Teaching Python to Secondary Students: A Backward Design Process

Dr. Wesley A Brashear, Texas A&M University
Dr. Sandra B Nite, Texas A&M University

Sandra Nite is trained as a mathematics educator and educational researcher. She is Director of AP Institutes in Mathematics and Computer Science in the College of Arts and Sciences. Besides her Ph.D. in Mathematics Education, she holds master's degrees in mathematics and music and lifetime Texas secondary teaching certification for mathematics, computer science, biology, chemistry, composite science, music, English, and English language arts as well as Master Math Teacher (8-12) certification. Dr. Nite has 17 years' experience teaching high school mathematics, science, and computer programming. She initiated a high school computer science program 1978, when such programs were rare. She taught preservice mathematics courses for elementary and middle school teachers and mathematics methods for elementary teachers. She has served as PI or Co-PI on secondary STEM summer camps for eight (8) years. Since joining High Performance Research Computing, she has helped developed cybersecurity curricula for summer camps, serving at the K12 expert on those camps. For the past 20 years she has worked extensively with professional development curriculum for secondary teachers, both inservice and preservice, to increase content knowledge and pedagogical content knowledge. As Co-PI for 3 years and PI for 7 years on Teacher Quality grants, she designed and developed over 1,000 hours of professional development. As Co-PI on a National Science Foundation Robert Noyce Scholarship Program grant, she designs and delivers professional development for the scholarship recipients She has not only worked with teachers in Texas and other states in the U.S. but also with teachers from Turkey and Qatar. Dr. Nite's research agenda has focused heavily on bridge programs for engineering calculus, STEM secondary education, and STEM teacher professional development. She also conducts research in music education. She is currently working in High Performance Research Computing at Texas A&M University.

Richard Lawrence, Texas A&M University
Dhruva Chakravorty, Texas A&M University

Teaching Python to Secondary Students: A Backward Design Approach

Abstract

Informal workshops and educational events are often restricted in the number of contact hours or opportunities for extensive in-depth coverage of foundational material. This is not an issue when educators are building on existing skill sets or covering a limited scope of material, but it is a challenge that needs to be addressed when teaching students a skill like programming - a broad topic which students might not have previously been taught. Project based learning is an effective pedagogical tool for teaching computer science, and the end product or goal is often a solution to a particular coding problem or a software application that performs a given task. However, students must be provided with some degree of foundational knowledge in order for this approach to be utilized. The content and extent of this knowledge is dependent on the focus and difficulty of the project, and can be particularly difficult to establish when working with students without prior programming experience. Furthermore, when teaching high-level general programming languages, the expansive suite of built-in tools coupled with additional third-party packages or libraries present a dense landscape of topics from which to develop curricular materials. To meet these challenges, we developed an effective approach to teaching Python programming to secondary students with no prior programming experience in a week-long summer camp. The method we used employs project based learning and highly curated foundational lessons. This approach begins with the identification of an appropriate capstone project that falls within the theme of the camp (e.g. coding, cybersecurity, data science) and that can be completed by students with minimal instruction from camp staff. These projects should also be able to incorporate more advanced programming techniques than those that are covered during the camp to keep all students engaged, including those with previous coding experience or natural aptitudes for programming. For example, the capstone project for one cybersecurity-themed camp required students to develop a simple application that could accept user input (a password) and then assess the quality or strength of that password and provide the user with feedback. In its simplest implementation, this application requires a basic understanding of the following concepts: 1) the basic elements of Python code (data types, variables, operators), 2) programming syntax, 3) built-in functions and methods, 4) acquiring user input, and 5) flow control (e.g., for loops, conditional statements). Many topics covered in traditional introductory programming courses are not required for students to be able to complete the capstone project; concepts like data structures, indexing, and user-created functions can be utilized in the project, but are presented as optional material for advanced students. The limited instructional time afforded during these week-long camps necessitates that instructional material be restricted to what we consider the essential “building blocks” required for students to successfully produce the final product. This stream-lined curriculum enhanced with optional bonus material ensures that both novice and experienced students remain engaged and are equipped with the tools necessary to challenge their skills while they build their own applications. In this paper we detail the process to design Python learning objectives for capstone projects, in which we start with the end product in mind, determine the Python programming elements needed, and work backwards to ascertain the order in which the elements should be taught. We also discuss the optional topics embedded in the Python Colab notebook for advanced students - topics designed to give the advanced students more options in designing their product.

Introduction

Informal educational programs for K-12 students (e.g. summer camps) offer a unique chance to foster interest and awareness of STEM fields outside of formal education settings and can represent the only opportunities some students have to engage in specific STEM fields prior to attending college [1] [2]. These informal programs must therefore be designed to cultivate a deep appreciation and interest in these topics while also addressing limitations or constraints such as a minimal number of contact hours and disparities in prior knowledge or experience. How can we meaningfully cover an advanced subject over a limited timeframe in such a way that 1) fosters individual student interest in the subject matter; 2) allows advanced students to remain engaged and novice students to not be overwhelmed; and 3) leaves each student with a sense of accomplishment and a better fundamental understanding of the covered material? Herein, we discuss our approach to these challenges using a backward design approach [3] to teach coding in Python to secondary students in week-long summer camp sessions.

The backward design process consists of three stages. First, teachers must identify the desired results or instructional outcomes. That is, what would we like the students to learn or understand or what skill would we like them to obtain. Next, teachers must determine what constitutes acceptable evidence for successful attainment of the desired result. This could include any number of assessment methods, including basic checks for understanding, quizzes/tests, or completion of a specific task or project. Once these steps are completed, the teacher can then plan the necessary learning experiences and instruction required for the students to demonstrate their progress towards the desired outcomes [3].

The backward design process can be used when developing experiential project-based learning approaches toward student learning objectives [4] [5]. In project-based learning, students are given an “ill-defined task with a well-defined outcome” [6]. This approach is particularly well-suited for engaging students while teaching coding, as a specific project or task can be completed with numerous approaches and with varying degrees of complexity. For example, students can be tasked with something as simple as creating a program that can take user input, perform a basic operation on the input, and return the result. A simple version of this script could be written with 2-3 lines of code, whereas a more advanced version might incorporate user-defined functions, flow control, and error handling. Both programs successfully complete the “ill-defined task”, but the complexity can be modified to engage students with varying degrees of previous experience. This flexibility is highly advantageous when teaching a cohort of students with the disparate levels of maturity and experience inherent in an informal educational environment such as a week-long summer camp.

Methodology

Summer Camp Structure

We have offered week-long summer camps for secondary students since 2017 [7]. These camps are in-person, day-only programs centered around a specific theme relevant to computing (e.g. cybersecurity, data science). Activities throughout the week include hands-on coding lessons in Python, guest lectures, university campus and data center tours, and group projects (see Table 1). Each camp hosts 35-50 students ranging in age from 13 to 17. Students are divided into groups of 4 to 6 students with whom they will work on group projects and activities throughout the week. Groups are balanced to include similar distributions among age and gender.

Implementation of the Backward Design Process

The first step in designing the curriculum for coding instruction for the week is to identify an appropriate capstone project that aligns with the theme of the camp (e.g. cybersecurity, data science). Capstone projects should be able to be completed by students with minimal oversight from the camp staff after students have participated in relatively few foundational lessons and exercises. These projects should also be able to incorporate more advanced programming techniques beyond those covered in the camp to keep students with previous coding experience or those with natural aptitudes for programming engaged.

Once the project has been established, the Python coding lessons held throughout the week can be planned (see Table 1, Fig. 1). The selection of Python lesson topics might not follow the order in which topics are covered in a formal class setting: lessons should be curated to cover only the material needed for the students to complete the project. For example, some capstone projects might require external modules or advanced data structures that might need to be covered instead of simpler concepts or built-in functions. We visualize this approach using an analogy to building with bricks (Fig. 2). A formal course might take a complete bottom-to-top approach in terms of building a more complete understanding of a particular coding language, completing each row or layer of foundational material before moving on to the next. Our approach only uses those building blocks necessary for the students to complete a project that is relevant to the camp's theme and for which they will gain a sense of accomplishment. This patchwork approach allows students to achieve a basic understanding of some components of coding in the limited time offered by week-long camps while also (hopefully) inspiring them to continue to learn independently outside of these educational settings or to consider enrolling in formal coding classes.

Table 1. Example schedule for a week-long computing camp for secondary students.

Time	Monday	Tuesday	Wednesday	Thursday	Friday
8:00	Arrival and Dropoff				
8:15	Summer Camp Orientation	Group projects			
8:30		Camp-themed Game	Camp-themed Game	Camp-themed Game	Camp-themed Game
9:00	Introductions and Laptop Checkout	Secure System Lecture	Non-coding computing activity	Coding in Python	Data Center Tour and Project Preparation
9:30	Escape Room				
9:50	Break	Break	Break	Break	
10:00	Escape Room	Industry Speaker	Safe Online Behavior and Social Engineering	Campus Tour	
10:30	Camp Theme Discussion				
11:00	Break	Break	Break		Break
11:10	Coding Python	Academic Guest Lecture and Activity	Project Preparation		Applied Concepts Activity
11:30					
12:00	Lunch	Lunch	Lunch	Lunch	Lunch
1:00	Flying Drones and Drone Security	Introduction to AI/ML	RaspberryPi Activity	Academic Guest Speakers	Camp Surveys and Project Preparation
1:50	Break	Break	Break	Break	Break
2:00	Group Project Overview	Coding in Python	Coding in Python	Cryptography activity	Group Project Presentations, Camp Certificates, and Closing
2:50	Break	Break	Break	Break	
3:00	Enterprise IT Guest Speakers	Coding in Python	Project Preparation	Capstone Python Project	
3:50	Break	Break	Break	Break	
4:00	Group Projects, Daily Reflections, and Laptop Check-In				
4:40	Dismissal / Pickup				

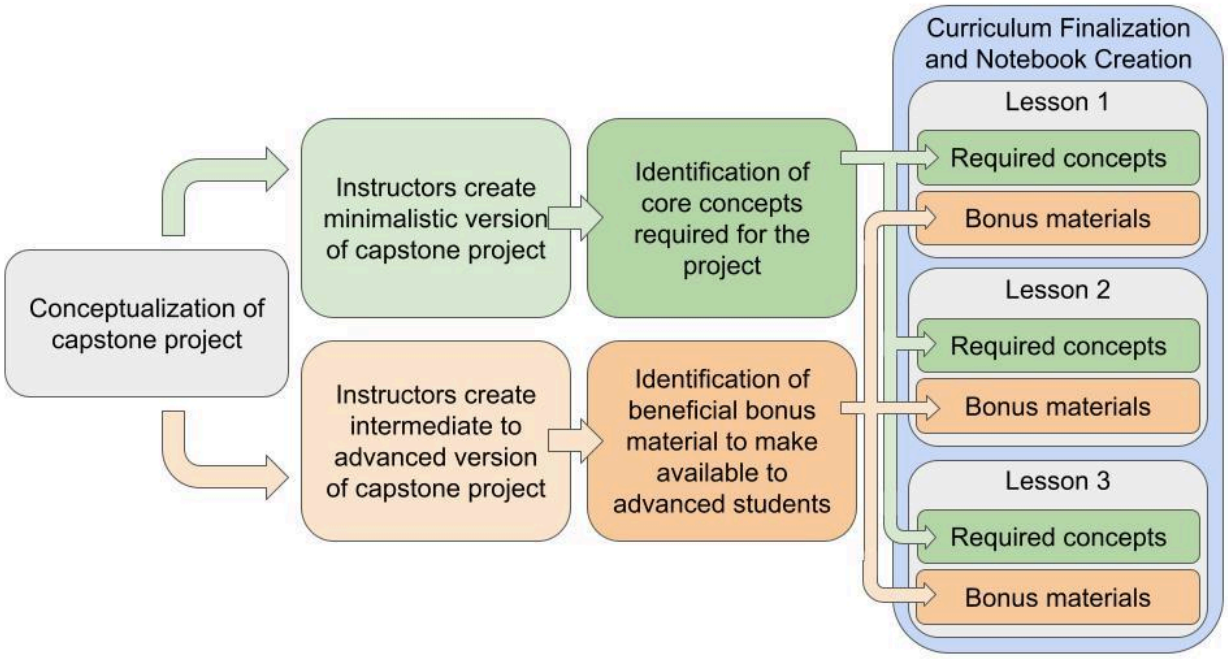

Figure 1. Backward design flowchart for developing lessons in coding for a week-long summer camp.

If we consider teaching a particular subject as analogous to building a structure, traditional education approaches often seek to build a strong foundation covering introductory topics to completion before advancing to intermediate or advanced topics. To overcome time constraints, we employed a backwards design approach to identify only those skills necessary to complete a particular project. The blue bricks in Figure 2 were topics needed for a project in the summer camp. The project required a knowledge of methods at the highest level. Instructors identified other topics needed to complete the project as well as topics needed at lower levels to provide the foundation for the higher levels needed. This provided the framework for creating a Google Colab to prepare students for the project assignment.

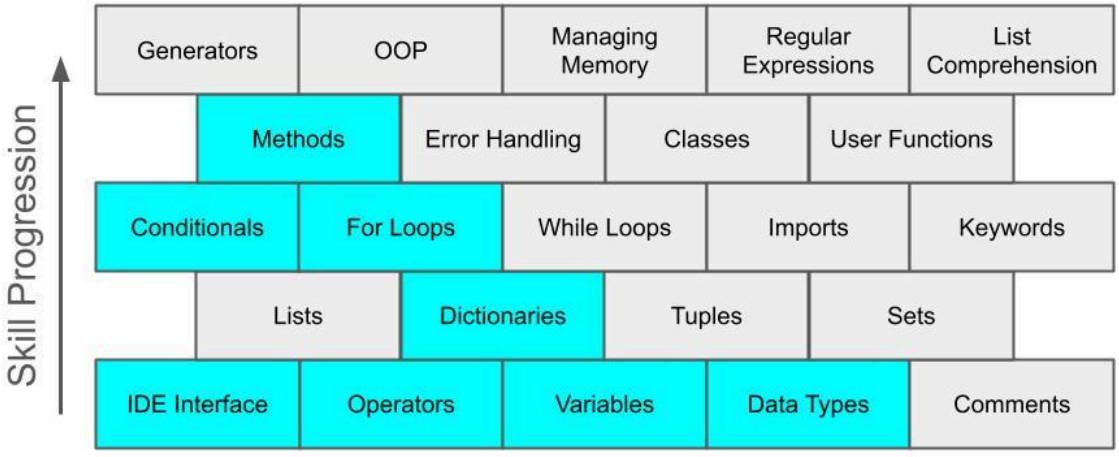

Figure 2. Skill progression for teaching Python programming.

Results

Capstone Project Example 1 - Password Strength Checking Program

Project Description. This capstone project was chosen for one of our GenCyber [8] camps focused on cybersecurity. The task given to the students is to write a Python program that can evaluate the quality of a password based on its content and length (e.g. number of characters, presence of both upper and lower case letters, numbers, and special characters). A simple version of this program can be completed with relatively little programming knowledge: variables, conditional statements, Python keywords, and arithmetic operations (Figure 3a). These topics can be covered sufficiently within the time allotted for coding instruction during the summer camp. For more advanced students, this project can also incorporate additional coding methods or techniques, such as user-defined functions, data structures, flow control, error handling, modules, and user interactivity (e.g. Figure 3b).

***Python Lessons*.** We used Google Colab to build instructional notebooks for each summer camp. This platform is a free service in which students are able to login to run Jupyter-style notebooks that contain code chunks/cells and explanatory text. Students are first introduced to the Google Colab interface as instructors go over topics like the environment runtime, code execution, and

A)

```
# Set pw variable as password you wish to check
pw = ''

for char in pw:
     if char in 'abcdefghijklmnopqrstuvwxy':
          lower_count=lower_count+1
     elif char in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
     upper_count=upper_count+1
     elif char in '0123456789':
          numeral_count=numeral_count+1
     else:
          other_count=other_count+1

score=upper_count*lower_count*numeral_count*other_count
print(score)
```

B)

```
def GetUserPassword():
    password = input("What password would you like to check?\n")
    return(password)

def GetPasswordContents(pw):
    upper = False
    lower = False
    number = False
    special = False
    length = len(pw) >= 8
    for char in pw:
        if char.isupper() == True:
            upper = True
        if char.islower() == True:
            lower = True
        if char.isnumeric() == True:
            number = True
        if char.isalnum() == False:
            special = True
    return([upper, lower, number, special, length])

def ScorePassword(pw_contents):
    if pw_contents.count(True) == 5:
        return("Your password is great!")
    else:
        return("Your password could use some work.")

if __name__ ==  '__main__':
    password = GetUserPassword()
    password_contents = GetPasswordContents(password)
    user_answer = ScorePassword(password_contents)
    print(user_answer)
```

Figure 3. Two examples of python programs that evaluate the strength of a password. A) A simple version of the script that utilizes relatively few coding concepts, but does include comments, variables, flow control, and operators. B) A more advanced example of the same python program that includes user-defined functions, built-in methods, and user interactivity.

notebook navigation. Students execute their first code during this session: a version of the traditional ‘Hello World’ program [9].

The second Google Colab notebook used during this summer camp covered several introductory topics concerning the elements of code. We first cover comments in code, discussing the syntax in Python and why it is considered good practice to use comments in general. We then cover data types, discussing strings, integers, floats, and logical data. We also use this section to cover syntax highlighting the syntax specific to Google Colab (e.g., valid string data displayed in red text, logical data in blue). Next we discuss simple arithmetic operators in Python, including the order of operations and conditional operators. Our backward design approach helped us identify the utility of the Python ‘in’ operator for our capstone project, and this was covered in detail with exercises, following the conditional operators. Lastly, we covered the assignment operator and variables, including arithmetic with variables and variable errors. This notebook also contains bonus exercises and deeper coverage of data types, operators, and variables [9].

The next Google Colab notebook covered functions and methods in Python. In order to get user input for the capstone project, we covered the Python ‘input’ function and showed students how to store the output of that function in a variable. We also covered the ‘range’ function, which can then be utilized in the next notebook to teach concepts like for and while loops. The next notebook covers flow control (including for and while loops), where we teach the basic syntax of loops in Python, giving examples of iterating over ranges of numbers and characters in string variables. We then built on these concepts and combined loops, conditional statements, breaks, and functions from additional Python libraries (e.g. ‘random’). Each of these topics again contain bonus materials to help ensure engagement from all students [9]. Upon completion of the notebook covering flow control and loops, students are well-situated to create their own Python script to “grade” the strength of a user-defined password.

Capstone Project Example 2 - Encrypting Messages to Friends

Project Description. This capstone project is from another GenCyber camp [8], during which the topic of cryptography was covered during several lectures and hands-on (non-coding) exercises. This project tasked the students with using the RSA (Rives-Shamir-Adleman) cryptosystem [10] to send and receive encrypted messages with other students at the summer camp. The Google Colab notebook for this exercise utilized the Python rsa package to create public and private keys and then use these keys for encryption/decryption. The background required for this specific project is limited, but an understanding of the dictionary data structure in Python is beneficial to both disambiguate shared terminology (i.e., key) and to better understand the syntax of the methods specific to the rsa package.

Python Lessons. Our initial Python lessons closely aligned with the first two Google Colab notebooks described, for the previous capstone project. For this specific camp, however, we devoted an entire notebook to covering data structures in Python. We began by covering lists, discussing the basic Python syntax and terminology and the concept of ordered and unordered data structures. We then covered data types of elements in lists, as well as useful functions for working with this and other data structures (e.g., 'type', 'len'). We next covered dictionaries, again covering basic terminology (i.e., key, value) and syntax. We then covered indexing in Python, first using strings as an example to explain 0-based versus 1-based indexing, as well as the backwards indexing functionality in Python. These concepts were then applied to data structures in subsequent examples. We then covered modifying data structures, which allowed us to build on indexing to cover slicing. Lastly, we covered methods specific to different data structures, including '.sort()', '.append()', '.keys()', '.values()' and '.items()' [9].

Assessment

Because of the informal learning environment at the summer camp, there was no pretest/posttest of student learning for programming in Python. The application does ask them to rate their programming experience so that we have an idea of how many are new, intermediate, or experienced in programming in general. It would be difficult to test their Python knowledge because asking if they know about variables, loops, etc. would not really tell us the level of competency in using these concepts. We would need a task at the beginning and end of the camp, which time does not permit. However, campers completed daily reflections the first four days of the camp, and a reflection for the week on the final day of the camp. In the reflection for the days that included sessions in Python programming, we asked students how much they learned. Table 2 shows the percentage of students who selected each choice to the question about how much Python they learned on the first two days of camp. On the first day (Monday), the total attendance for the two one-week camps was 68, and the total on Tuesday was 70.

Table 2. Percentage of campers who made each choice on the question about how much Python they learned that day.

Choice	**A lot**	**Quite a bit**	**A little bit**	**Not at all**
Monday	29%	34%	24%	13%
Tuesday	33%	36%	19%	12%

In the free response section about what students liked most or learned most from among the plethora of camp activities on Monday and Tuesday (days Python was taught) Python was mentioned as a favorite activity in 34 of the 68 responses and as an activity from which they learned the most in 44 of the 70 responses. It was also mentioned on Friday 17 times in response

to the greatest learning experience of the entire week. Below we have shared a few of our favorite responses from students:

"I learnt the most from the python activities since I was able to learn a language in demand [in] *today's society, and the* [opportunity] *to learn it from amazing instructors and staff."*

"My favorite activity this week was Cryptography with Python because I got to encrypt and decrypt messages and send my public key to my friends so that they can encrypt or decrypt my secret message. I learned that people in World War II used Cryptography to send secret messages without the enemy knowing."

"My favorite activity for the week was doing cryptography in Python because I enjoyed decrypting secret messages that my friends sent. This enhanced my Python skills immensely."

Discussion

Although the backward design process is fairly often used in curriculum design, especially when students must take a standardized test at the end, much of current teaching in programming is based on starting with foundation skills and building on those skills, not leaving out anything that the learner might need later. This makes sense when teaching a programming language for which learners need a deep understanding of all or most topics in the language. However, when an important purpose of the programming instruction is to arm students with the tools needed for a specific project, and there is not time to teach everything in depth, the backward design process is very effective. In addition, student success is important to strengthen and maintain their desire to learn more [13]. Coding in summer camps has been shown to increase STEM interest [11][12] However, learning a programming language can be a daunting task. Thus, teaching Python programming with the backward design process in camps, clubs, or after school activities can give students a sense of success and increase students' interest in learning more about Python or other programming languages [14]. This strategy could also be used in formal education with a careful selection of projects that lead to learning the programming language in depth over the course of the year. Over the long run such strategies can broaden participation in computing by increasing interest and confidence in learning programming languages.

Acknowledgements
We gratefully acknowledge the GenCyber Program for the support to offer cybersecurity camps free of charge to U.S. students.

References
[1] J. R. Warner, C. L. Fletcher, R. Torbey, and L. S. Garbrecht, "Increasing capacity for computer science in rural areas through a large-scale collective impact model," *Proceedings of*

the 50th ACM Technical Symposium on Computer Science Education, *SIGCSE '19, Minneapolis, MN, USA, February 27–March 2, 2019,* New York: ACM, 2019. pp. 1157-1163.
[2] C. De Lira, R. Wong, O. Oje, G. Nketah, O. Adesope, A. Ghods, "Summer programming camps - exploring project-based informal CS education in a rural community," *International Journal of Computer Science Education in Schools,* vol. 5, no. 4, ISSN 2513-8359, Sep. 2022.
[3] G. Wiggins, J. McTighe, "What is Backward Design?" in *Understanding by Design,* Alexandria: Association for Supervision and Curriculum Development, 1998. [Online]. Available: https://educationaltechnology.net/wp-content/uploads/2016/01/backward-design.pdf [Accessed December 18, 2024].
[4] J. Dewy, *Experience and Education*. New York: Macmillan Company, 1938.
[5] Y. Liao, M. Ringler, "Backward design: integrating active learning into undergraduate computer science courses," *Cogent Education*, vol. 10, no. 1, 2204055, Apr. 2023.
[6] R. M. Capraro, M. M. Capraro, and J. Morgan, *STEM Project-Based Learning: An Integrated Science, Technology, and Mathematics (STEM) Approach*. Rotterdam, The Netherlands: Sense Publishers, 2013.
[7] S. B. Nite, T. J. Gray, S. Lee, and S. Stebenne, "Engaging Secondary Students in Computing and Cybersecurity," in *Practice and Experience in Advanced Research Computing, PEARC '24, July 21-25, 2024*, New York: ACM, 2024. 5 pages. https://doi.org/10.1145/3626203.3670624
[8] T. Ladabouche, S. LaFountain, "GenCyber: Inspiring the Next Generation of Cyber Stars," *IEEE Security and Privacy*, vol. 14, no. 5, 2016, pp.84-86. https://doi.org/10.1109/MSP.2016.107
[9] *Github link with notebooks. Removed to retain anonymity.*
[10] R. L. Rivest, A. Shamir, L. M. Adleman, "Cryptographic communications system and method". U.S. Patent #4405829.
[11] P. LePendu, C. Cheung, M. Salloum, P. Sheffler, P. and K. Downey. "Summer coding camp as a gateway to STEM". *In Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (SIGCSE '20), March 11-14, 2020, Portland, OR, USA. ACM, New York, NY, USA, 1 page. DOI: https://doi.org/10.1145/3328778/8.337263
[12] A. Bicer, Y. Lee, R. M.Capraro, M. M. Capraro, L. R. Barroso, D. Bevan, and K. Vela. "Cracking the Code: The Effects of Using Microcontrollers to Code on Students' Interest in Computer and Electrical Engineering". *In 2018 IEEE Frontiers in Education Conference* (FIE) (pp. 1-7). IEEE.
[13] S. B. Nite, A. Bicer, K. C. Currens, and R. Tejani. "Increasing STEM Interest through Coding with Microcontrollers," *In Proceedings of 2020 IEEE Frontiers in Education Conference: Education for a Sustainable Future. Uppsala, Sweden (online due to COVID).*. https://doi.org/10.1109/FIE44824.2020.9274273
[14] S. B. Nite, D. C. Rice, and R. Tejani (2020). "Influences for engineering majors: Results of a survey from a major research university," *Proceedings of the 2020 American Society for Engineering Education (ASEE) Virtual Annual Conference.* DOI: 10.18260/1-2--34825