Modern Course Design and CS Materials

Kalpathi Subramanian, Erik Saule
krs@uncc.edu, esaule@uncc.edu

The University of North Carolina at Charlotte

BRIDGES Summer Workshop 2022

Table of Contents

High Level Picture

Structured in Module

Content is grouped themes

Dependencies between modules

- Catch up modules
- Mandatory Modules
- Optional Modules

Goals

- Topics
- Course Learning Outcomes
- Program Learning Outcomes
- Competencies
- Pedagogical Strategies

A Variety of Content

- Lectures
- Videos
- In-class activities
- Assignments
- Exam
- Projects

Accreditation

- SACS
- ABET
- Quality Matters
- *insert your accreditation body here*

What is Alignment?

Properties of how content flow in

- Program
- Course
- Module
- Materials

That could apply to

- Topics
- Outcomes
- Competencies

That could be in term of

- What they cover
- What they assume students know

Aligning Modules with Course Objectives

Courses usually have objectives that come from program descriptions and assessments.
How do we ensure that the content of the class actually serve these higher objective? We want to align the objective modules with the objective of the course.
Two main properties to check:

- Are all the course objectives covered appropriately by a module objective?
- Are there module objectives that serve no course objective?

Alignment within Module

Typical module structure

- Exposition to new concept (lecture, textbook)
- Clarification of concept (discussion, hands-on activity)
- Reinforcement of concept (problem, programming assignment)

Properties you want

- The clarification should not introduce new concepts
- The reinforcement should strengthen the exposition and clarification topics
- The materials should cover the topics the module is meant to cover
- The materials should not wander too far from the module objectives

Assessment

Exam should never introduced new concepts

Plan for ITCS 6114: Algorithms and Data Structures

Overreaching Learning Outcomes

OLO1. Articulate that design, complexity, and correctness of algorithms and data structures matter in the real world

OLO2. Design correct and low complexity algorithms and data structures by employing standard techniques

OLO3. Analyze and implement given algorithms and data structures

OLO4. Recognize faulty algorithmic logic

Detailed Learning Outcomes

On Complexity

DLOC1. Interpret complexity notation and their implications on the performance/resource consumption of algorithms (OLO1, OLO2)

DLOC2. Articulate the real-world implication of the design of algorithms and data structures in term of performance (OLO1)

DLOC3. Derive the complexity of algorithms using various techniques (for instance, master theorem, amortized analysis, and average case analysis) (OLO1, OLO2, OLO3, OLO4)

DLOC4. Prove the NP-Completeness of classic problems (OLO2, OLO4)

DLOC5. Leverage the P != NP conjecture to recognize dubious algorithmic claims (OLO4)

On Correctness

DLOCo1. Recognize and prove the invariant of data structures and algorithms (OLO2, OLO4)

On Data Structures

DLOD1. Design, analyze, and implement tree-based indexes (OLO2, OLO3)

DLOD2. Design, analyze, and implement hash-based indexes (OLO2, OLO3)

DLOD3. Design, analyze, and implement classic algorithms on graphs (OLO2, OLO3)

On Algorithmic Techniques

DLOA1. Create, analyze, and implement divide and conquer algorithms (OLO2, OLO3)

DLOA2. Create, analyze, and implement greedy algorithms (OLO2, OLO3)

DLOA3. Create, analyze, and implement dynamic programming algorithms (OLO2, OLO3)

Week 1. Sep 8.

Lecture:

1. Introduction.
2. Complexity notations [DLOC1].

Activity:

1. Proving simple complexity notation properties [DLOC1].
2. Interpreting complexity notation in term of practical cost or feasibility [DLOC1, DLOC2].

Week 2. Sep 15.

Lecture:

1. Analyzing simple algorithms [DLOC3].
2. Invariant and correctness [DLOCo1].
3. Simple recursive complexity formulas [DLOC3].

Activity:

1. Given simple algorithms (binary search, insertion sort, simple nearest neighboor), prove their correctness and complexity [DLOCo1, DLOC3].
2. Implement and benchmark insertion sort and simple nearest neighboor [DLOC1, DLOC2].

Week 3. Sep 22.

Lecture:

1. Divide and Conquer [DLOA1].
2. Merge sort [DLOA1, DLOCo1].
3. Master Theorem [DLOC3].

Activity:

1. Solve some other problem using D&C [DLOA1].
2. Implement and benchmark Merge Sort [DLOC2].

Week 4. Sep 29.

Lecture:

1. Tree-based indexing [DLOD1].
2. Invariant of data structure [DLOCo1].
3. Using BST for associative array [DLOD1, DLOC2].

Activity:

1. Run BST manually on toy example [DLOD1].
2. Design, analyze, and implement a tree base index for nearest neighbor query [DLOD1, DLOC2].

Week 5. Oct 6.

Lecture:

Curriculum Guidelines

What are they?

Usually they are recommendation of what should/could be taught across a program.
Expressed in term of topics, learning outcome, and competencies. Not in term of courses.
Usually make recommendation on how much one should learn in a particular topic, sometimes specified in number of hours.

How can we use them?

Give us a reference of what we should/could be teaching.
Am I covering all that? Should I? Why not?
Give us a common language to communicate between instructors.

General Guidelines: ACM/IEEE CS 2013

Structured in

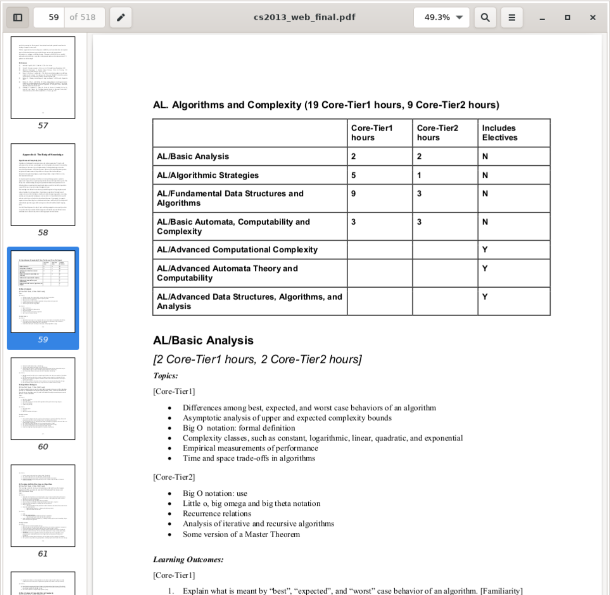
- Knowledge Area
- Knowledge Unit

Topics and Learning Outcomes are classified as

- Tier-1
- Tier-2
- Elective

Other general guidelines:

- Data Science
- Computer Engineering
- Upcoming revised CS



AL. Algorithms and Complexity (19 Core-Tier1 hours, 9 Core-Tier2 hours)

	Core-Tier1 hours	Core-Tier2 hours	Includes Electives
AL/Basic Analysis	2	2	N
AL/Algorithmic Strategies	5	1	N
AL/Fundamental Data Structures and Algorithms	9	3	N
AL/Basic Automata, Computability and Complexity	3	3	N
AL/Advanced Computational Complexity			Y
AL/Advanced Automata Theory and Computability			Y
AL/Advanced Data Structures, Algorithms, and Analysis			Y

AL/Basic Analysis

[2 Core-Tier1 hours, 2 Core-Tier2 hours]

Topics:

[Core-Tier1]

- Differences among best, expected, and worst case behaviors of an algorithm
- Asymptotic analysis of upper and expected complexity bounds
- Big O notation: formal definition
- Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential
- Empirical measurements of performance
- Time and space trade-offs in algorithms

[Core-Tier2]

- Big O notation: use
- Little o, big omega and big theta notation
- Recurrence relations
- Analysis of iterative and recursive algorithms
- Some version of a Master Theorem

Learning Outcomes:

[Core-Tier1]

1. Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm. [Familiarity]

Specific Guidelines: NSF/IEEE-TCPP PDC 2012

Structured in domains:

- Programming
- Algorithm
- Architecture

More descriptive.
Bloom levels.
Other specific guidelines: graphics, security

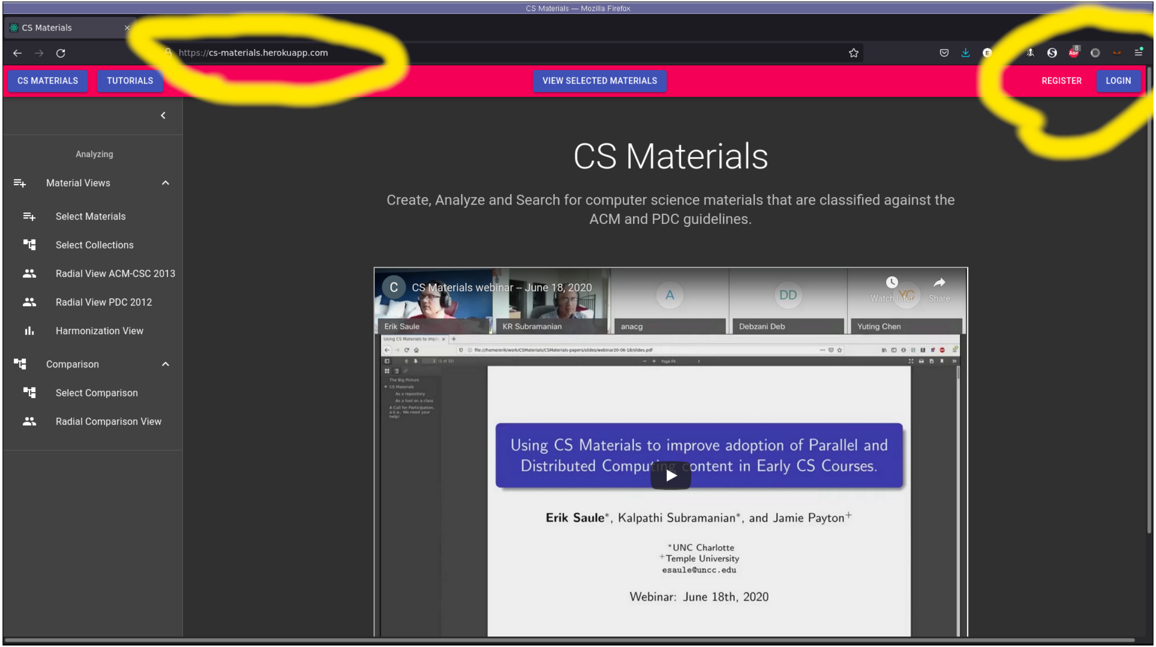
A Lingua Franca

CS Guidelines give us a fairly detailed description of what is in CS.
We can use them as ontologies to describe in a common language what a course of a class material is like.

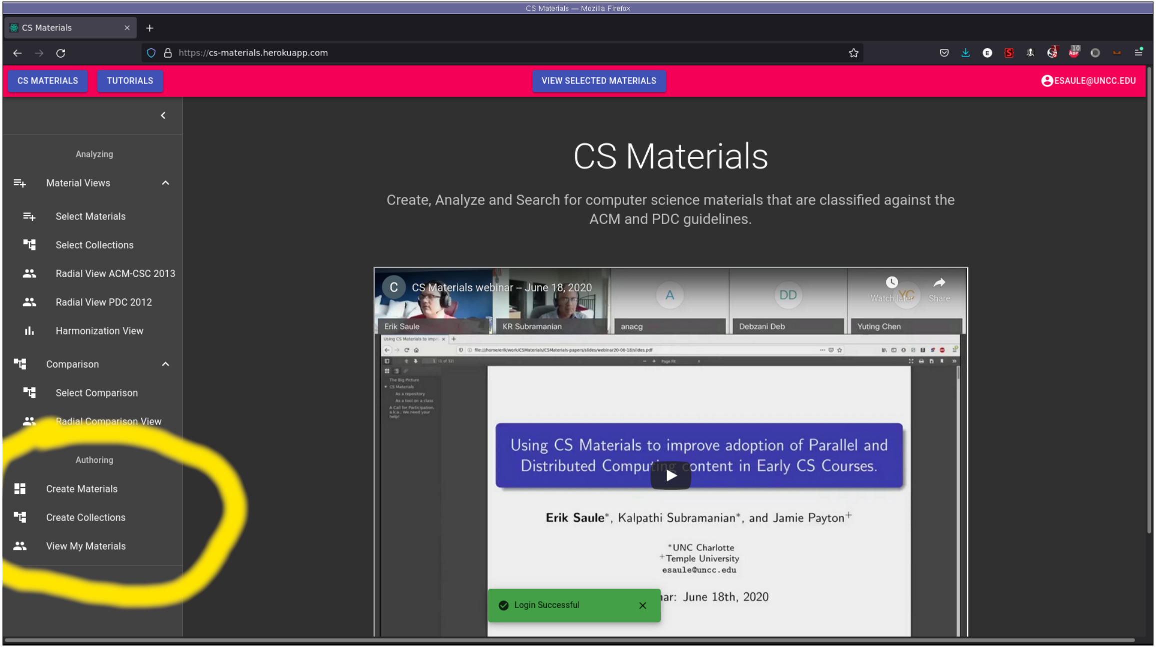
What do you think is in a lecture entitled UNCC-ITCS-2214-Saule-Graphs?

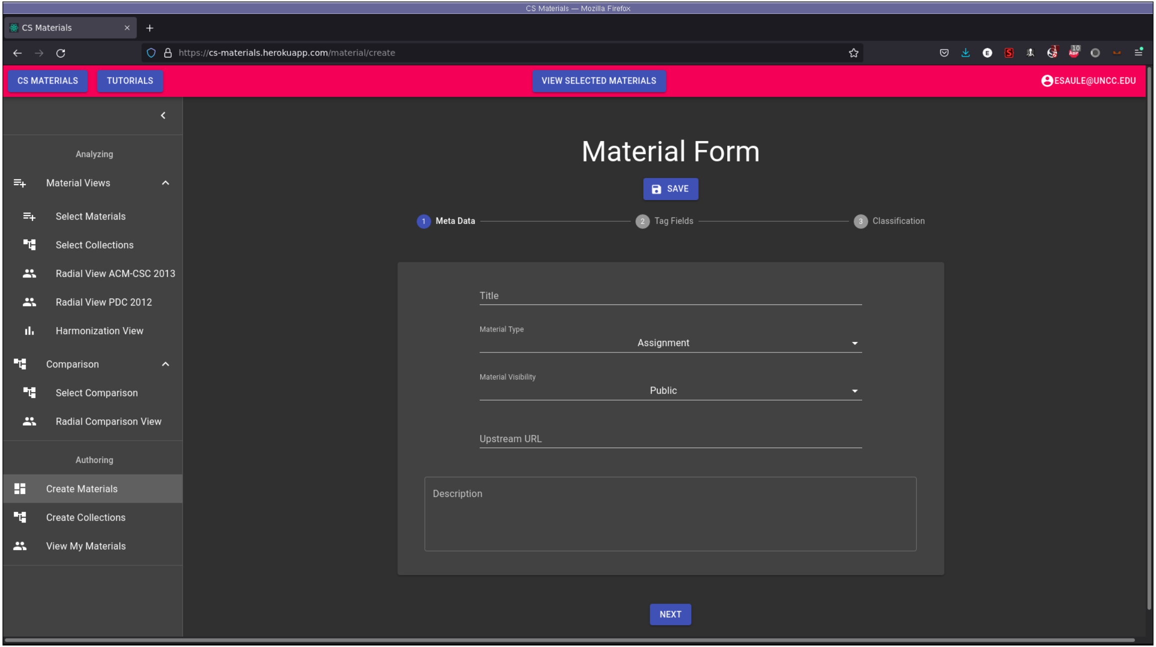
- Depth- and breadth-first traversals
- Representations of graphs (e.g., adjacency list, adjacency matrix)
- Reflexivity, symmetry, transitivity
- Illustrate by example the basic terminology of graph theory, and some of the properties and special cases of each type of graph/tree.
- Undirected graphs
- Directed graphs
- Weighted graphs
- Iterative and recursive traversal of data structures

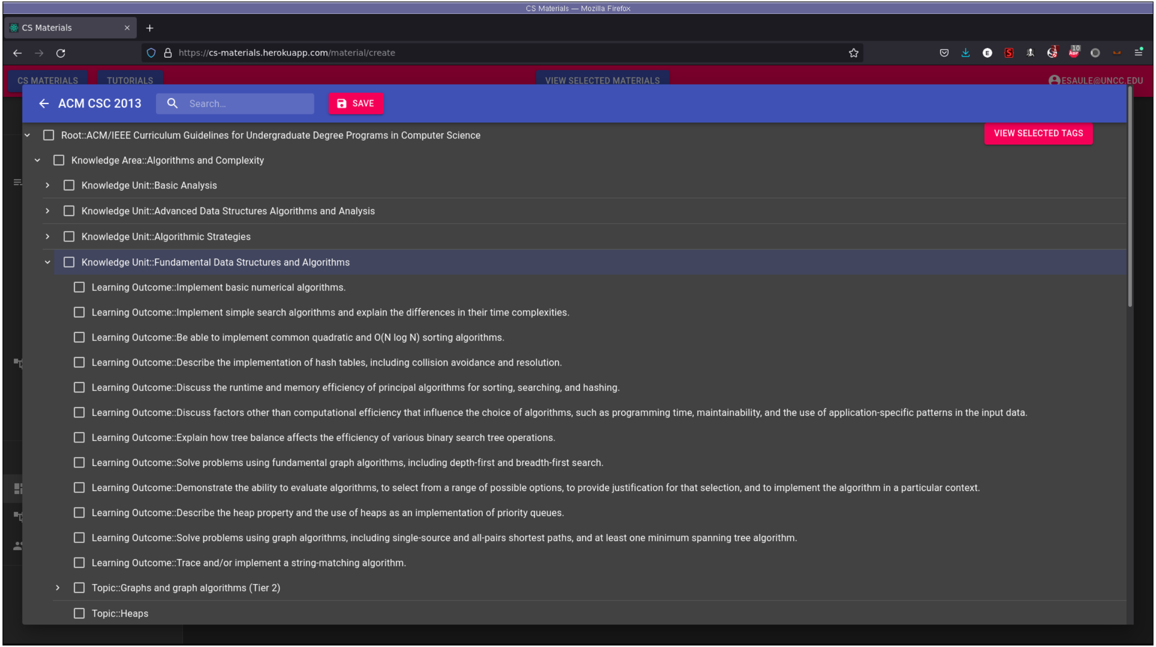
Table of Contents



CS Materials — Mozilla Firefox
CS Materials
https://cs-materials.herokuapp.com
CS MATERIALS
TUTORIALS
VIEW SELECTED MATERIALS
REGISTER
LOGIN
Analyzing
Material Views
Select Materials
Select Collections
Radial View ACM-CSC 2013
Radial View PDC 2012
Harmonization View
Comparison
Select Comparison
Radial Comparison View
CS Materials
Create, Analyze and Search for computer science materials that are classified against the ACM and PDC guidelines.
CS Materials webinar – June 18, 2020
Watch later
Share
Erik Saule
KR Subramanian
anacg
Debzani Deb
Yuting Chen
Using CS Materials to improve adoption of Parallel and Distributed Computing content in Early CS Courses.
Erik Saule*, Kalpathi Subramanian*, and Jamie Payton+
*UNC Charlotte
+Temple University
esaule@uncc.edu
Webinar: June 18th, 2020


CS Materials — Mozilla Firefox
CS Materials
https://cs-materials.herokuapp.com
CS MATERIALS
TUTORIALS
VIEW SELECTED MATERIALS
ESAULE@UNCC.EDU
Analyzing
Material Views
Select Materials
Select Collections
Radial View ACM-CSC 2013
Radial View PDC 2012
Harmonization View
Comparison
Select Comparison
Radial Comparison View
Authoring
Create Materials
Create Collections
View My Materials
CS Materials
Create, Analyze and Search for computer science materials that are classified against the ACM and PDC guidelines.
CS Materials webinar – June 18, 2020
Watch later
Share
Erik Saule
KR Subramanian
anacg
Debzani Deb
Yuting Chen
Using CS Materials to improve adoption of Parallel and Distributed Computing content in Early CS Courses.
Erik Saule*, Kalpathi Subramanian*, and Jamie Payton+
*UNC Charlotte
+Temple University
esaule@uncc.edu
June 18th, 2020
Login Successful


CS Materials — Mozilla Firefox
CS Materials
https://cs-materials.herokuapp.com/material/create
CS MATERIALS
TUTORIALS
VIEW SELECTED MATERIALS
ESAULE@UNCC.EDU
Analyzing
Material Views
Select Materials
Select Collections
Radial View ACM-CSC 2013
Radial View PDC 2012
Harmonization View
Comparison
Select Comparison
Radial Comparison View
Authoring
Create Materials
Create Collections
View My Materials
Material Form
SAVE
1 Meta Data
2 Tag Fields
3 Classification
Title
Material Type
Assignment
Material Visibility
Public
Upstream URL
Description
NEXT


CS Materials — Mozilla Firefox
CS Materials
https://cs-materials.herokuapp.com/material/create
CS MATERIALS
TUTORIALS
VIEW SELECTED MATERIALS
ESAULE@UNCC.EDU
ACM CSC 2013
Search...
SAVE
VIEW SELECTED TAGS
Root::ACM/IEEE Curriculum Guidelines for Undergraduate Degree Programs in Computer Science
Knowledge Area::Algorithms and Complexity
Knowledge Unit::Basic Analysis
Knowledge Unit::Advanced Data Structures Algorithms and Analysis
Knowledge Unit::Algorithmic Strategies
Knowledge Unit::Fundamental Data Structures and Algorithms
Learning Outcome::Implement basic numerical algorithms.
Learning Outcome::Implement simple search algorithms and explain the differences in their time complexities.
Learning Outcome::Be able to implement common quadratic and O(N log N) sorting algorithms.
Learning Outcome::Describe the implementation of hash tables, including collision avoidance and resolution.
Learning Outcome::Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing.
Learning Outcome::Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data.
Learning Outcome::Explain how tree balance affects the efficiency of various binary search tree operations.
Learning Outcome::Solve problems using fundamental graph algorithms, including depth-first and breadth-first search.
Learning Outcome::Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context.
Learning Outcome::Describe the heap property and the use of heaps as an implementation of priority queues.
Learning Outcome::Solve problems using graph algorithms, including single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm.
Learning Outcome::Trace and/or implement a string-matching algorithm.
Topic::Graphs and graph algorithms (Tier 2)
Topic::Heaps

Study of Coverage

We can easily understand what one course is covering.
We can understand across multiple offfering of the same course what that particular course is about.
We can identify different “flavors” of that course.

Have you ever searched for materials?

Let's look at Nifty Assignments

Nifty Assignments

The Nifty Assignments session at the annual SIGCSE meeting is all about gathering and distributing great assignment ideas and their materials. For each assignment, the web pages linked below describe the assignment and provides materials -- handouts, starter code, and so on.

Applying for Nifty is now done as its own track with a similar deadline to special sessions. The format and content of the .zip you submit is unchanged. See the info page for ideas about what makes a nifty assignment and how to apply for the Nifty session.

Please email any suggestions or comments to the nifty-admin email: nifty-admin@cs.stanford.edu
Nick's Home

Nifty Assignments 2021

Sankey Diagrams - Ben Stephenson	CS1 Sankey diagram - neat data visualization algorithm
Rocket Landing Simulator - Adrian A. de Freitas and Troy Weingart	CS1 Rocket Landing Simulator - fun algorithm
Covid Simulator - Steve Bitner	CS1-CS2 Covid 2D infection simulator - timely if scary
Linked List Labyrinth - Keith Schwarz	CS2 Neat memory / debugger skill exercise, custom per student

Nifty Assignments 2020
Thanks to our presenters for getting everything together including videos for this COVID-interrupted year.

Typing Test - John DeNero et al	CS1 Fill in algorithm of fun typing-speed test. **(Video)** (intentionally silent)
Color My World - Carl Albing	CS1 or later: Students are given a data file, but no description about what it represents. Can they solve the mystery by generating a reasonable image?
Bar Chart Racer - Kevin Wayne	CS1 - use real data to make a animated bar chart - captivating! **(Video)**
DNA - Brian Yu, David J. Malan	CS1 or CS2 Neat DNA project. **(Video)**
Recursion to the Rescue - Keith Schwarz	Nifty recursion projects using tied to real-world applications. **(Video)**
Decision Makers - Evan Peck	Two hour exercise illuminating algorithms and life

Nifty Assignments 2019

Nifty Post It - Jeffrey L. Popyack	CS0-CS1 Hands On Manipulative
Hawaiin Phonetic Generator - Kendall Bingham	CS1 Fun Text
Motion Parallax - Ben Dickson	CS1 Awesome Graphic Experience

Metadata

Summary	Students develop a program to map raw data files into a colorful images.
Topics	visualization, big data, image processing - color maps.
Audience	Use as an early assignment in an HPC class, Scientific Programming class, Data Science/Analysis class, or a Graphics/Image processing class. Appropriate for CS1 or higher students familar with loops, file io, argument parsing, and image processing. The starter code is written in Python.
Difficulty	This assignment is appropriate for various levels, depending on the initial conditions: starter code (or not), existing color maps (or not) and time alloted. A late-semester CS1 class given the starter code and a week.

Strengths	• Solving the mystery of what the image "looks" like
	• Working with *real-world* data to get visual, graphical feedback.
	• Allows for some artistic flair resulting in variations among solutions
	• Depending on the assignment write up there are open ended options including:
	  ○ creating different colormaps for different images;
	  ○ scaling the data to fit a given image size;
	  ○ a "smarter" program to deduce the image size from the data file;
	  ○ statistical analysis of the data to drive the choice of color map values

Weaknesses	• When creating a colormap from scratch it can be tricky to get color assignments that are both visually pleasing (artistic) and pull out the desired details, though that is part of the point of this assignment. • Use of graphics makes unit testing more challenging.
Dependencies	

Curriculum Guidelines as Features

Features

The problem in classic search is that it is hard to find good matches because people use imprecise textual descriptions.
Curriculum guidelines give us a well established precise features

Search

Give a set of materials that use these topics/outcomes

Recommendation

Give a set of materials that match the same outcomes as these ones.

Table of Contents

Activities

Upload your materials in google drive

- Structure them in modules

Classify Module 1 in CS Materials

- Create a CS Materials account
- Create a material for your first module
  - Title it with your name, your class name, and the module number and name
  - Classify against ACM CS 2013
  - (You may want to save often)