Instrumental Genesis: For Loop and 'len()' as Artifacts

Antonio Estevan Martinez IV
California State University, Long Beach

In this contributed report I present results from a teaching experiment broadly focused on students utilizing machine-based computing as a mediating tool to learn fundamental concepts related to set theory and logic. Specifically, I highlight the mathematical activity of one student and his unique approach to finding the intersection of three sets. To understand how this student leveraged the computer programming environment, I used the analytical framework known as instrumental genesis, which can be utilized to investigate the confluence of an artifact (often a piece of technology) and the human mind to solve a mathematical problem. Results indicate that the student was able to use computational tools such as For Loops and the len(), or "length," function as artifacts in his solution to finding the set intersection.

Keywords: Computing, Programming, Set Theory, Instrumental Genesis

Introduction

In this report I present the results of an investigation into student thinking in which students were tasked with using Python, the text-based computer programming language, to learn fundamental concepts related to set theory and logic. The call for research to investigate the ways in which technology and, more specifically, computers can be used to facilitate the teaching and learning of mathematics is not new (e.g., Fey, 1989; Papert, 1980; Perlis, 1962). However, previous approaches to address this need have varied in the medium through which the technology is used, such as video games (Levine et al., 2020), graphing calculators (Leng, 2011) and virtual reality (Bogusevschi et al., 2020). Also, these areas of technology integration have been a focus at multiple levels throughout the educational system (Ball et al., 2018; Oates, 2011). Moreover, Lockwood and Mørken (2021) explicitly made a call for research in undergraduate mathematics education with a focus on machine-based computing. They define machine-based computing as "the practice of precisely articulating algorithms that may be run on a machine" (p. 2). An important aspect of machine-based computing is the development of an algorithm rather than its performance or implementation alone. This is an important component of machine-based computing that distinguishes it from other forms of computing such as the use of Desmos where one may only be required to input parameters to create a visual representation of a function. Thus, Lockwood and Mørken consider machine-based computing to be a separate construct that includes the use of programmable calculators, writing packages in Geogebra, and using text-based or block-based programming languages. As such, the focus of my study was to introduce machine-based computing (Python) as a mediating tool to work with mathematical ideas in the context of mathematical logic and set theory.

Additionally, I emphasize a need to focus on machine-based computing in the context of mathematics for three main reasons. The first is that data science, and computer science more broadly, are becoming increasingly valuable skills in an age where fields such as artificial intelligence and machine learning are in position to be the most influential drivers of change for our society (World Economic Forum, 2016). The second purpose draws on Lockwood et al.'s (2019) commentary on the relationship between computing and students' mathematical activity in that computing will be a focus for mathematics educators soon to come. By encouraging the use of computational software as a tool for students to learn and connect ideas in undergraduate mathematics, I am encouraging a movement for where mathematics education will be going in the coming decade. Lastly, many areas of mathematics lend themselves to machine-based

computing environments, which suggests that the computational environments may serve as useful tools for learning the mathematical content. For this study, I focus on a fine-grain analysis of how programming may influence student reasoning and learning of mathematical set theory and logic. The research question driving this study is the following: *How does Python mediate students' reasoning of important concepts related to set theory and logic?* In the next section I describe a theoretical analysis framework which can be used to document students' mathematical activity while using a piece of technology.

Theoretical Framework

As alluded to in the introduction, programming is starting to become a fundamental aspect of learning undergraduate mathematics. For example, Buteau et al. (2020) investigates how an undergraduate student may use programming to learn both theoretical and applied mathematics. Rather than investigating student work with a particular mathematical concept, the authors articulated the use of a specific theoretical framework to study the use of programming as a tool for learning mathematics. The framework is known as the *instrumental approach* (Artigue, 2002; Guin & Trouche, 1998; Trouche, 2004). Importantly, Buteau et al. (2020) highlight that the instrumental approach has been used in the past for various artifacts such as graphing calculators, spreadsheets, applets, etc. but has not yet been used in the case where a programming language was considered as the artifact. They do provide illustrative examples of the relationship between mathematical inquiry and programming through the lens of an instrumental approach, but they do not provide an in-depth analysis of one specific mathematical concept. Therefore, to investigate how Python supports students' learning and advancing mathematical activity related to set theory and logic, I utilize the instrumental approach.

The description of the instrumental framework starts with the distinction between *artifacts* and *instruments*. An artifact may be a physical object, such as a graphing calculator, but also may be a formula, graph, or other objects that are central to a certain mathematical task (Roorda et al., 2016). Once the artifact has been determined, the integration of the artifact into a learner's mathematical activity is known as the instrument. This includes the use of the artifact to problem solve and as Trouche (2004) describes it, "an instrument can be considered as an extension of the body, a functional organ made up of an artifact component...and a psychological component" (p. 285). In the context of this study, the artifact is the computer programming environment, Python. As mentioned, programming environments have previously been utilized with this framework in mathematics education, but never with a focus on students' advancing mathematical activity in relation to set theory and logic.

The development of the instrument is known as *instrumental genesis* (Artigue, 2002) and is tied directly to the artifact and to the mental actions of the learner in their use of the artifact to carry out a given task. This means that the learner may be afforded particular lines of reasoning but also may be constrained in their mental activity given the particular features of the artifact itself. This psychological component, the mental processes of the learner to carry out a particular task, is referred to as an *instrumented action scheme* (Trouche, 2004). The idea of a student's scheme draws on the theory of constructivism and is described by Vergnaud (2009) as "the invariant organization of activity for a certain class of situations" (p. 88). To clarify, this definition encompasses the assimilation of familiar situations which learners respond to with their learned rules or already-established ways of understanding as well as addresses the adjustments necessary to address novel situations in which a learner is required to adapt, modify or reorganize their psychological thought processes. In the literature regarding the instrumental

approach, schemes are defined as consisting of four main features (Trouche, 2004; Vergnaud, 2009). These features are summarized well by Buteau et al. (2020) as:

1. The goal of the activity, with sub-goals and expectations.
2. Rules of action: stable behaviors of the subject.
3. Operational invariants, which can be theorems-in-action (propositions considered as true) or concepts-in-action (concepts considered as relevant).
4. Possibilities of inferences. These possibilities are essential for the adaptation of the scheme to the specific features of the situation. (p. 1026)

As Buteau et al. (2020) state, the rules of action and operational invariants may present themselves through students' mathematical activity in a situation where a student says "When I want to do this [aim of the activity] … I always act like this [rule of action] … because I think that [operational invariant]" (p. 1027). The rules of action are stable methods of activity that the student will rely on to accomplish a task, based on the belief or conceptual understanding (operational invariant) of how something works. The operational invariants are broken down into two categories, theorems-in-action and concepts-in-action. The theorems-in-action are constructed ways of understanding how something works. The concepts-in-action are the related mathematical concepts that are pertinent to the goal of the activity. For example, if a student was asked to use a graphing calculator to find the local max and min of a function given a certain domain, an example of a rule of action would be 'enter the function in the calculator'. A concept-in-action might be 'zero slope at local min and max,' and a theorem-in-action might be 'the window display of the graphing calculator must capture the specific domain that is asked in the problem in order to be sure of the local max and min.' Lastly, the possibilities of inference would be situations in which the student encounters a problem that might result in new rules of action and operational invariants. In accordance with constructivism, technically these schemes are constructed within the learner's mind and thus are not directly observable, however, they can be inferred by an instructor or researcher based on the "regularities and patterns in students' activities" (Drijvers et al., 2013, p. 27). That is, for the purposes of this study, a "scheme" is a model that I construct based on the actions of the student.

As Roorda et al. (2016) highlight, at the core of the instrumental approach is the relationship between one's scheme and their *technique*, or actions in response to a given situation. In contrast to a student's scheme, the technique consists of the observable actions that are carried out by the student. It is important to highlight that while the observable actions inform the researcher's interpretation of the learner's scheme, their actions may also contradict the scheme that was originally developed based on previous actions. So, the student's techniques are in-the-moment observable actions of the student that may or may not coincide with the student's scheme. The student's techniques are linked with conceptual elements that help frame an understanding for the student's scheme. To capture the idea of instrumental genesis in more detail, it is broken into two subcomponents, instrumentalization and instrumentation, each describing a distinct directional relationship between the artifact and the learner (Artigue, 2002; Trouche, 2004). *Instrumentalization* is the reshaping of the artifact in the learner's mind as to what the artifact can do to accomplish a certain task. Instrumentalization occurs through the use of the artifact and develops over time as the learner understands the functionality of the artifact and its capabilities. This includes the use of an artifact by the learner in unexpected or unintended ways. An example of this could include a learner storing mathematical results and theorems in their graphing calculator as a memory aid for future use. *Instrumentation* works in the opposite direction in which the artifact itself, with its built-in constraints and potential uses, is shaping the ways in

which the learner conducts their mathematical activity. In Table 1 I provide a hypothetical example of what a student's subset scheme might look like from an instrumental genesis perspective using the framework provided by Roorda et al. (2016).

Table 1: Hypothetical Example of a Student's Subset Scheme

Instrumentation Scheme	Techniques	Conceptual Elements	Technical Elements
Subset Scheme	Draw a Venn Diagram	To determine if a set is a subset, one must conclude that every element of the set is an element of the other set	Write a For Loop to check that each element is an element of the larger set

Roorda et al. (2016) provide examples of how an artifact may shape a learner's mathematical activity in their description of a student named Andy's schemes regarding the concept of the derivative. One specific example is Andy's use of the dy/dx-option on the graphing calculator to calculate the steepness of a graph at a particular point. Andy used this option in other examples throughout the longitudinal study when they needed to find instantaneous rate of change and did not coordinate between the dy/dx-option on the calculator and the symbolic representations that were covered in class. The authors conclude that while Andy's graphical and numerical representations of the derivative were strong, these representations were being developed independent of Andy's symbolic representations, a negative consequence of Andy's overreliance and strong use of the graphing calculator. Trouche (2004) comments that both instrumentalization and instrumentation work together in tandem as part of the instrumental genesis process, and thus refers to a learner's scheme as the instrumented action scheme as opposed to other terms such as *instrumentation scheme* as used by Roorda et al. (2016). For the purposes of this manuscript, I will adhere to Trouche's approach to the development of a learner's scheme (as perceived by the researcher) and refer to it as an instrumented action scheme or just scheme for short.

Methods

Participants were recruited from a four-year Hispanic-Serving Institution and were purposefully selected (Patton, 1990) in that they had already taken, or are currently enrolled in differential or integral calculus and not enrolled in an Introduction to Proofs course. Thus, the students in this study were not familiar with mathematical logic or set theory. Additionally, the students had little-to-no prior experience programming. The design of the study was a conjecture-driven teaching experiment (Confrey & Lachance, 2000) in which the conjecture was the following: *programming can serve as an experientially real context in which students will be able to connect mathematical logic and set theory.* There were 10 students that participated in the study divided into four groups with two to three students in each group. Due to the ongoing COVID-19 pandemic, the teaching experiment sessions were conducted and recorded via Zoom. For each session, I shared my screen which displayed two windows. The first window was a Jamboard[1] slide deck to present the tasks to the students, and to capture their ideas and diagrams.

[1] Jamboard is a Google interface that serves as a collaborative and interactive canvas.

In a window next to Jamboard, I had an Integrated Development Environment that was used to run Python code.

For analysis, I utilized Roorda et al.'s (2016) framework for identifying the techniques, conceptual elements, and technical elements as the basis for developing the students' instrumented action schemes. Recording the conceptual elements was done through the identification of the mathematical concepts that the students were learning. In analyzing the techniques used by the students (e.g., the output students produced, diagrams they drew, logical statements they considered), I watched the Zoom video recordings to document what exactly the students were producing in relation to conceptual elements. To document the technical elements (e.g., the code the students typed, the mathematics they wrote in Jamboard), I analyzed the Zoom video recordings to document what the students typed and when they typed it. This analysis was done by tagging the audio transcriptions using MAXQDA with Concepts, Techniques, Tech-Elements as the main codes for analysis. If a student typed something without saying out loud what they had typed, I used the 'memo' feature within MAXQDA to document what the student typed. This way I was able to analyze all three aspects of the students' instrumented action schemes within MAXQDA. By coding the techniques, conceptual elements and technical elements, I was then able to construct each students' instrumented action schemes. Importantly, the students' schemes were constructed with the four features as described by Trouche (2004) and Vergnaud (2009) which included the goals, rules, operational invariants and the possibilities for inference. Due to the page length constraints, I highlight the results for only one of the students on one problem within the larger teaching experiment. Additionally, the data presented here came from the fourth session out of five total sessions with this group of students.

Results

In this section I present the work of Alonso (pseudonym), a participant in Group 4 as they worked on a problem in which they were asked to "find the common elements across all three sets." The three sets, A, B, and C were given to the students in the study and contained both strings and integers. The students were also asked to "draw a diagram of what this set relationship might look like first before you write any code." I highlight Alonso's work due to the unique nature of his approach to solving this task. No other student in my study attempted to solve the task in the same way that Alonso did. First, I will briefly present some information that will be pertinent to understanding Alonso's solution method. First, repeated elements in a set, in Python and in mathematics in general, are not counted multiple times for the total number of elements that belong to the set. For example, if we define A = {1, 2, 3, 4, 5, 5}, the number of elements that belong to A, otherwise known as the *cardinality* of A, is five. Second, the cardinality of a set can be determined in Python using the 'len()' function. This function finds the length, or the number of items in the iterable data object. Also, it is important to note that before drawing the diagram, Alonso wanted to clarify what I meant by "diagram." I told him that I would "leave it up for interpretation." Figure 1 is a screenshot of Alonso's diagram, which he used as an opportunity to write pseudocode, a step-by-step process outlining an algorithm that would solve the problem. Of the ten students in the study, Alonso and his partner were the only two to write pseudocode. The other students either drew Venn diagrams, circled certain elements in the sets, or did some type of representation of what the intersection process might look like.

The first process in Alonso's pseudocode was to iterate through all the values, or elements, in A and add those elements to the set B and the set C. This step is fairly straightforward but requires a little interpretation as to what is meant by 'set B and C' in the second line. One could interpret this to mean the logical operator 'and,' because he refers to it as a 'set' instead of 'sets,'

and would thus possibly imply the intersection of the two sets B and C. However, in his description of the pseudocode, Alonso clarified this line when he said, "So, if you had set A, you would add it to both set B, and C. So that all the values from set A get cycled through the two sets." This confirms to me that Alonso was referring to two separate sets, set B and set C.

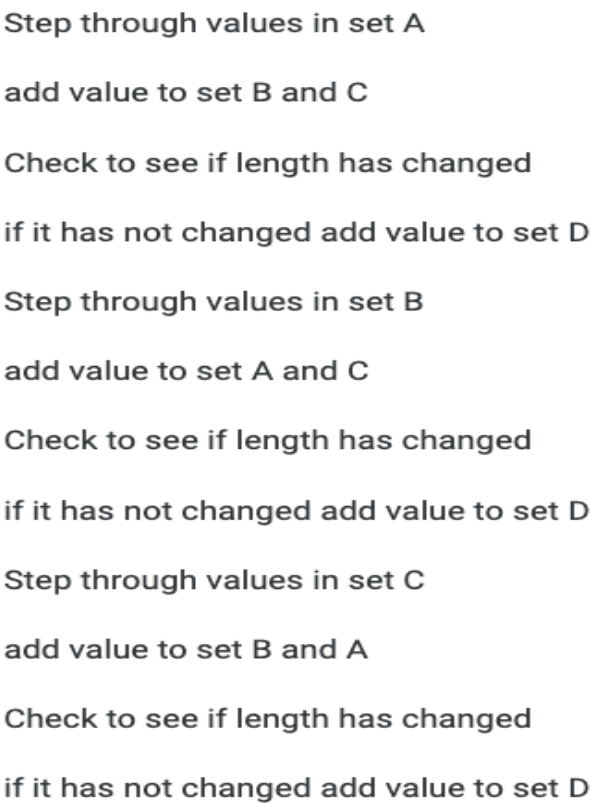

Figure 1. Alonso's Pseudocode for Set Intersection

The next step in Alonso's pseudocode was to check the cardinality of the two sets B and C. As Alonso described it, "With each iteration you check to see if the length has increased. If the length has not increased, then you know that there has been a repeat." I am interpreting this statement as a process of adding each element from A to both sets B and C. One then finds the cardinality of the sets B and C. If there is a change in the cardinality from before the element in A is added to the sets B and C, then one can determine that this element is not shared with the set. For example, let A = {1, 2, 3} and B = {4, 5, 6}. If we add the element 2 to B, then the cardinality of the set B would change from three to four since 2 is not an element of B. The next step in Alonso's pseudocode is to add the element to a new set, D, only if the cardinality does not change when added to sets B and C. An overview of the whole computational process is described by Alonso in the following way:

> This would require three For Loops and at the end you would just print the length of D and you would get the number of common elements. Or you could just print D to get which elements are in common.

The three For Loops that Alonso mentioned represent three iterative processes to add all the elements from each of the three sets to the other two remaining sets. Of course, three For Loops are not required. We did not go through the process of writing out Alonso's code as we were

running close to the end of time during the session and we still needed to hear from Alonso's partner about his diagram, but given that Alonso's method was so detailed, my hypothesis is that having the actual code would not change anything about his scheme or his conception of finding the intersection of multiple sets.

Due to Alonso's solution method, I refer to his scheme as the *Monitor Change in Cardinality Scheme.* As for his scheme, it is evident that the goal is to find the intersection of the three sets. I see two rules-of-action for Alonso. First, '*Construct an algorithm that would select each element in all sets'* and second, '*verify whether or not each element existed in each of the other sets*.' One theorem-in-action would be, '*Using a For Loop, one can iterate through every element of a defined set in Python.*' A second theorem-of-action is, '*One can determine the existence of common elements by monitoring any change in the cardinality of a set once an element has been added to the set.*' These theorems in action led Alonso to the solution method presented in his pseudocode. Given that this is just one problem from the larger teaching experiment, there is no identification of a possibility of inference for Alonso. A summary of Alonso's scheme can be found in Table 4.4.

Table 2: Alonso's Monitor Change in Cardinality Scheme

Instrumentation Scheme	Techniques	Conceptual Elements	Technical Elements
Determine Set Intersection by Monitoring Change in the Cardinality	Add an element to a set and calculate the cardinality of the set	The cardinality of a set changes when a new, non-repeated element is added	By writing a For Loop, one can pass through every element in a set. These elements are added to the other sets and the cardinalities of the other sets are calculated. If there is no change in the cardinality from before the element was added to after, for all sets, then the element is a repeated element and can be added to a new set.

Conclusion

Without Python, it is unlikely that Alonso would have come to that solution method to determining the set intersection of three sets, which is evidence on its own that Alonso constructed an instrument. Moreover, one important realization from analyzing these results was that Alonso used the For Loop and the 'len()' function as the artifacts in the construction of an instrument to solve a problem. That is, For Loops and functions are computational tools that were not originally designed to find the intersection of sets, but were co-opted by Alonso to solve the mathematical task of finding the intersection of three sets. This is a nuanced elaboration of how I initially interpreted the role of Python as the primary artifact when I designed this study. Specifically, when I originally conceptualized this study, I considered Python itself as the primary artifact in the students' instrument development. I was thinking about text-based code in general, not specific computational tools like For Loops and functions, which is what emerged through Alonso's work. The brief example presented in this manuscript illustrates that it is possible to understand how machine-based computing can be used to develop and enhance one's understanding of a mathematical idea.

References

Artigue, M. (2002). Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work. *International Journal of Computers for Mathematical Learning, 7*(3), 245–274.

Ball, L., Drijvers, P., Ladel, S., Siller, H. S., Tabach, M., & Vale, C. (2018). *Uses of technology in primary and secondary mathematics education*. Springer.

Bogusevschi, D., Muntean, C., & Muntean, G. M. (2020). Teaching and learning physics using 3D virtual learning environment: A case study of combined virtual reality and virtual laboratory in secondary school. *Journal of Computers in Mathematics and Science Teaching*, *39*(1), 5-18.

Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2020). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*, *51*(7), 1020-1041.

Fey, J. T. (1989). Technology and mathematics education: A survey of recent developments and important problems. *Educational Studies in Mathematics*, *20*(3), 237-272.

Guin, D., & Trouche, L. (1998). The complex process of converting tools into mathematical instruments: the case of calculators. *International Journal of Computers for Mathematical Learning, 3*(3), 195–227.

Leng, N. W. (2011). Using an advanced graphing calculator in the teaching and learning of calculus. *International Journal of Mathematical Education in Science and Technology*, *42*(7), 925-938.

Levine, B., Mauntel, M., Zandieh, M., & Plaxco, D. (2020, February). Chase that Rabbit! Designing vector unknown: A linear algebra game. In S. S. Karunakaran, Z. Reed, & A. Higgins. *Proceedings of the 23rd Annual Conference on Research in Undergraduate Mathematics Education*.

Lockwood, E., DeJarnette, A. F., & Thomas, M. (2019). Computing as a mathematical disciplinary practice. *The Journal of Mathematical Behavior*, *54*, 1-18.

Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in RUME. *International Journal of Research in Undergraduate Mathematics Education*, 1-13. https://doi.org/10.1007/s40753-020-00129-2

Oates, G. (2011). Sustaining integrated technology in undergraduate mathematics. *International Journal of Mathematical Education in Science and Technology*, *42*(6), 709-721.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Perlis, A. (1962). The computer in the university. In M. Greenberger (Ed.), *Computers and the world of the future* (180–219). MIT Press.

Roorda, G., Vos, P., Drijvers, P., & Goedhart, M. (2016). Solving rate of change tasks with a graphing calculator: A case study on Instrumental Genesis. *Digital Experiences in Mathematics Education*, *2*(3), 228-252.

Trouche, L. (2004). Managing complexity of human/machine interactions in computerized learning environ- ments: Guiding students' command process through instrumental orchestrations. *International Journal of Computers for Mathematical Learning, 9*(3), 281–307.

Vergnaud, G. (2009). The theory of conceptual fields. *Human Development, 52*(2), 83–94.

World Economic Forum. (2016). The future of jobs: Employment, skills and workforce strategy for the fourth industrial revolution. *Global challenge insight report. Geneva: World Economic Forum*.