INTRODUCTION TO PROGRAMMING – INSTRUCTOR MANUAL

Introduction and the Processing Approach

While this course is taught using the Processing IDE and language, teaching Processing to students is fundamentally the same as teaching any other Programming language. The educational goals by the end of the course are essentially the same as many introductory courses: students learn the basic tools (conditionals, loops, arrays, functions), and learn how to solve problems with them.

The primary shift that instructors will find with this course is that there is a large de-emphasis on String processing. Many text-books and courses spend a great deal of time on getting textual input from a user (usually in the form of prompt, full-line input), validating the input (generally with a fragile while loop), parsing that input (often with arbitrary, language-specific APIs such as substring, string searching, or tokenizing), and then formatting text output for a console. From a pedagogical perspective, such operations are useful for a student to learn how to work with raw data and a given set of tools to solve a goal. However, there are several limitations:

- Students must memorize complex APIs (e.g., Scanner.hasNext, String.indexOf) that often include concepts they have not yet learned, while they are still trying to learn the basics about variables, syntax, etc.;
- These are language dependent, and experienced programmers will look up the particular APIs as needed anyway;
- There are better, more robust tools for String processing that professionals generally use;
- The dialog model of pausing a program to get a single line of input from a user is dated. Very few instances of this exist on modern computing platforms, so it is more difficult for the students to relate to the work they are doing;
- Formatting fixed-width text output is a highly dated problem, relegated primarily to command-line tools, further making it difficult for students to relate to the work;

- The token-wrangling benefits gained from such String tasks can be analogously learned when working with arrays.

As such, while Strings and characters are covered in the course, they are primarily for basic operations only.

The second big shift that instructors will notice is that students learn state machine management. While this is not taught explicitly in the course, Processing is setup such that the primary program calls a draw function that is called on a timer, defaulting to 60 times a second. A component of problems that students solve is to, each time they draw, to evaluate the current state of the program, update the state, and then draw. When making an interactive program that reads the mouse or keyboard, this state management is non-trivial.

Although Processing is graphics based, the mathematics and geometry used has been kept to a minimum. Students may have to refresh their basic sine and cosine trigonometry (grade 9 level) to go from polar to Cartesian coordinates (given an angle and a radius, what is the x, y). In addition, students may struggle slightly with the spatial nature of some problems, although there are only a few basic templates that get re-used throughout the course.

Instructors will notice a shift from the classic input->process->output model that has generally been used to teach computer science, toward an interactive system model where the beginning and end are less well defined, and the main problem is managing real time input. There is an argument that the latter is much more reflective of modern computer programming. Any interactive system (desktop application, tablet or smart phone application), network application, robot system, etc., fall more in line with the event-driven and state management model, and the instances in computing of the classic command-line input->process->output are dwindling.

While this sounds complex, in reality instructors will find that teaching the course is essentially the same as before, except with less frustrating String syntax, and more engaging results.

Pedagogical Approach

This course is designed to teach students programming through extensive practice. Programming is something that cannot be learned from reading and observation only, and must be practiced, similar to learning a musical instrument- you can read sheet music all you want, but it won't help you learn to play it on a piano. As such, there are assignments that have significant grading weight, to encourage students to get their hands dirty with real problems.

For evaluation, most of the marks are encompassed in the course's exam components. There are several reasons for this. One, this is a purely individual exercise, whereas programming assignments can be collaborative, or can benefit from on-line help. Two, exams provide an opportunity to probe a student's deep reaches of material understanding, that they should have gained through practice, but can be hard to include in a programming assignment.

To support the practice model of learning, each unit has a large number of exercises that students should practice. The book conveys the importance of doing these exercises. Further, each exercise is marked in difficulty, as either Bronze, Silver, or Gold, and students are told that they can use the exercises to gauge their own understanding of the material. To pass the course they should be comfortable with the Bronze exercises. If they expect a B, they should be doing the silver exercises. A students should be working through many of the gold exercises.

In general, it is not recommended to post or share solutions to the exercises. These should be seen as an engagement opportunity, where students can engage the instructor or each other to work through bugs. Getting stuck on a bug, and going back and reviewing, is an extremely effective learning method. Conversely, when provided with sample solutions, students often peek at it and see it as a reasonable solution, without necessarily understanding the nuances behind each aspect.

Tour of the Materials

A collection of 19 Units have been provided for students, covering the range of topics in the course. Pre-Midterm generally goes up until Unit 11 (just before loops), with the remainder for the final.

The units cover every aspect of the course, including downloading and starting the required software.

Each unit targets a specific programming concept, and the provided material will fully explain the concept to the student and work through several examples, in addition to providing do-yourself exercises.

Getting Started

The best way for an instructor to get started is to download and install Processing, and to work through the examples at the end of each chapter. Instructors can expect that student questions and problems will fall in line with what they have seen previously, in regular Java. For example, struggling with a nested loop or boolean logic, and less on the geometry and graphics.