Code Camp at Knowles: Teaching Teachers to Program

Code Camp 2016 was a three-day professional development workshop where Knowles Fellows learned about programming for the classroom. In it, teachers learned how to use JavaScript to build web-based programs and control Google spreadsheets, and how to use programming concepts and computational thinking to teach math and science content.

In the previous two blog posts, I outlined why I think **programming has a place in all classrooms** (not just computer science ones) and how we approached **designing and teaching the Code Camp workshop**. This time, I would like to share two example problems that our teachers worked on during the workshop, both of which addressed key elements of computational thinking. In the first, our participants used JavaScript to add functionality to a class spreadsheet for creating fair groups from a class list. The second example utilized and highlighted many aspects of computational thinking without actually involving programming.

To start with a programming example, consider the problem of breaking students into groups. Imagine Dr. Jones needs to divide the 30 students in her chemistry class into six groups for a lab. She wants to make sure the groups are heterogeneous in terms of gender, likelihood to struggle with the material, and ethnicity. Those criteria for "goodness" (often called a **utility** function) are not always explicitly discussed; getting them out in the open is an important part of the software design process.

For Code Camp, this was a programming problem we gave our teachers: create a system that would take a list of names and an appropriate group size and convert that list into a randomized group assignment. We approached it as a Google sheets script, a fragment of JavaScript that can be run inside a spreadsheet, since

most teachers already use spreadsheets or spreadsheet-like gradebook programs, and leveraging spreadsheets means we can focus on the randomizer algorithm instead of worrying about persistently storing class lists and so on. To ease the work, I wrote all of the code necessary for loading spreadsheet data into a program and outputting from a program into a spreadsheet in an example **Google Sheet**.

While it may sound a bit contrived, this is a programming challenge with which the teachers in Code Camp enjoyed their engagement. There are a lot of different kinds of extensions to this project that are possible as well. For example, the program could be modified to make sure that the strongest students in class are not all in the same group, or ensure that no group consists of students of all the same gender. Furthermore, the project itself encapsulates some very important programming skills: taking input in a list form, iterating over that list to create some output value that conforms to a utility function, and outputting the new value somewhere the user can see.

Once each group created a basic solution to the problem of randomly assigning students to groups, we presented a list of potential extensions to the problem that they could choose to explore. Those extensions were designed to have varying levels of difficulty and appeal to different sorts of problem-solving.

> Add a date to each output sheet so you can remember which is which
> Ask user what group size they want
> Ask user what to do with overflow students: do you want two groups of three or one group of two?
> Change groups to not include certain pairs who might distract each other
> Don’t let students be in the same group as yesterday
> Port the group assigner to the web using our HTML skills
> Class scheduler – Each student has three preferences for a course, each course has a limited number of participants. Pick a grouping that puts each student into a desired course, optimizing for first choice.

For our non-programming task, we introduced computational thinking with a seemingly straightforward problem: create a flowchart that represents the decision-making process you go through to decide what you will wear for the day.

This task addresses a number of the principles of computational thinking, most importantly: simulation (developing a model to imitate real-world processes),

algorithm design (creating an ordered set of instructions for solving similar problems or doing a task), and data representation (organizing data in appropriate forms). It is also easily-approachable by people at many levels of mathematical and scientific expertise, but has a number of surprisingly nuanced outcomes.

If you take a moment to think about what your own flowchart may look like, try answering some of these questions:

> Does your process result in you wearing the same clothes over and over?
> Does your process account for the weather or formal dress codes?
> Does your process account for how you feel about your clothes?
> Does your process result in color coordination between different parts of your clothing?

Decision making, especially around everyday practices, is often the kind of thing that seems simple enough at first thought, but becomes much more complicated the more one thinks about it. Writing computer programs requires the programmer to break everything down as far as possible, taking nothing for granted.

Our Fellows created a number of flowcharts, which each encapsulate different beliefs about what is important when making decisions about clothing.

Both of these projects (the group randomizer and clothing selection flowchart) ended up expanding to become more than just a short activity during the workshop. As **previously noted**, the Code Camp workshop included about two hours each day for independent project work.

During independent work time, some of the Code Camp teachers expanded on the clothing selection flowchart activity to convert it into an in-class activity they could use with their students to introduce computational thinking processes. They built more robust flowcharts for themselves and wrote a lesson for their students.

Other of the Code Camp teachers extended the group-assigner discussed above and went in a completely different direction. What they wanted to build was a random phone contact spreadsheet. They took the same tools (student list spreadsheet and Google scripting) and created a project that would email them (the teacher) one person from their class list that had not been contacted recently as a reminder to do one home-call each evening, even for students who are doing

well. The email would include that student's contact information, parent information, and recent grade information, and would get automatically created at the same time each evening by the spreadsheet. On the surface, this seems quite different from the group randomizer, but from a programming perspective, it is quite similar—take an input list, iterate over it doing stuff to it, and then put the output somewhere the user can access it. In this case, send an email rather than adding a new page to a spreadsheet.

Throughout the Code Camp workshop, we gave a large number of smaller problems for everyone to explore. We then gave different extensions to each problem, so people could decide what kinds of challenges they wanted to give themselves, and we provided less-structured time so that they could focus on specific domains of programming or classroom design. This is why I claim that programming instruction is so amenable to problem-based learning models: we can create scenarios where learners are engaged with problems of their own choosing, at different levels of difficulty and with different pathways to success. In every case, however, people are still engaging with core principles of computer science and computational thinking: decision making, algorithms, utility functions, input-output, randomization, logic, loops, and data representation.