Nifty Assignments

Nick Parlante
Julie Zelenski
(moderators)
Stanford University
nick.parlante@cs.stanford.edu
zelenski@cs.stanford.edu

Baker Franke
Code.org
baker@code.org

Arvind Bhusnurmath,
Karen Her, and Kristen Gee
University of Pennsylvania|
bhusnur4@seas.upenn.edu
karenher@seas.upenn.edu
kgee@seas.upenn.edu

Eric Manley
Timothy Urness
Drake University
eric.manley@drake.edu
timothy.urness@drake.edu

Marvin Zhang, Brian Hou,
and John DeNero
University of California, Berkeley
zhangmarvin@berkeley.edu
brian.hou@berkeley.edu
denero@cs.berkeley.edu

Josh Hug
University of California, Berkeley
hug@cs.berkeley.edu

Kevin Wayne
Princeton University
wayne@cs.princeton.edu

Keywords
Education; assignments; homeworks; examples; repository; library; nifty; pedagogy

Abstract
I suspect that students learn more from our programming assignments than from our much sweated-over lectures, with their slide transitions, clip art, and joke attempts. A great assignment is deliberate about where the student hours go, concentrating the student's attention on material that is interesting and useful. The best assignments solve a problem that is topical and entertaining, providing motivation for the whole stack of work. Unfortunately, creating great programming assignments is both time consuming and error prone.

The Nifty Assignments special session is all about promoting and sharing the ideas and ready-to-use materials of successful assignments.

Each presenter will introduce their assignment, give a quick demo, and describe its niche in the curriculum and its strengths and weaknesses. The presentations (and the descriptions below) merely introduce each assignment. A key part of Nifty Assignments is the mundane but vital role of distributing the materials – handouts, data files, starter code – that make each assignment ready to adopt. The Nifty Assignments home page, http://nifty.stanford.edu, gathers all the assignments and makes them and their support materials freely available. If you have an assignment that works well and would be of interest to the CSE community, please consider applying to present at Nifty Assignments. See the nifty.stanford.edu home page for more information.

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA
ACM 978-1-4503-3685-7/16/03.
http://dx.doi.org/10.1145/2839509.2844678

Greedy Mountain Paths (CS1) - Baker Franke
The basic premise: using a 2D array of elevation data representing a mountainous region of the U.S., find the "best" way to walk through the mountains. Topographic data lends itself nicely to graphical representations. Students write their own algorithms to find paths through the mountains (choosing their own level of complexity) and render them by drawing a colored line through the image. The initial suggestion for a path is to do a "greedy walk" from west to east, choosing a path that represents the least amount of change in elevation from step to step.

This assignment is great because it offers a context in which you can introduce 2D arrays and build quickly from typical array processing tasks to some very interesting and novel algorithms in a way that is approachable to introductory students while offering many different degrees of difficulty to choose from. While it can be a "big" 2D array assignment – covering file I/O, graphics, heuristics and greedy algorithms – there are many options for the instructor to scale it down or use components of the assignment to focus on a particular topic. The assignment is probably best situated in the latter half of a CS1 course, it assumes students have had "typical" experience programming up through linear processing of arrays.

Restaurant Recommendations (CS1) - Brian Hou, Marvin Zhang, and John DeNero
In this assignment, students apply basic machine learning concepts to predict user ratings and cluster restaurants around a university campus, using the Yelp Academic Dataset. The assignment primarily illustrates CS1 concepts such as data abstraction, sequence processing, and key-value pairs. The machine learning ideas are all introduced by the project text as a way of motivating course fundamentals.

The project builds a single interactive browser-based visualization. First, restaurants are clustered by location using the K-means algorithm. Second, ratings for restaurants (visualized using a color scale) are predicted using linear regression (one dimension only; no matrix inversions required). Test cases provided with the project ensure that these pieces combine correctly.

Interested students have a wide array of possible extensions. They can use the Yelp API to make recommendations based on their own Yelp ratings. To improve the accuracy of predictions, students can implement other regression techniques or extract

better features. To improve the visualization, students can dive into web programming by using the Google Maps API to allow for more interactivity (e.g. zooming and dragging) or implementing variants of the main visualization (e.g. a weighted Voronoi diagram).

The niftiest aspect of the assignment is that students are exposed to the idea of machine learning in an intuitive and familiar setting, without math or confusing terminology. They particularly like the real-world application of recommending restaurants by analyzing large amounts of data.

Rack-O (CS1) - Arvind Bhusnurmath, Karen Her and Kristen Gee

The Rack-O assignment is based on a card game which is simple to explain but with enough complexity to make an interesting programming assignment. Rack-O is a card game that involves rearranging your hand of cards in order to have an increasing sequence. The cards are just numbered cards. There are no suits. The programming of this game can be easily done via manipulation of lists. We do the assignment in Python.

The overall goal for a student is to write a program such that a user can play Rack-O against a computer player. For the computer player, students are asked to come up with a strategy and then translate it into code.

What makes the assignment nifty is that students soon realize that strategies in this game are surprisingly easy to program. They spend a fair bit of time coming up with newer and more interesting ones. Some of the students take it upon themselves as a challenge to write a computer player that will beat the grader.

We also bring the physical game to the lab session in which this assignment is introduced. Most students have not seen it before and they have fun playing the physical version before programming it.

Movie Review Sentient Analysis (CS1/CS2) - Eric Manley and Timothy Urness

Sentiment analysis is a text mining task that attempts to figure out the general attitude (e.g. positive/negative attitude) conveyed by a particular piece of text. It can be used for things like analyzing product reviews or determining how the Twittersphere feels about a particular presidential candidate's debate performance.

For the assignment, students write sentiment analysis programs by training them on movie reviews from the Rotten Tomatoes website which have been manually labeled with a score of how positive or negative the review is. Their programs will be able to automatically determine that reviews like “The film was a breath of fresh air” have positive sentiment while reviews like “It made me want to poke out my eyeballs” carry negative sentiment.

The assignment is nifty because it involves real data used with a simple algorithm to achieve compelling results. It does not require any special libraries or software and can be adapted to emphasize a variety of different programming topics. We have used the assignment in CS1 (in both Java and Python) to emphasize file I/O, early control structures, accumulators/counters, and the min/max algorithm. We have used it in CS2 (in C++) to emphasize string manipulation and hash tables. It can also be used to introduce students to the increasingly popular discipline of data science within early programming courses, which was our primary aim in support of our new Data Analytics major. Assignment extensions involving validation, visualization, and algorithm improvement are all possibilities.

HugLife: Testing Creatures (CS1/CS2) - Josh Hug

Nearly every CS instructor has enjoyed watching little creatures running around a two-dimensional grid-based world, spreading across the grid gleefully or racing towards ecological catastrophe.

In HugLife, students program the AI and metabolic properties of competing species, and set them loose to vie for space. Naturally, any errors a student makes will be highly difficult to diagnose by simply running the simulation. Even minor errors can upset the delicate ecological balance of HugLife. Ultimately, the key lesson of this assignment is disciplined use of testing to support a complex system.

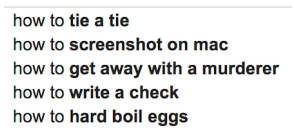
In the first part of the assignment, students write the code for the Plip species, a photosynthetic creature that slowly gains energy and replicates when it has become sufficiently plump. Then, before they're allowed to set their Plips loose on the world, they write a test suite that ensures that the Plips will behave properly. Finally, once the Plip has been tested, students are allowed to deploy them into the universe, which they will promptly fill with green goodness.

Students then repeat the exercise for the Clorus species, a blue carnivore that loves nothing more than consuming poor Plips. With our suggested AI rules and metabolic features, the system evolves into a not-quite stable equilibrium that can be addictive to watch. Students can tweak the existing creatures, or add new creatures of their own.

Most recently, this assignment was run as a two hour lab in Java that introduced testing of large systems. However, it can also be used as an introduction to inheritance and object oriented programming. Implementations available in Java and Python.

Autocomplete Me (CS2) - Kevin Wayne

Write a program to implement *autocomplete* for a given set of strings and weights. That is, given a query string, find all strings in the set that start with the query string, sorted in descending order of weight. Autocomplete is an important feature of many modern applications. For example, Google uses autocomplete to display suggestions in real time, as the user types a web search query.



In one version of the assignment, students implement autocomplete by combining sorting with binary search. The built-in sort suffices for sorting the terms either by query or weight. However, the variant of binary search required is nonstandard (and not available in Java), so students must design and implement their own version. To enable the binary search, students must also implement an interesting custom order for the terms (comparing two strings lexicographically, but using only their first *k* characters). The assignment is nifty because

* It is a familiar and authentic application, for which performance is critical.
* It brings sorting and binary search—often dull and poorly motivated topics—to life.
* The accompanying interactive GUI and real-world datasets make it fun for students.
* It does not require any scaffolding code.