Evaluating Active Learning Approaches for Teaching Intermediate Programming at an Early Undergraduate Level

Dhruva K. Chakravorty
High Performance Research Computing (HPRC) at Texas A&M University
College Station, Texas, USA

Marinus "Maikel" Pennings
HPRC at Texas A&M University
College Station, Texas, USA

Honggao Liu
HPRC at Texas A&M University
College Station, Texas, USA

Zengyu "Sheldon" Wei
HPRC at Texas A&M University
College Station, Texas, USA

Dylan M. Rodriguez
HPRC at Texas A&M University
College Station, Texas, USA

Levi T. Jordan
HPRC at Texas A&M University
College Station, Texas, USA

Donald "Rick" McMullen
HPRC at Texas A&M University
College Station, Texas, USA

Noushin Ghaffari
HPRC at Texas A&M University
College Station, Texas, USA

Shaina D. Le
HPRC at Texas A&M University
College Station, Texas, USA

Derek Rodriguez
HPRC at Texas A&M University
College Station, Texas, USA

Crystal Buchanan
HPRC at Texas A&M University
College Station, Texas, USA

Nathan Gober
HPRC at Texas A&M University
College Station, Texas, USA

ABSTRACT

There is a growing need to provide intermediate programming classes to STEM students early in their undergraduate careers. These efforts face significant challenges due to the varied computing skill-sets of learners, requirements of degree programs, and the absence of a common programming standard. Instructional scaffolding and active learning methods that use Python offer avenues to support students with varied learning needs. Here, we report on quantitative and qualitative outcomes from three distinct models of programming education that (i) connect coding to hands-on "maker" activities; (ii) incremental learning of computational thinking elements through guided exercises that use Jupyter Notebooks; and (iii) problem-based learning with step-wise code fragments leading to algorithmic implementation. Performance in class activities, capstone projects, in-person interviews, and participant surveys informed us about the effectiveness of these approaches on student learning. We find that students with previous coding experience were able to rely on broader skills and grasp concepts faster than students who recently attended an introductory programming session. We find that, while makerspace activities were engaging and explained basic programming concepts, they lost their appeal in complex programming scenarios. Students grasped coding concepts fastest using the Jupyter notebooks, while the problem-based learning approach was best at having students understand the core problem and create inventive means to address them.

DOI: https://doi.org/10.22369/issn.2153-4136/10/1/10

CCS CONCEPTS

• Social and professional topics → Computing education programs; Computing literacy; Computational thinking; Informal education; K-12 education; • Applied computing → Interactive learning environments;

Keywords

Undergraduate education, informal education, computing literacy, Python, Jupyter notebooks, active learning, problem-based learning, scaffolding strategies, Raspberry Pi

1. INTRODUCTION

Offering incoming (freshman) undergraduates opportunities to learn basic computing skills is an idea that is rapidly gaining popularity in higher education. Due to the rapid proliferation of computing in science, technology, engineering, and mathematics (STEM) disciplines, these efforts need to be supplemented with intermediate level student training programs as well [6, 7]. Such intermediate programming experiences need to provide a solid skill foundation in order to help early undergraduates develop complexity in programming skills. Proficiency in computer science widely differs from student to student, depending on their previous experiences with computing. In this paper, we investigate how important previous exposure to computing concepts is to computing education. While a wide range of programming constructs are considered introductory activities, a defined standard

for intermediate- level proficiency is currently lacking. The Python programming language helps address some of these concerns. The language has an easy-to-comprehend syntax and can grow in complexity to support analysis in numerous STEM disciplines. In addition, inexpensive computers, like the Raspberry Pi, open possibilities for educators to couple makerspace activities with Python programming in the classroom to serve as a scalable platform on which students can develop an intermediate-level skill set in computing. Informal efforts that use well-reviewed pedagogical approaches to education have been found to encourage participation in and adoption of computational thinking [3, 13, 23, 24]. Previous studies have shown that projects connected to greater societal impact or include elements of physical creativity are more likely to appeal to a broader audience [14, 16]. Using participatory technologies such as visualization, modeling, and robotics provide a number of opportunities in this space. By combining increasingly easy-to-access computing resources with a scaffolded teaching approach [10], the barrier to entry can be reduced [4, 20]. Reducing this barrier to entry is particularly relevant for informal efforts hosted by high performance computing (HPC) centers that support users who have a diverse range of research needs and computing prowess.

In this study, we explore different pedagogical approaches that utilize these technologies to introduce learners to complex programming scenarios suitable for the intermediate level [2, 17, 19]. In the subsequent sections of this paper, we present the conceptual framework of our program, and describe the methods used to evaluate three learning approaches. The paper next describes our efforts toward assessing the success and pitfalls of these approaches using capstone exercises, interviews and evaluations. We finally discuss the lessons learned over the previous year and summarize our findings in conclusion.

2. EXPERIMENTAL DESIGN

Instructing groups of students on coding practices who have a diverse range of skill sets and educational backgrounds remains a challenge in computing education. Scaffolded instruction methods offer avenues to support students with varied skill sets [10]. Active learning has been shown among high-ability trainees to produce significantly higher levels of metacognitive activity than procedural training. Here, we compare and contrast the benefits of well-reviewed approaches to scaffolded instruction and innovative active learning exercises in the context of Python programming over a week-long training session. The program's goals are to (a) increase participant engagement (b) develop a participant's understanding of complex computing concepts, and (c) pro- vide participants with a learning environment that employs hands-on exercises.

Complex coding projects can overwhelm the new or intermediate learner. Such learning is best facilitated in a tiered format where information is provided, comprehended, analyzed and employed before moving to the next step. Traditional approaches utilize handouts for students, presenting the code on the screen, and perhaps provide prepared versions of the code. Jupyter Notebooks provide interesting avenues in this space, because they provide a number of useful features [8]. Unlike traditional applications, these notebooks run as interactive web-browser applications that allows users to write Python code in cells. The output from the cells is easy to access and visualize because of its closeness to the Python code. Students can see the output from each portion of code as they are writing it. This allows the instructor to keep students engaged through exercises that focus on smaller pieces of a complex code and demonstrate how the output from each cell combines to form the larger program structure. The last advantage of Jupyter Notebooks is that, as a web-based effort, these notebooks are platform agnostic and can be run on any computer! This attribute makes them tremendously portable. A notebook that runs on a Raspberry Pi can be ported to run on a supercomputing cluster with graphical processing units [15].

There is a significant body of work describing the importance of makerspace activities and problem-solving approaches in informal education [22]. Specifically, here we explore three different approaches that use Raspberry Pi microcontrollers. We specifically report on the following qualitative and quantitative outcomes, (1) connecting coding to sensing and control of the real world through hands-on maker activities [14, 16]; (2) incremental learning of computational thinking elements [3, 13, 17, 23, 24] through guided exercises using Jupyter Notebooks [8]; and (3) problem-based learning with step-wise code fragments leading to a complete implementation of an algorithm in which students are presented a "narrative" program and goal, and work through converting specific objectives into code to write the program and achieve the goal.

To help ensure student engagement, these approaches used exercises in game design using the Raspberry Pi Hardware Attached on Top (HAT) sensor platform, image recognition using machine learning (ML) [11], and sharing secret messages using cryptography respectively. All three approaches included structured and unstructured components, handouts for students, and a number of advisors (1:4 ratio) available to assist if needed. In order to effectively judge the efficacy of each approach, we ensured that the exercises in these approaches did not build on each other. To avoid biases due to familiarity with an instructor, each approach was taught on a separate day by a different instructor (2 males and 1 female). Participants were informed of the nature of activities and approaches only on the day of the activity. This ensured that the participants were exposed to these learning activities for the first time on the day of the camp, and helped reduce artifacts arising from the participants having previous knowledge of these activities.

In order to create an early undergraduate environment that comprises of intermediate-level learners, we recruited a cohort of 23 participants who had recently graduated from high school or would do so in the near future. 7 of these 23 students were female. In all, 13 of these 23 students be- longed to groups that are traditionally underrepresented in computing. Recruiting was performed using social media, emails to listservers, our website and contacts at local school districts. Participant applications were managed via our website. As part of their applications, applicants were asked to completed a pre-training evaluation in the form of Likert Scale and open-ended questions. These questions helped establish the participant's familiarity with Python, programming, Linux and Raspberry Pi computers. Thanks to the exciting nature of offered projects, we received a number of applications from well-qualified participants. Selection to the training program was merit-based. All participants had basic Python programming skills defined by the ability to write scripts that employed loop constructs, had earned a GPA of above 3.5, and were interested in attending college. The participants also had some experience with the Linux operating system and text editors. The students belonged to two distinct learning groups. The first group of 12 students were self-identified intermediate-level Python learners. The second group included 11 learners who had participated in our introductory Python programming course 2 months prior to this exercise. For the purpose of brevity, these cohorts of participants are henceforth referred to as Group 1 and Group 2, respectively, in the remainder of this manuscript.

3. PEDAGOGICAL APPROACHES

Each participant received a Raspberry Pi that was preloaded with the Raspbian operating system, a Debian-based distribution of Linux. To ensure that all participants had a set of common Python programming skills, they were provided with the training material from our introductory programming camp. These materials contained information about introductory Python programming practices, the Linux operating system, GitHub, and the Gedit text editor. Trainees who had previously participated in our introductory camp program were taught using this material. Introductory computing skills were reviewed on the first day of the camp. Each slide in the lesson contained a small activity to allow for immediate application of the respective topic. To synthesize knowledge gained in the lesson, the students were tasked with their first maker activity; they performed variable assignments, basic arithmetic operations, and console output. The students created both string and integer variables to store numerical values. The purpose of these tasks was to teach students to explore through individual trial and error what actions could be executed on which data type. On completion of the first day, all students were able to successfully demonstrate the use of algorithms and loops in a review exercise. Surveys and in-person interviews further supported that all students were at a similar learning level after the first day's reviews. A brief description of the activities during the training exercises are provided in the Reproducibility Appendix in the order in which they were taught. Owing to the complexity of the topics covered, measures were taken to reduce the complexity of the problem set. Sessions included the use of PowerPoint slides for instruction and students received handouts containing pertinent information and review exercises. Instructors and their assistants were provided with handouts that included more details about the exercises along with possible solution sets.

3.1 Maker Spaces With Raspberry Pi HATs

Raspberry Pi computers have been successfully coupled with sensors to create a wide range of makerspace activities. While a wide variety of inexpensive discrete sensors and actuators, such as sound-buzzers, LEDs, and touch sensors are available, the integrated Raspberry Pi HAT sensor platform was used for these activities. The Pi HAT combines discrete components nicely into a portable and usable package. This platform provided participants with numerous common utilities including an LED display. The maker activities in this section had the trainees explore cybersecurity and gaming scenarios that used the Pi HAT to present a visual output on the LED display. The lesson began with an introduction to basic programming concepts, such as variable nomenclature, basic operators, and basic data types. The intention was to develop an intuition for translating common language commands into Python. The next segment began with an introduction to the Python list, list operations, comparison operators, user input, classes, and scripts. Students employed Git repositories to exercise version control during these activities. It followed the structure of the previous segment, with miniature activities on each slide and a maker activity recursively consolidating the segment information. Students were provided with a slide showing parts of the program. New concepts were highlighted as they were discussed during the session. The lesson continued as such, with each segment scaling up in difficulty and building on previously-learned concepts, as per a scaffolded instruction approach.

The students completed two key activities during this session. In the first exercise, they sequentially controlled the light pat- tern on an RGB LED. For the second exercise, the students created a Pi-stacking game on the Pi HAT that required players to stack colored tiles on top of each other. Student's engagement with maker activities throughout the lesson pro- vided opportunities for the instructor to manage the varied skill levels with extra focus given when needed. Successful completion of the coding exercises was used as a metric of student success.

3.2 Jupyter Notebooks in Machine Learning

During this session, we described a series of hands-on activities that introduced the participants to aspects of machine learning on the widely-used Keras and TensorFlow software [1, 5]. Participants were provided with an install script that automated most of the process to reduce the complications arising from having to install these software and their associated libraries. Students were first introduced to concepts in machine learning. They were taught how machine learning uses labels to categorize the subjects in images and how it predicts the subject in an image file based on what it learns from a training data set. Topics covered during this session included the need for training using established and respected data sets, emphasizing the need for higher levels of accuracy in terms of training and complexity of models, and finally envisioning scenarios where machine learning will make incorrect predictions. The introductory session was followed by interactive hands-on activities that introduced these learners to various aspects of ML. These included (i) using a training database to teach ML systems to recognize hand-drawn numbers in an image, (ii) accurately predicting the kind of flower seen in an image, (iii) improving the predictive ability of exercise by using convoluted neural networks, and (iv) identifying the objects in a given image downloaded from the internet [9]. As students worked through these exercises, they employed the popular MNIST and ImageNet data sets to explore logical regression models, transfer learning, Python imaging libraries and the scikit learning libraries [12, 18]. Hand-drawn images for these activities were created using the GIMP software. On completion of these activities, students were able to leverage existing modules to solve a real-world machine learning problem with a small training data set and computing constraints. Once again, student success was evaluated based on their ability to complete the session's hands-on activities. As a capstone, each student created two trained platforms. The first platform identified hand-drawn numbers in the range of 0 to 9, and the second platform classified images. Since Jupyter Notebooks were used, additional, more periodic metrics could be used to evaluate student performance. These metrics include qualitative evaluations of the student's use of programming concepts to manipulate training data sets and images. In our exercises, students came up with different ways to confuse the algorithm. One case was of particular interest: a trainee from Group 2 drew upon concepts from programming sessions to create an image distorted with static to understand how the ML programs would react to this data.

3.3 Real World Problem Solving Exercises in Cryptography

During the cryptography session, the instructor first presented the importance of cryptography in computer security. During this session, the emphasis was placed on discussing a relevant real-world problem and identifying approaches to solve it. Unlike the previous activities, these problems could be solved by implementing a number of different approaches. Much time was spent discussing the fundamentals of modern cryptography: modular arithmetic and one-way functions. Computationally difficult functions such as discrete logarithms and the prime factorization problem are not typically covered in introductory courses, so these concepts are worth spending more time on to ensure the students have a strong understanding of the underlying

complexity of cryptography. Additionally, the session on cryptography provides an opportunity to talk about how brute force solutions to problems are often computation- ally infeasible and require the problem set be compressed, a common issue in high performance computing. To apply the material, the instructor presented the Diffie-Hellman Key Exchange algorithm, an accessible example of cryptographic concepts. As part of evaluating the effectiveness of this approach, three exercises were completed during this activity. Each progressive exercise would have more possible solutions. The trainees perform an encryption exercise followed by a decryption exercise.

The session was followed by a capstone exercise where students were challenged to decrypt the communications of others in the class. To demonstrate how it is critical to encrypt secure communications over non-secure lines, all messages, including key exchange negotiations, were broadcasted to the entire room. The students were given examples of the Diffie-Hellman algorithm implemented in Python. The intended recipient should be able to successfully decrypt the message, while unintended recipients will find it computation- ally infeasible. After this exercise, the students were able to describe the need for cryptography and apply their knowledge of modular arithmetic to computing problems.

4. EVALUATIONS, SURVEYS AND ASSESSMENTS

The program and the activities in the program were evaluated using a variety of approaches. Data collected as part of the registration process helped identify the participant's base line skills. As described in previous sections, development of competencies is gauged by the student's ability to complete in-class exercises and activities. To test competency on a topic, students were asked to complete in-class exercises. As described in the previous section, each of the three Python sessions included capstone projects that required the students to apply the knowledge gained from the session. Finally, surveys were completed by students at the end of the camp. These surveys asked questions about the activities taught during the camp. The survey consisted of questions that asked students to rate activities based on a Likert scale. The survey also included a number of open- ended questions that gave participants an opportunity to include additional details about the exercises. To compensate for the limited size of the sample group, surveys were augmented with data from in-person interviews and student performance in activities. Some of the open-ended questions in our post-program survey were:

1. How did you learn about this camp?
2. Why did you register for this camp?
3. What are the difficulties that you faced in this camp?
4. What do you think are the strengths of this camp?
5. What specific content/concepts in the camp were particularly challenging for you?
6. What specific content/concepts in the camp were particularly easy for you?
7. How do you plan on using what you learn in this camp?
8. Who would you recommend the camp to?

Essay-style questions that allowed campers to explain their perceptions of the camp sessions were:

Please add any additional comments and details about topics that you felt were easy or difficult to learn. What was the most enjoyable aspect of a given session? What else you would like to have learned during these sessions?

(a) Linux review session

(b) Python review session

(c) Coding Games

(d) Artificial Intelligence activities

(e) Secret sharing activities

(f) Virtual Reality activities

The Likert Scale (1-5 scale) questions were:

1. How easy was the course for you?
2. How satisfied are you with the camp?
3. Please rate your proficiency in Python before attending the Intermediate camp.
4. Please rate your proficiency in Python after the Intermediate camp.
5. How likely are you to recommend the camp to others?
6. Please rate how likely you are to tell your teachers about this camp.
7. Please rate how likely you are to participate in conferences, science fairs, or other STEM programs in the coming school year?

These surveys provide us a rationale for why people are attending our classes and help with the recruiting efforts. Post-program surveys will be administered at the 6-month and 1-year anniversaries of the program. We will use these data to build a quantitative model that includes a longitudinal aspect. We are particularly interested in understanding how the participants used these skills in future efforts. We hope to develop a profile of the kind of students or groups that are most likely to participate in intermediate programing efforts. In the future, we will partner with research groups on campus for an Internal Review Board (IRB) approved study to investigate whether participants in these programs are able to meet the stated learning objectives as well.

5. RESULTS AND CONCLUSIONS

All participants completed the week-long program. An important consideration while evaluating these data is that these students had self-selected themselves to participate in intermediate computing exercises. Our data indicates that the choice of instructors did not impact student learning in the three models. Each approach and the associated exercises were appropriate for an intermediate program (90%), and were found to be equally engaging (80% or higher). Based on the performance of capstone (and review) exercises, we find that students who had been exposed to computing a year or more ago (Group 1) were better prepared for the program as compared to students who had recently participated in an introductory-level programming camp (Group 2). There was no discernible difference in the academic prowess of these two groups of participants. We assume that students who have had previous exposure to computational thinking would have had more time to synthesize new ideas and develop an understanding of the concepts. Another possible rationale for this observation is that, as self-described learners, Group 1 participants have learnt additional coding and computing concepts that were not taught to Group 2 participants during the introductory camp. Some of the major finding from these the approaches are:

Maker Spaces With Raspberry Pi HATs. While these activities scored highly in terms of processing ideas and taking ownership of ideas, the activities required the students to demonstrate Python programming skills. Students who had not previously developed these skills struggled in exercises that applied them to specific problems (30%).

Jupyter Notebooks in Machine Learning. Participating students were successfully able to manipulate the Jupyter Note- books to change training sets (100%), images (100%), and the number of epochs (100%). While all students could per- form the exercises, the conceptual underpinnings on ML were unclear to some (40%). Understanding of ML concepts was demonstrated by students working to confuse the model by finding images outside of the training data set (30%).

Problem Solving Exercises in Cryptography. Students appreciated that this approach connected to real world problems. The interactive nature of this session provided students opportunities to develop hypotheses to solve problems, and validate them (100%). In these exercises, students had the freedom to select one of many approaches to solve the problem at hand. Instructors were available to provide assistance, but did not direct the students. The data shows that the trainees found this approach to be more challenging in scenarios where there were a large number of possible solutions to solving a problem (80%). As such, this approach is perhaps better suited where students can directly apply the gained knowledge to solving a problem. Throughout this experience, we have found that teaching students new computing concepts, such as navigating a Linux environment, using the command line, and writing code, is more productive when done through an interactive format rather than using a lecture format that is interspersed with a few activities. With an interactive format, students are more motivated to follow along with the instructor and other students by participating in the activities. The various ways in which the problem could be solved increased the complexity of this approach for a number of students (80%). In agreement with existing literature, our data indicate that students with varying degrees of programming are best suited with scaffolded learning approaches like Jupyter Notebooks for application specific training. A problem-solving approach, though slower, encourages greater interactions and deeper learning of the subject matter. The makerspace activities provided the least amount of scaffolding and were less successful than their counterparts at incorporating increased levels of complexity in programming. Throughout the exercise, we find that, while the scaffolded elements allowed students to complete exercises, the lower learning gains indicate that it is best suited for complex topics where a single approach is attempted. The strategies described in this work were found to be effective for a group of intermediate learners and can be adopted in undergraduate curricula. Taken together, these strategies can help attract students to become the next generation of computer scientists, especially from groups that are currently underrepresented in the field.

6. SUSTAINABILITY & LESSONS LEARNED

The training program was designed with sustainability in mind. Student training in computing is a critical area where demand currently outweighs supply. Post-training surveys indicate that the program's format was well received by the community of students. Data from this training pro- gram shows that intermediate-level coding can be effectively combined with a number of fun activities that engage early undergraduate students and encourage them to participate in computer science. The largest challenges lie in presenting programming concepts at a level that can be comprehended and learnt by a diverse set of students. While certificates of attendance were provided to the participants, we are looking into making these efforts "transcriptable" so that employers can recognize them. While pre-training assessments were per- formed, in future iterations, we hope to assess student skills and competencies both pre- and post-program using evaluation scheme in the style of Student Assessment of Learning Gains (SALG) [21]. Representative samples of students and their parents will be interviewed. Some of the topics will include their experience in the camp, motivation to pursue careers in computing, STEM, and cybersecurity and ways to refine the offerings. The seemingly vast availability of Python teaching tools and the low cost of computing plat- forms makes the effort inherently sustainable. An abundance of free tutorials, educational makerspace coding activities, and intuitive Python interpreters have helped reduce the amount of programming that a user had to know prior to using large-scale computing or HPC resources. These approaches present exciting opportunities to engage students in programming, a critical step, to get them to learn and contribute to computing efforts.

7. SUPPORTING INFORMATION

All training materials developed by Texas A&M High Performance Research Computing (HPRC) are available for download free-of-charge on the Texas A&M HPRC website. Future adoptees can access the material at https://hprc.tamu.edu/training. Surveys, machine learning installation scripts, and a Linux review exercise are included as part of the Reproducibility Appendix. Agendas, registrations forms, sample announcements, templates to track participants, Trello event-boards and other such materials will be made available by the authors upon request. Please send us feedback about your adoption experience to help@hprc.tamu.edu.

8. ACKNOWLEDGMENTS

The authors thank staff, student workers and researchers at Texas A&M HPRC, the Laboratory for Molecular Simulation, TexGen, Division of Research, the Texas A&M Engineering Experiment Station (TEES) and Texas A&M Provost Information Technology for supporting the program. We gratefully acknowledge support from the National Science Foundation for award number 1649062, "NSF Workshop: Broadening Participation in Chemical and Biological Computing at the Early Undergraduate Level", and award number 1730695, "CyberTraining: CIP: CiSE-ProS: Cyberinfrastructure Security Education for Professionals and Students." We acknowledge the instructors and support staff of these courses: Narendra Chaudhary, Michael Dickens, Lisa Perez, Yang Liu, Keith Jackson, and Lan Li.

9. AUTHORS EMAIL ADDRESSES

The authors email addresses are – Dhruva Chakravorty, chakravorty@tamu.edu; Donald McMullen, mcmullen@tamu.edu; Honggao Liu, honggao@tamu.edu; Noushin Ghaffari nghaffari@tamu.edu; Dylan Rodriguez, dylan@tamu.edu; Shain Le, sle@tamu.edu; Nathan Gober, ngober@tamu.edu; Zengyu Wei, lele0027@tamu.edu; Levi Jordan, ljordan56@tamu.edu; Marinus Pennings, pennings@tamu.edu; Derek Rodriguez, whomps@tamu.edu; and Crystal Buchanan, crysb818@tamu.edu

10. REFERENCES

[1] Mart´ın Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large- scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16), Kimberly Keeton and Timothy Roscoe (Eds.), Vol. 16. USENIX Association, Savannah, Georgia, USA, 265–283

[2] Paul Michel Baepler, J.D. Walker, D. Christopher Brooks, Kem Saichaie, and Christina I. Petersen. 2016. A Guide to Teaching in the Active Learning Classroom: History,

Research, and Practice (1st ed.). Stylus Publishing, LLC, Sterling, Virginia, USA.

[3] Karen Brennan and Mitchel Resnick. 2012. New Frameworks for Studying and Assessing the Development of Computational Think- ing. In Proceedings of the 2012 Annual Meeting of the American Educational Research Association. American Educational Re- search Association, Vancouver, British Columbia, Canada, 25.

[4] Dhruva K. Chakravorty, Marinus “Maikel” Pennings, Honggao Liu, Zengyu “Sheldon” Wei, Dylan Rodriguez, Levi T. Jordan, Donald “Rick” McMullen, Noushin Ghaffarri, and Shaina D. Le. 2018. Effectively Extending Computational Training Using Infor- mal Means at Larger Institutions. Practice and Experience in Advanced Research Computing (2018).

[5] Franc ̧ois Chollet et al. 2015. Keras. https://keras.io

[6] Carol L. Fletcher. 2014. Building the Texas Com- puter Science Pipeline: Strategic Recommendations for Success. Technical Report. Texas Regional Collabo- ratives. https://www.thetrc.org/web/assets/files/pdfs/ Building-the-Texas-CS-Pipeline Fletcher.pdf

[7] National Science Foundation. 2018. Program Solicitation NSF 18-537: Computer Science for All. https://www.nsf.gov/pubs/ 2018/nsf18537/nsf18537.pdf

[8] Chris Holdgraf, Aaron Culich, Ariel Rokem, Fatma Deniz, Maryana Alegro, and Dani Ushizima. 2017. Portable Learning Environments for Hands-On Computational Instruction: Using Container- and Cloud-Based Technology to Teach Data Science. In Proceedings of the Practice and Experience in Advanced Re- search Computing 2017 on Sustainability, Success and Impact, David Hart and Maytal Dahan (Eds.). ACM, New York, NY, USA, 32.

[9] Andrej Karpathy. 2017. Stanford University CS231n: Convolu- tional Neural Networks for Visual Recognition. http://cs231n. stanford.edu/syllabus.html

[10] Martha Larkin. 2002. Using Scaffolded Instruction To Optimize Learning. ERIC Digest. (Dec 2002).

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. Nature 521, 7553 (2015), 436.

[12] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. [n. d.]. The MNIST Database of Handwritten Digits. http://yann. lecun.com/exdb/mnist/

[13] James J. Lu and George H.L. Fletcher. 2009. Thinking About Computational Thinking. ACM SIGCSE Bulletin 41, 1 (Mar 2009), 260–264. https://doi.org/10.1145/1539024.1508959

[14] Robin R. Murphy. 2016. Emergency Informatics: Using Computing to Improve Disaster Management. Computer 49, 5 (May 2016), 19–27. https://doi.org/10.1109/MC.2016.135

[15] Hyesung Park, Kihyun Kim, and Cindy Robertson. 2018. The Impact of Active Learning with Adaptive Learning Systems in Gen- eral Education Information Technology Courses. In Proceedings of the Southern Association for Information Systems Confer- ence (SAIS 2018). Association for Information Systems, Atlanta, Georgia, USA.

[16] Jim Parsons and Leah Taylor. 2011. Improving Student Engage- ment. Current Issues in Education 14, 1 (2011).

[17] Michael Prince. 2004. Does Active Learning Work? A Review of the Research. Journal of Engineering Education 93, 3 (2004), 223–231.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115, 3 (2015), 211–252.

[19] Kem Saichaie, D. Christopher Brooks, Phillip Long, Robert Smith, Richard Holeton, Carole Meyers, Adam Finkelstein, and Shirley Dugdale. 2017. 7 Things You Should Know About Research on Active Learning Classrooms. In ELI 7 Things You Should Know. EDUCAUSE, Washington, DC, USA.

[20] Jinsil Hwaryoung Seo, Michael Bruner, Austin Payne, Nathan Gober, Donald “Rick” McMullen, and Dhruva K. Chakravorty. 2018. Using Virtual Reality to Enforce Principles of Cybersecurity. Journal of Computational Science Education (Nov 2018). (to appear).

[21] Elaine Seymour, Douglas J. Wiese, Anne-Barrie Hunter, and Susan M. Daffinrud. 2000. Creating a Better Mousetrap: On- line Student Assessment of their Learning Gains. In Proceedings of the National Meetings of the American Chemical Society Symposium. American Chemical Society, San Francisco, California, USA.

[22] Kimberly Sheridan, Erica Rosenfeld Halverson, Breanne Litts, Lisa Brahms, Lynette Jacobs-Priebe, and Trevor Owens. 2014. Learning in the Making: A Comparative Case Study of Three Makerspaces. Harvard Educational Review 84, 4 (2014), 505–531.

[23] Jeannette M Wing. 2008. Computational Thinking and Thinking About Computing. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 366, 1881 (2008), 3717–3725. https://doi.org/10.1098/ rsta.2008.0118

[24] Aman Yadav, Ninger Zhou, Chris Mayfield, Susanne Hambrusch, and John T Korb. 2011. Introducing Computational Thinking in Education Courses. In Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, Thomas J. Cortina, Ellen L. Walker, Laurie Smith King, and David R. Musicant (Eds.). ACM, Dallas, TX, USA, 465–470. 457110.