textbook

This course has a textbook. It is Think Python by Allen B. Downy. This is an excellent book on programming and is part of a series written by Downy who is a Computer Science professor at Franklin W. Olin College of Engineering. You can purchase this as a print book or read it online at http://www.greenteapress.com/thinkpython/html/thinkpython001.html. This book will be used throughout the course series. Downy has many other excellent books on computing topics at Green Tea Press. I encourage you to check this site out in detail.

I recommend the following books for getting into the deeper topics of Python and programming in general.

- **Learning Python by Mark Lutz**—Aimed at the advanced student or seasoned professional, this book teaches Python in depth, but assumes you know a great deal of computer science. This is a great book to take you to the higher levels.
- **Programming Python by Mark Lutz**—Mark's books are encyclopedic in their scope. They not necessarily the best texts to learn from, but they are tremendous references. And, I'm referring in part to their size. They're phonebook sized texts which cover almost every topic in depth.
- **Python Cookbook by David Beazly and Brian K. Jones**—O'Reilly Publications has cookbooks for almost every language there is in computer programming. These books are invaluable for the practical examples and discussions of common problems in programming.
- **Natural Language Processing by Steven Bird, et. al.**--This book is about the Natural Language Toolkit for Python. This was originally intended for Linguists. However, it covers a variety of topics which pertain to the work Librarians do in the realm of Digital Libraries. This is your goto reference when it comes to web scraping and processing text. It's written by an old colleague of mine.
- **System Analysis & Design Methods by Jeffry Whitten and Lonnie D. Bently**--This is a standard textbook in Systems Analysis courses. This book does not teach programming. Rather it teaches how to analyze business structures and solve problems. This book contains a wealth of information about modeling data and systems. This is the heart of System Analysis and it is all too often overlooked in both Computer Science and Information Science programs. This textbook is expensive. Look for it used and get the cheapest one you can find. The information in it does not really expire.

Course Overview

I will be presenting a great deal of information in a very short period of time. I doubt we'll achieve mastery, but I hope to develop some level of competency or at least understanding. Every lesson will have a forum and a chat session associated with it in which we can address questions or concerns. I encourage students to help one another. The goal is to learn, not to compete.

Statements and Expressions

In this lesson, we learn the basics of writing programming commands. We will also see an overview of data types and structures. This lesson will take you through the steps of building complete

and valid program statements from simple expressions

First Programs

This lesson teaches how to string statements into complete programs. In this lesson, you'll learn about control structures such as for loops, while loops, and if statements. The programs you write in this lesson will be very simple, but the skills used will translate to much larger bodies of code.

Data in Depth

This lesson goes into Python's basic data structures in depth. An overview of objects will be presented. Operations on variables and their use will be covered.

Functions and Dictionaries

The them of this lesson is modularity. We'll learn how to place code into functions, organize our programs and use functions effectively. We'll also cover modules and how to import code from outside our programs. The lesson finishes with an in depth treatment of Dictionary data types, a fundamental abstract data type used in all programming languages.

Assignments and Evaluation

Included in each lesson you will find a set of exercises. Each one requires you to write statements and short programs. Each assignment is worth twenty points.

Working Environment

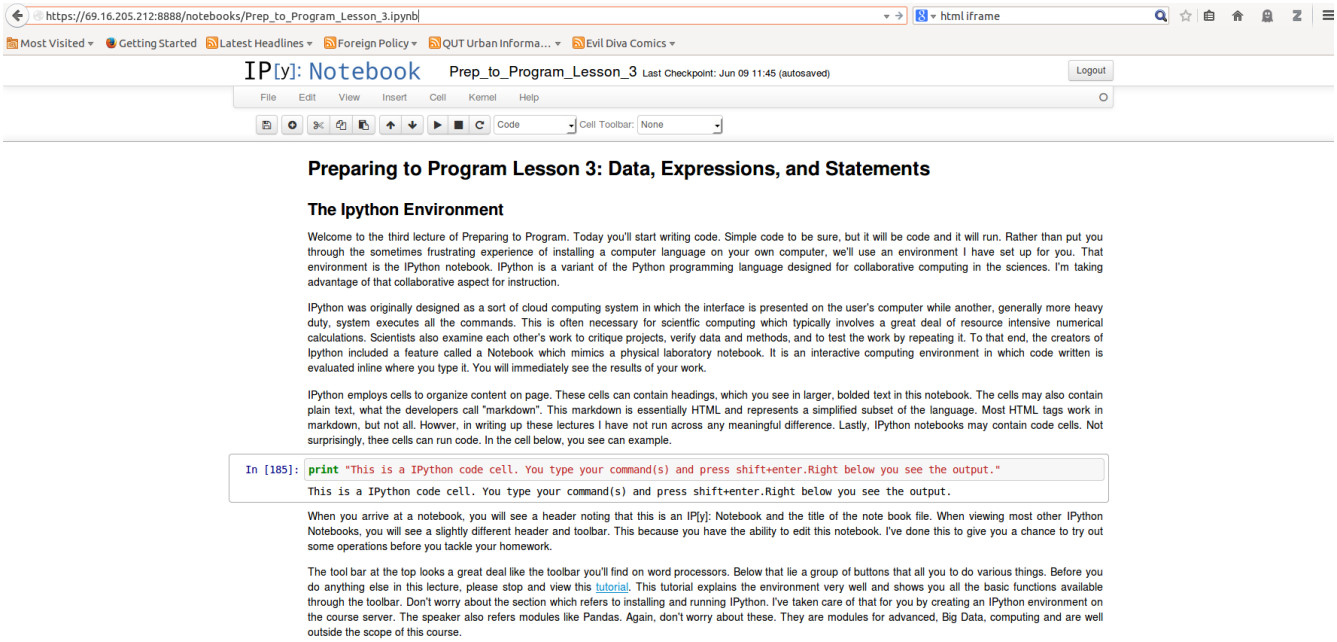
For this class, we will be using a web based programming environment. Rather than put you through the sometimes frustrating experience of installing a computer language on your own computer, we'll use an environment I have set up for you. That environment is the IPython notebook. IPython is a variant of the Python programming language designed for collaborative computing in the sciences. I'm taking advantage of that collaborative aspect for instruction.

IPython was originally designed as a sort of cloud computing system in which the interface is presented on the user's computer while another, generally more heavy duty, system executes all the commands. This is often necessary for scientfic computing which typically involves a great deal of resource intensive numerical calculations. Scientists also examine each other's work to critique

projects, verify data and methods, and to test the work by repeating it. To that end, the creators of Ipython included a feature called a Notebook which mimics a physical laboratory notebook. It is an interactive computing environment in which code written is evaluated inline where you type it. You will immediately see the results of your work.

I provide a quick introduction to the Ipython notebook in the lesson, but the lesson itself is an Ipython notebook. For that reason, it's probably a good idea to go over the way an Ipython notebook works here. An Ipython notebook is a dynamic web page which may be edited by the user, namely you. Ipython runs on its own independent web server which means the lessons and exercises have a different port number than you're used to. For instance, let's look at an example url https://69.16.205.212:8888/notebooks/Prep_to_Program_Lesson_3.ipynb. First off, it uses an IP address, numbers rather than words. Second, you notice and additional 4 digit number following the colon, ":", at the end. That number is a port number. Specifying it on the end means that the url takes you to the Ipython notebook server rather than the regular web server on this machine. Why does Ipython use a dedicated server? The answer is that Ipython uses some very customized transactions to take what you type into on the web page and executed in the Python programming environment underlying everything. Each student will have their own Ipython notebook to use for their work and to make notes in. Each student will have their own notebook with the lesson and homework incoporated into a single page. The notebook urls and passwords will be assigned in a special course forum. Please give it a try immediately and let me know how things work. One issue you will encounter is a message from your browser about the ssl certificate not being recognized. This is because I had to create ssl certificates that I signed myself. As I am not a recognized signing authority, your browser will protest and ask if you want to create a security exception. Do so. I apologize for this slight glitch, but Ipython is still somewhat experimental and does things a little differently.

Once you enter the password, you will see a page that looks something like this:



The first thing you see at the top of the screen is the toolbar. This toolbar has two levels. One is menubar that has several pulldown menus. The second level consists of various buttons for commonly used tasks such as saving a notebook or navigating the page. Below the toolbar, the page is divided into into cells. In this particular page, the title headings are their own cells. The explanatory text is plain HTML markup. Same as any other web page. The cell with a rectangle drawn around it is different. It contains Python code and that cell can be executed, meaning the code inside it will be executed and the results, if any displayed below. Any time you run code, the overall page appearance may change depending on what the code does. If it prints something, you will see that below code cell. Likewise any sort of graphics and error message.

I also encourage students to download and install Python on their own systems. Python is a mature and robust language with a great many third party distributions and versions, such as Ipython. One I recommend is Active State Python. Active State produces refined and well supported distributions with easy to use installers. Their basic, individual distribution is free. You can find it at http://www.activestate.com/activepython/downloads.