Sample of Work - Lesson Plan: Intro to Engineering through Programming

Boris Taratutin – December 16, 2011

# Lesson Summary

This will be a short lesson serving to be an introduction to engineering, taught through the lens of software/programming. It will last 2.5-3 hours, and is intended for undergraduate students who have no experience in programming (really – no sort of background required!), but are curious to learn some basic skills in the area. Within this short time span, we’ll cover everything from the basics of Python (a modern programming language) to applications of software/programming in the world around us, explore some more advanced programming concepts, work together through a number of small programming problems/challenges, and also spend some time discussing how programming might be applied to areas outside of engineering – specifically, the majors and interests of the people in the class. Students will leave with skills, confidence, and resources to pursue programming post-this lesson.

The lesson format will be an interplay of interactive lecture, balanced with hands-on programming and time for experimentation. There will be non-trivial teamwork and discussion, and plenty of support from peers and the instructor – so much so, that by the end of the class, I guarantee that everyone will have written an interesting and non-trivial computer program of their own!

# Goals for the Lesson (non-measurable)

* Introduce non-engineering students to the field of engineering, through programming
* Explore potential stereotypes or misconceptions students may have about engineering (being ‘too hard’ or ‘too abstract’, etc.)
* Help students find some sort of personal relation/meaning to programming (whether through fun, or a potential application to their field/major)
* Provide students with the tools and resources to continue learning programming (should they so desire to) after the lesson
* Build student self-efficacy and confidence in programming to enable them to continue learning programming (should they so desire to) after the lesson

# Objectives for the Lesson (measurable)

# By the end of the class, students should be able to…

* Have written multiple simple Python programs and be familiar (in terms of recognizing and being able to work with) with the fundamental structures/concepts in programming: *for* loops, *while* loops, *if-else* statements, methods, objects, etc.
* Have solved and implemented a non-trivial programming challenge or problem of their interest
* Be able to recognize the uses and applications of software in the world around them
* Leave the class with an idea for how they might apply programming to their personal interests/pursuits
* Walk away from the class and be able to point at a concrete piece of knowledge, conceptual change, or skill gained as a result of the lesson

# Materials

* Every student should bring a laptop (if they have one), I may need to borrow laptops otherwise
* Python installed on their laptop
* Projector for displaying slides
* (if lesson extension is to be run)
  + ~3-5 Arduinos
  + Physical, controllable objects (+ safety for them) – lamps, fans, LEDs, motors
  + Wires, misc. ECE components

**Audience**

* Undergraduate students at a Liberal Arts University with limited or no experience in engineering/programming

# Procedure

*Pre-Class*

* *Objective: minimize overhead in class and get a sense for where the students’ interests and abilities stand, so can better tailor the class to their interests and ability levels.*
* Students fill out pre-class interest survey (to be designed soon) & download & install Python on their computers (if can’t, no problems – will ask them to show up before class and I’ll help them)

*Introductions [10-15 min]*

* *Objective: establish an informal atmosphere where students feel comfortable with each other & the professor, so that, once we get started, they won’t be afraid to try, fail, and ask for help. Make resources and feedback mechanisms available to students*
* Write up (on board) timeline for what we’ll cover today (the outline form of this plan)
* Ask students to power up computers (will need them in a sec.)
* Round of introductions (start with self – some sort of interesting questions)
  + “Since we all may potentially not know each other, let’s do some introductions”
  + Start with self, and give facts: name, year, major, and [improvise] (one fun thing looking forward to over break?)
* Give students link to course materials and resources [link]
* Give students my e-mail, and tell them if I’m moving too slow or too fast or something really isn’t clear, can raise hand and ask to clarify, or g-chat them to me.
* Mention that there are students with very different backgrounds in programming in this class, and while we’re working through problems, definitely don’t be afraid to ask me questions, and/or to ask for help from your neighbors – peer learning & instruction is a great way to learn, whether you’re giving or getting help. Plus now that we’ve done introductions, we’re all one big class now
* Overall, this is supposed to be fun, and kind of expose you to another (new) field – hopefully you’ll find something interesting and worthwhile. There’s not (obviously) going to be any sort of grading (except for a small reflection at the end) and it ultimately what matters is if you took something away from the class

*Background [10-15 min]*

* *Objective: Orient students to what we’ll be covering in the lesson today, and provide motivation via personal anecdotes and contextualization*
* “So, as a bit of background..”
* Explain how I got into programming (HS, Olin, went away from it, came back in the end, and now away)
* Explain “why programming” as an intro to engineering
  + Easier to get into than electrical or mechanical (own experience anecdote)
  + Easier to teach self and learn on own (lots of resources online)
  + Highly applicable to a number of fields (will go over in more detail, but everything from web to research analytics to gaming to social media to military to personal projects..)
  + Ultimately, what’s cool about programming is you control everything. You start with a blank slate, and anything you add to it is your doing. If you’ve ever worked with circuits, they’re like little black boxes- things go wrong and you don’t know if you connected something wrong or if the chip died or… programming is (though it often doesn’t appear to be so), much more clear.
  + But the general principles of engineering are the same – the same design process and problem-solving, etc.
* Explain “why Python” as a programming language
  + Simpler than other languages (“hello world” example ppt)
  + Intuitive, quick to get up and running, lots of importable modules (xkcd comic)
  + Open-source (do ya’ll know what open-source is?)
  + Lots of free resources for learning it
  + Lots of companies are now starting to code with it (Twitter, Google, Facebook)
* Without further ado, let’s jump in!

*Jumping In & Getting Coding [30-45 min]*

* *Objective: Introduce Python & basic programming practices & concepts; get students familiar with them through hands-on work and writing of code.*
* Open up Python, do “hello world” example (interactive console mode)
  + Explain “scripting” (can prototype & play in console – without writing code)
* **[Jumping into Python.docx]**
* What did we just learn? (students respond, one are a time, around a circle)
  + Data Types (int, string)
  + List, Dictionary
  + String formatting
  + Variables
  + For Loops, While Loops
  + If-Else
  + Comparisons
  + Objects, Classes
  + docs.Python.org
  + Importing modules
* Summary of Programming Concepts so far

[Break] [5 min]

*Interactive Lecture about Programming Applications [30-35 min]*

* *Objective: Give an overview of the breadth and depth in the field of programming, and contextualize it in engineering and in design. Provide students with knowledge about the applications of programming, so that they may start to see programming and engineering in the world around them, and how they might be able to apply either/both to their fields/interests.*
* *[This section will be taught through the interactive lecture style, using powerpoint and leaving time for questions]*
* So now that you have some hands-on experience with Python and programming, hopefully this will provide some more context for thinking about programming. Here, I’ll talk about the broader field of programming, it’s applications, examples, and give an overview of all the things you can do with computer science.
* Perhaps through some of the examples that we worked through, you can start picking out the bigger pictures of programming (with Python) – it’s useful for.
* I’m now going to go over some of the areas of programming, and give examples and anecdotes about my experiences and knowledge. Don’t worry so much about taking notes here – the powerpoint is available online (link) which you can always reference later. This section is intented to kind of paint the ‘big picture’ of programming, all of its far-reaching aspects and how and where and why it’s used. With this in mind, I’m going to try to go relatively quickly through it, so don’t worry if you don’t memorize the individual concepts; it’s more intended to gie you an overall view of the field and (hopefully) be a bit interesting!
  + Ok, so what is software used for?
  + Easily prototyping & getting your code up & running – writing little scripts to do interesting things (file renaming example)
  + *Automating* long and/or tedious tasks (counting, generating, printing)
    - Parsing and/or counting long stretches of data (ie. Number of photos taken/day)
  + *Simulating* different scenarios (randomness, or modeling)
    - Chemical Reactions
    - Physics
    - Trebuchet Simulation
    - House heating
    - [Mod Sim course pitch]
  + Visualizing and making sense of data
    - Code-generated visualizations (i.e. network analyses)
    - Wikipedia browser
    - Twitter feel/mood
  + A way to ‘cheat’ (jk) on your math homework (math equations)
    - Linear algebra, matrix reducer
    - ODE solver
  + Web
    - Facebook, Google, Reddit, Wikipedia – they all run off of code
    - Different code; HTML & CSS for structure, Javascript, Java, & jquery for active interactios; C, Python, Java for back-end; SQL & PHP for databases (introduce all of these concepts)
  + Your desktop
    - You have to look no further than your desktop to see examples of software – somebody had to write everything; from MS word, to an image display to everything
  + Software-Hardware Connection
    - Ok so now that we’re mentioned your computer, it would be a good time to explain the connection between programs and the real world.
    - The question here is – how does a computer work? (how does code you write actually translate into changes on your screen) – can anybody take a guess? Throw some ideas out
      * Talk about Python Code, “compiled” into C, which is then translated to assembly/binary, which are basically “instructions” for the machine
      * What is the machine? Integrated chips system map
        + Explain hard drive, RAM, CPU
      * What are these chips made out of? (Transistors)
      * Make the connection between 101010100 and transistors. Then, multiply by a billion, and you get CPU speed. Cool, huh?
  + Robotics
    - The next logical connection, (since we’re on the topic of connecting software & hardware), is Robotics & Mechatronics
    - Robots example (big dog, igvc, path-planning, vision, gps, waypoints)
      * Sensors, gathering data, accelerometers
    - Mechatronics example
      * Mechatronic dino toys & consumer products
  + Mechatronics
    - Elevators
    - Bank systems (ATMs)
  + AI
    - Neural networks
    - Genetic Algorithms
    - NLP
    - Image Processing (Read: Photoshop)
    - Twitter Spam Bot Identification
  + Computer Science
    - Data Structures
    - Algorithms
* Broad statement/wrap-up about programming
  + Programming has something for everyone – from technical algorithms for Google or robotics to high-level API design for Microsoft or Facebook to simulation, data processing, or interface design & mockups
* Connect programming to engineering, in a broad sense [attached ppt]
  + What is engineering all about?
  + Teaching yourself, solving real-world problems, building things, iteration
* Contextualize engineering in the overarching process of design [attached ppt]
  + Needs, Problems, Opportunities, Ideation, Requirements, Refinement, Engineering, Implementation

*Applications of Programming to Students’ fields/interests [25-35 min]*

* *Objective: get students thinking, on an individual and class-wide basis, how programming might apply to them, how they might use the skills and ideas learned in the future?*
* Transition from the previous section by saying that we’ve now explored three things:
  + The motivating reasons for studying programming, and in particular, Python
  + The basics of the Python programming language
  + A number of applications of programming/engineering
* ..and how now the goal is to relate what you’ve learned to your own interests and goals. (Give an overview of what this section will contain; how think-pair-share will work, etc.)
* Think-pair-share activity:
  + (Explain how think-pair-share works)
  + For this activity, I want you to think about two things:
  + “what is a small programming challenge that you think you might be interested in working on at the end of this lesson?” (a series of example lessons would be shown on a ppt)
  + “given the numerous examples and applications that we’ve covered in this lesson, what is a potential bigger programming problem that you think would be interesting to tackle – that might apply to your bigger personal goals and interests?”
* For the ‘share’ part of the activity, each student would go around and verbalize their interest in a small problem to work on (which would help formulate teams after the discussion).
* After this round of sharing, we would transition to a more discussion-based setting, where students could share their ideas for how programming might apply to their fields of study/interests/goals, or express their lack of ideas. We would also discuss general perceptions and feelings of engineering and programming as a discipline, potentially through the lens of gender.

*Project-Based Programming Challenge [0-35 min]*

* *Objective: really try to solidify skills in programming via a hands-on, project-based exercise. Build student self-efficacy through working (and succeeding) on a programming challenge (at their level of ability/comfort). It will also be an opportunity for students to work more closely with the instructor and either get more tutor-like help, or even go on to learn wholly new topics not covered in the lesson.*
* Here the students have a choice – stick around and work on a programming challenge, to further develop their skills and build self-efficacy, or to be done with the class (take the post-class survey and leave)
* Students will be encouraged to pair up based on problem interests (or work independently, if they strongly desire), and told that the instructor will be available for help, clarification, and further instruction on any personalize topic.

*Post-Class Feedback Survey [2-10 min, post-class]*

* *Objective: get feedback from the students about how the course went; both qualitatively (through their analysis), and ‘quantitatively’ (through their ability to articulate ‘take-away’ or learned points).*
* Students will be asked to write out “1 minute reflection” cards – about what they learned, what they liked/thought was interesting, what could be improved, what they will take away, and what they will probably forget
  + Alternatively, they can fill out an on-line survey with the same questions, either post-class, or later at any given time

# Pedagogical Methods Used

*An articulation of the pedagogical methods embedded in the above procedure plan – not necessary, but makes the pedagogical methods used above more explicit.*

* *Pre-class surveys* to get a feel for students’ interests & abilities – and tailor/structure the course appropriately
* *Hands-on, problem-based and project-based* learning to develop the concrete skills in Python, and build interest and competence
* *Interactive Lecture* to deliver content to students – basic programming principles, broad-level applications, (things they can’t deduce on their own)
* *Pre-class introductions* to get everyone feeling comfortable with each other and start to build a sense of community in the class
* *Use of feedback* - both from the teacher to the students (to provide direction, encourage them, and build self-efficacy) and student to teacher (to shape class direction and have a say in the class once it’s done)
* *Discussion (think-pair-share)* to change up the format, build relatedness in the classroom, and help students develop a clear vision for how programming
* *Further Suggestions/Resources* to empower students to continue seeking and learning even once the class is done

# Assessment

The main form of assessment for this lesson will be instructor check-ins on student work, to be carried out throughout the lesson. This will be an opportunity to provide feedback to the students and help them along in the process. Ultimately, the ability of students to complete the small programming challenges and come up with ideas for application of programming will be the biggest metric for assessing the success of the class. I will potentially include some sort of knowledge-acquisition feedback metrics in the post-class survey.

# Acknowledgements

Special thanks go out to the number of people who shared their thoughts on students’ perceptions of engineering, and their experiences in teaching programming to non-engineeirng students: Ijeoma Ekeh, Amon Millner, (more to come).

*A more explicit explanation of the integration/application of motivational concepts (if desired) is included in the same-directory document, “Motivational Aspects of Lesson Plan.docx”.*