ISSN: 2473-3857
v4 n4641

Reflections on Applying the Growth Mindset Approach to Computer Programming Courses

Katherine Carl Payne
kpayne@wtamu.edu

Jeffry Babb
jbabb@wtamu.edu

Amjad Abdullat
aabdullat@wtamu.edu

Department of Computer Information and Decision Management
West Texas A&M University
Canyon, Texas 79016, USA

Abstract

This paper provides motivation for the exploration of the adoption of Growth Mindset techniques in computer programming courses. This motivation is provided via an ethnographic account of the implementation of Growth Mindset concepts in the design and delivery of two consecutive computer programming courses. We present components of a Growth Mindset approach to STEM education and illustrate how they can be implemented in computing courses. Experiences of an instructor in two consecutive computer programming courses at a private regional undergraduate university in the southwestern United States are described and analyzed. Six students enrolled in the required Java course in the fall semester opted to take the elective Python course in the spring semester.

Keywords: Growth mindset, reflective teaching practice, computer programming, STEM education, pedagogy

1. INTRODUCTION

As educators in STEM courses, we frequently hear students utter the phrase "I'm not a technical person" or "I'm not good with computers." I receive messages from concerned students expressing this sentiment every semester. With each iteration of this disclaimer, I have thought more about the following questions: 1) How do students reach the conclusion that they do not innately possess technical skills? 2) What are the implications of students informing their teachers that they are "not technical people"? and 3) What can I do as an instructor to promote students' success in technical courses when they have these self-perceptions?

The work of Carol Dweck on what is termed the "Growth Mindset" has received significant attention in primary and secondary math education and has recently experienced a renaissance through application to math higher education (Silva & White, 2013). Her work and the work of Jo Boaler have informed my understanding of student self-perception and motivation and have provided the impetus for designing a Growth Mindset approach to computing courses (Boaler, 2016).

The goal of this paper is to provide motivation for the continued exploration of adopting Growth Mindset concepts in programming courses. By relating experiences with students in two

ISSN: 2473-3857
v4 n4641

consecutive programming courses—Java and Python—we provide motivation for and describe the implementation of Growth Mindset concepts in a technical setting. To provide a theoretical framework for the course experiences, the Growth Mindset approach is first defined and characterized and a summary of its use and effectiveness in higher education is provided. The course experiences are related in the "confessional style" from the perspective of one of the authors, in which some observations of the everyday experiences of the students are recounted (Schultze, 2000).

After a review of the Growth Mindset approach, the course experiences are described in three sections. The first two sections relate the experience of the Java course, including my initial approach to its design and delivery, the challenges faced by the students and myself during the first part of course, and how the course was redesigned after the midterm to address these challenges by applying Growth Mindset concepts. The third section describes how the experience of the Java course influenced the students to take another programming course with me and how Growth Mindset concepts informed the design and implementation of the Python course. A reflective discussion of both courses and conclusions follow.

2. THE GROWTH MINDSET

The Growth Mindset approach is characterized by a combination of attitudes towards students in the classroom and actions taken to promote such attitudes in course design and delivery. Jo Boaler indicates that people with a Growth Mindset believe that students' intelligence increases with hard work, whereas those with a "fixed mindset" believe that while students can learn, they cannot change their basic level of intelligence (Boaler, 2016). In practice, a Growth Mindset is characterized by the following:

1. Problem-driven learning
2. Emphasis on learning concepts over memorization of facts
3. Emphasis on the value of the learning process
4. Framing mistakes as a positive and integral part of the learning process
5. Collaborative problem-solving

Application in Higher Education

There has been a call for an increase in both the provident of STEM subjects in both the K-12 and post-secondary arenas of education (Breiner et al., 2012; Brown et al., 2011; Charette, 2013). In these cases, this call focuses on subjects which generally contain a bit more analytical/mathematical content than would be the case for other subjects. There is also evidence that simply guiding students to STEM, and emphasizing its importance, is not a guarantee for student engagement and flourish in the subjects (Bell, 2016).

Among the reasonable questions to ask is what pedagogies may be necessary to accommodate an influx into STEM such that students not previously oriented to the subjects will succeed? The Growth Mindset, established particularly in mathematics, holds the potential to widen the net of those who are both engaged in and find self-discovery and growth in learning STEM material.

To wit, we explore where Growth Mindset techniques have been applied in higher education and with what success. We focus specifically on motivation as, given recent shifts to viewing the benefits of education as being rooted in competency (Trotter and Ellison, 1997), motivation remains a primary component of successful outcomes (Maurer et al., 2003).

Much of the work on Growth Mindset centers on a proposition that the essential nature of a person (self-image, social role, traits, and motives) is of importance in a discussion of education, as it leads to the development of the knowledge and skills and to demonstrable competency (Spencer and Spencer, 1993). Thus, both what students encounter in technical challenges and the performances we observe are at the "surface" of observation. Whereas, components such as traits and motivations, though not as readily observable, constitute aspects of a "mindset" that can at least be influenced.

The literature on Growth Mindset tends to paint a dichotomy whereby mindsets, and thus intelligence, is either *fixed* or *growth*. At a finer point, growth modes fostered pedagogies of discovery and habits rather than a simple inventory of the challenges of the material and a map of the territory to be conquered. The difference can be subtle and, as much as when one is asked whether they are a "good person," few will want to identify with or own any behaviors ascribed to being within the "fixed" mindset.

A mindset can be somewhat of a fleeting and innocuous endeavor as much of progress and definition happens outside of observability. Improvisation (Weick, 1998), entrepreneurship (McGrath et al., 2000), and global orientation

(Gupta and Govindarajan, 2002) have each been outlined as requiring a concomitant "mindset" for their most efficacious application.

It is difficult to pinpoint a single conclusive definition for mindset; however, it seems to be similar to worldview, Weltanschauung, or outlook. As an internal yet influenceable component of an individual, it is certainly attached to sense making, motivation, and belief.

In its essence we see the distinction between fixed and Growth Mindsets as follows:

- *Fixed*: Intelligence and capability are set from an initial pool from which the individual draws upon. The volume of this "pool" is set and not anticipated to expand.
- *Growth*: Intelligence and capability are cultivated through habits that include effort, persistence, and motivation

Among the aspects of the work of Dweck (2008) and Boaler (2013) on mindset are those that center on the student-teacher interactions during the processes of learning and assimilating new and difficult material. Their work centers on the nature of feedback and what the object of that feedback is. In cases where students faced adversity, it was praise for their processes of persistence, hard work, and focus that led to more lasting and positive outcomes (Mueller and Dweck, 1998). This is a principal characteristic of a Growth Mindset. On the other hand, praise ascribed to the individual—good job, you are very smart—reinforced an emphasis on innate qualities rather than earned qualities. Thus, much as muscle is created through breaking and then strengthening tissues via exercise, the new neural pathways created via the adversity of challenge mostly leads to an expansion of capability and efficacy (Boaler, 2013).

These postulates lead to questions regarding the design of learning experiences in the classroom and the overall pedagogical strategies needed particularly in STEM topics such as computer programming. Notoriously difficult to learn (Dalbey and Linn, 1985; Jenkins, 2003; Pea and Kurland, 1984), programming seems to be a promising candidate for the application of the Growth Mindset.

Forays and successes in the use of the Growth Mindset are discernable in the literature, particularly so in STEM areas. O'Rourke et al. (2014) discuss using Growth Mindset in a gamified structure to develop incentives to persist to good effect. To the degree that a Growth Mindset elicits creativity, Karkowski (2014) has explored the challenges of measuring this mindset. Hernandez et al. (2013) conducted a longitudinal study of the manner in which the Growth Mindset encouraged participation in STEM for underrepresented students and found some success as well. Overall, there is growing evidence that the Growth Mindset is effective in the STEM context and worth consideration and implementation (Murphy and Thomas, 2008).

An additional important contextual consideration for Growth Mindset is not entirely rooted in its inherent potential to help the learning process, but it its foundational tenets—that attitudes about learning matter. The basis for these attitudes has some grounding in students' formative processes. Thus, many students have a context and background that would suggest wide variance in inputs to foundational attitudes towards learning. For instance, many first-generation college students may not have a framework in their households that paints an experiential picture of the habits and attitudes required for success (Terenzini et al. 1996). This would also be the case for the non-traditional student wherein the expectations inherent in the work required to excel in programming may not be innate and readily identifiable (Collier and Morgan, 2008). Thus, it is against this backdrop that we proceed to describe the case for and the means by which a focus on Growth Mindset provided a wider pathway to success for more students in what is always a fairly challenging endeavor: computer programming. The experiences are presented from the first-person perspective of one of the authors.

3. JAVA PROGRAMMING: THE INITIAL APPROACH

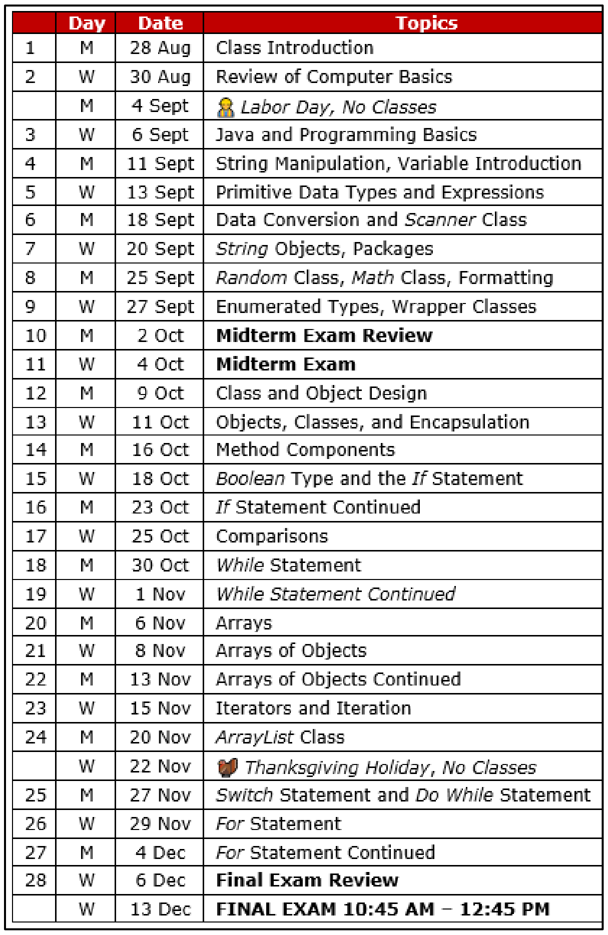
In the fall of 2017 I joined the management information systems (MIS) faculty at a private undergraduate school in the southwestern United States with an enrollment of around 11000 students, 79% of whom demonstrate a need for financial assistance, and many of whom are first-generation college students. Other faculty had expressed to me that, in addition to the challenges commonly faced by undergraduates, many of these students were under the impression that they did not "belong" in college. The self-held belief of first-generation college students and students from underrepresented groups in college that they "don't belong" is a common phenomenon in higher education (Foltz, Gannon, & Kirschmann, 2014).

Determined to serve the students at the university to the best of my ability, I took inventory of my teaching philosophy. It was my hope that with the right philosophy I could help students change negative self-perceptions into self-confidence and willingness to tackle challenging technical material.

The basic tenet of the teaching philosophy I had cultivated stated that given enough time, opportunity, and motivation, any student could learn anything. This optimistic philosophy was developed through my own experiences as a student, teaching assistant, and instructor. It was with this philosophy and familiarity with traditional approaches to teaching computer programming that I designed a Java course for MIS students.

Course Design

The Java programming course was initially designed in the style of an undergraduate object-oriented computer science course.



	Day	Date	Topics
1	M	28 Aug	Class Introduction
2	W	30 Aug	Review of Computer Basics
	M	4 Sept	*Labor Day, No Classes*
3	W	6 Sept	Java and Programming Basics
4	M	11 Sept	String Manipulation, Variable Introduction
5	W	13 Sept	Primitive Data Types and Expressions
6	M	18 Sept	Data Conversion and *Scanner* Class
7	W	20 Sept	*String* Objects, Packages
8	M	25 Sept	*Random* Class, *Math* Class, Formatting
9	W	27 Sept	Enumerated Types, Wrapper Classes
10	M	2 Oct	**Midterm Exam Review**
11	W	4 Oct	**Midterm Exam**
12	M	9 Oct	Class and Object Design
13	W	11 Oct	Objects, Classes, and Encapsulation
14	M	16 Oct	Method Components
15	W	18 Oct	*Boolean* Type and the *If* Statement
16	M	23 Oct	*If* Statement Continued
17	W	25 Oct	Comparisons
18	M	30 Oct	*While* Statement
19	W	1 Nov	*While Statement Continued*
20	M	6 Nov	Arrays
21	W	8 Nov	Arrays of Objects
22	M	13 Nov	Arrays of Objects Continued
23	W	15 Nov	Iterators and Iteration
24	M	20 Nov	*ArrayList* Class
	W	22 Nov	*Thanksgiving Holiday, No Classes*
25	M	27 Nov	*Switch* Statement and *Do While* Statement
26	W	29 Nov	*For* Statement
27	M	4 Dec	*For* Statement Continued
28	W	6 Dec	**Final Exam Review**
	W	13 Dec	**FINAL EXAM 10:45 AM – 12:45 PM**

Figure 1: Pre-Midterm Java Course Syllabus

Java Software Solutions 9th edition by Lewis and Loftus was used as the course textbook.
The course syllabus provided at the beginning of the fall semester is shown in Figure 1. As shown, the first two classes described course expectations and provided a review of computer basics. By the end of the first two weeks, the students were required to install Eclipse or a similar editor and the latest version of the Java Development Kit (JDK) on their personal laptops for use in the third week of classes.

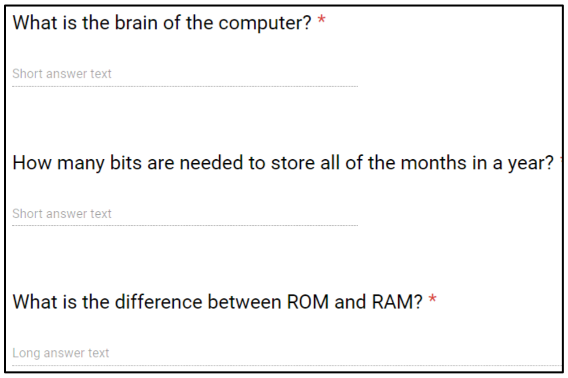
In the classes that followed up to the midterm exam, the students were introduced to the basics of Java programming, from writing print statements to using built-in packages. After the midterm exam, the students were to learn how to write their own classes, objects, and methods so that they could be used when discussing the basic control and data structures.

The course assessments, shown in Table 1, included two exams each worth 30%, programming assignments worth 29%, and participation and attendance worth 11% of the students' grades.

Assessment	Percentage
Attendance & Participation	11%
Programming Assignments	29%
Midterm Exam	30%
Final Exam	30%
Total	100%

Table 1: Java Course Assessments

Students were given the opportunity to earn attendance and participation credit by responding to three to five reflection and short answer prompts delivered via an online form. Every class except the day reserved for the midterm exam was accompanied by post-class questions that were graded on effort and completion rather than correctness. An example of post-class questions is shown in Figure 2.


What is the brain of the computer? *

Short answer text

How many bits are needed to store all of the months in a year?

Short answer text

What is the difference between ROM and RAM? *

Long answer text

Figure 2: Post-Class Questions

Course Delivery

The few classes that did not include programming were delivered primarily in a lecture format. Lesson plans for subsequent classes followed the format shown in Table 2.

Item	Minutes
Administration	5
Solutions to post-class questions	5
Previous topic(s) review	10
2-3 new topic demonstrations	30
Student practice	10
Solution to student practice	10
Post-class questions	5

Table 2: Java Pre-Midterm Lesson Plan

The first part of each class was spent discussing course administration and reviewing the topics from the previous class. After the review, I introduced two to three new programming concepts—first through lecture and then through demonstrations in Eclipse. The students then had time to practice a programming problem on their own before being presented with a solution. At the end of class, students had the opportunity to complete the day's post-class questions related to the new topics.

Discussion

There were aspects of the course design and delivery that were both effective and ineffective at promoting student learning. Post-class reflections are a tool I have used in many courses with positive feedback from students. An updated version of the "index card method," they are a quick way to measure students' understanding of the topics presented in class. Students appreciate the time at the end of class provided to complete the questions, that they count towards their overall grade, and are graded based on completeness rather than correctness. When gathered together, they are also a useful study guide for exams.

The main weakness of the course delivery at this point in the semester was the pattern by which new topics were introduced and practiced. New topics were shown on presentation slides and then demonstrated in Eclipse, where students were encouraged to follow along. I then presented a similar programming problem for students to work out on their own. What followed was a ten-minute period in which most students stared at blank editors on their laptops while I walked around the classroom to observe their progress (or lack thereof).

Formative feedback is one of the most powerful tools we have as instructors to facilitate student learning (Shute, 2008). Where the post-class reflections were effective at providing feedback, the time allowed for students to solve a problem on their own was not—or at least did not seem to be. The blank laptop screens (one or two with open social media instead of a programming editor) left me with the same uneasy feeling and questions after every class: 1) Why don't students seem to understand the new material after it is presented? 2) Are they afraid to make mistakes when solving the problem on their own? 3) Are they simply waiting for me to return to the front of the classroom to provide the solution?

Even when fewer new topics were presented as we approached the midterm exam, what I had hoped would be a daily opportunity for me to observe student programming techniques and provide feedback remained what seemed to be a pause in instructor activity before students were given a solution.

Towards a Growth Mindset

Discussions with students outside of class indicated that the limited amount of practice time they had, as well as the method by which new topics were introduced made them afraid to solve problems on their own and less motivated to learn how to program. What was needed was a change in class delivery that would improve students' attitudes towards their ability to solve programming problems. A Growth Mindset approach to course redesign would provide a problem-driven environment in which students would build confidence as they wrote their own solutions collaboratively and received feedback.

4. JAVA PROGRAMMING: MIDTERM EXAM AND COURSE REDESIGN

A former high school teacher of mine once snarkily remarked that he hoped that most of the learning in his course occurred *before* exams, but that occasionally we find ourselves learning the most after they are graded and returned. Such was my experience as an instructor after administering the midterm exam in the Java course and attempting to grade it. The exam was like those I was given as an undergraduate student in programming courses—some multiple-choice questions, short answer questions, and a larger programming problem to solve on paper. Because cheating was reported to be an issue with computer-based exams in the program, I opted for a paper-based exam despite an understanding that writing code on paper is cognitively much different than writing it on the computer.

The students' exam submissions validated the uneasy feelings I had in class watching them during what was meant to be practice time. On average, they missed more than half of the multiple-choice questions, provided scattered solutions to the short answer problems, and—most significantly—left the larger programming problem almost completely blank. I recall flipping through exam pages to find mostly empty space and leaving the pile an ungraded and unsolved pedagogical puzzle on the living room sofa.

The exams and my difficulty grading them were a clear call for course redesign. As I explained to the students, we could either ignore the exam results and continue working through the course content as indicated in the syllabus, or we could try something different. After careful consideration, I offered the students the opportunity to complete the programming portion of the exam as homework, redesigned the syllabus, and reformulated a pattern for class lesson plans.

Adapted Course Design

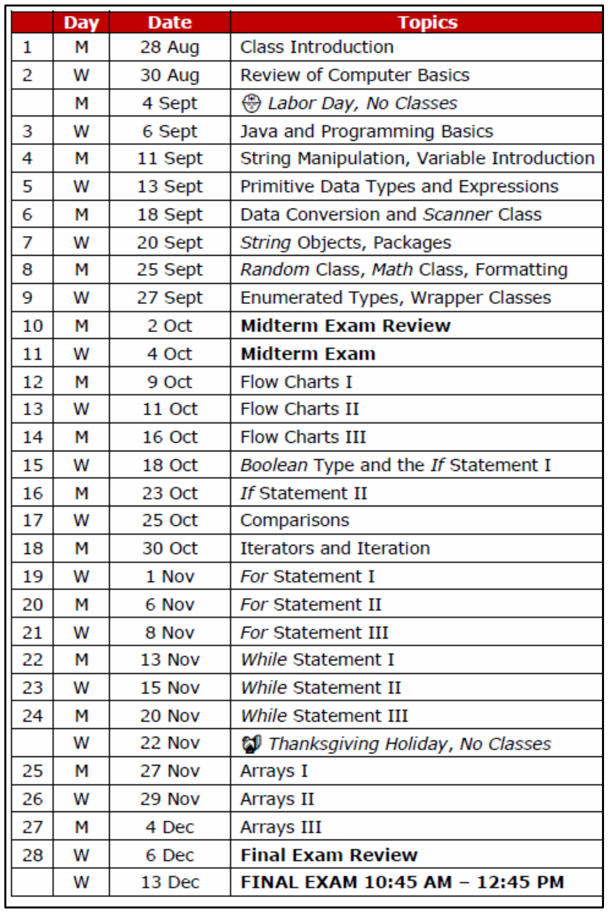
Most of the students in the course could not solve the programming problems presented to them or recall parts of the Java language in isolation. It was as though they were asked to hold a conversation in a foreign language without being able to remember any of the vocabulary. In addition, they seemed unable to separate the logic needed to solve the problems from the language itself.

It was my goal that by the end of the course the students be able to solve programming problems presented to them using the Java language. With this goal in mind and an awareness that the students still perceived programming as alien, I redesigned the syllabus. The post-midterm syllabus shown in Figure 3 had object-oriented concepts removed and additional time devoted to each of the basic programming control structures.

After debriefing the students on the midterm exam and its implications for the rest of the course, we learned how to represent the solution to problems using logical flow charts. Examples of flow charts used in class appear in Appendix A.

Adapted Course Delivery

The revised lesson plan format is shown in Table 3. After discussing solutions to the post-class questions and reviewing the programming problem from the previous class, students were given paper copies of one or two flow charts that illustrated the logical flow of solutions to problems.



	Day	Date	Topics
1	M	28 Aug	Class Introduction
2	W	30 Aug	Review of Computer Basics
	M	4 Sept	*Labor Day, No Classes*
3	W	6 Sept	Java and Programming Basics
4	M	11 Sept	String Manipulation, Variable Introduction
5	W	13 Sept	Primitive Data Types and Expressions
6	M	18 Sept	Data Conversion and *Scanner* Class
7	W	20 Sept	*String* Objects, Packages
8	M	25 Sept	*Random* Class, *Math* Class, Formatting
9	W	27 Sept	Enumerated Types, Wrapper Classes
10	M	2 Oct	**Midterm Exam Review**
11	W	4 Oct	**Midterm Exam**
12	M	9 Oct	Flow Charts I
13	W	11 Oct	Flow Charts II
14	M	16 Oct	Flow Charts III
15	W	18 Oct	*Boolean* Type and the *If* Statement I
16	M	23 Oct	*If* Statement II
17	W	25 Oct	Comparisons
18	M	30 Oct	Iterators and Iteration
19	W	1 Nov	*For* Statement I
20	M	6 Nov	*For* Statement II
21	W	8 Nov	*For* Statement III
22	M	13 Nov	*While* Statement I
23	W	15 Nov	*While* Statement II
24	M	20 Nov	*While* Statement III
	W	22 Nov	*Thanksgiving Holiday, No Classes*
25	M	27 Nov	Arrays I
26	W	29 Nov	Arrays II
27	M	4 Dec	Arrays III
28	W	6 Dec	**Final Exam Review**
	W	13 Dec	**FINAL EXAM 10:45 AM – 12:45 PM**

Figure 3: Post-Midterm Java Syllabus

Before any code was written, we discussed the logic shown in the flow charts. What followed was a demonstration of a programming solution to one of the flow charts and time for students to devise solutions on their own or in pairs. I observed the students' work and offered feedback before returning to the instructor's station to demonstrate a programming solution to the problem.

Item	Minutes

Administration	5
Solutions to post-class questions	5
Discussion of previous problem	10
New problem and flow chart	15
Student practice time	15
Solution demonstration	10
Lab time for homework	10
Post-class questions	5

Table 3: Java Post-Midterm Lesson Plan

The remainder of the class time was reserved for students to work on the current homework assignment and complete the post-class questions. During this time students often asked questions and received feedback. This pattern of instruction continued for the rest of the semester.

Discussion
With these adaptations in course design and delivery came a change in the classroom atmosphere evident in the practice and homework time provided to the students. Blank editors were replaced with attempts at solving programming problems that facilitated feedback and a better understanding of course concepts.

The introduction of flow charts let us study the logical flow of programming control structures separately from the language itself. This approach has been adopted with success in computer science classrooms (Myers, 1986). More importantly, however, it enabled a more problem-driven approach to the delivery of each subsequent lesson.

A problem-driven approach to technical topics is essential for fostering a Growth Mindset. This approach was augmented by the additional time provided to students to work on problems and homework in class. These features of the Growth Mindset approach, including a focus on exploring concepts through problem solving, collaboration, and time to work through programming mistakes led to a change in student motivation in the class evidenced by the effort they showed on subsequent homework assignments and the final exam.

The atmosphere of the class had evolved from one of fear of unfamiliar technical topics to one of motivated curiosity. Students expressed that while programming was difficult that it seemed useful to them and they liked that they "were actually doing something" in class. One student described the joy of "hitting the button in Eclipse" and seeing successful results that spurred her onward. As further evidence of the success of the course redesign, several students that were not graduating in the fall semester asked me if I would be teaching another programming course in the spring because they wanted to continue to learn.

5. PYTHON PROGRAMMING: A GROWTH MINDSET APPROACH

Many of the Growth Mindset characteristics implemented in the Java course resulted in drastic improvement of student motivation and performance. When considering the design of a Python course for the spring semester, I reflected on the effectiveness of applying Growth Mindset techniques and wanted to further extend their application.

Course Design
One of the key characteristics of a Growth Mindset approach that had not been fully realized in the Java class was that of developing a positive attitude towards failure. Students were given the opportunity to practice making mistakes in longer practice periods, but the effectiveness of these experiences had not been maximized. The Python course was thus designed with a focus on allowing for time to mistakes, debugging, and developing positive attitudes toward student failure.

The Growth Mindset theory informed this design such that class time would be taken to directly address the debugging process. As an instructor, I wrote specific incorrect programming snippets so that students could develop a repertoire of programming issues. In doing so, I was able to model a positive attitude toward failure and reframe students' programming errors as part of the learning process, rather than insurmountable roadblocks.

The Python course was thus designed as a problem-driven course in which students developed a positive attitude towards making mistakes and, as a result, gained further motivation for learning. The course as a whole was presented to the students as an opportunity to solve data analytics problems using a programming language. The end goal of the class was to be able to perform preliminary analysis on data obtained from Twitter using Python.

Course Delivery
The delivery of each class lesson followed a similar approach to that of the redesigned Java course with some important modification. An example lesson plan is shown in Table 4.

Item	Minutes
Administration	5

Solutions to post-class questions	5
Discussion of previous problem	10
New problem with errors	15
Student practice time	15
Solution demonstration	10
Lab time for homework	10
Post-class questions	5

Table 4: Python Lesson Plan

A problem-driven approach to each class was taken but supplemented by the deliberate inclusion and discussion of errors in the preliminary solutions presented. I followed the same pattern for each new problem presented by discussing different solutions, making mistakes while programming solutions, and finally arriving at a feasible solution. Mistakes were made part of the process in an organic way such that I was able to react to the errors in the Python terminal so that students could not only see solutions, but the process involved in finding solutions. I took special care to react to errors as though they were additional puzzles to be solved, rather than failures that reflected on my competence as a programmer.

Discussion

The focus on modeling an attitude towards making mistakes further improved student motivation and performance in the Python course, as evidenced by the experiences I had during the student practice time. As I walked around the classroom to observe student progress, I saw effort from every student to solve the problems presented, even from those that had been the least confident in the Java course. The students' efforts allowed me to create a mental catalog of student approaches to the programming problems that I shared when returning to the front of the classroom. Demonstrating the students' different approaches to their peers reemphasized our focus on concepts over the memorization of programming facts.

Further evidence of an improvement in motivation and confidence was demonstrated by the increase in frequency of student questions and comments in class. One student that had frequently expressed anxiety to me in office hours about the course material confidently responded to a demonstration of a Python concept in class that she had found a different method to solve the problem that made more sense to her. Her resourcefulness and her willingness to share a different approach in front of the class indicated to me that she had experienced tremendous growth in confidence and motivation as a student. Another student that had struggled with the material in the Java course was observed spending ten to fifteen minutes after class debugging and asking additional questions.

6. RESULTS AND DISCUSSION

The Java and Python courses and their accompanying course design, redesign, and analysis were an informative exercise in reflective teaching practice that explored the effectiveness of applying Growth Mindset concepts to computer programming courses and technical material, in general. It would be difficult to compare student performance on course assessments in a Java and Python course because of the differences in material. However, the increase in student motivation and confidence that occurred after the midterm in the Java course and throughout the Python course was evident in the frequency and depth of student participation and motivated curiosity in both courses. Students that had little to no prior programming experience transformed from timid students with negative self-perceptions about their technical abilities into more confident problem solvers that had developed a willingness to try and make mistakes while learning difficult material.

These experiences are not presented to say that applying Growth Mindset techniques will guarantee automatic success and increased performance in students that lack confidence with technical courses. They are, however, evidence that adopting such an approach can provide a framework for students to build confidence and increase motivation and effort when confronted with more challenging material.

It would be prudent to point out that this paper is an account and recollection of course curricular adjustments and design as they relate to meeting the challenges of under-motivated and under-performing students, in situ. As such, this account constitutes a post hoc account rather than acute a priori design. While we are convinced that, for a student population with not simply a "normal distribution," but rather one with an appreciable component of students with incomplete study habits, there is an urgent imperative to adopt the growth mindset to establish the conditions that might lead to success. A follow-on to our experiences here would be a rigorous experimental design which might establish more than the notional and perhaps conjectural ascription of effects to the Growth Mindset in the case report. We do not demure on the improvements we observed, but

ISSN: 2473-3857
v4 n4641

we also do not control for other conflating influences which may have also been present.

7. CONCLUSION

In this paper, we reviewed the progressive redesign of a course as a response to an occasion warranting pedagogical attention. Given challenges such as low motivation, deficient study habits and orientation, and a lackluster response to straight-forward pedagogy, the Java class described provided the impetus to explore the lessons of the Growth Mindset. Many of those who influence the decision to embrace instruction and courses leading to efficacy and competency in STEM will often have an intellectual orientation to the perceived benefits of garnering these competencies but lack the background in experience to orient their mindset and habits to comprehension, growth, and excellence.

The changes in course design that most affected student motivation and performance were the implementation of a problem-based approach and the emphasis on the proper approach to errors and failure. Positioning programming as a problem-solving activity incited student curiosity and alleviated the fear that many of them associated with using the computer as a programming tool. Modeling an appropriate response to failure and errors as part of the programming process also encouraged students to work through solutions, rather than give up when something went wrong.

The challenges described, ameliorations attempted, and overall improvements we can ascribe to the Growth Mindset are offered as a case demonstrating both the progress of students and of the reflective realizations arising in the educator. We propose that the Growth Mindset holds equal promise for students and educators alike in maintaining a reflective practice designed to track the process of growth and habit of growth apart from any magnitudinal benchmarking.

This is not to say that a finite set of material should be accomplished within the timebox of a term. What we do observe is that a Growth Mindset allows the student and instructor to put "first things first" such that the processes for comprehension and improvement area paramount and the "quantity" of accomplishment would concomitantly follow.

We encourage colleagues to explore the Growth Mindset further for themselves, try some of it out, and perhaps report back in their own studies and cases. Far too many students are grasping for the promise of STEM and falling short when their habits are not matched to the rigor of engagement and perseverance required for excellence. We have found early promise in the principles of the Growth Mindset.

8. ACKNOWLEDGEMENTS

The authors would like to acknowledge the students of the Java and Python courses that provided post-course reflections.

9. REFERENCES

Bell, D. (2016). The reality of STEM education, design and technology teachers' perceptions: A phenomenographic study. International Journal of Technology and Design Education, 26(1), 61-79.

Boaler, J. (2016). Mathematical Mindsets. Jossey-Bass, San Francisco, CA.

Brown, R., Brown, J., Reardon, K., & Merrill, C. (2011). Understanding STEM: current perceptions. Technology and Engineering Teacher, 70(6), 5.

Breiner, J. M., Harkness, S. S., Johnson, C. C., & Koehler, C. M. (2012). What is STEM? A discussion about conceptions of STEM in education and partnerships. School Science and Mathematics, 112(1), 3-11.

Charette, R. N. (2013). The STEM crisis is a myth. IEEE Spectrum, 50(9), 44-59.

Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: A literature review. Journal of Educational Computing Research, 1(3), 253-274.

Dweck, C. S. (1986). Motivational Processes Affecting Learning. *American Psychologist*, 41(10, 1040-1048.

Dweck, C. S. (2008). Mindset: The new psychology of success. Random House Digital, Inc.

Foltz, L. G., Gannon, S., & Kirshmann, S. L. (2014). Factors that Contribute to the Persistence of Minority Students in STEM Fields. *Planning for Higher Education Journal*, 42(4), 1-13.

ISSN: 2473-3857
v4 n4641

Gupta, A. K., & Govindarajan, V. (2002). Cultivating a global mindset. Academy of Management Perspectives, 16(1), 116-126.

Jenkins, T. (2002, August). On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences (Vol. 4, No. 2002, pp. 53-58).

Maurer, T. J., Wrenn, K. A., Pierce, H. R., Tross, S. A., & Collins, W. C. (2003). Beliefs about 'improvability' of career-relevant skills: relevance to job/task analysis, competency modelling, and learning orientation. Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior, 24(1), 107-131.

McGrath, R. G., Mac Grath, R. G., & MacMillan, I. C. (2000). The entrepreneurial mindset: Strategies for continuously creating opportunity in an age of uncertainty (Vol. 284). Harvard Business Press.

Murphy, L., & Thomas, L. (2008). Dangers of a fixed mindset: implications of self-theories research for computer science education. In ACM SIGCSE Bulletin (Vol. 40, No. 3, pp. 271-275). ACM.

Myers, B.A. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 59-66.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. New ideas in psychology, 2(2), 137-168.

Schultze, U. (2000). A confessional account of an ethnography about knowledge work. *MIS quarterly*, 3-41.

Schute, V. J. (2008). Focus on Formative Feedback. *Review of Educational Research*, 78(1), 153-189.

Silva, E. & White, T. (2013). Pathways to Improvement: Using Psychological Strategies to Help College Students Master Developmental Math. Carnegie Foundation for the Advancement of Teaching Report Stanford, California.

Spencer, L. M., & Spencer, S. M. (1993). Competency at work. New York: John Wiely & Sons, 5.

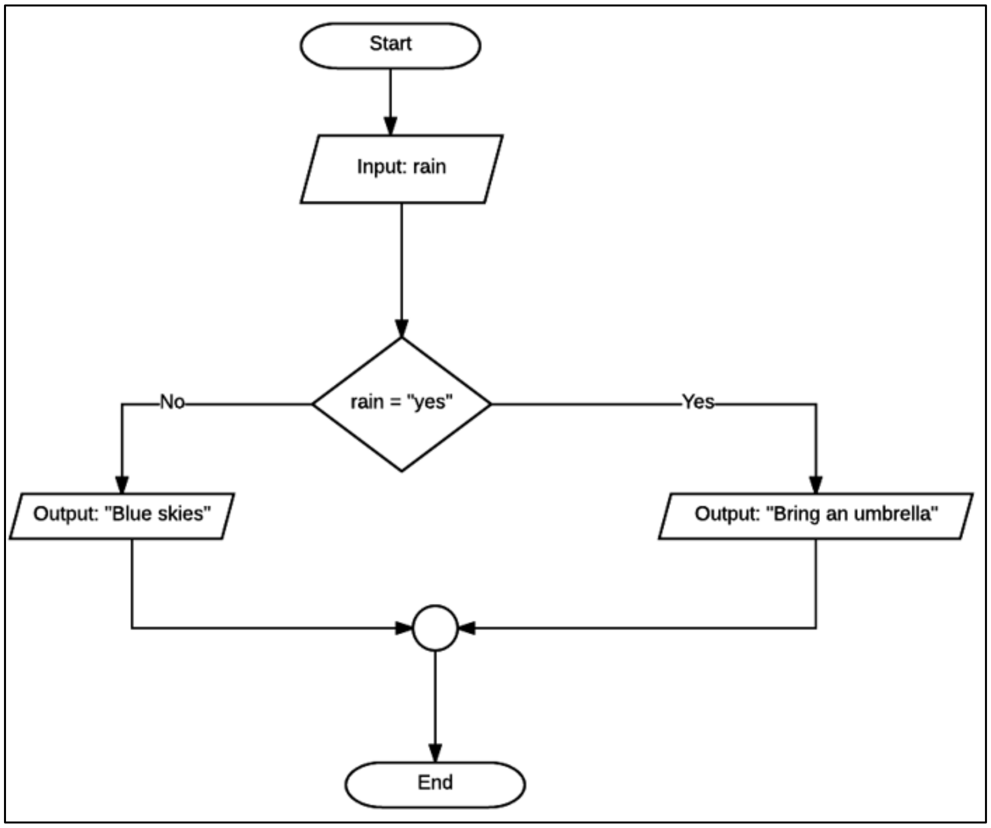
Terenzini, P. T., Springer, L., Yaeger, P. M., Pascarella, E. T., & Nora, A. (1996). First-generation college students: Characteristics, experiences, and cognitive development. Research in Higher education, 37(1), 1-22.

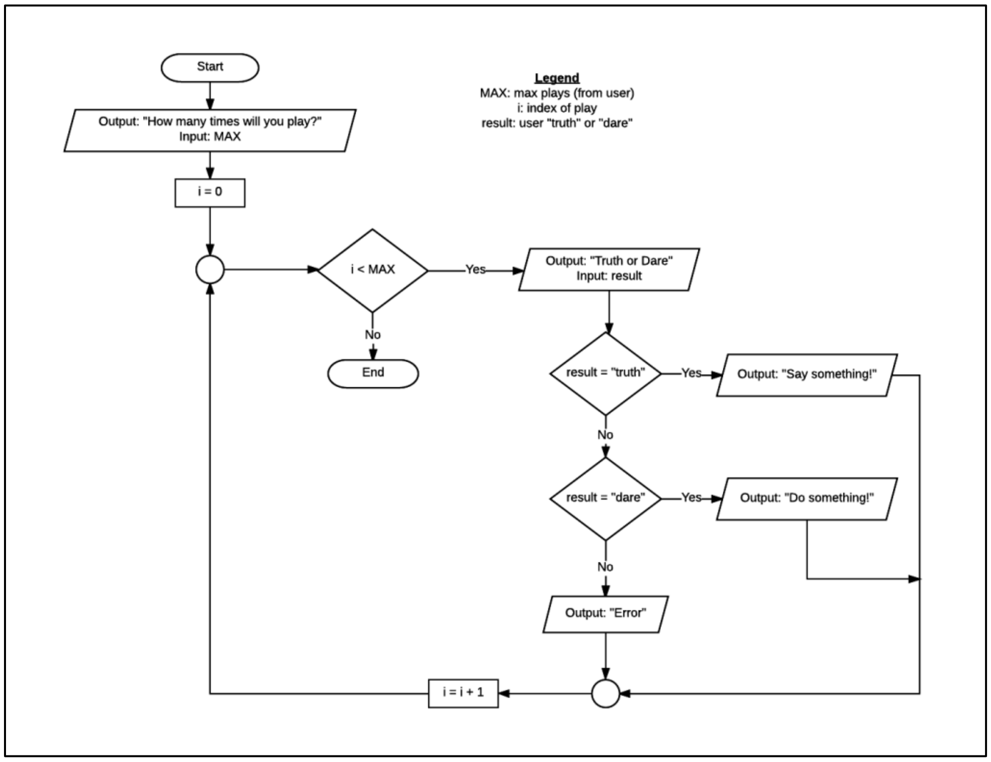
Trotter, A., & Ellison, L. (1997). Understanding competence and competency. School Leadership for the 21st Century.–London: Routledge, 36-53.

Weick, K. E. (1998). Introductory essay—Improvisation as a mindset for organizational analysis. Organization science, 9(5), 543-555.

ISSN: 2473-3857
v4 n4641

Appendix A: Problem Flow Charts


Start
Input: rain
rain = "yes"
No
Yes
Output: "Blue skies"
Output: "Bring an umbrella"
End


Start
Legend
MAX: max plays (from user)
i: index of play
result: user "truth" or "dare"
Output: "How many times will you play?"
Input: MAX
i = 0
i < MAX
Yes
No
End
Output: "Truth or Dare"
Input: result
result = "truth"
Yes
Output: "Say something!"
No
result = "dare"
Yes
Output: "Do something!"
No
Output: "Error"
i = i + 1