Serving CS Formative Feedback on Assessments Using Simple and Practical Teacher-Bootstrapped Error Models

by

Kristin Victoria Stephens-Martinez

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Armando Fox, Chair
Professor Marcia Linn
Assistant Teaching Professor John DeNero

Fall 2017

Serving CS Formative Feedback on Assessments Using Simple and Practical Teacher-Bootstrapped Error Models

Abstract

Serving CS Formative Feedback on Assessments Using Simple and Practical
Teacher-Bootstrapped Error Models

by

Kristin Victoria Stephens-Martinez

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Armando Fox, Chair

The demand for computing education in post-secondary education is growing. However, teaching staff hiring is not keeping pace, leading to increasing class sizes. As computers are becoming ubiquitous, classes are following suit by increasing their use of technology. These two defining factors of scaled classes require us to reconsider teaching practices that originated in small classes with little technology. Rather than seeing scaled classes as a problem that needs management, we propose it is an opportunity that lets us collect and analyze large, high dimensional data sets and enables us to conduct experiments at scale.

One way classes are increasing their use of technology is moving content delivery and assessment administration online. Massive Open Online Courses (MOOCs) have taken this to an extreme by delivering all material online, having no face-to-face interaction, and allowing the class to include thousands of students at once. To understand how this changes the information needs of the teacher, we surveyed MOOC teachers and compared our results to prior work that ran similar surveys among teachers of smaller online courses. While our results were similar, we did find that the MOOC teachers surveyed valued qualitative data – such as forum activity and student surveys – more than quantitative data such as grades. The potential reason for these results is that teachers found quantitative data insufficient to monitor class dynamics, such as problems with course material and student thought processes. They needed a source of data that required less upfront knowledge of what the teacher wanted to look for and how to find it. With such data, their understanding of the students and class situation could be more holistic.

Since qualitative data such as forum activity and surveys have an inherent selection bias, we focused on required, constructed-response assessments in the course. This reduced selection bias had the advantages of needing less upfront knowledge and focused attention on measuring how well students are learning the material. Also, since MOOCs have a high proportion of auditors, we moved to studying a large local class to have a complete sample.

We applied qualitative and quantitative methods to analyze wrong answers from constructed-response, code-tracing question sets delivered through an automated grading system. Using

emergent coding, we defined tags to represent ways that a student might arrive at a wrong answer and applied them to our data set. Since what we identified as frequent wrong answers occurred at a much higher rate than infrequent wrong answers, we found that analyzing only these frequent wrong answers provides a representative overview of the data. In addition, a content expert is more likely to be able to tag a frequent wrong answer than a random wrong answer.

Using the wrong answer to tag(s) association, we built a student error model and designed a hint intervention within the automated grading system. We deployed an in situ experiment in a large introductory computer science course to understand the effectiveness of parameters in the model and compared two different kinds of hints: reteaching and knowledge integration [28]. A reteaching hint re-explained the concept(s) associated with the tag. A knowledge integration hint focused on pushing the student in the right direction without re-explaining anything, such as reminding them of a concept or asking them to compare two aspects of the assessment. We found it was straightforward to implement and deploy our intervention experiment because of the existing class technology. In addition, for our model, we found co-occurrence provides useful information to propagate tags to wrong answers that we did not inspect. However, we were unable to find evidence that our hints improved student performance on post-test questions compared to no hints at all. Therefore, we performed a preliminary, exploratory analysis to understand potential reasons why our results are null and to inform future work.

We believe scaled classes are a prime opportunity to study learning. This work is an example of how to take advantage of this chance by first collecting and analyzing data from a scaled class and then deploying a scaled in situ intervention by using the scaled class's technology. With this work, we encourage other researchers to take advantage of scaled classes and hope it can serve as a starting point for how to do so.

To Christopher Satoaki Martinez

Who supported me throughout this process.

To Varick Takashi Martinez

Who came in at the end of this work to brighten my days.

Contents

List of Figures

List of Tables

Acknowledgments

I want to first acknowledge my advisor Armando Fox for his invaluable support throughout my years in graduate school. Also, I want to recognize the other professors that mentored me in grad school. Marti Hearst served as my advisor for a time and significantly shaped my thinking. Marcia Linn taught me how to conduct education research. Vern Paxson advised me on my master's work in computer networking and gave me the push I needed to find I wanted to research computer science education. John DeNero gave me insight into the pedagogy of introductory computer science learning and machine learning. Dawn Song gave me my first taste of computer science education research. Additionally, the department staff that supported me throughout my time at UC Berkeley deserve a special acknowledgement.

Next, I want to acknowledge my friends. My Thrive in Science graduate student group supported me through the hard times, gave me perspective, and helped celebrate my triumphs. In addition, my fellow department graduate students encouraged me in this crazy shared experience called graduate school. Also, thank you to my friends and family for helping me polish this text. Finally, I want to explicitly acknowledge edX and Coursera colleagues as invaluable in helping with this work.

In my research, I want to give a special acknowledgment to Mario Martinez, who served as a fantastic sounding board and gave me so much advice on my statistical analysis. In addition, I could not have accomplished so much without all the undergraduate students that worked with me: Michelle Tian, Hannah Huang, Kelly Liu, Hayden Sheung, Spenser Chiang, Steven Chi, Krishna Parashar, Regina Ongowarsito, Sreesha Venkat, Nikunj Jain, Maia Rosengarten, Anwar Baroudi, and Kavi Gupta.

Finally, this work was supported by the UC Berkeley Chancellor's fellowship, the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1106400, a summer internship with Stanford's edX development team, and another summer internship at Coursera.

Chapter 1

Introduction

Computing education is experiencing growing demand for post-secondary education [61], but teaching staff growth is not keeping pace [60]. This disparity is leading to increasing class sizes, with some brick and mortar classes reaching over 1,000 students [1, 4]. At the same time, Massive Open Online Courses (MOOCs) create large online classes, which enable enrollments of tens of thousands of students[1] at once [13]. Due to the size of these classes, it is harder for teachers to understand how well students are learning both from the perspective of the whole class and for each student. A teacher's time is valuable because of her deep understanding of course dynamics and how to teach the class material. However, this precious time is spread thinly across the students. We need to find ways to best utilize this resource that is growing more and more limited.

Another vital factor to consider is the increased use of technology in the class enabled by the ubiquity of computers. This increase has been due to: (1) the need to manage larger classes, (2) convenience, and (3) pedagogical innovations, such as clickers [17]. This increased use of technology also means teachers and researchers can collect more data than ever before and this data is highly dimensional. This data can enable teachers to create a fine-grained view of each student's learning and a better aggregate view of the entire class. How can teachers navigate this data deluge to determine what information is important?

We define a scaled class as having these two aspects: a large number of students and high use of technology. These scaled classes are part of the modern computer education class and require us to reconsider our teaching practices, which originated in non-scaled, small classes with minimal technology. However, rather than merely asking how teaching practices should change, we need to take advantage of the unique opportunity presented by the scaled class. We argue that the two factors of a scaled class enable us to better study learning. The large class size enables us to identify class trends that previously had insufficient evidence because they were discernible from only a few outlier data points. Additionally, technology enables us to collect highly dimensional data and run experiments at a large scale. This scale will

[1] MOOC platforms usually refer to students as learners for consistency, however, we will refer to all those taking a course of some kind as a student.

require fewer meta-analyses and reduce confounding variables such as cohort and teacher differences.

Given a scaled class, we propose a research process where we first use the technology already in place to collect data from a large class. Next, we apply a mix of quantitative and qualitative methods to gain pedagogical insights. Then, we use these insights to (1) provide teachers with feedback on the state of learning from individual students to the entire cohort and (2) design pedagogically grounded intervention experiments.

This work is one instantiation of this research process. To begin, in Chapter 3, we report on the results of surveying MOOC teachers to understand what information sources they value during a particular phase of running or preparing for their MOOC, with the goal of creating an information display for MOOC teachers. Through our survey, we found that:

1. Quantitative data sources, such as assignment grades, although useful, are not enough. Instead, understanding the activity in discussion forums and student surveys were of interest to 97% of the surveyed MOOC teachers who answered questions on the use of information sources.

2. Teachers do not believe chat logs are a valuable information source for understanding student behavior.

3. Generally, MOOC teachers want the same sources of information as teachers of smaller-scale, distance-learning courses, based on prior work.

4. Respondents reacted positively to mockups of both previously-used and novel visualization techniques, indicating they would use these while monitoring a running course and for review when preparing for a new offering.

5. Teachers expressed widely varying views on the types of data and visualizations they would find useful.

This keen interest in the qualitative forum and survey data is surprising because it has an inherent biases. Prior work shows forum use is typically limited to a small percentage of students [13, 20], and survey data has inherent self-selection bias. However, one likely reason for this interest is because MOOC teachers found quantitative data insufficient to monitor class dynamics, such as problems with course material and student thought processes. To see such issues using quantitative data the teacher had to know: (1) that such problems could happen and (2) how to check if they were happening. With qualitative data, the MOOC teachers were potentially seeking a data source that required less upfront knowledge to find problems. This way their understanding of the students and the class could be more holistic.

Therefore, to have the advantages of qualitative data's less upfront knowledge requirement, reduce the selection bias, and focus attention on measuring how well students are learning the material, we used code-tracing questions that were required, univalent (having a single correct answer), and constructed-response. In addition, to address the potential problems caused by the large number of auditors in MOOCs, we instead collected and analyzed data from a large, in-person introductory computer science course. Compared to MOOC data, this data is a much more complete sample of student behavior. Also, the data's constructed-response nature provides us with qualitative data that allows us to consider student learning with less upfront knowledge compared to what a purely quantitative approach would require.

In Chapter 4, we report our findings from analyzing this data collected from the auto-graded, question-answer, code-tracing system in the large, in-person introductory computer science course. In our analysis, we first sought to better understand whether it is possible to analyze such qualitative data. Then, if we found it is possible, we investigated: (1) if we could apply the results of analyzing one course offering's data to subsequent course offerings, (2) how best to analyze this data, and (3) what insights we can gain from this data. We found:

1. Only a ≈5% subsample of the most frequent wrong answers cover ≈60% or more of the wrong responses. This result meant we merely needed to inspect a small subsample of wrong answers to understand the majority of student behavior.

2. The frequent wrong answers are consistent in how much they overlap for a given question set and a pair of course offerings, but the level of overlap varies between question sets and course offering pairs.

3. The wrong answer's frequency should be taken into account to choose the subsample, for which emergent coding is a reasonable analysis technique.

4. The student difficulties we found included misconceptions identified in prior work and new difficulties not reported, such as ways students struggle with programming-language-specific constructs and data structures.

To research the advantages of scaled classes, in Chapter 5, we report how we used the insights gained from analyzing the students' wrong answers to deploy a hint intervention experiment. Our goal was to enable teachers to offload to the computer the task of delivering hints for common difficulties. This offloading would then give teachers more time to focus on students struggling with rarer difficulties.

In this experiment, we added a feature in the auto-grading system from the large introductory computer science course that collected the original data set. This feature used a student error model built from the results of Chapter 4 to automatically deliver hints to students as they answered code-tracing questions. In addition, we compared two different kinds of hints: reteaching and knowledge integration. Reteaching hints re-explain the concept(s) associated with the tag. Our knowledge integration hints are inspired by Gerard et al.'s work [28] with the goal to push the student in the right direction without re-explaining the concept she is struggling with. We reach this goal with hints that remind the student of an idea or ask her to compare two or more ideas. From our experiment we found:

1. It is straightforward to deploy an intervention experiment at scale by taking advantage of technology that already exists in the class.

2. That co-occurrence between wrong answers yields useful information about machine-marked-wrong answers without having to inspect them.

3. We can build our student error model to identify a student's difficulties using both the results of Chapter 4 and co-occurrence information.

4. That we currently do not have sufficient evidence that giving hints to students using this model improves performance on post-test questions.

5. Preliminary results using qualitative and quantitative techniques provide a basis for us to start understanding when hints do help.

Before these three chapters, we include an overview of related work in Chapter 2. In Chapter 6 we discuss in more detail how a scaled class can help student learning, lessons learned from this work, and potential directions for future work. Finally, in Chapter 7, we conclude with some final thoughts on our research.

Chapter 2

Related Work

Research areas similar to our work include learning analytics (LA), education data mining (EDM), Intelligent tutoring systems (ITSs), and online learning tools. We can characterize research areas that analyze large education data sets using three aspects: (1) where the data is from, (2) how the data is analyzed, and (3) how the insights from the analysis are used. We can break down the data source further into how many courses it came from and how many learning domains it covers. Our work's characteristics overlap with each of these related research areas, but does not share the same three with any.

For example, the three characteristics of LA [26] – and especially early warning systems [10, 62] that sought to predict student outcomes in time for an intervention – collect their data from many courses and many learning domains. This data is usually collected at the campus level from their online learning tools. Analysis techniques used in LA include network analysis, information visualization, statistical tests, and machine learning with a foundation in learning theory. LA's goals are usually to predict student outcomes so that we can improve learning with actionable recommendations.

Closely related is EDM [9, 65]. EDM data is collected from a single course or many, usually from the same learning domain. EDM techniques include information visualization, association analysis, clustering, classification, pattern analysis, and regression. EDM research focuses on the technical challenges of processing the raw education data into information that improves our understanding of learning.

ITSs also collect data from many classes on the same learning domain. However, their data analysis focuses on creating student or knowledge models. ITS research then uses these models to determine what feedback to give students within the ITS.

Finally, broader than ITSs are online learning tools. Researchers in this area [21, 37, 48] collect data on the tool's usage in many courses on the same learning domain. The data is analyzed with a mix of quantitative and qualitative methods with the goal of improving the tool.

Our work within the above framework has the source of data coming from a single course, which by definition is on the same learning domain. We also use mix methods to analyze the data, and the insights go towards improving the course. The critical difference is the data comes from a single course. In our case, it consists of mainly constructed responses – where the student creates the answer – from code-tracing assessments – where the student is predicting the output of code. Moreover, because these are code-tracing questions the answer is univalent (only one answer) and easy to grade. As a result, we can automatically grade student responses, which enables assigning many of these questions due to the low cost per student.

The data is large because it comes from a scaled class and the answer space is technically infinite due to the constructed nature of the answers. However, while the answer space is unlimited, students are not randomly picking wrong answers from this space because these answers are from code-tracing questions. The lack of randomness means a significant portion of this "infinite space" will be unlikely to students because the code drives them towards particular subspaces of wrong answers. All of this results in a large data set that is qualitative yet quantitatively analyzable, meaning it is possible to analyze the data using mixed methods. The primary methods we apply are: (1) analyzing the frequency of wrong answers and (2) emergent coding [57]. We use the insights from this analysis to inform an intervention experiment to improve the course by delivering hints to students as they answer the formative, constructed-response, code-tracing assessments.

Chapter 3

Monitoring MOOCs: Which Information Sources Do Teachers Value?

3.1 Introduction

In brick-and-mortar classrooms, teachers often use face-to-face interaction with individual students during lecture and office hours to understand student performance in the course and how they interact with the course materials. Many recent Massive Open Online Courses (MOOCs) including providers such as edX and Coursera have enrolled tens of thousands of students per offering, with a few enrolling hundreds of thousands. At such scales, individual interaction with every student is infeasible, and most interactions are through the software platform, rather than face-to-face. Fortunately, a MOOCs' large scale and the fact they are offered via a heavily instrumented online environment provide teachers with a rich source of information they previously lacked: instrumented activity from interactions with the e-learning platform.

Historically, data visualization has been an effective way to explore large data sets in which identifying interesting patterns is more productive than scrutinizing individual data points. Since MOOCs are relatively new, little work has been done on visualizing the rich sources of information available in them. Current MOOC platforms offer only a small set of visualizations of basic quantitative information.

To help explore this space, we investigate it from two angles. First, we implemented a prototype teacher dashboard for the edX platform called the Metrics Tab (see Figure 3.1) that is currently available only to a small number of test users. Second, and the focus of this paper, we administered a survey to investigate the following questions:

This work was done with Marti A. Hearst and Armando Fox [75].

1. What information sources do MOOC teachers prefer to help identify key trends and behaviors in both student performance and student interaction with course content?

2. Which of these sources are most useful to teachers during the three phases of course preparation, course administration, and course postmortem?

3. How should these sources be presented to develop tools and visualizations that teachers will find most useful?

92 MOOC teachers answered the survey. Survey questions included visualizations of information sources, two of which were modeled after those in the Metrics Tab, as well as three additional designs. Teachers were asked to judge the understandability and usefulness of each design.

The results support the following primary findings:

1. Quantitative data sources, such as assignment grades, are not enough: understanding discussion forum activity was of interest to 97% of those surveyed that answered questions on the use of information sources. This interest is despite (1) a lack of related work on visualizing discussion forum activity at scale and (2) previous work showing that forum use is typically limited to a small percentage of students who are not necessarily representative of the overall enrollment [13, 20].

2. Teachers do not think chat logs are a valuable information source for understanding student behavior.

3. By and large, MOOC teachers want the same sources of information as teachers of smaller-scale, distance-learning courses, as found in earlier surveys.

4. Respondents reacted positively to mockups of both previously-used and novel visualization techniques, indicating they would use these to monitor a running course and to review materials when preparing for a new offering, but were less likely to use them in preparing new material.

5. Teachers expressed widely varying views on the types of data and visualizations they would find useful: some preferred data and visualizations that would support quantitative analysis such as correlation; others conducted courses focused more on discussion than quantifiable grades and therefore considered quantitative analysis not useful, and so on.

Below, we present related work in Section 3.2; describe the survey procedures in Section 3.3; describe the visualizations in Section 3.4; present the results in Section 3.5 and 3.6; discuss the ramifications of these results; and close with recommendations for future work for the design of monitoring interfaces for MOOC teachers in Section 3.7.

Information Visualized	**Related Work**
Performance: Grades on assignments, cumulative performance on problems for a particular concept	[43, 27, 54, 53, 56]
Access and Activity Patterns: What, how much, and when content has been opened, how long a student stays on a piece of content, student navigational path through the content, when a student turned in an assignment	[43, 2, 27, 30, 33, 34, 54, 53, 56, 84]
Forum Discussions: Author of a post, when the post was made, structure of follow-up posts, how many posts a student made, how many follow-up posts are in threads each student made, number of posts read by a student	[29, 30, 54, 53, 56]
Student Demographics: Location, reason for taking the course, age, learning style	[34, 84]

Table 3.1: Types of information and related work that visualizes each.

3.2 Related Work

Teacher surveys

Monitoring student learning has been promoted as a best practice in education literature since the 1970s [22]. Two surveys of e-learning teachers – one in 2003 by Mazza et al. [55] (98 participants) and another in 2006 by Zinn et al. [87] (49 participants) – agreed broadly on several points. Respondents stated that the most important phenomena to monitor are individual students' performance, per-student performance compared to the class as a whole, common misconceptions shared by many students (as manifested by common wrong answers to exercises, for example), and a range of metrics regarding student activity patterns. Mazza et al.'s respondents also said that forum behavior was a valuable way to gauge participation, but email or chat data was not.

Visualizations of student information

Visualizations have been used as a form of educational data mining [65]. However, very little related work in visualizing student information has focused on MOOCs, and modern MOOC platforms such as edX and Coursera provide limited teacher-facing visualizations. Table 3.1 shows information categories prior work has commonly visualized, ranging from standard graphs to innovative designs.

Standard graphs used by prior work [43, 2, 27, 30, 33, 34, 54, 53, 56, 84] include: scatter plots, bar indicators, bar and stacked bar charts, line graphs, Cumulative Distribution Function (CDF) line graphs, pie charts, and heat maps. These graphs are used by prior work in one of two ways: (1) to provide a set of visualizations showing different kinds of information or (2) as a supporting graph in a complex visualization.

The prior work that provides visualizations for multiple categories of information [43, 27, 30, 34, 54, 53, 56] usually has the goal of giving teachers an overall picture of their course's e-learning experience. These visualization systems include: Goldberg et al.'s [30] early WebCT [31] visualizations; Hardy et al.'s [34] e-learning tracking visualizations; Mazza et al.'s Coursevis [54, 53] and GISMO [56]; Gaudioso et al.'s [27] visualizations for dotLRN and PDinamet; and Khan Academy's Coach monitoring system [43]. It is important to note that while these systems provide an overview of the course, they often are intended for courses of only tens to the low hundreds of students and visualize each student individually (such as show each student as their own row in a heat map). As a result, a majority of these visualizations would not scale to the size of a MOOC unless judicious filtering is applied first.

Innovative visualizations used by prior work usually use known visualization techniques in a novel way. These include directional and non-directional node graphs, three-dimensional graphs, timeline spiral graphs, icons, and line graphs. Node graphs are used by Hardless et al. [33] to show a timeline of student activity, which they call an activity line. Williams and Conlan [84] use a node graph to show the navigational path through content. Finally, Gibbs et al. [29] use a directional node graph, with node placement conveying time, to show how forum posts relate to each other.

Mazza et al. [54, 53] also visualize forums with a three dimensional scatter plot that the user could explore. The timeline spiral graph by Aguilar et al. [2] displays student access and activity patterns. This graph used mainly bars for both supporting information and spiraled around a center where each 360-degree spin was an easily understood unit of time (*e.g.,* 24 hours, 1 week). Icons used by Khan Academy's Coach tool [43] highlight points in bar charts when students earned badges. Two prior works that use line graphs in innovative ways are Hardy et al.'s [34] line graph with shading to depict a student's path through the material and Williams and Conlan's [84] line graph as a connected sparse scatter plot depicting a student's learning style.

Four interaction techniques used in the prior work include sorting, filtering, drill down, and clustering. Sorting was usually available in any visualization that provides a tabular view of information, such as Goldberg et al.'s [30] WebCT tabular student views and Khan Academy's Coach tool [43] that shows students individually. Aguilar et al.'s [2] timeline spiral allows users to filter by time, activity, course, and student. Hardy et al.'s [34] e-learning tracking visualizations allow filtering by time and any subset of students. It also incorporates an understanding of course hierarchy, which provides an ability to drill down through this hierarchy. Gaudioso et al.'s [27] visualizations for dotLRN and PDinamet includes a clustering feature that automatically groups students based on access patterns. It provides a way to view aggregate information of the students in the groups and compare these aggregates to each other. Huang et al. [38] also use clustering in their node graph to show syntax similarity between student code submissions.

Evaluation in prior work mainly involved interviews and focus groups [29, 33, 54, 53, 56, 84] and usually reported a mix of both positive responses to the system and a need for future improvements. Three prior works did not include an evaluation section [2, 30, 34]. Gaudioso et al.'s [27] work with dotLRN and PDinamet considered the dropout/success rate of the classes before and after the visualizations were provided to the teachers and found a marked improvement. However, there is no discussion of whether the improvement with the dotLRN system is due to the visualizations or the revamped material that happened at the same time. They also conducted a questionnaire on student and teacher satisfaction, which found that a majority of both groups were satisfied with the course and system. Mazza et al.'s [54, 53, 56] work on Coursevis and GISMO performed the most thorough evaluation, looking at the system's extent of required functionality, effectiveness, efficiency, and usefulness through a combination of an experimental study, interviews, and a focus group. Their results are positive across all their criteria.

3.3 Survey Procedure

We used SurveyMonkey to administer a survey estimated to take about 30 minutes. We identified 539 potential participants by collecting teacher names from the web page of courses offered on the three largest MOOC platforms: edX, Coursera, and Udacity.

The survey consisted of five parts:

1. **Background information** about the teachers.
2. **Specific details about one MOOC**. If a teacher taught multiple MOOCs, we asked them to choose one and answer all following questions in terms of that MOOC.
3. **Course Monitoring Goals** and asking which information sources help achieve the desired understanding.
4. **Mockups** of five different visualizations of information that may be useful for monitoring a MOOC and evaluating their efficacy.
5. **Open-ended response** for additional thoughts.

The next section describes the mockups in more detail.

3.4 The Metrics Tab and Visualization MockUps

The teachers were asked to evaluate the potential usefulness and understandability of five visualizations of source information for monitoring MOOC activity. We derived two of these visualizations from designs in the prototype Metrics Tab, a new tab in the edX Teacher Dashboard. Figure 3.1 shows this visualization in detail.

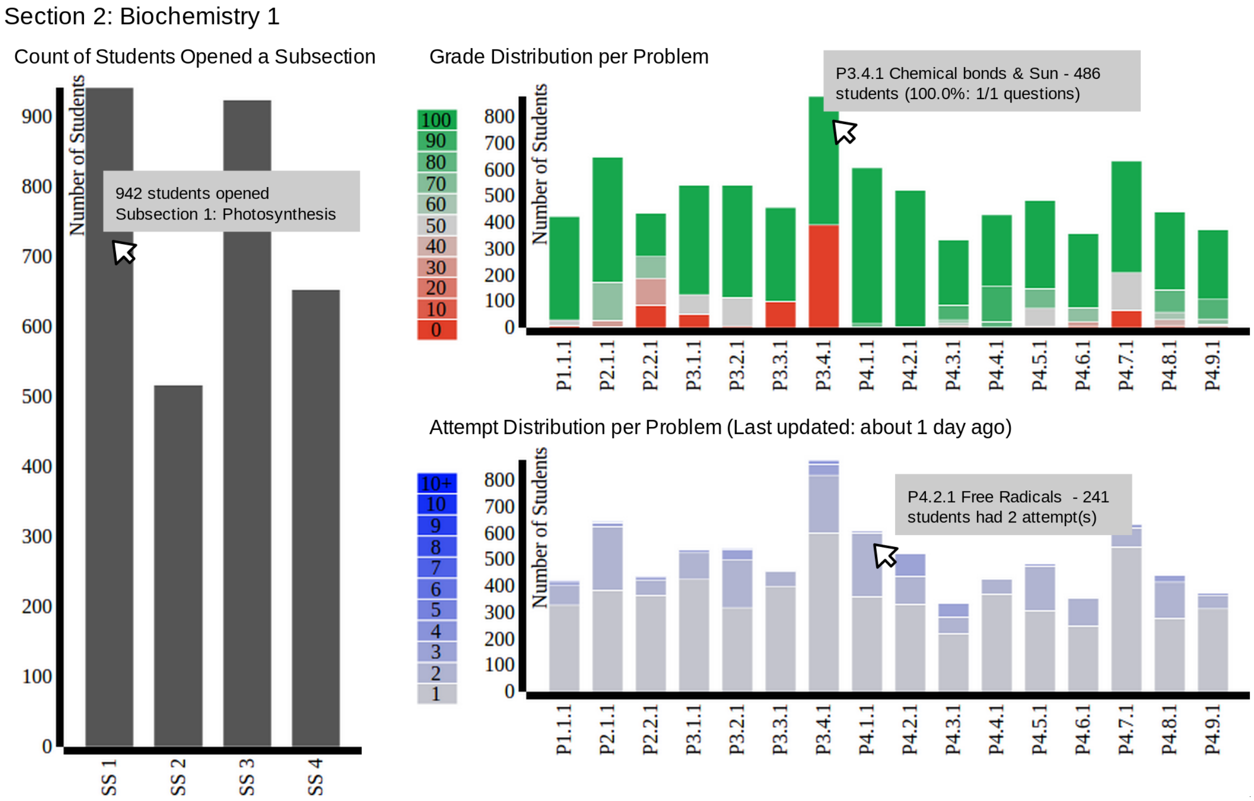


Figure 3.1: Mockup of a single section in the prototype edX Metrics Tab with a tooltip visible for each graph.

We based both the mockup and the implemented prototype visualizations on ideas from previous work and conversations with teachers before we administered the survey. The mockups we decided to use also served as a preliminary evaluation of the Metrics Tab.

Metrics Tab

The goal of the Metrics Tab is to provide teachers a quick to consume dashboard display of available information in their course. The Metrics Tab separates the course's information by section and shows the same dashboard display seen in Figure 3.1 for each section. The section was chosen as the level of granularity because edX usually uses a section to contain a week's worth of material, with subsections allowing further division of the week's content.

The left grey bar chart shows how many students opened each subsection in the section; that is, viewed at least some of the content in that subsection at least once. When the user hovers the cursor over a bar in this graph, the name of the subsection and the exact number of students that opened that subsection will appear in a tooltip, as seen in Figure 3.1.

The upper right red and green stacked bar graph shows the grade distribution for each problem in the section. It shows every problem regardless as to whether the problem is included in the students' course grade or not. If students are allowed to submit an answer multiple times, as is common in MOOCs, it only shows the grade for their last submitted answer (since the last submitted answer is used when calculating a student's grade). For a given bar in the graph, the color represents the grade for all the students in the bar, and the height represents how many students received that grade. The color gradient for grades, seen to the left of the graph, goes from red, grey, to green[2] (0, 50, and 100 percent). Hovering the cursor displays the teacher-defined description of the problem, the number of students in the bar, their percentage grade, the number of points earned, and the number of possible points.

Finally, the bottom right blue stacked bar graph shows the distribution of the number of attempts per problem in the section. On both edX and Coursera, MOOC teachers can choose the number of times students may attempt each problem. The color gradient is grey to blue, which maps from 1 to 10+ attempts. Students that attempted more than 10 times are grouped together because some problems allow unlimited attempts, which students do take advantage of. Hovering over a bar reveals the teacher-defined description of the problem, the number of students in the bar, and the number of attempts.

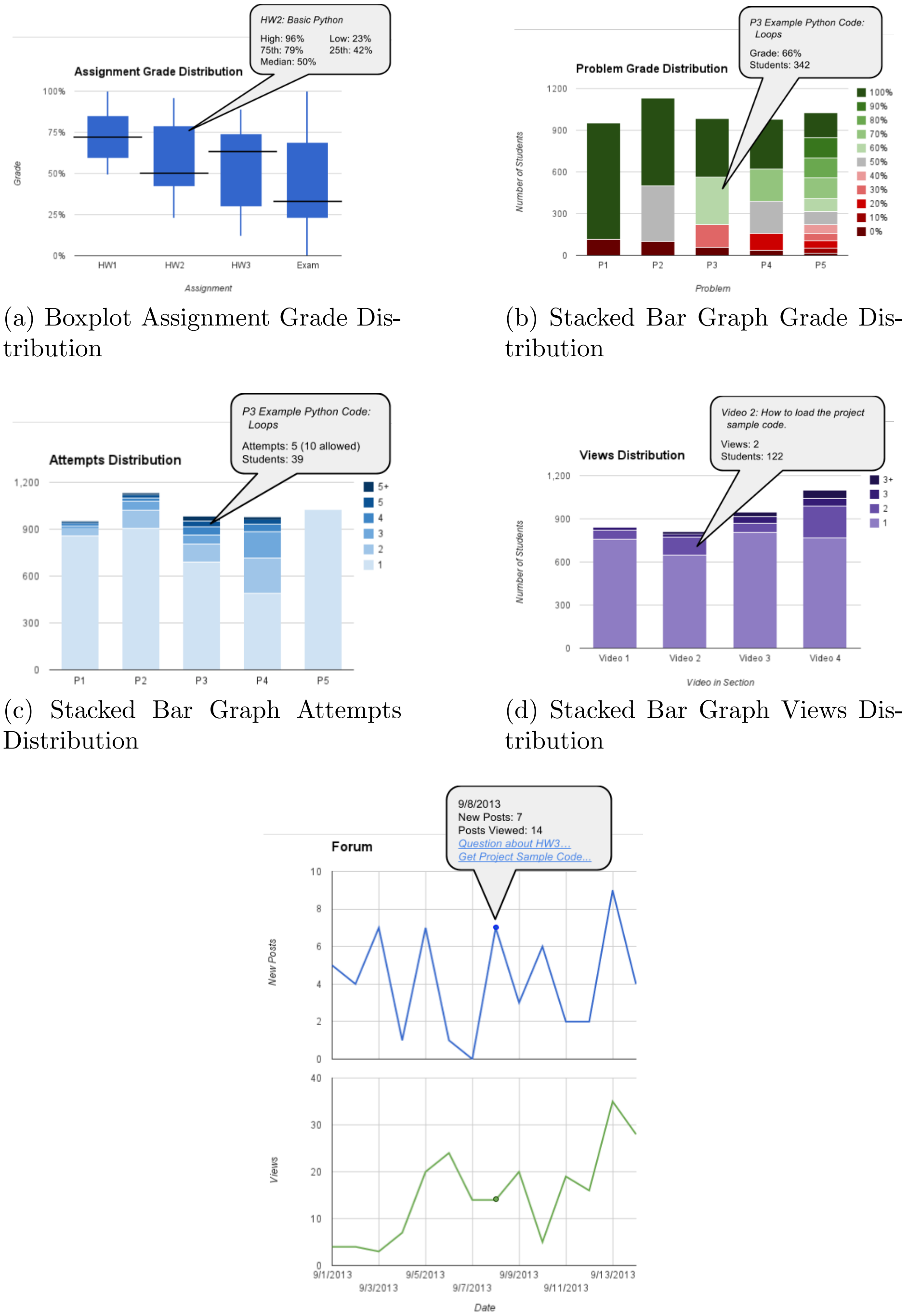
Visualization Mockups

The five mockups we presented to teachers in the survey appear in Figure 3.2. The callout bubble in each mockup represents the tooltip if the user hovers the cursor over that or a similar part of the graph. Mockup 3.2a is a boxplot diagram of grade distributions, which is included as a standard visualization. The tooltip shows the name of the homework assignment or assessment item, the high and low scores, the median score and the 75th and 25th percentile scores. Mockup 3.2b is very similar to the upper right graph in the Metrics Tab, and Mockup 3.2c is similar to the lower right graph of the Metrics Tab. Mockup 3.2d is similar to Mockup 3.2c but shows views of materials rather than attempts at homework problems. Finally, Mockup 3.2e shows two line graphs of forum usage data: number of new posts per day and number of views per day. The tooltip is for both graphs. On hover, the points with the same date are highlighted. The tooltips text includes the date, the number of posts for that day, the number of posts viewed on that day and the titles of the most popular posts.

In the survey, each mockup in Figure 3.2 included a description on how to read the graph and any interactions with it.

Participants were asked to provide Likert responses to (a) whether the mockup is useful and (b) whether it is easy to understand. Next, we asked when the teacher might use it: (1) when preparing new material, (2) when preparing by reviewing past courses, and (3) while the course is running. We also ask if they have any other comments about the mockup (open-ended response).

[2] This color scale is inappropriate for red-green color-blind viewers, and so in future iterations will be changed.



HW2: Basic Python
High: 96% Low: 23%
75th: 79% 25th: 42%
Median: 50%

(a) Boxplot Assignment Grade Distribution

P3 Example Python Code: Loops
Grade: 66%
Students: 342

(b) Stacked Bar Graph Grade Distribution

P3 Example Python Code: Loops
Attempts: 5 (10 allowed)
Students: 39

(c) Stacked Bar Graph Attempts Distribution

Video 2: How to load the project sample code.
Views: 2
Students: 122

(d) Stacked Bar Graph Views Distribution

9/8/2013
New Posts: 7
Posts Viewed: 14
Question about HW3...
Get Project Sample Code...

(e) Line Graph Forum

Figure 3.2: Mockups shown to survey participants. The callout bubbles display the tooltip if the user hovers the mouse over a part of the graph.

#	Run (N=92)	Created (N=91)
1	67%	85%
2	13%	9%
3	4%	1%
4+	9%	1%
N/A	6%	4%

Table 3.2: Survey respondents' MOOC experience for running and creating 1 to 4+ MOOCs.

	100s	1,000s	10,000s	Total
Engineering	3	10	7	20
Science	2	25	10	37
Humanities	1	8	10	19
Other	1	3	7	11
Total	7	46	34	87

Table 3.3: Cross between the estimated number of students in the course and the course's area.

3.5 Survey Results

Of the 539 teachers solicited, 92 teachers (17%) started the survey and 67 (73%) completed it. Of the 91 teachers that chose to answer the question about gender, 73% identified as male, 25% as female, and 2% chose not to specify.

Characteristics of Courses

Of those teachers who ran a MOOC, more than two-thirds had done so only one time, while 13% had done so twice (see Table 3.2). That said, many of these teachers have experience with large in-person courses. 31% said they had run courses with greater than 250 people more than 4 times, and another 25% had done so 3 or fewer times. 85% of survey respondents reported creating one MOOC, 9% created two, one individual created three, and one created four courses (see Table 3.2). Teachers have used a wide range of platforms, with Coursera and edX being the most frequent; Figure 3.3 shows the usage counts of the others reported. "Other" refers to platforms the teachers provided to us in the survey. These platforms include Google, Desire2Learn, and an institution-specific MOOC platform.

For the questions that followed, if teachers had taught more than one MOOC, they were asked to choose one and answer in reference only to it. 91% of the courses completed with 1,000s to 10,000s of students. Table 3.3 shows the estimated number of students in the course at the time of completion crossed with the subject matter of the course. The courses marked "other" include social sciences, business, education, and interdisciplinary studies. Interestingly, humanities courses were frequently among the largest MOOCs.

Course Monitoring Goals

We asked the survey participants to consider nine different tasks or goals they might have when running or planning a MOOC, summarized in Table 3.4. We asked teachers to assess ten information sources regarding their efficacy for these nine goals and rate them in terms of if they currently *use*, *would use if available*, or *do not use/would not use* that resource.

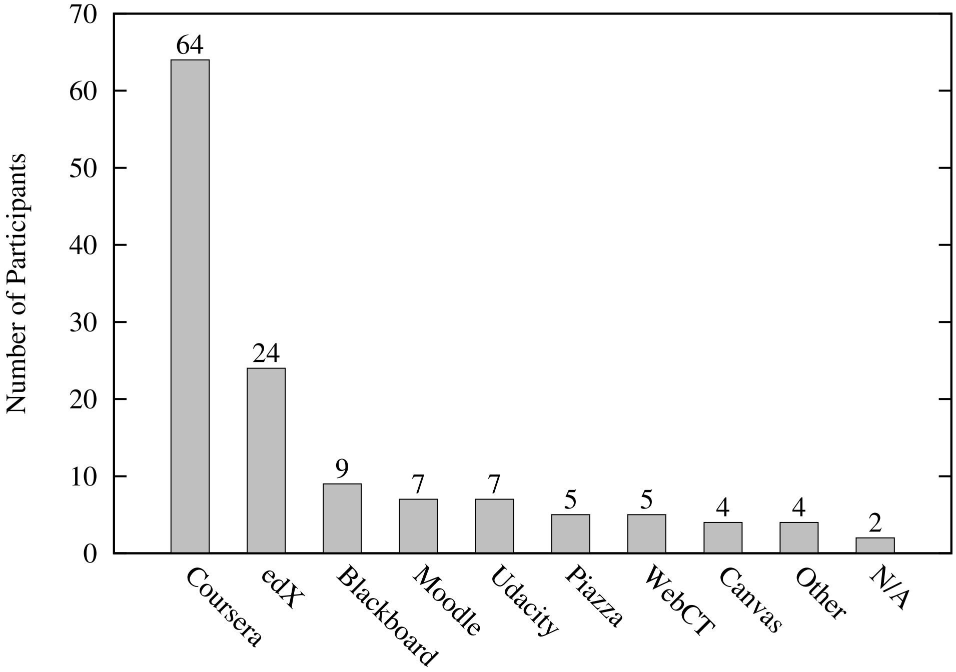


Figure 3.3: Platform usage statistics.

1	Problems with the current assignment.
2	Struggling students and what they are struggling with.
3	The difficulty of an exam problem.
4	Appropriateness of course difficulty level for students.
5	Most engaging content for the students.
6	Most difficult parts of the course.
7	Effectiveness of teaching assistants.
8	How to improve presentation of a topic.
9	Content students considered least interesting.

Table 3.4: Short descriptions of Course Monitoring Goals.

F	Discussion Forum
CS	Class Survey
SD	Discussion with Students
TA	Ask the teaching assistants
AG	Assignment grades
PAn	Student answers to problems
VP	Student's view pattern of online content
PAt	Number of times students attempt a problem
SCQ	Grades for self-check questions
CL	Chat room logs

Table 3.5: Resources Potentially Used for Course Monitoring.

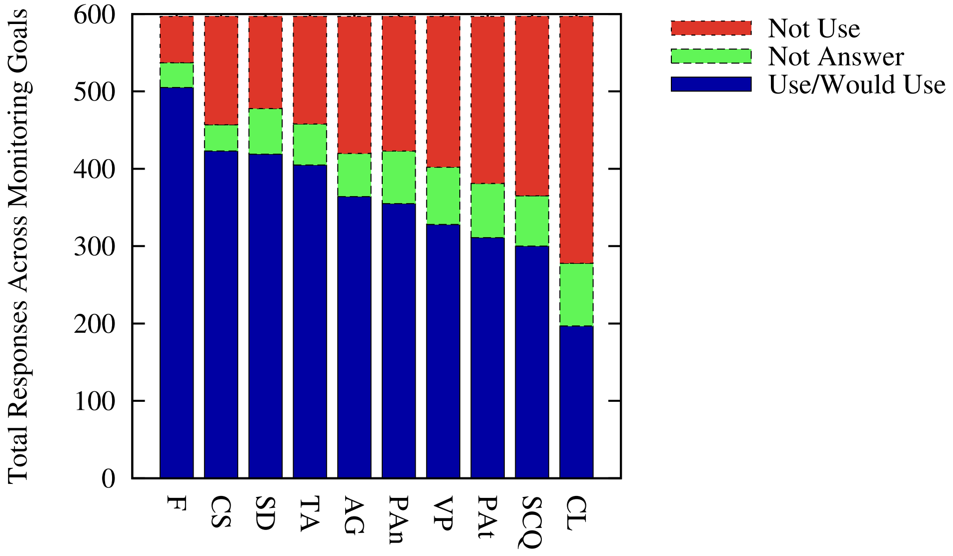


Figure 3.4: For each information source, the number of participants who use it or would use if available (combined into a single category), do not use, or did not answer. Did not answer indicates that the participant chose an answer for a subset of the information sources for that Monitoring Goal. It is shown here to see the relative rate of responses.

Asking about usage is different from prior work [55, 87], which asked survey participants to rate the level of interest or importance of an information source. We also purposely chose resources that teachers are likely familiar with to reduce the need for the teachers to guess if a resource is useful. Table 3.5 shows this list of resources.

The main findings from this section of the survey are:

Qualitative information is important. Figure 3.4 shows the raw counts of responses across monitoring goals for each information source. (Visual inspection did not reveal significant differences when we examined subsets of courses by area, but we did not conduct statistical tests to confirm this.) The lower segment (blue, solid border) indicates those who currently use this resource or would use it if available. The top segment (red, dotted border) indicates the counts of those who would not or do not use this resource across all tasks. For some questions, participants chose to answer usage for a subset of information sources. We assign the center (green, dashed-lines border) bar for the not-answer responses to show the relative rate of response.[3] The figure is ordered by highest to lowest raw response counts of use or would use if available.

Figure 3.4 shows that across all tasks, teachers see discussion forums as the most useful and chat logs as the least useful. In particular, the preference of forums over class surveys (the second-highest-used information source) is significant (Fisher's exact test, $p < 0.001$) as is the difference between chat logs and self-check questions (the second-lowest-used information source; Fisher's exact test, $p < 0.001$).

If we visualize the responses by Course Monitoring Goal, the response patterns suggest a grouping as shown in Figure 3.5. The figure suggests the relative usefulness of different information sources within a group of goals is more uniform than it is outside the group. To make comparison easier, each separate stacked bar graph is the same as Figure 3.4, except it is the percent of teachers that answered for that Course Monitoring Goal instead of the raw counts. Results are not significantly biased because 59 to 75 teachers answered each question.

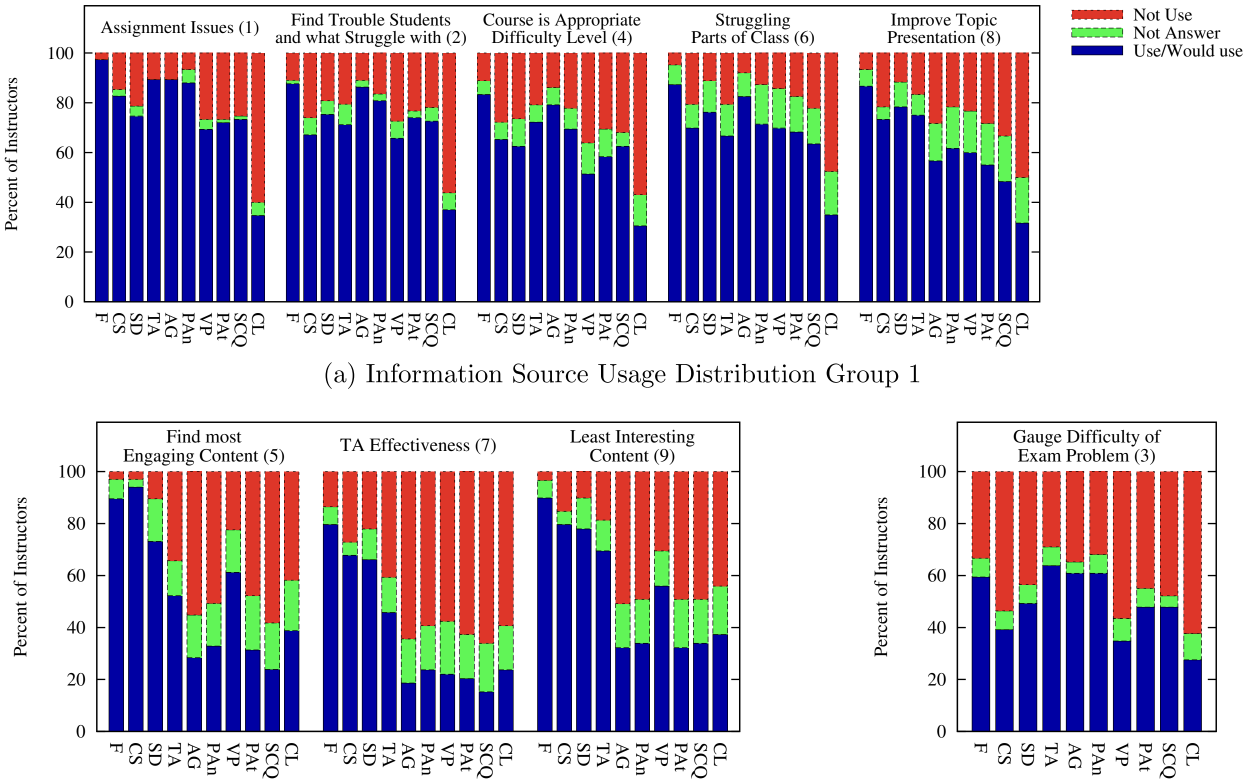
The first group of five Monitoring Goals (1, 2, 4, 6, and 8, in Figure 3.5a; the numbers correspond to Table 3.4), could be characterized as quantitative questions about course material difficulty or presentation of course materials. For these Monitoring Goals, all information sources but chat logs have 97% to 48% of question respondents say they use/would use the information source. Chat logs have only 37% to 31% question respondents say they use/would use it.

The second group of three Monitoring Goals (5, 7, and 9, in Figure 3.5b) could be characterized as a qualitative assessment of student engagement or teacher effectiveness. In these goals, participants were most enthusiastic about the "softer" information sources such as forums, discussions with students, class survey, and discussions with TAs. The percent of teachers that say they use/would use these information sources range from 94% to 46%. While there was much less enthusiasm for quantitative performance such as assignment grades, problem answers and attempts, and self-check question grades, use/would use range from 41% to 15%. View pattern usage is the least similar between the goals, where it is used very little for the TA effectiveness goal, but used much more for the other two engagement goals.

Monitoring Goal 3 – gauging exam problem difficulty in Figure 3.5c – does not display a similar pattern to the others, with no definite winners among the information sources.

Chat room logs are rarely used and considered unimportant. Chat room logs are more not used than used. We can see this from the earlier discussion of Figure 3.4

[3] Most participants completed the entire survey, 73%. However, because this section contained 9 questions requiring answers for 10 resources, some participants became fatigued (as indicated by their free-text comments) and either skipped portions of this part of the survey or did not complete the survey beyond this point.



(a) Information Source Usage Distribution Group 1

(b) Information Source Usage Distribution Group 2

(c) Information Source Usage Distribution Group 3

Figure 3.5: Percent of participants that answered for each usage option, as well as the percent that answered part of the question but not for that information source option. The Monitoring Goals are grouped based on their usage distributions. Each goal has a short description and the number corresponds to Table 3.4. Letters along the x-axis stand for the information source, see Table 3.5.

and looking at chat logs across all questions in Figure 3.5. Across all Monitoring Goals the percent of teachers saying they use/would use chat logs ranged between 39% and 24%. We exclude chat logs from the remainder of the discussion.

Teachers' opinions of what is useful largely confirm earlier surveys. Respondents' opinions as to what information they would find useful are mostly consistent with the Mazza et al. [55] and Zinn et al. [87] surveys. In those surveys, the most important information concerns overall student performance relative to the class and information about what materials students interacted with and for how long (activity/view patterns). A majority of respondents also identified that information as useful. Respondents also agree with Mazza et al. that qualitative information from forum postings is important, but analysis of chat logs is not. However, respondents of both prior surveys placed higher importance on viewing per-student performance information and per-student mastery information. We speculate that such information is less useful in MOOCs, in which attention to individual students is rare.

Open-ended portions of the survey revealed a wide range of teacher views. Most survey questions, and each section of the survey, solicited open-text comments. The responses showed teachers' preferences ranging from simple numbers with no visualization to very complex data analysis. Complex analysis tool requests included A/B testing, correlational analysis, auto-clustering of students by teacher-chosen parameters, and detailed view pattern information including paths through the material.

An interesting dichotomy arises between courses where quantitative assessment is foremost and more experientially-oriented courses. Several teachers stated they were not interested in grades, problem answers, and other performance-based metrics because their goal was to provide students with a learning experience and not the ability to quantitatively prove they learned the material. These teachers stated they ran their course based on discussions and team interactions.

Some teachers were not worried about certain monitoring goals. One teacher stated there were too many students to worry about finding struggling individuals. Two teachers said course difficulty was not a concern. While one teacher stated that the course difficulty was fixed at the beginning and could not change while the course was running, the other said they structured their course to work at multiple difficulty levels.

Responses to Mockups

Before going into detail of the mockups responses, it is important to note a caveat to these results. As a reminder, the survey asked participants if they considered each mockup useful and easy to understand and to predict when they would use the mockup. Since the survey asked participants what they think they will like and do, as opposed to what they actually liked and did, there are limitations to these result's generalizability because what a person thinks they will like or do does not necessarily match what they will actually like or do.

The results of the mockup section are:

At least a majority of teachers considered each mockup useful and understandable. Figures 3.6 and 3.7 show participants' responses to the visualization mockups. The familiar box plot (Mockup 3.2a), when applied to student grades, was most often viewed as useful (74%) and understandable (78%), followed closely by the visualization (Mockup 3.2c) showing the number of student attempts at assignments (71% useful and 73% understandable). The number of times materials were viewed (Mockup 3.2d) was also considered useful information by two-thirds of respondents (66%). Multiple people expressed concern that the stacked bar visualization of the grade distribution (Mockup 3.2b) was difficult to understand, with only 52% agreeing that it was easy to understand. The visualization of the forum usage (Mockup 3.2e) was also not overwhelmingly supported, with only 55% of respondents agreeing or strongly agreeing that it was potentially useful.

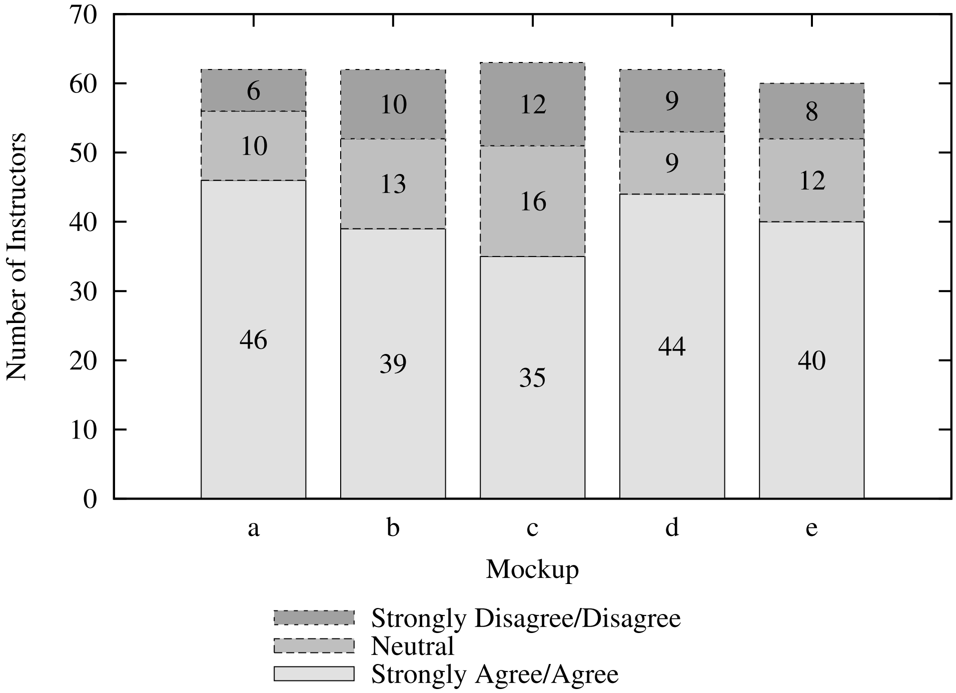


Figure 3.6: Likert scale responses to the statement "This visualization is useful." Letters correspond to Figure 3.2's subfigures.

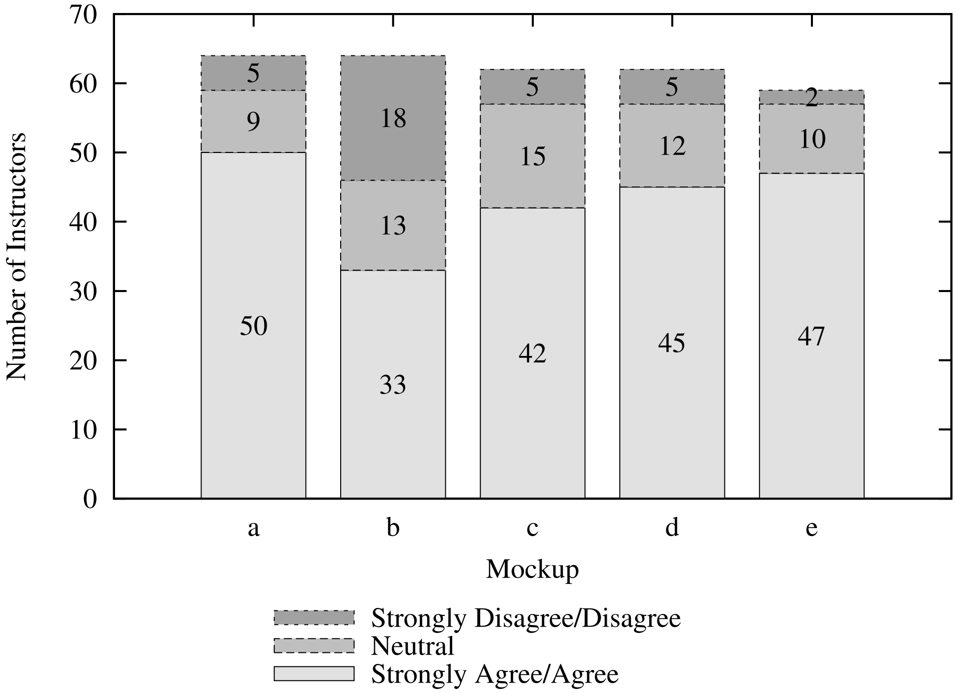


Figure 3.7: Likert scale responses to the statement "This visualization is easy to understand." Letters correspond to Figure 3.2's subfigures.

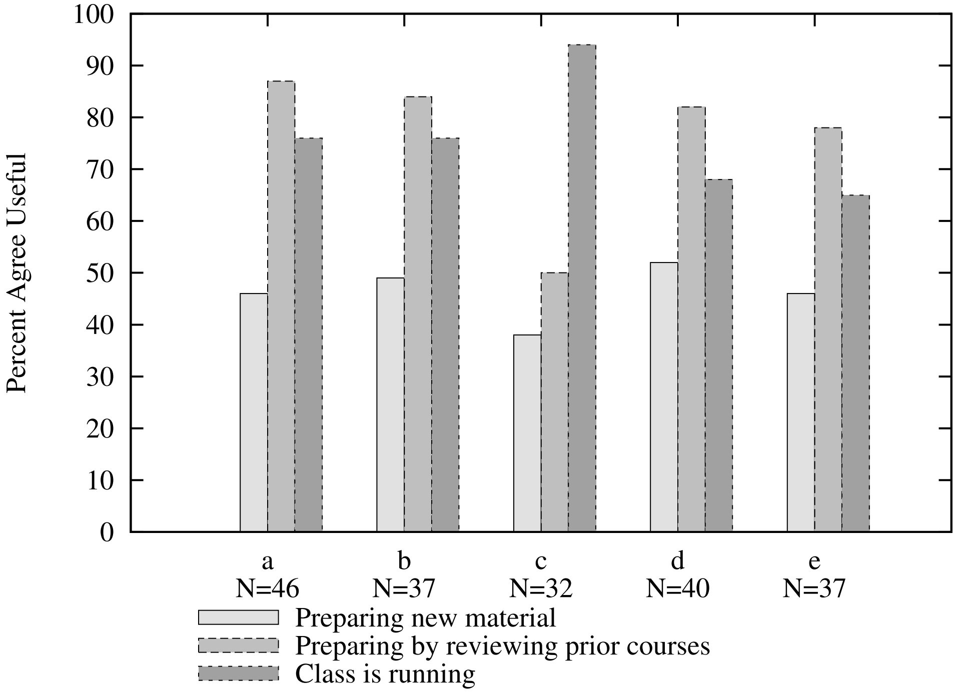


Figure 3.8: Responses to the question "When would you use this mockup," with choices in terms of three phases in a "MOOC cycle": preparing new material, preparing by reviewing the previous course runs, and while the class is running. Letters correspond to Figure 3.2's subfigures.

There is a relative lack of interest in the forum visualization. Only 55% of teachers stated the forum usage visualization would be useful. Comments about this visualization stated it is not fine-grained enough, it is not more useful than current statistics, the number of posts is not a useful indicator, and up-and-down votes are more important indicators.

Of those that considered the visualization useful, they would mainly use the visualizations while the course is running and all visualizations but forums after the course is over while preparing for a future offering. For those who did indicate that a given visualization was potentially useful, they were asked to indicate which circumstances it would be best used. Figure 3.8 presents the results; participants could mark more than one choice in each case. In all cases, at least two-thirds of respondents who found the visualization useful wanted to use it while the course was running. Moreover, by an even larger margin, teachers wanted to see every visualization except the forum visualization (c) when reviewing a past course in preparation for a future offering.

Relatively fewer participants considered the visualizations useful while preparing new course material. 38% to 52% of respondents indicated preparing new material for a course would be a good use of the visualizations.

3.6 Metrics Tab Usage Experiences

We released a variation of the Metrics Tab to a small number of test users. This variation included the open subsection count graph (Figure 3.1 left, grey) and grades graph (top right, green and red); the attempts graph was not available. We interviewed two users that used the Metrics Tab while their course was running. Also, at the time of publication, we became aware of another publication that used the Metrics Tab [32]. We report the Metrics Tab usage experiences below.

One user we interviewed was the TA of a MOOC that started with about 8,000 students and ended with about 500 completing the course. The Metrics Tab was one of the TA's primary methods for tracking student activity. The open subsection count graph was used to monitor how many students were still active in the course and if they were looking at all of the content. The grades graph was closely monitored to see how many students were doing the problems and which problems might have issues that needed to be resolved, such as input errors or ambiguities in the question.

The second interviewee was a teacher that ran a small online course of about 100 students. This teacher also used the open subsection count graph to see what content the students looked at. This teacher shared a story in which the Metrics Tab drew their attention to a student error. At the beginning of the course, many students were unaware of any but the first subsection and had to be informed that there were other subsections. The teacher also liked the grades graph and used it to monitor the students' lecture quiz grades.

The final usage experience of the Metrics Tab is from Grover et al. [32]. They used the edX platform to teach an introductory computer science course for middle and high school students and reported on the pedagogy of their course and how they leveraged the Teacher Dashboard for curriculum assessment. They used the Metrics Tab to monitor quiz data, specifically using the grades graph to find what content the students found challenging and what content needed revision.

All three experiences confirm the survey finding that student performance information is important. The interviewees' use of the open subsection graph aligns with the survey results, which found that student viewing patterns are important. Although only 52% of those surveyed found the Mockup 3.2b (which is based on the Metrics Tab's grades graph) easy to understand, neither of those interviewed had trouble interpreting the grades graph. The two interviewees were both engineers and so may be more familiar with reading graphs than other teachers.

3.7 Discussion

The survey results show that discussion forums were the most frequently preferred source of information across monitoring tasks, suggesting that effort should be invested in making forums more useful for students and more effective for providing information to teachers.

It is crucial to bear in mind, however, that prior work has found that only a small proportion of MOOC students are active on forums [13, 20], and therefore forum posters are not necessarily representative of all students in the course. Most likely teachers are aware of these limitations, and this may explain why other methods for eliciting an understanding of students' views – including student surveys, student discussion, and asking TA's for feedback – follow discussion forums as the perceived most useful information resources.

Notwithstanding these caveats, research to improve forums could significantly aid teachers' goals. For example, better methods to automatically group similar issues – and to alert students to previously posted issues as they type – will help consolidate issues for the benefit of both teacher and student. User interface improvements can also help. At present, some forum tools such as Piazza allow the teacher or students to mark individual issues as resolved but do not make it easy to group a set of posts and mark them as similar and then resolved. A more "dashboard-oriented" view of forum posts, oriented towards the teacher, could be a significant time-savings improvement both for monitoring issues and topics in the forum and for processing posts as they are responded to by the teacher and teaching assistants.

This idea can be taken still further to create more of a "bug report" or "issues tracker" type interface approach to teaching a MOOC, similar to how problems are tracked with software engineering projects. As the teacher or teaching assistants learn about problems – whether via forum, survey, or quantitative metrics such as low scores on a homework problem – the issue could be entered into such an interface. Quick surveys or polls could be issued to see if the perceived gaps or problems are widespread and the results entered into the tool. After the correction is made, the problem could be marked as resolved.

Since surveys are private, those students who are not comfortable posting on the forum may be more willing to express their views in a survey in order to have them taken into account.

Finally, automated methods can be used to find which students appear to be struggling and send them survey questions or encourage them to read the posts on the forum or post their questions, which will then be seen by the teachers.

A potential drawback of the work reported here is it primarily asked teachers about familiar information sources. Researchers are developing very sophisticated log analysis tools (*e.g.,* [38, 44]) that can produce profiles of student behavior that could be surfaced to the teacher in innovative ways. Future work must investigate the efficacy of these approaches.

Another drawback is that MOOC teachers who come primarily from in-person class backgrounds may have preferences for technologies that work well in those environments and against those that do not, such as chat rooms. It may be the case that online chat will work better in MOOCs. More generally, after being exposed to new techniques in action, teachers may form different opinions.

The open-ended comments written by respondents revealed an impressive diversity of views that indicated what is useful to one teacher may not be as useful to another. Teachers ranged from being very pressed for time and wanting to see just a few summary numbers to wanting complex correlational analysis. While some courses are administered with a heavy quantitative evaluation focus, others are more oriented around discussion. The emphasis on student grades in the survey seemed inappropriate to the latter teachers. This diverse range of teacher desires and course styles suggests that what is most useful and effective could be teacher- or course-specific and that the ideal MOOC data visualization should be flexible enough to meet these different needs.

3.8 Summary

This survey of 92 MOOC teachers confirmed the findings of prior surveys of teachers of conventional online courses, which found that teachers value seeing student performance, activity patterns such as what materials students look at, and forum behavior to gauge participation. A standard boxplot of distribution of course grades was perceived to be both understandable and useful by a large majority of respondents, as was a novel design in which stacked bar charts show the number of repeated attempts at solving problems.

However, for those who wish to design visualizations for MOOC teachers, a major takeaway from this work is that views of quantitative measures are insufficient. Rather, teachers believe they need to hear what students have to say – be it from discussion forums, student surveys, or the impressions of their teaching assistants – to achieve their full range of course monitoring goals. Thus future work should focus on how to obtain the thoughts of a wide range of students taking the course, and how to summarize and present this information to the teacher in a useful manner.

Chapter 4

Taking Advantage of Scale by Analyzing Frequent Constructed-Response, Code-Tracing Wrong Answers

4.1 Introduction

The goals of formative assessment are to provide feedback to the student to improve their attainment in cases of error and to inform the teacher as to how to modify or improve pedagogy to address weaknesses in student attainment [12]. In large-enrollment courses, selected-response questions (*e.g.,* multiple choice) are often used as both formative and summative assessment instruments because they can be mechanically graded. However, it is difficult to write selected-response questions whose distractors effectively target common student misunderstandings [64]. Writing constructed-response questions (CRQs), such as filling in blanks, is easier because the teacher does not need to create specific distractors. Moreover, requiring the student to construct a response may also provide richer insight into their level of understanding compared to asking them to identify a correct choice from a list due to the qualitative nature of the data.

However, manually analyzing constructed responses to determine student errors can be prohibitively time-consuming in large-enrollment courses. We considered the wrong constructed responses from code-tracing questions and set about to answer the following research questions:

This work was done with An Ju, Krishna Parashar, Regina Ongowarsito, Nikunj Jain, Sreesha Venkat, and Armando Fox [76].

- **R1**: Can analyzing a small subsample of wrong constructed responses yield information about student difficulties that makes it worth the time investment?
  - **R1.A**: If so, assuming the same questions are used in subsequent course offerings, can the information so gained be applied to future course offerings, further amortizing the time investment?
  - **R1.B**: If so, how should that subsample be chosen and how large must it be?
- **R2**: What insights about student difficulties can be gained from analyzing the subsample?

We address these questions by examining a corpus of 332,829 responses to 92 code-tracing, univalent (having a single correct answer) CRQs administered on a terminal based question-answer, auto-graded system. The data comes from responses by 4,068 students in 3 offerings of a large-enrollment introductory computer science (CS) course. We found that a ≈5% subsample of the most frequent wrong answers covers ≈60% of the wrong constructed responses. The frequent wrong answers are consistent in how much they overlap for a given question set and course offering pair, but the level of overlap varies between question sets and course offering pairs. As a result, frequency should be taken into account when choosing a sample to inspect.

In addition to discovering student difficulties similar to those found in prior work, we also identify new difficulties not reported in prior work such as language-specific constructs and data structures.

After defining terminology and surveying related work, we describe our data set and the emergent coding process we used to analyze it. We then report on how our analysis informs the answers to our research questions. We conclude with a discussion on known and potential uses for our findings.

4.2 Terminology

- *Machine-marked-wrong answer (MMWA)* - A (question, string) pair the automated system marks as incorrect.
- *Wrong answer* - A MMWA that is confirmed incorrect, as opposed to a false positive marked incorrect by the automated system.
- *Response* - A (student, MMWA) pair; many students may give the same MMWA.
- *Tag* - Human experts' interpretation of a specific student difficulty that could lead to an observed student error.
- *Taggable* - A wrong answer is taggable if at least one tag can be applied to it.

4.3 Related Work

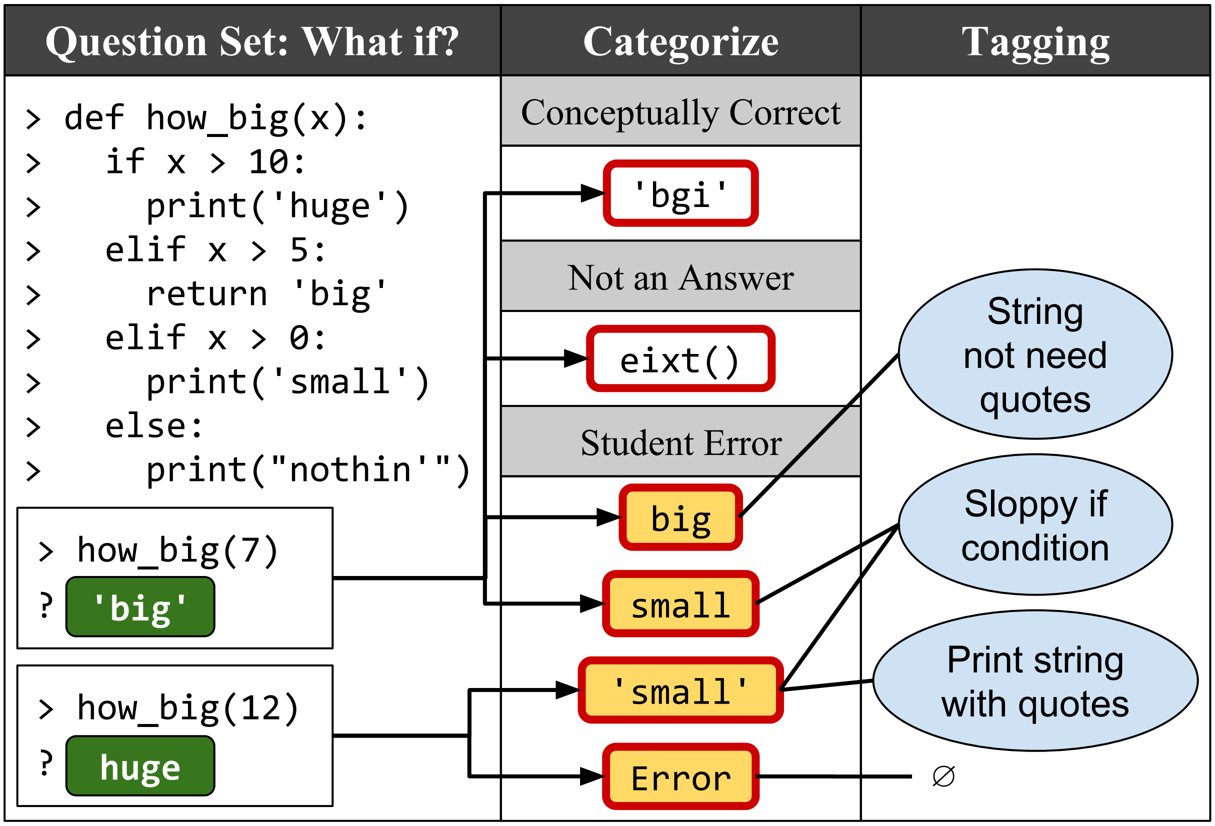
Misconceptions leading to student programming errors have been intensively studied. Clancy [19] reviews potential causes of programming misconceptions and inappropriate attitudes that interfere with learning programming. Sorva [73] provides an extensive catalog of novice misconceptions about introductory programming content. We find similar misconceptions, but also new misconceptions about data structures and language-specific constructs less studied in prior work. In addition, because we focused on *how* students got wrong answers, we also found student difficulties with syntax and sloppy reading or writing.

While machine-gradable, selected-response exercises with carefully-constructed distractors can reveal common student errors, effective distractors are hard to create [64]. This difficulty in creating distractors is especially pronounced in introductory programming courses, where teachers' beliefs about student errors often only weakly correlate with the errors students actually make [15]. Others have therefore attempted to extract information about CS students' misunderstandings from various types of CRQs, such as code explanations [50, 68] or code submissions [41, 46, 58, 66], programming process byproducts such as error logs [18, 39], and univalent CRQs, as in our work. Univalent CRQs are particularly appealing: like other CRQs, they can reveal useful information about student difficulties without requiring the creation of distractors in advance, but unlike other CRQs, they can be easily machine-graded.

The closest work to ours is Sirkiä and Sorva's analysis of students' missteps when using a visual simulation tool for program tracing [70]. Students use the tool to indicate their expectations of each execution step and the tool records every student mistake. The authors identified 200 mistakes each made by at least 10 students, and they analyzed the 26 most popular mistakes to find them to be either a usability-related issue, previously-known conceptual difficulty, or previously-unreported conceptual difficulty. In contrast, our code-tracing questions elicit only a single answer from the student, namely the overall result of running the code. In addition, our *tags*, which code the way(s) students could arrive at a wrong answer, are the equivalent of student mistakes but we allow multiple tags that in combination or separately could cause the wrong answer. Some questions include intermediate `print` statements so we can gain more fine-grained information as the student traces the code. In this regard, both systems encourage students to fix earlier errors before continuing their code-tracing, so we view them as complementary.

Finally, others outside CS have also used the content of wrong answers to evaluate a student's current knowledge. For basic arithmetic problems, Tatsuoka categorized student errors using a two-dimensional Rule Space, with regions of this space representing erroneous rules students use to solve the problems [79, 78]. We instead assign as many tags as necessary to capture the individual or combined student errors that would lead a student to arrive at a wrong answer.

Another way to understand student difficulties is to construct a student model. This is the approach of Repair Theory [14] and systems such as PROUST [40] and MARCEL [74]. Building such models requires enumerating both the student's knowledge and a "bug" list



```
> def how_big(x):
>   if x > 10:
>     print('huge')
>   elif x > 5:
>     return 'big'
>   elif x > 0:
>     print('small')
>   else:
>     print("nothin'")
```

```
> how_big(7)
? 'big'
```

```
> how_big(12)
? huge
```

Figure 4.1: The flow of assigning categories and tags to MMWAs. Left: question set example with two questions on control flow with the correct answers in dark green round-corner rectangles. MMWAs for these questions are in the middle, all with a bold red border. These MMWAs are categorized as: “conceptually correct”, “not an answer”, or “student error”; the top two categories are false positives. Wrong answers with yellow/non-transparent rounded-corner rectangles are either taggable (top three answers have tags, shown in blue circles) or not taggable (bottom answer).

representing ways to mutate that knowledge. While our technique does not result in an explicit student model, it provides insight into common student difficulties without this up-front cost.

4.4 Data Collection and Analysis

Our data comes from three offerings of a large-enrollment introductory CS course. This course teaches programming and the basics of programming abstraction using Python and Scheme. One formative-assessment activity consists of univalent CRQs involving code-tracing: a typical question (Figure 4.1, left) presents 1 – 20 lines of code and asks the student what the interpreter's state will be at various points during execution. A *question* *set* groups questions about a similar topic, for example, lambda expressions.

Data Collection and Preprocessing

The questions are administered through an automatic system running in a terminal window. This system poses each question and prevents the student from proceeding to the next question until the current one has been answered correctly. Unlimited attempts are allowed. Grading is based on completion of the question sets. We record and timestamp every student response; this corpus forms the basis of our data set.

We cleaned the data by removing all blank answers and any duplicate responses made by the same student. Since the questions are answer-until-correct, every student will have the same set of correct responses, so we examine only their MMWAs. Therefore, the following discussion of responses only concerns the MMWAs. We did not do any merging of answers, as we found that fixing common typos such as `'TRue'` for `'True'` barely changed our results.

We collected data from the Fall 2015, Spring 2016, and Fall 2016 offerings of the course. Different teachers taught the fall versus spring offerings, while the large teaching assistant (TA) staffs were more similar between the Fall 2015 and Spring 2016 offerings than the two fall offerings. We report our findings on 11 question sets.

As shown in Figure 4.1, one weakness of the automatic question administration system is its inflexibility in grading, which can result in two types of false positives. A *typo* might be the student entering `'bgi'` rather than the correct answer `'big'`; a human teacher would likely recognize that the student was trying to provide the correct answer. A *mode error* might be a student typing `'eixt()'` rather than `'exit()'` when intending to exit a session. These responses are also marked as wrong, even though the student was not attempting to answer the question at all. In the next section, we explain how we deal with such responses in our analysis process.

Figure 4.2 and Table 4.1 summarize information about the question sets we used, ordered chronologically with some order swapping between course offerings. Although some question sets showed significant variation in the total number of unique MMWAs across course offerings (Figure 4.2) or percentage of students making at least one mistake when tackling that question set (Table 4.1), we find that the properties of the frequent wrong answers remain relatively consistent, as we describe in the next section. A noteworthy outlier is Question Set 1, which has outlier behavior in almost all of our analyses because it tests simple Boolean logic and is, therefore, easier than the other question sets. In addition, in Fall 2016 that question set was optional, and stronger students (who would be more likely to get all the questions correct on the first attempt) may have simply skipped it.

Tagging Process

Three content experts inspected the MMWAs. Two were experts who did well in the course and continued to higher-level courses; the third expert was a former TA for the course.

Question Set	# of Questions	Fall 2015			Spring 2016			Fall 2016		
		Students	Responses	% Students Wrong	Students	Responses	% Students Wrong	Students	Responses	% Students Wrong
1 Booleans	3	1,271	1,337	54	848	1,046	61	1,093	3,135	83
2 Short Circuit	10	1,283	9,570	95	847	5,234	94	1,692	19,032	99
3 if...else	11	1,293	8,145	97	840	5,843	97	1,022	6,901	99
4 Loops	4	1,278	4,744	81	830	3,766	87	1,675	13,103	96
5 Lambdas	12	1,239	23,124	99	827	19,316	99	1,614	37,666	99
6 HOF	6	1,203	9,698	93	783	8,624	96	1,581	27,121	99
7 OOP	5	907	4,146	92	767	4,147	94	1,516	5,431	90
8 OOP	19	1,042	12,344	99	767	10,760	99	1,510	28,199	99
9 Link Lists	9	1,010	5,543	91	753	4,525	92	1,479	6,478	87
10 Scheme Lists	11	1,040	17,050	99	739	12,791	99	359	3,614	100
11 Iterators	2	870	5,834	92	722	4,562	94	-	-	-

Table 4.1: Statistics on the question sets used for this analysis for all course offerings. “HOF” stands for Higher Order Functions and “OOP” for object-oriented programming. “Students” is the number of students attempting that question set; low values are often due to the question sets being optional in certain course offerings. “% Students Wrong” is the percentage of students who made at least one error on any question in the question set.

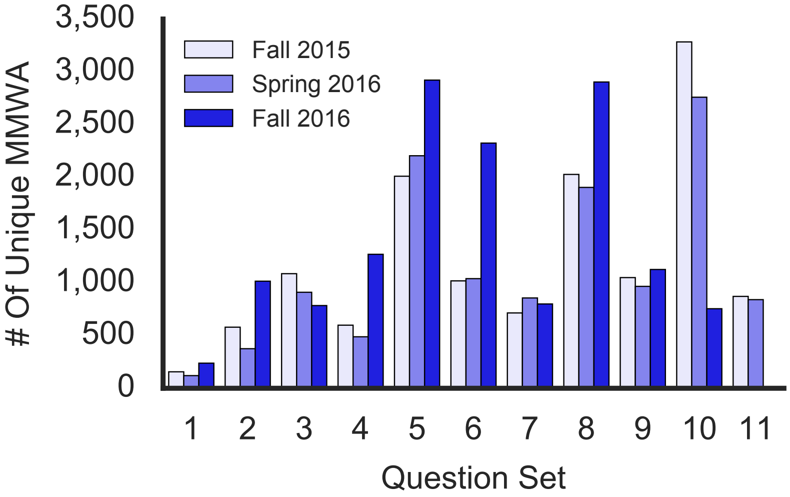


Figure 4.2: Distribution of the number of unique MMWAs for each course offering and each question set. Ex: Question Set 6 had ≈1,000 unique MMWAs for Fall 2015 and Spring 2016 but ≈2,250 for Fall 2016.

For each question set, we first chose a subset of MMWAs. For this subset of MMWAs, we completed two phases with multiple steps each.

Our process uses emergent coding to create and assign the tags [57]. Section 4.5 includes details of how much time each phase took, what MMWAs we inspected, inter-rater-reliability, and results of the tagging process.

Phase 1: Tag Creation

1. Propose Tags: One expert inspects the MMWA set to generate a set of proposed tags with a name, description, and example. (Time here recouped during Phase 2, Step 1 & 2)

2. Finalize Tags: All three experts discuss the proposed list until deciding on a revised list of finalized tags.

Phase 2: Tagging Answer Set

1. Categorize: Two experts separately inspect the MMWA set and assigned each MMWA a category (middle of Figure 4.1).

    - **conceptually correct:** marked wrong due to a typo
    - **not an answer:** marked wrong but was not intended as an answer because of a mode error or misunderstanding what text contains the question.
    - **student error:** conceptual errors and carelessness, further discussed in a later section.

2. Assign Tags: Those answers categorized as "student error" are assigned zero or more tags (right of Figure 4.1).

3. Consolidate/Discuss: The two experts consolidate their assignments into a single set of category and tag(s), discussing until they reach an agreement.

4. Review/Confirm: During initial training of experts, a third expert inspects consolidated assignments and confirms them. If this expert disagrees, there is a discussion among all three experts until they reach an agreement.

4.5 Results

R1: Useful to Examine Small Subset of MMWAs?

Our main findings are that frequent MMWAs appear *much more* frequently than infrequent ones, and that, for most students, most of their wrong answers are in the frequent set. Therefore, inspecting a small subset of these most frequent MMWAs results in good coverage of cumulative responses covered, rapidly decreasing response coverage, and good coverage of an individual student's MMWAs. This section provides details to support the above findings, which affirmatively answer R1. The results presented refer to the Spring 2016 offering, as there was little difference among the three course offerings. We will note differences as appropriate.

Frequent wrong answers are very frequent. Figure 4.3 shows the behavior of the 1,000 most frequent MMWAs. Even though Figure 4.2 shows a wide range in the number of unique MMWAs per question set and per course offering, the cumulative percent of responses covered quickly reaches 50% using no more than the top-100 MMWAs for each question set. In other words, to cover the majority of responses from students, less than 100 MMWAs will need to be inspected per question set, with most question sets needing less than 50. This behavior is confirmed by the percent of students that submit a given unique MMWA quickly dropping to below ≈5% by 100 MMWAs, even though almost all question sets have over 500 unique MMWAs.

For most students, most of their MMWAs are frequent. As a result, if we, use the top-100 unique MMWAs as a simple definition of "frequent MMWAs," we can ask how many students have a given percentage of their MMWAs within that top-100 frequent set. As the CDF in Figure 4.4 shows, for any given question set, 80% or more of the students have the majority of their MMWAs in the top 100. Therefore, most of a student's MMWAs are in the top 100, and the infrequent wrong answers are coming from many students as opposed to a concentrated subset of students. This result gives us greater confidence that by inspecting only the most frequent MMWAs, we are examining at least some, if not the majority of, MMWAs from every student.

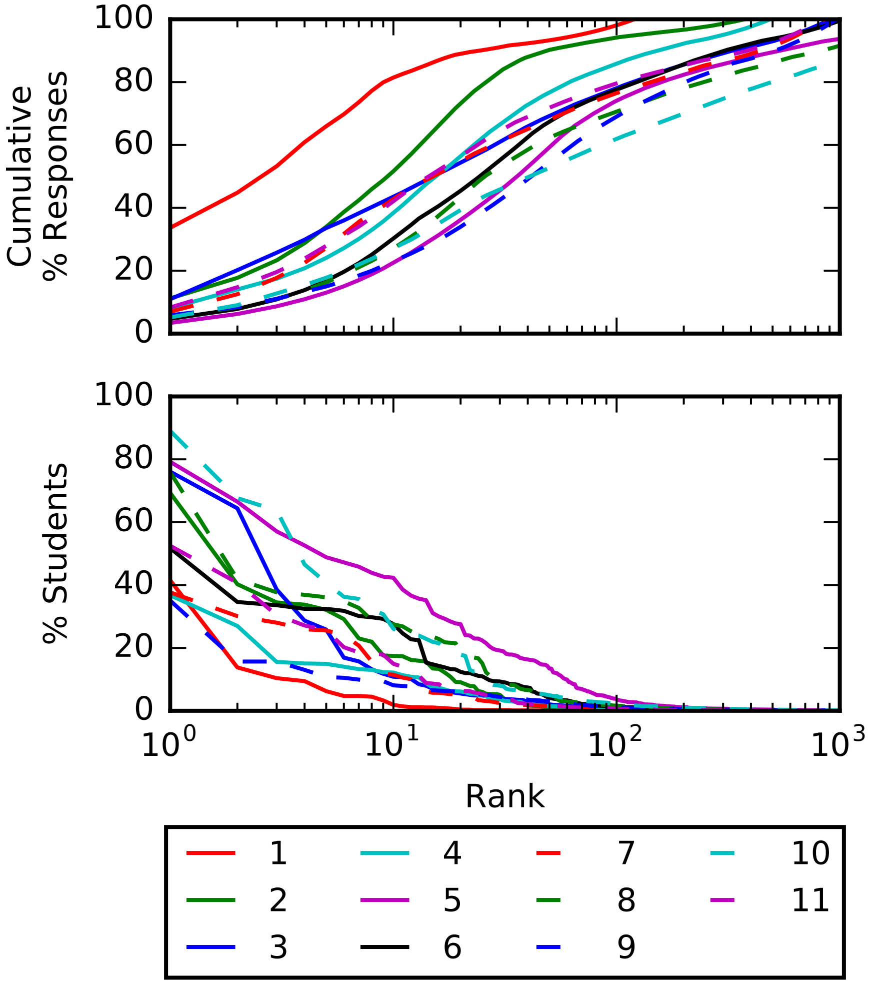


Figure 4.3: Frequency of the unique MMWAs in Spring 2016. Each line represents a question set. The x-axis (log scale) is the 1,000 most frequently occurring MMWAs ordered by frequency. Upper: Cumulative percent of responses covered by up to the Xth most frequent answer, *e.g.,* Question Set 1's top 10 MMWA covered ≈80% of wrong responses. Lower: Percent of students that submitted the Xth most frequent MMWA, *e.g.,* Question Set 11's 10th most frequent MMWA was submitted by ≈42% of the students that submitted a wrong response to this question set.

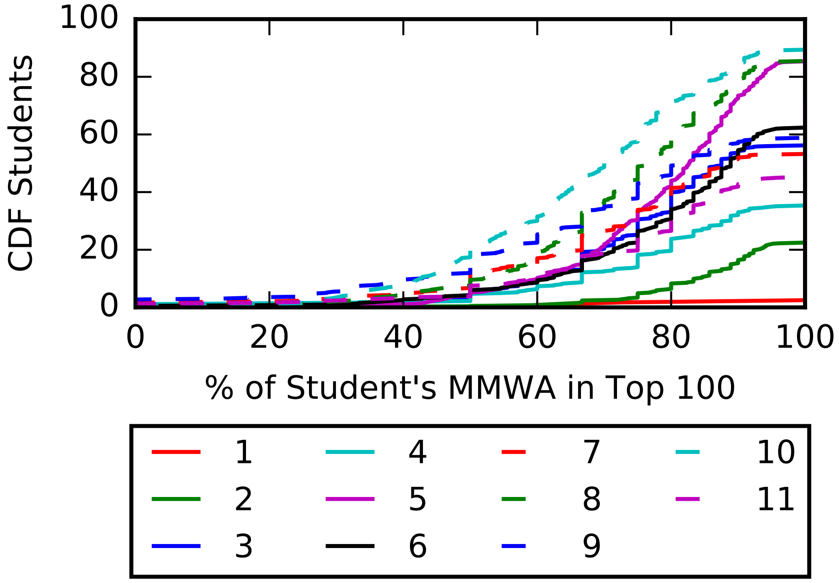


Figure 4.4: CDF showing the percent of students that have at least the x-axis percentage of their MMWAs in the top 100. Ex: For Question Set 10, ≈75% of students have at least ≈80% of their MMWA in this question set's top 100.

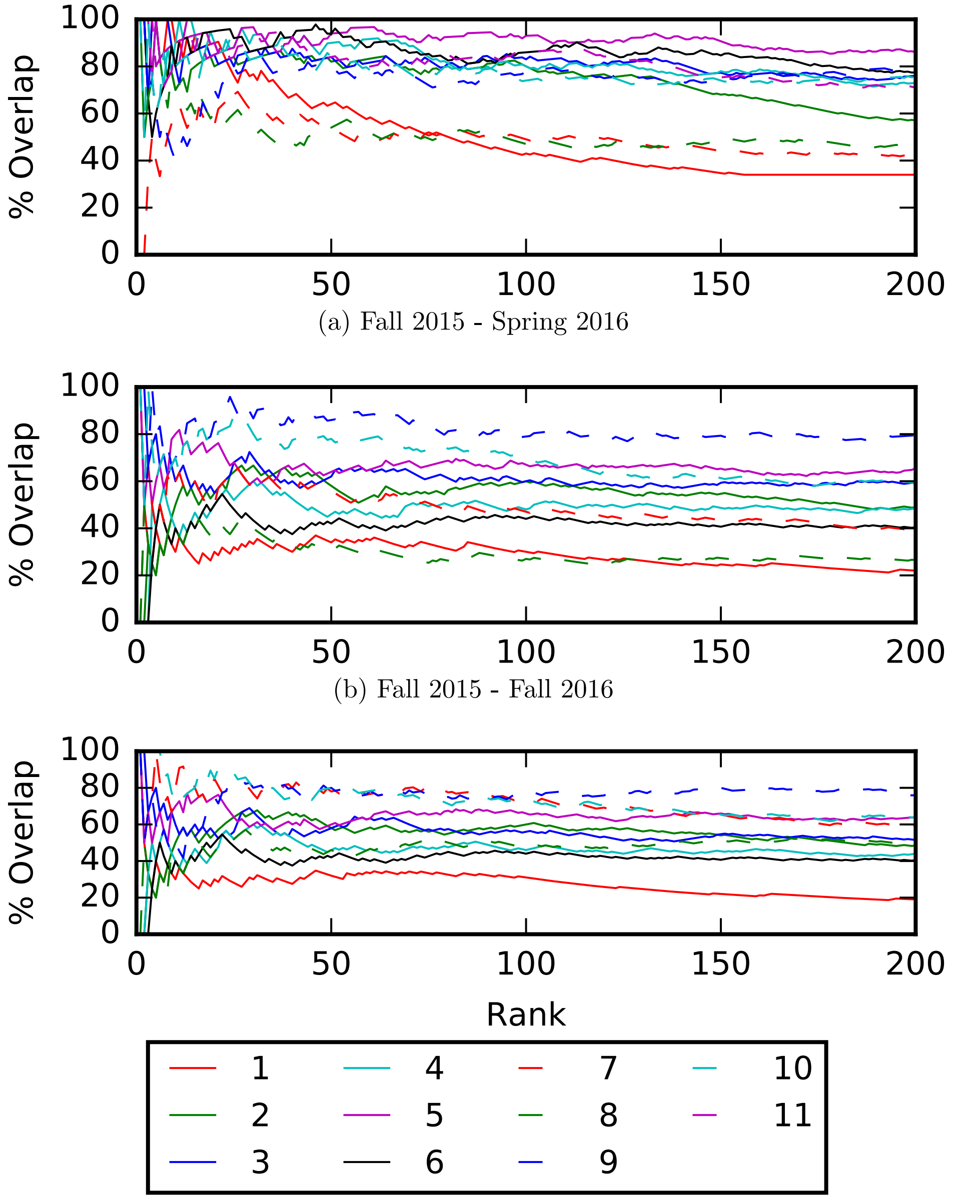
R1.A: Are frequent MMWAs stable across course offerings?

Figure 4.5 shows the overlap of the most frequent MMWAs between a pair of course offerings. (When ordering MMWAs, we broke ties arbitrarily by sorting the answer text alphabetically.) The beginning of the graph is noisy due to a small denominator (the x-axis value). The amount of non-overlapping frequent MMWAs is an estimate of how many MMWAs would need to be inspected in a new offering of the course.

For some question sets, there is a high level of overlap between course offerings. Almost all question sets for all pairs of course offerings stabilize starting at ≈50 MMWAs and stay stable until ≈150 MMWAs or beyond. 150 is well past the number of MMWAs we need to inspect to cover the majority of responses. Therefore, there will always be some tagging that can be reused, but the amount depends on the question set and other factors that are currently unclear.

The differences between the pairs of course offerings is also unclear, primarily because the Fall 2015 and 2016 offerings were taught by the same teacher, yet have lower MMWA overlap than Fall 2015 with Spring 2016. Since the course material did not change, we would expect that a change of teacher would result in lower overlap than the same teacher teaching the same material twice. Our best guess as to why this is happening is that the TA staff between the Fall 2015 and Spring 2016 overlapped much more than with the Fall 2016 TA staff. Each course offering had 50 to 87 TAs versus a single lecturer, so it is possible that a teaching effect happening at the TA level is causing the differences in the MMWAs overlap.

Figure 4.6 shows the result of an empirical Monte Carlo simulation on the Spring 2016's



(a) Fall 2015 - Spring 2016

(b) Fall 2015 - Fall 2016

(c) Spring 2016 - Fall 2016

Figure 4.5: Percent of MMWAs that appear in a pair of course offering's X most frequent MMWAs, *e.g.*, comparing Fall 2015 and Spring 2016, ≈85% of Question Set 6's top-100 MMWAs overlapped between the course offerings. Note: Fall 2016 did not have Question Set 11.

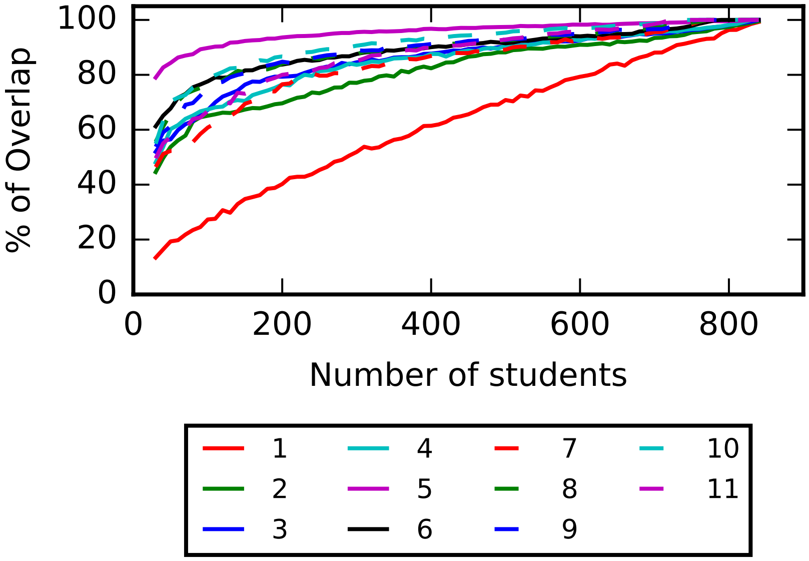


Figure 4.6: Empirical Monte Carlo analysis results when sampling the x-axis value of students 50 times and plotting the mean overlap across the samples with the entire course offering's top-100 most frequent wrong answers for each question set. Ex: For Question Set 5, when we sampled 100 students 50 times, the mean overlap of the top-100 MMWA with the entire cohort's top-100 MMWA was ≈90%.

data using the following steps and the simple definition of the top-100 unique MMWAs as "frequent": (1) sample successive values of N students (x-axis value), 50 times, (2) for each sample compute the overlap of the top-100 most frequent MMWAs between the sample and the entire offering, and (3) plot the mean overlap across the samples.

Figure 4.6 shows high overlap is achieved between 150 and 250 students, with marginal returns afterward. Question Set 1 once again is an outlier, most likely due to how few unique MMWAs it had. Therefore, given our data, frequent MMWAs are stable for a much smaller course than the size we had available (enrollments between 800 and 1,700).

R1.B: How to choose subsample, and how large?

Using taggability (whether or not a wrong answer has a tag) as a proxy for information about student difficulties, our main finding is that frequently-occurring wrong answers are more likely to yield information about student difficulties than rarely-occurring ones, suggesting that the subsample should be created by choosing the most "popular" wrong answers.

We arrived at this conclusion by applying our tagging process to two different MMWA sets. One focused on only the *frequent* MMWAs per question set, hereafter the *FrequentSet*. The other was *all* MMWAs submitted by a subsample of 50 randomly chosen *students* for each question set, hereafter the *StudentSet*. The primary deciding factor for choosing each MMWA set was how much human-expert resources we had to tag these MMWAs. For the

Tagging Step	**Human effort required**
Finalize Tags	≈ 10 mins./tag
Assign Tags	≈ 1.5 mins./answer
Consolidate/Discuss	≈ 0.5 mins./answer
Review/Confirm	≈ 0.1 mins./answer

Table 4.2: Human-expert time required for each tagging step. Total time was ≈87 tagger-hours to create tags and ≈36 expert-hours to tag the FrequentSet. We also spent ≈127 expert-hours to tag the StudentSet, required only for validation and not integral to the technique.

FrequentSet, we chose ≈500 MMWAs to tag; for the StudentSet, after getting a better sense of the resources required, we chose ≈2,000. Table 4.2 summarizes the person-hours required.

For each question set, we ranked the MMWAs by frequency and then used thresholds on two metrics: (1) *total coverage* – which includes the most "popular" MMWAs that cover 60% of all responses, and (2) *marginal coverage* – which then adds further MMWAs (still ranked by frequency) as long as each additional MMWA covers at least 0.4% more responses. The interaction between these metrics (Figure 4.3) led us to explore their value ranges jointly; our parameter values resulted in 508 MMWAs to tag.

Our inter-rater agreements, when categorizing wrong answers in the FrequentSet and StudentSet, were 96.2% and 86.7%, respectively. The fraction of tags given by both experts was 33.2% for the FrequentSet and 46.3% for the StudentSet. The tag overlap is lower than we would like, which could be for three possible reasons. First, since a question set tested a main topic, a misapplied tag was usually misapplied for multiple answers. Second, in some cases, one expert used a more specific tag than the other, for example, "sloppily evaluating a variable" versus "sloppily evaluating an *attribute* of a variable" (an important distinction in Python). During consolidation, we always used the more specific tag. We did not compensate the inter-rater agreement scores for either of these situations. Third, taggers might need more training. The FrequentSet was the first time tagging for our experts, so training occurred while tagging. The increase in agreement between the FrequentSet and StudentSet supports this reasoning, despite there being more wrong answers to tag in the latter.

Despite the low agreement on tags, however, the agreement on categories is high, supporting the assertion that frequent wrong answers are more likely to be taggable. In addition, the primary focus of this work is on whether the wrong answer was tagged (and therefore whether we can gain insights from it), as opposed to which tag(s) it received.

By design, there are many more MMWAs in the StudentSet than the FrequentSet (see Figure 4.7). However, the number of MMWAs per question set is much more varied, making this method of choosing MMWAs more likely to result in higher variability in which MMWAs are chosen. In addition, we found the StudentSet included all the MMWAs in the FrequentSet because (in accord with intuition) we were more likely to randomly choose a student that gave

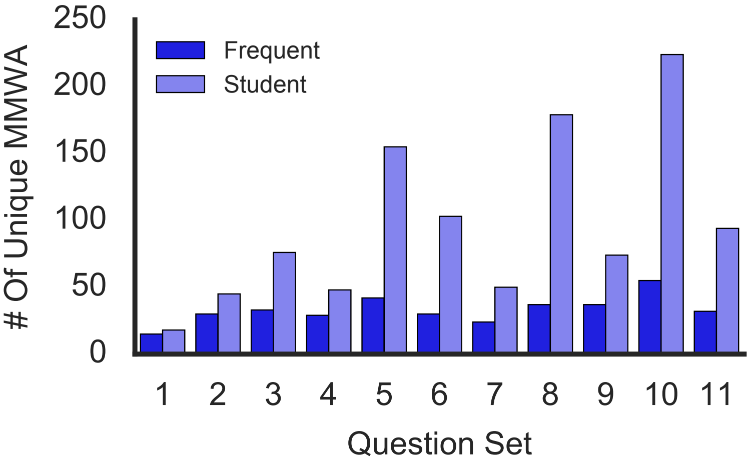


Figure 4.7: Number of MMWAs in each MMWA set per question set. All MMWAs in the FrequentSet also appeared in the StudentSet and therefore are counted in both bars. Ex: For Question Set 6, the FrequentSet had ≈50 MMWA and the StudentSet had ≈250.

	Frequent			Student		
	Not an Answer	Conceptually Correct	Student Error	Not an Answer	Conceptually Correct	Student Error
Mean	1.0%	5.8%	93.1%	4.4%	8.1%	87.5%
Median	0%	4.5%	94.6%	3.2%	4.9%	90.8%
Variance	0.1	0.3	0.4	0.2	0.4	0.4

Table 4.3: Statistics on the % of MMWA per category for the FrequentSet and StudentSet.

a particular frequent MMWA than a specific rare one. Therefore, Figure 4.7 represents the Frequent MMWAs twice, once in the FrequentSet and once in the StudentSet. These findings are further evidence that the frequent MMWAs would be sufficient to yield information on student difficulties that is representative of all MMWAs.

Table 4.3 also shows that across question sets, we categorized more MMWAs as "conceptually correct" or "not an answer" for the StudentSet than the FrequentSet. Since our interests are mainly in MMWAs categorized as "student error" and the FrequentSet yields relatively more of these, we have further support that frequency is a good way to choose which MMWAs to inspect.

Finally, Figure 4.8 shows that the percentage of taggable MMWAs is higher for the frequent wrong answers than the infrequent ones for all but one question set. While the question sets are ordered chronologically, it is unclear why the percentages are converging but not stabilizing. However, since almost all question sets have a higher percentage of taggable wrong answers for the frequent wrong answers than infrequent wrong answers in the StudentSet, we count this as further evidence that frequent MMWAs are more informative than rare ones.

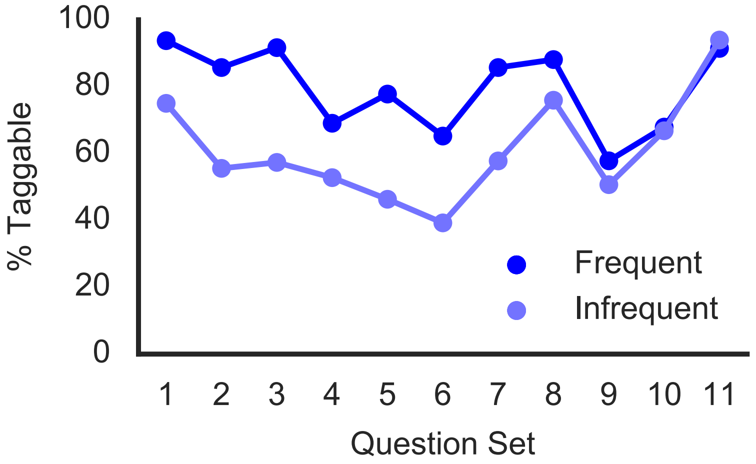


Figure 4.8: Percent of taggable wrong answers between the frequent and infrequent wrong answers in the StudentSet. Ex: For Question Set 6, ≈67% of the frequent wrong answers in the StudentSet were taggable and 40% of the infrequent wrong answers were taggable. Note: All MMWA in the FrequentSet are the frequent wrong answers in the StudentSet.

R2: What insights can be gained from the subsample?

We created a total of 166[2] tags, which is more than the catalog of novice misconceptions provided by Sorva [73]. In addition, 65% of our tags did not fit the topics in the catalog. This lack of fit is because our tags focus on what the student did to *create* the wrong answer, as opposed to the *conceptual idea* the student misunderstood. In Table 4.4 we list the topics from the catalog and ones we created, the number of tags for that topic, and an exemplar tag. We only include exemplar tags due to space. A full list of our tags can be found in Appendix A.

"Language-Specific Constructs" are tags about student difficulties specific to the language, such as idioms, constructs, and implementations. "Syntax" tags focused on ways students were wrong due to syntax errors or misunderstandings. "Sloppy" tags focused on ways students either read the code poorly or submitted their answer without proofreading. Finally, we created the topic "Data Structures" because, even though it is not as well studied in prior work, we found student difficulties with data structures using our analysis. We were able to do this because our data set included questions testing concepts with linked lists, regular lists, dictionaries, sets, and trees.

These insights were gained from question sets created by teaching staff before this work and without a rigorous, data-driven design process. We believe more insights could be gained through an iterative process where current insights inform the design of new questions, whose wrong answers are then analyzed, and hopefully more insights are gained. Such an iterative process we leave to future work.

[2] In the published related work of this chapter [76], the tallies are different due to a counting error. The values in this chapter are correct and do not otherwise affect the results.

Topic	# of Tags	Exemplar Tag		
		Name	Description	Example Code (in Python)

General, Control, OOP, References, Misc	27	Sequential `if` statements are `if...else`	This WA demonstrates that the student believes two `if`'s next to each other are actually an if..else clause	`>>> x = 5`
				`>>> if x <= 5: a = 1`
				`>>> if x > 3: a = 2`
				`>>> print(a)`
				`1`

Variable Assignment	22	Expression not evaluated	This WA demonstrates that the student does not recognize the need to evaluate an expression and instead a code snippet is “passed around.”	`>>> a = 1 + 2`
				`>>> a`
				`1 + 2`

Calls	14	Evaluating a function name is a function call	This WA demonstrates that the student believes when the name of a function is in a line of code (but not called) the function is being called.	`>>> f = lambda x: 1`
				`>>> f`
				`1`

Language Specific Constructs*	34	List comprehension does not return a list	This WA demonstrates that the student believes a list comprehension does not return a list, but just a value.	`>>> [x for x in range(3)]`
				`0`

Syntax*	34	List does not need commas	This WA demonstrates that the student believes a list does not need commas.	`>>> [x for x in range(3)]`
				`[0 1 2]`

Sloppy*	15	Sloppy sorting	This WA is wrong because the student is being sloppy in how they are sorting the values	`>>> sorted(['b','a','c'])`
				`['a','c','b']`

Data Structures*	18	Link lists cannot cycle	This WA demonstrates that the student believes linked lists cannot link back to themselves, so if a line of code does this, it is as if it did not happen.	`>>> l = Link(1, Link(2, Link(3)))`
				`>>> l.rest = l`
				`>>> l.rest.rest.first`
				`2`

Table 4.4: The number of our tags per topic in Sorva's catalog [73] with exemplars. Those with * are topics we created.

Research Question Summary

> **R1**: Can analyzing a small subsample of wrong constructed responses yield information about student difficulties that makes it worth the time investment?

Yes, the frequent MMWAs constitute only a ≈5% subsample of the MMWAs in the data set, yet the wrong answers in the FrequentSet are more likely to be taggable.

> **R1.A**: If so, assuming the same questions are used in subsequent course offerings, can the information so gained be applied to future course offerings, further amortizing the time investment?

How much can be applied to another course offering is currently inconclusive. The amount of overlap for a given question set and pair of course offerings is consistent between the most frequent ≈50-150 wrong answers. However, how much it overlaps between a question set and pair of course offerings is highly dependent on the question set and factors that are currently unclear, such as teaching effects.

> **R1.B**: If so, how should that sample be chosen and how large must it be?

We believe the best way to choose MMWAs is by first ordering them by their frequency and then choosing the most frequent, thresholding based on both the cumulative percent of responses covered and the marginal additional coverage of each additional MMWA. The parameter values can be chosen together based on the number of human-expert hours available for tagging, with the understanding that a lower threshold will affect the results of the set's representativeness and stability.

> **R2**: What insights about student difficulties can be gained from analyzing the subsample?

Using MMWAs from univalent-constructed-response, code-tracing questions with our tagging process, we found both misconceptions similar to those identified in prior work and new misconceptions based on topics appearing in our assessments but not used in prior work, such as difficulties with language-specific constructs and data structures. In addition, we found many ways students can create wrong answers that are not necessarily misconceptions.

4.6 Discussion

It is possible the question quality affects our results. Our data comes from questions that existed before this work, and they were designed by the course's teaching staff without a rigorous data-driven design process. Therefore, we had no control over the question quality. Also, the teaching staff designed these question sets with the intention of students going through them quickly for the sake of giving more class time to work on the code-writing questions. However, even if we improved the question quality, we believe our results are likely a lower bound. Higher quality questions would aim to reveal particular student difficulties. This targeting would then push students to construct a smaller set of unique wrong answers and therefore make the frequent MMWAs even more frequent.

Another concern is the "Sloppy" tags may represent genuine struggles with the material, as opposed to the students not reading the code carefully. Therefore, such errors should be taken seriously despite the potentially belittling name. In addition, we will gladly change the name if future work demonstrates its inappropriateness.

Finally, there is a potential caveat when using this tagging process across course offerings. If the wrong answers tagged in a previous offering of a course do not cover the current offering, we would need to collect and tag more wrong answers. The stability we found across cohorts leads us to believe we will eventually achieve the desired level of tagged frequent wrong answers or the number of wrong answers will never be so high as the initial effort of tagging. However, if the teacher is using the formative assessments to inform changes in teaching strategies, which we hope is the case, she is now tracking a moving target. The target moves because each student cohort is likely changing their common errors based on the instruction received.

4.7 Applications

When we shared our tags with the course's teaching staff, they used the tags to create univalent CRQs for the exams. Another use of the wrong answer taggings is for identifying common student errors in the class to develop targeted distractors for selected-response assessments. Teachers can also use such knowledge to change the teaching materials to target those errors proactively. In addition, we can analyze the MMWAs categorized as "conceptually correct" to reveal how the system is poor at marking answers correctly. These insights can lead to improvements in the automated system or the questions to reduce such errors. For the MMWAs categorized as "not an answer," we can use them to understand ways to improve the system or the questions. For example, this category led us to a confusing question where students were answering a question in the code's comment rather than predicting the code's output.

The tagging data could also become part of a continuous cycle of improving the code-tracing questions. Wrong answers and the tagging process can inform which questions are identifying actual student struggles versus noise by looking at the tags on a question's wrong answers and the ratio of the question's tags to frequent wrong answers. This information can then inform which questions to remove or change, as well as which tags need questions specifically designed for them.

Furthermore, a teacher dashboard reporting the common student errors in the class – informed by the work from Chapter 3 – could help the teacher target which student struggles to focus on when teaching. Such a dashboard could show the entire large class's everyday struggles, as well as filter by a single TA's discussion section or a particular group based on exam performance, similar to Aguilar et al.'s [2] and Hardy et al.'s [34] work. Also, the dashboard could update in real time as students work through the question sets. The real-time updating would enable a TA to monitor her current section's widespread errors and interrupt the entire class when necessary.

Another potential application is from related work on parameterizing the auto-generation of questions [16]. We can use tags to inform auto-question generation by targeting particular tags, as well as auto-generate the wrong answers for those tags. This auto-generation would allow a preliminary understanding of student struggles without first analyzing the student-generated wrong answers.

Finally, we can use wrong answer taggings to develop a model to detect student difficulties as they work through the automatically graded assessments. When the model detects a stable difficulty in the student, it could trigger the delivery of formative feedback in situ.

4.8 Future Work

Pilot study: Directly asking students for answer rationales

Directly asking students why they submitted an answer is a way to improve understanding of those wrong answers. Doing such would help us validate our existing tags, create new tags, and better understand how to apply tags such that we have fewer MMWAs with no tags.

We ran a pilot study asking students for a rationale for their most recent answer. Our prompt was "To help the class provide better hints to future students, please take a moment to explain your answer." We showed this prompt about once per question set after both correct and wrong answers, mainly focusing on correct answers and wrong answers categorized as student error but without tags. The student did not know the correctness of their answer until after they submitted the rationale. Before inspecting these rationales, we first filtered by removing all rationales with fewer than 20 characters.

When we inspected a sample of rationales collected after a wrong answer, we found we were able to apply existing tags to wrong answers that before had no tag, as well as create and apply new tags. In addition, we found three other trends: (1) non-normative generalizations, (2) realization of mistake, and (3) articulation difficulty.

For the first trend, students' rationales indicated they had formed a non-normative generalization about a programming concept. This trend is similar to generalizations found in Clancy's review [19]. These generalizations usually stemmed from an attempt to extrapolate from examples in the teaching material. For instance, a student saw only an example of the boolean `and` returning false when it arrives at the first false statement. This student then extrapolated that `and` only ever returns false and otherwise returns an error.

We identified rationales where students realized their mistake by students beginning to write text explaining their wrong answer then changing to a correct answer or an admission they now understand things differently. These rationales show that prompting for a rationale could also be a means of feedback.

Finally, articulation difficulty rationales are when the student's rationale includes pieces of code or keywords, but the rationale itself is incoherent. Rationales like this could be due to a lack of fluency in the programming language itself or some deep-seated misunderstanding of the concepts such that any explanation they attempt comes out too tangled to be understandable.

For a future experiment, we can conduct think-aloud interviews with students as a case study that will then inform a more systematic collection and study of rationales from students.

Using tagging data to understand learning better

Other potential future work could investigate the relationship between tags and the rest of the course material, such as what tags appear in multiple question sets or how often tags appear together. We can also link a tags appearance to a particular teacher or TA. Such data could bring insight into whether a specific teacher or TA influences what kind of struggles students go through.

We can also take into account the relationship between tags and course material within a student, such as investigating whether a student's common tags correlate with exam score, performance on particular questions, or proficiency on code writing assignments. In addition, tag relationships within a student could reveal if students tend to have specific sets of tags. If these "student tag profiles" exist, then teachers can design material with them in mind.

4.9 Summary

We set out to investigate if the information gained from analyzing responses from univalent-constructed-response, code-tracing questions is worth the effort expended. We analyzed a corpus of 332,829 responses to 92 questions by 4,068 students across 3 offerings of a large-enrollment introductory CS course. We found inspecting the frequent wrong answers are worth the opportunity cost because they are a small sample compared to all the unique wrong answers and cover a majority of the wrong responses. When comparing the overlap of the frequent wrong answers between two course offerings, our results show that the level of overlap is almost always consistent for the frequent wrong answers, but that level varies between question sets and course offering pairs.

In addition, we report on the insights we gained from inspecting these frequent wrong answers. We found similar misconceptions discussed in prior work. Our inspecting process focused on identifying ways students arrive at wrong answers, so we also identified student difficulties with syntax and how students can be sloppy when they read the code or answer questions. Finally, we readily found student difficulties with language-specific constructs

(Python and Scheme for this class) and data structures, areas with less prior work on student misconceptions.

Chapter 5

Delivering Hints on Constructed-Response, Code-Tracing Assessments Based on Wrong Answers

5.1 Introduction

Formative feedback is important. Specifically, we mean "information communicated to the learner that is intended to modify his or her thinking or behavior to improve learning" [67]. In computer science, code-tracing is an important, and potentially necessary, skill that supports code writing [49, 51, 81]. Therefore, we set out to improve student's code-tracing skills by using the insights we gained from inspecting frequent wrong answers from code-tracing questions in Chapter 4 to deliver formative feedback given as hints to students.

However, there is little work examining what kind of hints best helps with learning code-tracing. Therefore, we also investigated the effectiveness of two kinds of hints: reteaching and knowledge integration (KI) [28]. Reteaching hints re-explain the concept(s) associated with what the student is struggling with. KI hints push the student to re-evaluate their answer and understanding to help him overcome his difficulty without any more intervention.

In this chapter, we report on the process and results of conducting an in situ experiment with 1,057 students, answering 9 question sets, during a 15-week course. We built a student error model using the tags, representing student difficulties, from Chapter 4 and split the students into three groups: control, reteaching, and KI. We found:

This work was done with Krishna Parashar, Regina Ongowarsito, Nikunj Jain, Kavi Gupta, and Armando Fox.

- It is straightforward to deploy an intervention experiment at scale by taking advantage of technology already existing in the class.
- That co-occurrence between wrong answers yields useful information about machine-marked-wrong answers without having to inspect them.
- We can build our student error model to identify a student's difficulties using both the results of Chapter 4 and co-occurrence information.
- That we currently do not have sufficient evidence that giving hints to students using this model improves performance on post-test questions.
- Through qualitative techniques, our current experimental setup has Human-Computer Interaction (HCI) problems, allows students to brute-force guess the answer, and lets the model deliver too many hints.
- Through quantitative techniques, there are areas for further investigation, such as students submitting fewer wrong answers after a frequent wrong answer compared to an infrequent one, reteaching hints may help students more than KI hints, and the hint "Typo?" on conceptually correct wrong answers may also help students.

5.2 Related Work

Le et al.'s [47] review of AI-supported tutoring for learning programming noted that most CS tutors give feedback on code writing. Keuning et al.'s [42] review of programming exercise tools found a variety of feedback types that contained information on the assessed task, concepts, mistakes in the code, and next steps.

However, there is mounting evidence that code tracing is an important – if not strictly necessary – skill to learn how to write code [49, 51, 81]. Lopez et al. [51] studied student exam responses and found a correlation between performance on code-tracing tasks and code-writing tasks. They also used step-wise regression with code writing as a dependent variable to create a path diagram and found code tracing to be an intermediary level in the hierarchy with code explaining. Code constructs were at the bottom of the hierarchy. Venables et al. [81] extended and replicated Lopez et al.'s work using their own multiple-choice exam. Venables et al.'s results were broadly consistent in finding that code tracing is an important skill for code writing, with noting some sensitivity to the code-explaining questions. Lister et al. [49] performed a replication studying investigating the relationships between code tracing, code explaining, and code writing. Their findings were consistent with prior work, in that students who cannot trace code cannot usually explain it, while those that can write code usually have the ability to both trace and explain code.

We were able to find two works related to tutor feedback on code tracing. Baffes and Mooney's [7] tool ASSERT used theory refinement to automatically alter a student model to become consistent with a set of examples of a student's behavior. The tool assessed the students on C++ code-tracing using multiple-choice questions that asked students to classify the behavior of a piece of code, such as if it would error out. The feedback had two parts. One was a reteaching message that explained the knowledge altered in the student model to fit the student's answer. The second part was a counterexample pertinent to that knowledge. Kumar's [45] code-tracing feedback tool also used C++ code and assessed student understanding through multiple-choice questions. This tool allowed for assessing students with multi-stage, multiple-choice questions such that subsequent questions are dependent on prior ones. When a student is incorrect, a step-by-step explanation of the code is automatically generated and presented to the student.

These two works barely begin an investigation on the use of feedback on code-tracing assessments. Formative feedback in education literature has been long studied [35, 52, 59, 67]. There are multiple frameworks to inform feedback design and delivery and many factors to consider, which include: the feedback itself, how to deliver the feedback, the assessment the feedback is on, and aspects of the student receiving the feedback.

Our work adds to the current literature regarding the feedback given and the assessment the feedback is on. We use two different kinds of hints: reteaching, which re-explains the misunderstood concepts, and knowledge integration [28], which pushes the student to re-evaluate their understanding of the material. Our tool delivered hints on code-tracing, constructed-response assessments based on the results of analyzing wrong answers to questions in Python.

5.3 Data

We built a student error model that determined when to deliver hints to students on data from prior work in Chapter 4. Please see that chapter for full details. Here are only the details that pertain to Chapter 5.

First, below are terms we will use in this chapter, most of which we introduced in Chapter 4.

- *Machine-marked-wrong answer (MMWA)* - A (question, string) pair the automated system marks as incorrect.
- *Response* - A (student, MMWA) pair; many students may give the same MMWA.
- *Wrong answer* - A MMWA that is confirmed incorrect, as opposed to a false positive marked incorrect by the automated system.
- *Tag* - Human experts' interpretation of a specific student difficulty that could lead to an observed student error.
- *Taggable* - A wrong answer is taggable if at least one tag can be applied to it.
- *Conceptually correct* - A category for MMWAs in the tagging process where it is marked wrong due to a typo.

- *Not an answer* - A category for MMWAs in the tagging process where it is marked wrong but was not intended as an answer because of a mode error or misunderstanding what text contains the question.

- *Student error* - A category for MMWAs in the tagging process where it is marked wrong due to conceptual errors or carelessness.

- *Inspected MMWA* - A MMWA that has gone through the tagging process.

- *Uninspected MMWA* - A MMWA that has not gone through the tagging process.

The data is collected from an automatic, question-answer system running in a terminal, called `OK` [11]. `OK` administers questions in *question sets* that are answer-until-correct, such that the student cannot proceed to the next question until he answers the current one correctly. Students have unlimited attempts per question. At a weekly *lab*, students receive zero or more question sets and receive credit if they complete the question set. We record and timestamp every response submitted by a student.

From work in Chapter 4, we have two sets of inspected MMWAs. Each MMWA in these sets has a category ("conceptually correct," "not an answer," or "student error") and, if categorized as student error, assigned zero or more tags. One is the *FrequentSet*, which consists of each question set's most frequent MMWAs that cover a total of 60% of the question set's responses and all MMWAs that individually cover 0.4% of the responses. The other is the *StudentSet* that includes all MMWAs from a random 50 student subsample per question set.

5.4 Student Error Model

As we showed in Chapter 4, while tagging the frequent MMWAs covers a majority of the data set, it is only ≈5% of all MMWAs. Tagging solely frequent MMWAs means most MMWAs are not inspected and therefore not tagged. Also, inspecting more is not necessarily worth the time and human resources. Therefore, all the uninspected MMWAs a student submits is potentially lost information about that student. With this in mind, we decided to create a model that inferred information about the student using the tagged MMWAs and quantitative metrics about the MMWAs.

Why go beyond the frequent MMWAs?

To explain our rationale for going beyond the frequent MMWAs, we first need to define some methodological assumptions. First, we define a student as having a tag when he gives enough evidence that leads us to believe he is struggling with the difficulty represented by that tag. Second, we designate two wrong answers sharing a tag as sufficient evidence that a student has a tag. Third, we will understand *precision* as "how often the model correctly tags a student" and *recall* as "of the tags each student should have, how many does the model correctly assign." As long as we *do not* try to infer information, our precision will always be 1.0 (or perfect) with these assumptions. However, our recall may be lower than desirable. Therefore, we should take advantage of any additional information to improve our model, such that it allows us to trade some precision for an increase in recall.

In addition, because we are using data from assessments designed without our particular tags in mind, the model may not tag students when the student does have that difficulty. If the student answered more targeted assessments, she might have then provided sufficient evidence to show she has a tag. However, we cannot determine this with our current data set. Therefore, even if we make some inferences as to what tags students have, the tags we assign are likely a lower bound of what tags the student has.

Tag propagation using co-occurrence

Tag propagation in this context means within a student, as opposed to a wrong answer. As we mentioned above, we require a student to have two wrong answers that share a tag to indicate the student has the tag. We chose two wrong answers because it allowed for students to slip – where they have the knowledge to answer correctly but submitted a wrong answer – and not receive a hint prematurely. How many wrong answers to ideally require we leave to future work. We use co-occurrence, inspired from pattern mining, to propagate tags. The co-occurrence metrics we considered include coherence, cosine, and Kulczynski explored in Wu et al.'s [85] pattern mining work, as well as the odds ratio.

Our model with tag propagation applied tags to students at the lab level. We calculate at the lab level (1) because question sets within a lab are usually related and (2) to take advantage of all data within a student's code-tracing session. The model applied a tag to a student if the student has:

- *Primary rule*: Two wrong answers that share the tag OR
- *Propagation rule*: There is only one wrong answer for the tag and an uninspected MMWA who's co-occurrence metric with the tagged wrong answer is above a threshold.

To understand how well co-occurrence could help our model, we compared it to an uninformed baseline. The uninformed baseline replaced the co-occurrence threshold in the propagation rule with a probability to apply the tag to the student. We compared the models by plotting their precision vs. recall curve and calculating a modified area under the curve (AUC). For each model, we used the FrequentSet as the training set and the StudentSet as the test set.

To plot the curve for the uninformed baseline, we varied the probability at 200 evenly distributed points between and including 0 and 1. We then took the mean precision and recall from running the model 100 times per probability value and removed any precision or recall points on the curve that caused it to backtrack on itself. To plot the curve for the co-occurrence models, we varied the threshold. To calculate the precision and recall we defined a positive tagging as the model properly applying a tag to a student in the StudentSet and

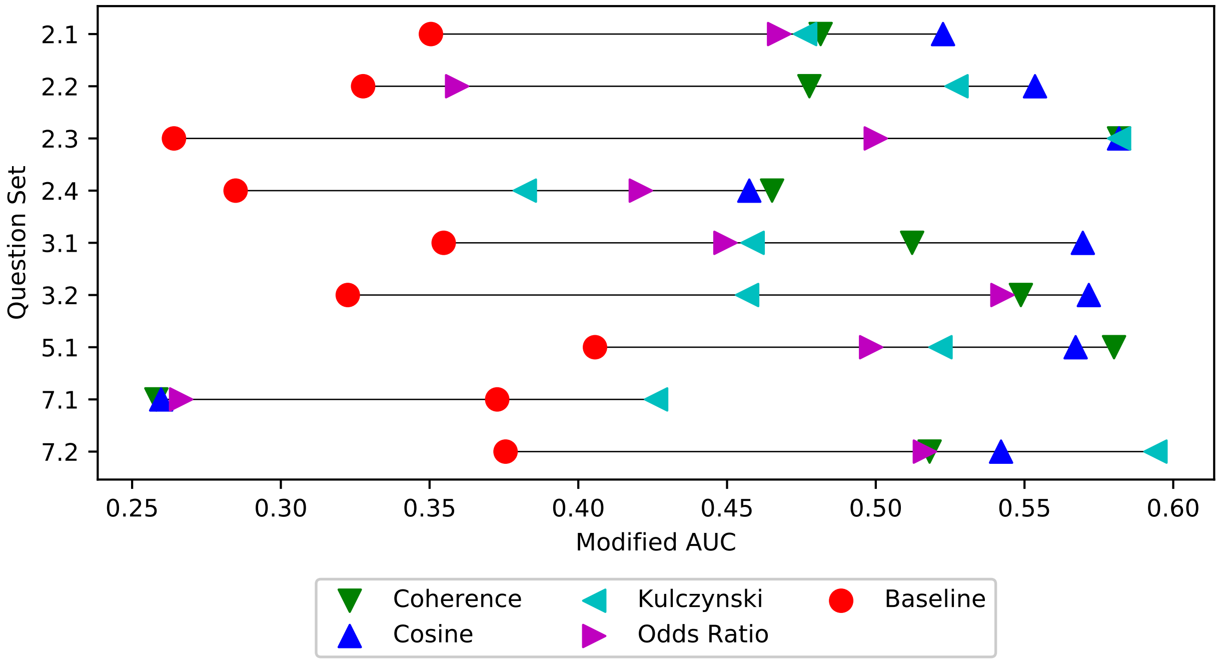


Figure 5.1: Dot plot comparing the modified AUC for each question set across the metrics and baseline. The decimal number for each question set represents first the lab it was in (left of the decimal) and its order in the lab (right of the decimal). Note: The x-axis range is not 0 to 1.

a negative as properly not applying a tag to a student in the StudentSet. Also, we give a positive and negative assignment only within the context of the tags known to appear in the question set(s) under consideration. For example, if we were considering two question sets and 20 unique tags appear in these question sets, each student will have a positive or negative assignment for all 20 tags. We judged the model on whether it correctly marked (true/false) each student's 20 tags as positive or negative. We modified calculating the AUC because the primary rule and our definition of how a student receives a tag caused high initial precision and recall values. For example, the Question Set 7.2 had precision and recall values between [0.64, 1.0] and [0.71, 0.90] respectively. Therefore, we defined our modified AUC by first anchoring each curve at its extreme points and then scaling the curve such that it fits inside a 1x1 square.

Figure 5.1 shows a dot plot comparing the modified AUCs for each question set. A question set's number represents the week on the left of the decimal and the order on the right of the decimal. From this figure, we can see all models for all question sets except Question Set 7.1 outperform the uninformed baseline. This outperformance leads us to conclude that co-occurrence does yield useful information on what tags to apply to a student. However, when comparing the ranking of the metrics for each question set, we find there is no consistent ordering. For example, Kulczynski is usually one of the weaker performing metrics, except for Question Sets 7.1 and 7.2, our two Object Oriented Programming (OOP) question sets. For Question Set 2.3, coherence, cosine, and Kulczynski tie for first place. In addition, 4 out of the other 6 question sets have cosine outperform all the metrics, but coherence is better for the other 2. This lack of consistent ordering leads us to conclude the best metric to use in the model is question set specific.

Therefore, to create a model for a question set we took the following steps:

1. Choose the metric with the best modified AUC,

2. Calculate the f2-score for all threshold values for the chosen metric, and

3. Use the threshold with the best f2-score

We decided to use the f2-score after comparing the resulting model when using the f1-score and f3-score. Intuitively, the number for an f-score means we value recall that many times more than precision. When we used the f1-score, our model rarely chose to propagate because the f1-score equally weighted precision and recall. With this equal weight and precision is 1.0 without propagation, there was no other combination of recall and precision with a better f1-score. In comparison, the f3-score usually resulted in a precision lower than desirable.

5.5 Kinds of Hints

After the model tags a student, we start giving that student hints associated with that tag. Concretely, we associate hints with tags or (tag, wrong answer) pairs. When a student provides a wrong answer associated with a particular tag and the model has also tagged that student, the student receives the hint(s). If the current wrong answer is uninspected, the student receives the hint(s) the other inspected wrong answer would receive. As for hints, we decided to compare two different kinds of hints: reteaching and knowledge integration. We include a full list of the hints in Appendix B for reteaching and Appendix C for KI.

Reteaching hints

The main kind of hint used in CS learning is a reteaching hint, which re-explains a concept (*e.g.,* short-circuiting) or an idea (e.g. syntax error) to the student because the student usually has been taught the concept already. To give students reteaching hints, we first associate a tag with one or more concepts. We then write a reteaching hint for each concept. We wrote a total of 75 reteaching hints.

For example, we have a tag to indicate that students have "swapped the meaning of boolean `and` with `or`." We associated three concepts to this tag: boolean `and`, `or`, and short-circuiting. When the model assigns this tag to the student, he would get three hints at once, as seen below. We placed `|`'s or bars around text meant to be code because the text is displayed in a terminal, where we could not make any assumptions about formatting.

Boolean `and`:

```
|and| either returns the first false value evaluated
in an expression or the last value of that expression
if all the operands are true.
```

Boolean `or`:

```
|or| either returns the first true value evaluated in
an expression or the last value of that expression if
all the operands are false.
```

Boolean Short-Circuiting:

```
In short-circuiting Python only cares about false-y
values while evaluating an |and| expression, and truth-y
values while evaluating an |or| expression. A
|FALSE-Y_VAL and VAL| will always evaluate to |False|,
and a |TRUTH-Y_VAL or VAL| will always evaluate to
|True| so as soon as one of these values is known, the
other subexpressions is superfluous, and therefore
Python does NOT evaluate them.
```

Knowledge Integration (KI) Hints

We based our Knowledge Integration (KI) hints on Gerard et al.'s work on science learning [28]. Their hints had four main components: (1) *elicit ideas* - make an observation about the response to connect to the student's initial ideas, (2) *add and distinguish ideas* - Ask question about the key missing or non-normative concept in the response, (3) *add and distinguish ideas* - Direct student to material about the question asked, and (4) *integrate ideas* - Ask student to use the evidence they have gathered to improve their response.

Our KI hints are a variant suited to the task of code-tracing. These hints push the student to re-evaluate their understanding without relying upon another explanation. We intend the hints to encourage students to overcome their struggles on their own, rather than to re-explain the concepts we suspect the student is having difficulty with. This hint design potentially makes KI hints more robust than reteaching hints because KI hints are less likely to over-diagnose a student's difficulty compared to targeting a particular concept. When writing such hints, we: (1) remind the student of one or more ideas or (2) ask the student to compare two or more ideas. An idea, in this case, we define as a concept, segment of code, syntax, and anything else having to do with code-tracing. We wrote and assigned hints to (tag, wrong answer) pairs. However, most tags have only one hint associated with them regardless of the wrong answer, and some tags share hints. We wrote 115 KI hints.

For example, for the "swapped the meaning of boolean `and` and `or`" tag, the KI hint would be:

```
        Remember the difference between |and| and |or|.
```

5.6 Experimental Setting

We ran our experiment during the 2017 15-week Spring semester in the same CS1 course as Chapter 4. This course's weekly schedule included 2 lectures by the teacher, 1 discussion with the lecture by a teaching assistant (TA) and with exercises, and 1 lab with self-paced computer exercises and a TA and multiple lab assistants (LAs) present.

We administered hints during lab in the code-tracing question sets delivered by the automatic system, `OK`. Labs usually covered material taught in the most recent lectures that students have not yet used in practice. The teaching staff designed the code-tracing question sets to be the student's first interaction with the new material, and they are meant to be easy. We had 9 question sets spread over 4 labs. Question Sets 2.3 and 2.4 were not required.

Changes to the automatic system

We ran the experiment by making only small modifications to the existing technology in the class, `OK` [11]. First, for each lab assignment's downloadable zip file, we added a data file with the model and hints. We included this data file with the assignment because students only downloaded the assignment once and we wanted to ensure they had the necessary information if they worked offline. Second, we altered `OK` to read this file whenever a student opened a question set. Third, we also altered `OK` to contact a server to retrieve the student's treatment group when the student started working on the lab assignment. Finally, as students submitted wrong answers, `OK` updated the model and delivered hints as appropriate. All alterations were straightforward and required little effort compared to the resources used to create the technology.

Treatment groups

We had three treatment groups: *control* with no hints, *reteaching*, and *KI*. Students were randomly assigned a treatment at the beginning of the course and kept the same treatment throughout the course. Due to a bug in our deployment, about three times as many students were in our control group than in each hint group. This bug resulted in 630 students in the control group, while reteaching had 215 students and KI had 212 students. There was 1 student assigned to multiple treatments due to an error in creating student accounts in the system. We removed this student from the data set.

Question sets and models

To create the model that identifies when to give students hints, the model's parameters of co-occurrence metric and threshold were first determined per question set. The chosen

Question Set	Metric	Threshold	AUC	F-score	Precision	Recall
2.1 Short Circuiting	Cosine	0.082	0.523	0.930	0.746	0.990
2.2 Loops	Cosine	0.130	0.554	0.879	0.823	0.895
2.3 Booleans	Cosine	0.189	0.582	0.938	0.938	0.938
2.4 if...else	Coherence	0.020	0.465	0.712	0.582	0.754
3.1 Lambdas	Cosine	0.229	0.570	0.862	0.744	0.898
3.2 Higher Order Functions	Cosine	0.204	0.572	0.815	0.745	0.835
5.1 List Comprehension	Coherence	0.039	0.580	0.851	0.842	0.853
7.1 OOP	Kulczynski	0.504	0.426	0.918	0.871	0.931
7.2 OOP	Kulczynski	0.503	0.594	0.835	0.734	0.865

Table 5.1: Each question set's chosen model parameters and statistics. The decimal number next to each question set represents first the lab it was in (left of the decimal) and its order in the lab (right of the decimal). Row lines separate question sets by lab.

parameters and relevant statistics can be found in Table 5.1. These parameters indicated which wrong answer pairs had a high enough co-occurrence with each other. Then, when administering hints, the model took into account all wrong answers regardless of their original question set. This disregard for the origin of the question set means that if a student submits a wrong answer with a tag during Question Set 7.1 and another wrong answer with the same tag but in Question Set 7.2, the student will receive a hint immediately after the latter wrong answer. Since we only considered the co-occurrence within a given question set, the propagation rule only worked within it.

Question sets had two subsets: intervention and post-test questions. First, in the *intervention questions* – the original set of questions designed by the teaching staff – students could receive hints. Second, in the *post-test questions*, students would not receive hints. We placed these questions after the intervention questions. Since the question sets are answer-until-correct, the students only answer them after receiving our hint intervention. We designed the post-test questions to surface the tags that appeared in the intervention questions.

The question sets had a total of 114 questions with 77 questions during intervention and 37 questions for the post-test. Question Set 7.1 did not have post-test questions because it covered the same topic as Question Set 7.2, OOP. In addition, students were highly encouraged to complete it before Question Set 7.2. All students that did Question Set 7.1 also did 7.2.

Question Set	In Treatment	Not in Treatment	%
2.1	969	86	8.9
2.2	954	84	8.1
2.3	373	35	8.6
2.4	348	34	8.9
3.1	932	49	5.0
3.2	812	48	5.0
5.1	880	25	2.8
7.1	879	12	1.3
7.2	871	11	1.2

Table 5.2: The number of students in treatment per question set and the amount lost due to missing information or computer error.

5.7 Results

1,057 students attempted at least one question set and 275 students that attempted all the question sets. 689 students did all seven required question sets. When we split the students that did all required question sets by treatment group, control had 411 (65.2%), reteach had 145 (67.4%), and KI had 133 (62.7%) students. These numbers seem low because we lost students per question set due to missing information (where we could not determine if they correctly received treatment) or computer error (where the treatment server did not respond to them). Table 5.2 shows the number of students in each question set that did and did not receive treatment. From this table, we can see the student loss per question set is similar within a lab and therefore assume this was a uniform problem across the data.

Measuring performance

As far as we can determine, there is no consensus at present as to how to best measure student performance on answer-until-correct questions [6, 5, 83]. The following is the method we used.

We use two different values to measure a student's performance on a set of questions. First, we take into account the difficulty D_i of question i as 1 minus the fraction of students that correctly answered question i on the first attempt. This definition of difficulty means that the higher the value, the more difficult the question. Second, for a student, we have the binary value $C_i = \{0, 1\}$ of whether the student correctly answered question i on the first attempt. Then, we define the performance of a specific student for a set of questions Q as:

$$\frac{\sum_{i \in Q} C_i \times D_i}{\sum_{i \in Q} D_i} \tag{5.1}$$

In other words, a student's performance on a set of questions can be defined as the questions they correctly answer on the first attempt divided by the questions they attempted with the questions weighted by their difficulty. Our method was designed with several factors in mind. First, weighting by the question's difficulty rewards students for answering harder questions correctly. Second, using whether the student answered correctly only on the first attempt emphasizes what the student knew initially without any confounds, such as what the student learned from any submitted wrong answers. Third, we divide by the weighted sum of all the questions in the set to ensure that we only take into account the questions the student attempted. By using only these questions, we purposely did not penalize students for not attempting a question since it is unclear what such an action indicates. The majority of students finished each question set they attempted since the students received credit based on whether they completed it. As a result, the data shows reasonable consistency in the sets of questions done (Q) between students.

No hints versus any hint

Before investigating the difference between reteaching and KI hints, the first question we considered is whether receiving any hint was better than receiving no hint. To do this, we combined our reteaching and KI treatment groups into one "any hint" group. Then, we performed a Welch Two Sample t-test between the control group, which received no hints, and the any-hint group on each student's performance on the post-test questions. The test resulted in $p = 0.64$, meaning a null result. As a result, we cannot reject the null hypothesis that there is no difference in performance on post-test questions between getting no hints and getting any hint. Since there is no statistically significant difference between these two groups, there was no need to investigate further distinctions between our two hint types.

Considering student prior knowledge

One potential explanation for the null result is that a student's prior knowledge affects his performance on the code-tracing questions. We defined prior knowledge as a student's performance on the earliest question sets in the course in Week 2's lab. Week 2's lab included the first four question sets, two of which were optional. We were able to calculate a prior for 957 students by using our previous Equation 5.1 on all the questions in these four question sets. We confirmed that there was no difference between the priors of our no-hint and any-hint group using another Welch Two Sample t-test, which resulted in $p = 0.62$. Finally, we ran an Analysis of Covariance (ANCOVA) with our two groups, the prior as our covariate, and the dependent variable the performance on all post-test questions not in Week 2. This resulted in $p = 0.47$. Once again, we received a null result.

5.8 Exploratory Analysis of Data

To understand why our results are null, we performed an exploratory analysis of our data using both qualitative and quantitative techniques. We qualitatively analyzed wrong answer sequences for a question set from students that gave long wrong answer sequences and received many hints. Our quantitative analysis involved defining metrics for groups of wrong answers or hints and comparing these metrics between groups.

Preliminary qualitative analysis

To qualitatively assess why our results are null, we inspected a sample of long wrong answer sequences within a student and question set. We chose our sample by taking the (student, question set) tuples with the highest count of hints and wrong answers. Through this process, we made three discoveries.

Human-Computer Interaction (HCI) problems in the hints and automatic system that cause wrong answers. We knew the system had some HCI problems from our conceptually correct MMWAs. However, we found more in our case study analysis, and while these problems likely do not cause a severe misunderstanding in a student, they probably cause frustration. An HCI problem caused by our hints were wrong answers that were correct except included `|` or bars, such as "`|False|`". Our hints likely caused this confusion because we used `|`'s in our hint text to denote code.

A chief system HCI confusion was that the keywords to indicate if an error occurred or Python would display nothing or a function pointer. The keywords students used to answer correctly were "Error," "Nothing," and "Function." Students instead would submit the name of the error that would occur instead of "Error," "`None`" instead of "Nothing," and the name of the function instead of "Function."

Another system HCI problem is a lack of text canonicalization, which is the main reason we categorize a MMWA as conceptually correct. These kinds of MMWAs potentially slow student learning because a student may change his conceptual understanding to a non-normative understanding when told he is wrong. In one instance, a student submitted a conceptually correct answer "function," but without proper capitalization, the system marked it wrong. This student then submitted 19 more wrong answers before answering the question correctly. In addition, the lack of submission canonicalization caused a problem with our hints. We found the system counted minor differences in formatting as separate wrong answers such that the model could deliver hints too early. For example, a student submitted "`1/0`" and "`1 / 0`" separately and received a hint on the second submission. While it is true that both of those wrong answers were frequent and given the same tag, it does not necessarily make sense for a student to receive a hint after submitting two wrong answers that are canonically the same answer.

Brute-force guessing by students can create long wrong answer sequences. A primary cause of long wrong answer sequences was students systematically guessing the answer. We found sequences of the numbers 0 to 10, different orderings of the three answer keywords

("Error," "Function," and "Nothing"), and different variations and combinations of pieces of problem text. For instance, consider the following sample of a student's sequence of wrong answers. The correct answer is "123":

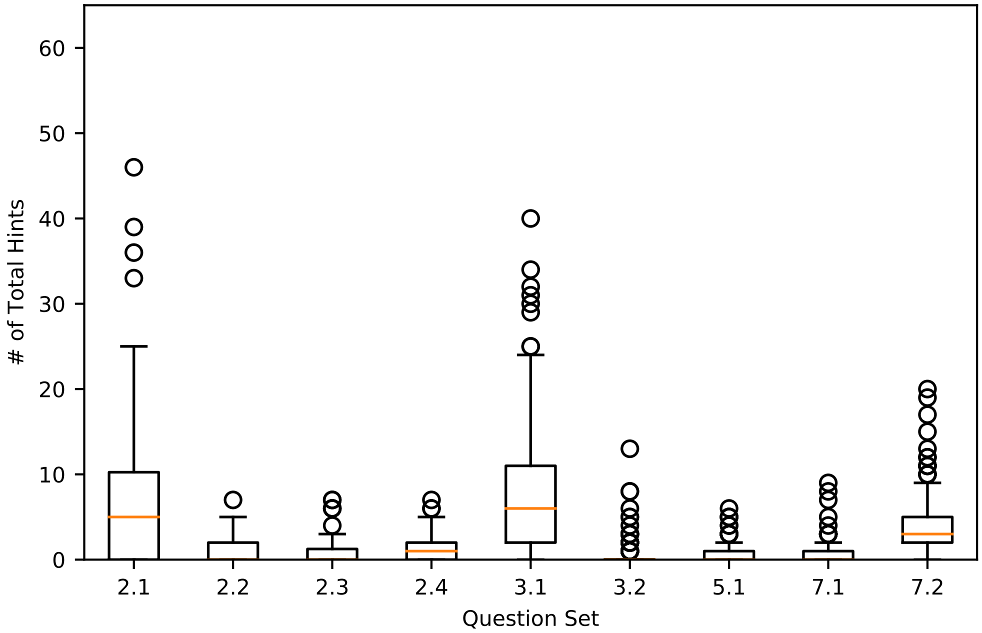
```
>>> c = lambda x: lambda: print('123')
>>> c(88)()
88
Function
Error
Nothing
0
print
c
()
print 88
lambda: print('123')
'123'
```

The model can propagate too aggressively, such that uninspected wrong answers received many tags and therefore many hints. For example, a student in the reteaching condition submitted "0" to the question "`True and 1 / 0 and False`". The correct answer is "Error". The model propagated so many tags that the student received six hints on the concepts: erroring when dividing by zero and Boolean `and`, `or`, short-circuiting, primitives, and order of operations. For the KI condition, we had a student receive ten hints after the below question, answer pair.

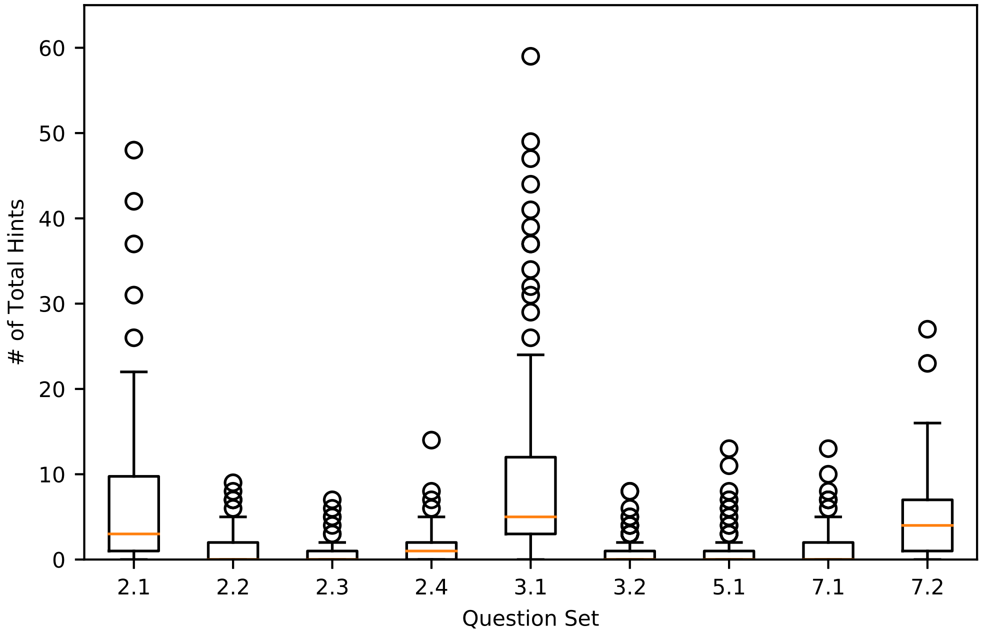
```
>>> b = lambda x, y: print('summer')
>>> c = b(4, 'dog')
b
```

The number of hints for a single answer in the KI condition especially surprised us because we never intended more than one KI hint to appear for a wrong answer.

To gain a simplified understanding of the number of hints given to each student for each treatment, we created Figure 5.2. The most striking feature here is that students in the reteaching condition (Figure 5.2a) receive fewer total hints compared to KI students (Figure 5.2b). This difference is also present when counting the number of unique hints per student. There are two likely reasons this happened. First, we have more unique KI hints than reteaching hints. Second, we assign reteaching hints per concept and KI hints per (tag, wrong answer) pairs, even if a KI hint is usually the same for a tag regardless of the wrong answer. This hint assignment means that for a given set of tags the set of reteaching hints are the unique concepts among those tags and the set of KI are almost always one hint per tag. In addition, the model propagates tags within a question set, which covers a small set of concepts. Therefore, if the model propagates many tags to a wrong answer in the question



(a) Boxplot number of total hints received by reteaching group



(b) Boxplot number of total hints received by KI group

Figure 5.2: Boxplot showing the distribution of the number of hints each student received per question set. Note: Since the data is integral, outlier points may represent more than one student.

	Count	**% 0**	**% 1**	**% >1**
Wrong Answer Set				
Frequent Control	32,536	41.7	20.3	38.0
Frequent Reteaching	10,338	43.5	20.3	36.3
Frequent KI	11,324	41.5	20.3	38.1
Infrequent Control	17,651	32.6	19.8	47.6
Infrequent Reteaching	5,791	33.3	20.0	46.8
Infrequent KI	6,758	30.5	18.9	50.6
Hint Set				
Reteaching	4,072	49.1	18.1	32.7
KI	4,473	42.6	20.2	37.2
Correct/Typo	64	84.4	9.4	6.3
Reteaching range function	76	64.5	18.4	17.1
KI range function	68	57.4	23.5	19.1
KI short circuiting	44	56.8	27.2	15.9
KI for stacked calls cause errors	52	23.1	15.4	61.5
Reteaching Assignment	188	26.1	17.1	56.9

Table 5.3: Statistics when looking at the number of wrong answers after a (student, question, event) tuple, where the event is from a set of hint(s) or wrong answers. The table's lines separate different types of sets. For example, of the 4,072 times the system delivered a reteaching message, 49.1% of the time a student answered correctly immediately after, 18.1% of the time a student submitted a wrong answer and then submitted the correct answer, and 32.7% of the time there was more than one wrong answer until the student was correct for that question.

set, it is reasonable for the tags' concepts to overlap. This overlap results in fewer total reteaching hints compared to one KI hint per tag.

Preliminary quantitative analysis

To start a quantitative understanding of why our results are null, we calculated the number of wrong answers after a (student, question, event) tuple. The events were the occurrence of a hint or wrong answer from a set. We then created summary statistics of these counts, see Table 5.3. The first column describes what makes up the wrong answer or hint set. The second column is the number of times a (student, question, event) tuple occurred. The last three columns are the percent of those counts where after the event there were no wrong answers (*i.e.,* the student's next answer was correct) "% 0", one wrong answer "% 1", or more than one wrong answer "% >1." For example, for the first row, there were 32,536 instances of a (student, question, frequent wrong answer) tuple. Of those events 41.7% had the correct answer after it, 20.3% had one wrong answer after it, and 38.0% had more than one wrong answer after it. Note, the way we are counting the tuples means that a frequent wrong answer (an event) could happen multiple times within a (student, question) pair. Below are some interesting potential findings for further investigation.

Students submit the correct answer in fewer wrong answers after a frequent wrong answer compared to infrequent wrong answers. The two groups in the upper part of the table are the statistics of the wrong answer sets for the frequent and infrequent wrong answers split by each treatment group. The counts for the control group are three times as high because there were three times as many students in that group. When comparing the "% 0" column, there is an average of 10.1% more for a frequent wrong answer than an infrequent one across all three treatment groups. This increase seems almost entirely taken from the "% >1" column.

One potential explanation for this increase is the brute-force guessing we saw in our case study. Most of the wrong answers when a student is brute-force guessing are infrequent wrong answers. These infrequent wrong answers mean that if a student is brute-force guessing they are submitting many infrequent wrong answers for a single question. Therefore, a given infrequent wrong answer is likely to have many wrong answers after it.

Reteaching hints may help students submit fewer wrong answers than KI hints. In the lower table are different hint sets. The first group is the statistics when a student received a reteaching or KI hint. By comparing these statistics to our frequent wrong answer set's statistics, one can see that reteaching hints may help students submit fewer wrong answers, while KI hints make no change. However, further work is needed to evaluate this possibility. We need to compare these statistics to the same set of statistics for the control group where a student would have gotten a hint but did not.

A hint suggesting the student has a typo for conceptually correct wrong answers considerably increased the "% 0" column. For both the reteaching and KI treatment groups, we delivered the same hint for conceptually correct wrong answers that simply read "Typo?" Comparing the statistics for this hint to the general reteaching and KI hint groups indicates that it drastically reduced the number of wrong answers after such wrong answers. Once again, however, we do not know if this is causal without comparing it to the control group.

In this case, there are several reasons why we believe it would be beneficial to show this hint for every conceptually correct wrong answer. First, we only delivered this hint 64 times, despite the many conceptually correct wrong answers in the data set. Delivering so few of these hints most likely happened because a particular student did not submit enough conceptually correct wrong answers and the model was too strict by requiring two conceptual correct wrong answers. A second reason to always show the "Typo?" message is evident in the HCI problem case study we discussed earlier in which a student submitted a conceptually correct MMWA and then submitted more MMWAs after he was told he was wrong and did not receive the typo hint.

Some hints decrease and others increase the "% 0" column. The next two groups of hint sets are the statistics for single hints. The first group is hints that decreased the number of wrong answers after the hint and the second group increased the number compared to the general statistics for each hint set. The difference is especially apparent when comparing the percentages in the "% >1" column to the overall reteaching and KI hint set. Our other hints spanned the values between these hints.

It is currently not clear why specific hint types performed better than others. The performance variance may be because our three exemplar “good” hints are more targeted than our two exemplar “bad” hints. However, there are other hints that are just as specific but which did not perform as well. Moreover, when comparing between reteaching and KI hints for specific tags, sometimes the hints perform similarly and other times not. Further work is needed to determine the properties of effective hints for both types, especially work that compares the results with the control group.

5.9 Discussion

Our results further understanding of the complexity of providing hints to students on constructed-response, code-tracing assessments. While our overall results on improving student performance on post-test questions are null, we did find hints that seemed to help students since when the hints are present there are fewer attempts on the question. Potential reasons why our overall result is null include:

- We need a different performance metric to best measure changes in student performance, such as methods found in related work [6, 5, 83].
- There is a topping out effect because the questions are too easy, which was a design goal for the teaching staff when they wrote the original questions that we used as intervention questions.
- We did not have enough post-test questions to measure the effect of the hints. The teaching staff limited the question sets’ lengths. Therefore, to give students sufficient questions during the intervention portion, we had few post-test questions and designed the questions to test for multiple tags at once.

Another potential reason our results are not as successful as related work is due to differences between our work and related work. One of the most similar related works with KI guidance is Vitale et al.’s [82] work comparing specific guidance – which is like our reteaching hints – with KI guidance. They found that specific guidance improves student outcomes on short-term assessments but not on a transfer task. KI guidance had the reverse effect: it improved student outcomes on the transfer task but not on the short-term assessments. A transfer task was out of scope for our experiment. However, a potential and readily available transfer task is the code-writing assessments immediately after the code-tracing question sets. If our experiment included an analysis of these assessments, we might have found a significant difference since a student must be able to trace code after writing it. Moreover, related work has found that code-tracing is an important skill that supports code-writing [49, 51, 81].

Besides scope, our KI hints and the learning setting differ from related work that used KI guidance [28, 77, 82]. As mentioned above, we did not include all four KI hint parts found in Gerard et al.'s work. Also, some of our hints are vague compared to Vitale et al.'s [82] work. For example, we have hints that merely suggest rechecking how a keyword works as opposed to Vitale et al.'s hints that guide a student to attend to a particular detail. Our learning settings are also different. Most work with KI guidance is in K-12 classes and has the teacher present and guiding students through the curriculum. Our work is in an undergraduate class that is much bigger than a single K-12 class and has students with greater autonomy regarding how and when they receive teacher help and guidance. These differences combined with Tansomboon et al.'s [77] findings that teachers can influence the effectiveness of a particular kind of guidance could explain why our KI hints did not improve student outcomes overall.

Finally, our KI hints may not have worked since learning how a scientific system works – the common learning domain for KI guidance – is not similar enough to learning a mental model of how computers work or a notional machine [72]. The two learning domains are similar in that both are complex systems and, when learning these systems, students bring their mental models and ideas of how they work. However, the specific details of the two learning domains differ. The former focuses on how the physical world works and encourages the use of the scientific method while the latter requires computational thinking. These differences between the two might cause too large of a divergence to successfully transfer the success of KI hints to the learning domain of computer science.

Regarding our reteaching hints, there are also experimental setting differences in our work compared to related work. Kumar [45] tested his tool in a lab setting. Baffes and Mooney's [7] experimental design had students use the ASSERT tool voluntarily, and they incentivized students by giving extra credit and having the tool cover final exam material. We conducted our experiment in situ on required class material. These differences may be why our reteaching hints are not as successful as these related works.

5.10 Future Work

There are two main areas for future work: a further investigation to understand our current results and how to improve the system or experiment.

Understanding our results

As in our exploratory analysis, there are both qualitative and quantitative techniques we can apply. A qualitative technique to better understand if our hints are understandable is user studies using the think aloud protocol. With user studies, we will investigate if the hint wording is confusing or misleading. We would focus on whether the student correctly understands the hint as they think aloud and whether the hint causes them to correctly understand the concept, for reteaching hints, or to re-evaluate their understanding, for KI hints.

Quantitatively, there are still more metrics we can mine from the existing data to tease out what happened. For example, if we look at the time between answers, we can gauge if students actually read the hints. We may find student behavior similar to that in Heckler and Mikula [36], who observed students initially read their explanations but stopped after a few questions. In addition, this would lead us to more student response sequences that we can study qualitatively. Another potential metric is comparing or running regressions between a student's performance to her number of hints, tags, question sets, and questions. Such a regression could help us better understand if there is a dosage effect.

We will also investigate other performance metrics. For example, Ahadi et al. [3] measured performance by using the $\log_2$ of the number of attempts by a student. This metric could replace whether the student correctly answered a question on the first try, as well as influence a question's difficulty metric.

Another area for future work would be to tag the wrong answers in the post-test questions and then compare the tags that the model assigned to the student during the intervention questions with the student's tags in the post-test questions. If the two tag sets are the same, it could mean the hints did not reduce the difficulties represented by those tags. If the tags from the post-test questions are strictly a reduced subset of the tags in intervention, it could mean the hints did help but not enough to improve performance in the post-test questions.

Improving the system or experiment

Improve HCI: From our qualitative case studies, multiple HCI problems need addressing. With user studies, we can find a better way to mark what is code in our hints. In addition, we can investigate more intuitive ways for students to submit when they think the answer will be an error or Python will display nothing or a function pointer.

Disincentivize brute-force guessing: As evident in our case study, at least some long wrong answer sequences are due to students brute-force guessing. Possible ways to limit guessing is rate limiting how many answers a student can submit within a minute. Another option would be that if a student submits over a threshold of wrong answers or the system detects guessing using what we found in our case studies, the system would prevent the student from submitting more answers until a TA or LA unlocks his session.

Improve question sets: Now that we have wrong answers from questions specifically designed to target our tags (the post-test questions), we can begin a cycle of improving the question sets. In this cycle, we would first remove intervention questions that had few taggable wrong answers. Then, we would tag the wrong answers from the post-test questions. We move those post-test questions that proved effective at surfacing tags to the intervention questions and their tagging data added to the model. Finally, we would design new post-test questions for tags without questions in the post-test question. In addition, we would discuss with the teaching staff ways of increasing the number of post-test questions.

Comparing the current model with a stricter model: In this experiment, we used only one model to deliver hints to students. To better understand if it is the propagation affecting student performance, we need to compare it to a model that does not propagate. This model would only use the primary rule to apply tags to students.

Use a student's entire history of wrong answers: Rather than using only the wrong answers in the current lab, we can include the student's whole wrong answer history. Expanding what we consider would require taking into account how long ago each wrong answer appeared and whether the student had the opportunity to give a wrong answer for that tag.

Modify when hints are delivered: In our exploratory analysis, we found students that submitted fewer wrong answers after receiving a hint pointing out they may have a typo on their conceptually correct wrong answer. However, this hint was not delivered often because the model was too strict. Therefore, for the conceptually correct wrong answers, we should always give students this hint.

Another finding in our exploratory analysis was the model sometimes propagates too many hints at once. We can mitigate this by adding another layer to the propagation logic so that if there are too many propagated tags, we would propagate only the tags with the highest co-occurrence metric, thus reducing the number of hints delivered.

Measure prior knowledge outside of the question sets: Defining student prior knowledge with the early question sets is confounded because students also received hints during that time. Therefore, we should measure prior knowledge either by not delivering hints in the early question sets or using a validated test in prior work, such as Tew and Guzdial's [80] or Park et al.'s [63] work.

Investigate more kinds of formative feedback: Related work has found that giving students the correct answer after they attempt a question improves outcomes [25, 24], but what about answer-until-correct? We could view answer-until-correct as an extension of the answer once and receive the correct answer when wrong because the student does eventually reach the correct answer, but she is generating it herself. In a future experiment, we could compare answer-until-correct with hints versus answer once with correct response (knowledge of correct response - KCR) and explanation. Answer-until-correct could have no hints and the hints from this work. KCR could have only the correct answer, a line-by-line textual explanation, or a step through code visualization. An additional variation would be to give the students the option to see the explanation even if they are correct.

5.11 Summary

We set out to investigate ways to deliver hints to students based on information about their wrong answers, as well as comparing two different kinds of hints. To do this, we altered existing technology in the course to deliver an in situ experiment with 1,057 students as they answered 9 question sets during a 15-week course. We also conducted a further analysis of a corpus of 332,829 responses from a previous offering of the course, originally from Chapter 4.

We found that altering existing scaled technology for our experiment was straightforward, especially when compared to the resource requirements to build the original technology. Our analysis of the data corpus from Chapter 4 found that co-occurrence yields useful information about machine-marked-wrong answers without individually inspecting them. We then used this information to create a student error model to determine when to deliver hints to students. However, we were not able to find a difference in student performance on post-test questions between our treatment groups.

Our exploratory analysis to understand our null results led us to find: (1) HCI problems, (2) that students create long wrong answer sequences due to brute-force guessing, and (3) our model is sometimes too aggressive in delivering hints to students. In addition, we found preliminary evidence that: (1) a student is more likely to get a question correct after submitting a frequent wrong answer than an infrequent one, (2) students correctly answer a question in fewer wrong answers after a reteaching hint than a KI hint, (3) the simplistic hint "Typo?" for conceptually correct wrong answers potentially drastically reduces the number of wrong answers after it for that question, and (4) both reteaching and KI vary in how many wrong answers the student submits between receiving the hint and submitting the correct answer.

Chapter 6

Future Work

We believe that a scaled class has distinct advantages for student learning. Explicitly, we define a scaled class as having far more than the typical class of 30 students and technology embedded in the class to help manage and enhance the class. It is important to study ways to handle large, technology-filled classes because they are part of our present and will continue to be part of our future. We *need* to investigate what can and cannot be scaled.

Rather than seeing scaled classes as a problem that needs management, we believe they are an opportunity to improve learning. Scaled classes have richer data regarding the data size, due to the number of students, and data dimensionality, due to the technology as potential data sources. This data can help us better understand learning with fewer confounds, such as teacher and cohort differences, as well as give us the statistical power to find class trends that before would look like outliers. In addition, the technology required for scaled classes provides a ready test site for innovation. Finally, the large class size enables us to find ways to enhance teaching through techniques like automation.

The objective of our work is not to design technology that can entirely replace the human teacher. Rather, we believe a scaled class can help student learning by informing and enabling the building of “intelligently designed systems that leverage human intelligence” [8]. Through analyzing this rich data, we can enhance the teacher’s understanding of the class. Then we can enhance teaching by using the insights of the analysis and the embedded technology.

In this work we learned:

- That despite the massiveness of this rich data, teachers want to understand their qualitative data, rather than only their more easily accessible quantitative data. (Chapter 3)

- It is possible to analyze this rich data using a mix of qualitative and quantitative techniques. (Chapter 4)

- It is straightforward to deploy an intervention experiment at scale if we use the already embedded technology. (Chapter 5)

- Hints to improve learning are complicated. (Chapter 5)

With these lessons in mind and the belief that scale can help a class, we believe there are many possibilities for future work.

6.1 Data Sources to Apply Mixed Methods

Office hours management app data

One way to manage office hours with a large class is with an app [23, 71]. Such apps collect an assortment of data, from simple queue arrival/removal timestamps to advanced information like student identity, the TA that helped the student, and the student's question.

We can enrich our understanding of this data by studying it within the qualitative context of the course's material, deadlines, exams, logistics, and mishaps (*e.g.,* a bug in the project), as well as individual student performance. Questions we could study include: (1) How do we better utilize TAs in office hours? (2) Are there better ways to organize office hours? (3) Does office hours usage correlate with improved performance in class? (4) What are the typical questions in office hours that could become part of non-office hours resources, such as "Frequently Asked Questions" or class tutorials?

Gradescope

Gradescope [69] is an online assessment grading tool for handwritten assignments and exams. It is a web-based platform that enables teaching staff to grade assessments using a dynamically evolving rubric per assessment. This data includes an association between a student, his assessment answer, and the rubric items awarded to that answer. If the teacher creates rubric items not only with grading in mind but also data analysis, we could measure student exam performance on specific concepts and link such information to the rest of the class's data. This information cohesion could lead to better understanding of the entire class and each student's mastery of the material. We could also investigate the correlations between concept mastery and other course material use and performance.

6.2 Rich, Robust Insights from Large Data

Measuring how long it takes students to complete work

The number of credits for a class often indicates the class's workload. However, there are few forcing mechanisms to keep a class's workload within its number of credits. This lack could be because it is hard to measure how much students work on a course. We could use how long a student spends on course material as a rough estimate for course workload. We could calculate such an estimate if the class has most course assignments administered through technology with timestamped log data. Then with this information, a teacher could calibrate his course material to match the expectations of the course's workload.

Use TA staff timings to predict student timings

A heuristic for predicting how much time students need for an exam is that the student will take at most three times as long as a teacher. We can empirically investigate this by measuring how long TAs take to complete the exam and recording how much time each student spends on her exam. Furthermore, we could apply this process to any course material, including: question sets, projects, homework, and coding problems. We could collect student timings actively – by asking students to submit them – or passively – by using the log data of the technology used to administer the material. Then once we know the empirical multiplier for the material, we could apply it to new material before exposing the material to students and, therefore, have a reasonable estimate of how long the new material would take a student to complete.

6.3 Deployable Scaled Experiment

A data-driven flipped class

Xie and Yang [86] compared providing students with a video of an expert solving a problem and explaining critical information such as decision-making, reasoning, and common errors versus providing an overall score and highlighting the student's errors. They found that the video improved student performance more than the overall score and student error highlighting. However, such videos may be too time-consuming to create, or there may be too many potential key pieces of information to explain in a single lecture. Instead, we could use the insight from this work to create a data-driven flipped class. We would organize the class by first collecting and automatically grading and assessing student work for the common principal information students are struggling with. During the lecture, the teacher would focus on this key information while modeling how to solve both problems from the students' work and new problems to show ways to transfer understanding. Finally, we could measure whether this new approach improved student outcomes.

Chapter 7

Conclusion

We define scaled classes as having both a large student body and a high usage of technology. We believe these scaled classes are an opportunity to explore how to effectively teach large classes without sacrificing quality, as well as learning in general. In this work, we present an instantiation of a research process in which we: (1) use the existing technology to collect highly dimensional data, (2) use quantitative and qualitative techniques to analyze the data, and (3) use the insights from the analysis to improve our understanding of learning and design pedagogically grounded intervention experiments.

We started this work by surveying MOOC teachers to understand how these teachers valued information sources similarly and differently than small online class teachers (Chapter 3). We found that MOOC teachers wanted qualitative data, despite the prohibitively large number of students. Next, we analyzed qualitative data from a scaled, local class and found that, despite the large data set, inspecting constructed-response wrong answers with both qualitative and quantitative techniques is not necessarily prohibitively resource intensive (Chapter 4). Moreover, we were able to gain valuable insights from this analysis that informed our last work. In our final work, we used the insights to deliver hints to students on constructed-response, code-tracing questions and found that it was straightforward to run the experiment by taking advantage of the existing scaled technology and that hints are complicated (Chapter 5).

We believe that teachers and students can greatly benefit if researchers take advantage of scaled classes. Our proposed research process is one way to do so, and this work presents one instantiation of this research process. We hope that it inspires others to do the same.

Bibliography

[1] *2016 Fall CS61A 001 Lec 001.* `http://classes.berkeley.edu/content/2016-fall-compsci-61a-001-lec-001`. Accessed: 2017-05-05. 2016.

[2] Diego Alonso Gómez Aguilar, Roberto Therón, and Francisco José García Peñalvo. "Semantic Spiral Timelines Used as Support for e-Learning." In: *Journal of Universial Computer Science* 15.7 (2009), pp. 1526–1545.

[3] Alireza Ahadi, Raymond Lister, and Arto Vihavainen. "On the Number of Attempts Students Made on Some Online Programming Exercises During Semester and Their Subsequent Performance on Final Exam Questions". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education.* ITiCSE '16. Arequipa, Peru: ACM, 2016, pp. 218–223.

[4] Assoicated Press. *Monstrous class sizes unavoidable at colleges.* `http://www.nbcnews.com/id/21951104/ns/us_news-education/t/monstrous-class-sizes-unavoidable-colleges/`. Accessed: 2017-05-05. 2007.

[5] Yigal Attali. "Immediate Feedback and Opportunity to Revise Answers". In: *Applied Psychological Measurement* 35.6 (2011), pp. 472–479.

[6] Yigal Attali and Don Powers. "Immediate Feedback and Opportunity to Revise Answers to Open-Ended Questions". In: *Educational and Psychological Measurement* 70.1 (2010), pp. 22–35.

[7] Paul Baffes and Raymond Mooney. "Refinement-based student modeling and automated bug library construction". In: *Journal of Interactive Learning Research* 7.1 (1996), p. 75.

[8] Ryan S. Baker. "Stupid Tutoring Systems, Intelligent Humans". In: *International Journal of Artificial Intelligence in Education* 26.2 (2016), pp. 600–614.

[9] Ryan SJD Baker and Kalina Yacef. "The state of educational data mining in 2009: A review and future visions". In: *JEDM-Journal of Educational Data Mining* 1.1 (2009), pp. 3–17.

[10] Rebecca Barber and Mike Sharkey. "Course Correction: Using Analytics to Predict Course Success". In: *Proceedings of the 2Nd International Conference on Learning Analytics and Knowledge.* LAK '12. Vancouver, British Columbia, Canada: ACM, 2012, pp. 259–262.

[11] Soumya Basu et al. “Problems Before Solutions: Automated Problem Clarification at Scale”. In: *Proceedings of the Second (2015) ACM Conference on Learning @ Scale.* L@S ’15. Vancouver, BC, Canada: ACM, 2015, pp. 205–213.

[12] John D Bransford, Ann L Brown, and Rodney R Cocking. *How people learn: Brain, mind, experience, and school.* National Academy Press, 1999.

[13] LB Breslow et al. “Studying learning in the worldwide classroom: Research into edX’s first MOOC”. In: *Research & Practice in Assessment* 8 (2013), pp. 13–25.

[14] John Seely Brown and Kurt VanLehn. “Repair theory: A generative theory of bugs in procedural skills”. In: *Cognitive science* 4.4 (1980), pp. 379–426.

[15] Neil C.C. Brown and Amjad Altadmri. “Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data”. In: *Proceedings of the Tenth Annual Conference on International Computing Education Research.* ICER ’14. Glasgow, Scotland, United Kingdom: ACM, 2014, pp. 43–50.

[16] Peter Brusilovsky and Sergey Sosnovsky. “Individualized Exercises for Self-assessment of Programming Knowledge: An Evaluation of QuizPACK”. In: *Journal on Educational Resources in Computing* 5.3 (Sept. 2005).

[17] Jane E Caldwell. “Clickers in the large classroom: Current research and best-practice tips”. In: *CBE-Life sciences education* 6.1 (2007), pp. 9–20.

[18] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. “The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior”. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research.* ICER ’15. Omaha, Nebraska, USA: ACM, 2015, pp. 141–150.

[19] Michael Clancy. “Misconceptions and attitudes that interfere with learning to program”. In: *Computer Science Education Research* (2004), pp. 85–100.

[20] Derrick Coetzee et al. “Should your MOOC forum use a reputation system?” In: *Proceedings of the 2014 Conference on Computer-Supported Cooperative Work.* 2014.

[21] Stephen Cooper, Wanda Dann, and Randy Pausch. “Alice: A 3-D Tool for Introductory Programming Concepts”. In: *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges.* CCSC ’00. Ramapo College of New Jersey in Mahwah, New Jersey, USA: Consortium for Computing Sciences in Colleges, 2000, pp. 107–116.

[22] Kathleen Cotton. *Monitoring student learning in the classroom.* Northwest Regional Educational Laboratory, 1988.

[23] *CS 61A Queue.* `https://oh.cs61a.org/`. Accessed: 2017-10-07. 2016.

[24] Roberta E Dihoff, Gary M Brosvic, and Michael L Epstein. “The role of feedback during academic testing: The delay retention effect revisited”. In: *The Psychological Record* 53.4 (2003), p. 533.

[25] Michael L. Epstein, Beth B. Epstein, and Gary M. Brosvic. "Immediate Feedback during Academic Testing". In: *Psychological Reports* 88.3 (2001), pp. 889–894.

[26] Rebecca Ferguson. "Learning analytics: drivers, developments and challenges". In: *International Journal of Technology Enhanced Learning* 4.5-6 (2012), pp. 304–317.

[27] Elena Gaudioso, Felix Hernandez-del-Olmo, and Miguel Montero. "Enhancing e-learning through teacher support: two experiences". In: *Education, IEEE Transactions on* 52.1 (2009), pp. 109–115.

[28] Libby F Gerard et al. "Automated guidance for student inquiry." In: *Journal of Educational Psychology* 108.1 (2016), p. 60.

[29] William J Gibbs, Vladimir Olexa, and Ronan S Bernas. "A visualization tool for managing and studying online communications". In: *Educational Technology & Society* 9.3 (2006), pp. 232–243.

[30] Murray W Goldberg. "Student participation and progress tracking for web-based courses using WebCT". In: *Proceedings of the Second International NA WEB Conference*. 1996, pp. 5–8.

[31] Murray W Goldberg, Sasan Salari, and Paul Swoboda. "World Wide WebCourse tool: An environment for building WWW-based courses". In: *Computer Networks and ISDN Systems* 28.7 (1996), pp. 1219–1231.

[32] Shuchi Grover, Roy Pea, and STephen Cooper. "Promoting Active Learning & Leveraging Dashboards for Curriculum Assessment in an OpenEdX Introductory CS Course for Middle School". In: *Learning @ Scale, Work in Progress*. ACM. 2014.

[33] Christian Hardless and Urban Nulden. "Visualizing learning activities to support tutors". In: *CHI'99 extended abstracts on Human factors in computing systems*. ACM. 1999, pp. 312–313.

[34] Judy Hardy, Mario Antonioletti, and S Bates. "e-learner tracking: Tools for discovering learner behavior". In: *The IASTED International Conference on Web-base Education*. 2004.

[35] John Hattie and Helen Timperley. "The Power of Feedback". In: *Review of Educational Research* 77.1 (2007), pp. 81–112.

[36] Andrew F Heckler and Brendon D Mikula. "Factors affecting learning of vector math from computer-based practice: Feedback complexity and prior knowledge". In: *Physical Review Physics Education Research* 12.1 (2016), p. 010134.

[37] Neil T. Heffernan and Cristina Lindquist Heffernan. "The ASSISTments Ecosystem: Building a Platform that Brings Scientists and Teachers Together for Minimally Invasive Research on Human Learning and Teaching". In: *International Journal of Artificial Intelligence in Education* 24.4 (2014), pp. 470–497.

[38] Jonathan Huang et al. “Syntactic and Functional Variability of a Million Code Submissions in a Machine Learning MOOC”. In: *AIED 2013 Workshops Proceedings Volume*. 2013, p. 25.

[39] Matthew C. Jadud and Brian Dorn. “Aggregate Compilation Behavior: Findings and Implications from 27,698 Users”. In: *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ICER ’15. Omaha, Nebraska, USA: ACM, 2015, pp. 131–139.

[40] W. L. Johnson and E. Soloway. “PROUST: Knowledge-Based Program Understanding”. In: *IEEE Transactions on Software Engineering* SE-11.3 (1985), pp. 267–275.

[41] W Lewis Johnson, Stephen Draper, and Elliot Soloway. *Classifying Bugs is a Tricky Business*. Tech. rep. DTIC Document, 1983.

[42] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. “Towards a Systematic Review of Automated Feedback Generation for Programming Exercises”. In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’16. Arequipa, Peru: ACM, 2016, pp. 41–46.

[43] *Khan Academy Coach Demo*. `http://www.khanacademy.org/coach/demo`.

[44] René F Kizilcec, Chris Piech, and Emily Schneider. “Deconstructing disengagement: analyzing learner subpopulations in massive open online courses”. In: *Proceedings of the Third International Conference on Learning Analytics and Knowledge*. ACM. 2013, pp. 170–179.

[45] Amruth N Kumar. “Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors”. In: *Technology, Instruction, Cognition and Learning (TICL) Journal* 4.1 (2006).

[46] Antti-Jussi Lakanen, Vesa Lappalainen, and Ville Isomöttönen. “Revisiting Rainfall to Explore Exam Questions and Performance on CS1”. In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. Koli Calling ’15. Koli, Finland: ACM, 2015, pp. 40–49.

[47] Nguyen-Thinh Le et al. “A review of AI-supported tutoring approaches for learning programming”. In: *Advanced Computational Methods for Knowledge Engineering*. Springer, 2013, pp. 267–279.

[48] Marcia C. Linn, Douglas Clark, and James D. Slotta. “WISE design for knowledge integration”. In: *Science Education* 87.4 (2003), pp. 517–538.

[49] Raymond Lister, Colin Fidge, and Donna Teague. “Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming”. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’09. Paris, France: ACM, 2009, pp. 161–165.

[50] Raymond Lister et al. "Not Seeing the Forest for the Trees: Novice Programmers and the SOLO Taxonomy". In: *SIGCSE Bulletin* 38.3 (June 2006), pp. 118–122.

[51] Mike Lopez et al. "Relationships Between Reading, Tracing and Writing Skills in Introductory Programming". In: *Proceedings of the Fourth International Workshop on Computing Education Research*. ICER '08. Sydney, Australia: ACM, 2008, pp. 101–112.

[52] B Jean Mason and Roger H Bruning. "Providing feedback in computer-based instruction: What the research tells us". In: *CLASS Research Report*. Vol. 9. University of Nebraska-Lincoln: Center for Instructional Innovation, 2001.

[53] Riccardo Mazza and Vania Dimitrova. "CourseVis: A graphical student monitoring tool for supporting instructors in web-based distance courses". In: *International Journal of Human-Computer Studies* 65.2 (2007), pp. 125–139.

[54] Riccardo Mazza and Vania Dimitrova. "CourseVis: Externalising student information to facilitate instructors in distance learning". In: *Proceedings of the International conference in Artificial Intelligence in Education*. Sydney, Australia, 2003, pp. 117–129.

[55] Riccardo Mazza and Vania Dimitrova. "Informing the design of a course data visualisator: an empirical study". In: *5th International Conference on New Educational Environments (ICNEE 2003)*. 2003, pp. 215–220.

[56] Riccardo Mazza and Christian Milani. "Exploring usage analysis in learning systems: Gaining insights from visualisations". In: *AIED05 workshop on Usage analysis in learning systems*. Citeseer. 2005, pp. 65–72.

[57] Matthew B Miles, A Michael Huberman, and Johnny Saldana. *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated, 2013.

[58] Craig S. Miller and Amber Settle. "Some Trouble with Transparency: An Analysis of Student Errors with Object-oriented Python". In: *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ICER '16. Melbourne, VIC, Australia: ACM, 2016, pp. 133–141.

[59] Edna H Mory. "Feedback research revisited". In: *Handbook of research on educational communications and technology* 2 (2004), pp. 745–783.

[60] National Center for Education Statistics. *What are the current trends in the teaching profession?* `https://nces.ed.gov/fastfacts/display.asp?id=28`. Accessed: 2017-05-05. 2017.

[61] National Center for Education Statistics. *What are the most popular majors for postsecondary students?* `https://nces.ed.gov/fastfacts/display.asp?id=37`. Accessed: 2017-05-05. 2017.

[62] Abelardo Pardo et al. "Generating Actionable Predictive Models of Academic Performance". In: *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*. LAK '16. Edinburgh, United Kingdom: ACM, 2016, pp. 474–478.

[63] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. "Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment". In: *Proceedings of the 2016 ACM Conference on International Computing Education Research.* ICER '16. Melbourne, VIC, Australia: ACM, 2016, pp. 93–101.

[64] Michael C Rodriguez. "Three options are optimal for multiple-choice items: A meta-analysis of 80 years of research". In: *Educational Measurement: Issues and Practice* 24.2 (2005), pp. 3–13.

[65] Cristóbal Romero and Sebastián Ventura. "Educational data mining: a review of the state of the art". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 40.6 (2010), pp. 601–618.

[66] Otto Seppälä et al. "Do We Know How Difficult the Rainfall Problem is?" In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research.* Koli Calling '15. Koli, Finland: ACM, 2015, pp. 87–96.

[67] Valerie J. Shute. "Focus on Formative Feedback". In: *Review of Educational Research* 78.1 (2008), pp. 153–189.

[68] Simon and Susan Snowdon. "Multiple-choice vs Free-text Code-explaining Examination Questions". In: *Proceedings of the 14th Koli Calling International Conference on Computing Education Research.* Koli Calling '14. Koli, Finland: ACM, 2014, pp. 91–97.

[69] Arjun Singh et al. "Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work". In: *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale.* L@S '17. Cambridge, Massachusetts, USA: ACM, 2017, pp. 81–88.

[70] Teemu Sirkiä and Juha Sorva. "Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises". In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research.* Koli Calling '12. Koli, Finland: ACM, 2012, pp. 19–28.

[71] Aaron J. Smith et al. "My Digital Hand: A Tool for Scaling Up One-to-One Peer Teaching in Support of Computer Science Learning". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education.* SIGCSE '17. Seattle, Washington, USA: ACM, 2017, pp. 549–554.

[72] Juha Sorva. "Notional Machines and Introductory Programming Education". In: *Trans. Comput. Educ.* 13.2 (July 2013), 8:1–8:31.

[73] Juha Sorva et al. *Visual program simulation in introductory programming education.* Aalto University, 2012.

[74] James C. Spohrer and Elliot Soloway. "Simulating Student Programmers". In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1.* IJCAI'89. Detroit, Michigan: Morgan Kaufmann Publishers Inc., 1989, pp. 543–549.

[75] Kristin Stephens-Martinez, Marti A. Hearst, and Armando Fox. "Monitoring MOOCs: Which Information Sources Do Instructors Value?" In: *Proceedings of the First ACM Conference on Learning @ Scale Conference*. L@S '14. Atlanta, Georgia, USA: ACM, 2014, pp. 79–88.

[76] Kristin Stephens-Martinez et al. "Taking Advantage of Scale by Analyzing Frequent Constructed-Response, Code Tracing Wrong Answers". In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ICER '17. Tacoma, Washington, USA: ACM, 2017, pp. 56–64.

[77] Charissa Tansomboon et al. "Designing Automated Guidance to Promote Productive Revision of Science Explanations". In: *International Journal of Artificial Intelligence in Education* 27.4 (2017), pp. 729–757.

[78] Kikumi K Tatsuoka. "A probabilistic model for diagnosing misconceptions by the pattern classification approach". In: *Journal of Educational and Behavioral Statistics* 10.1 (1985), pp. 55–73.

[79] Kikumi K Tatsuoka. "Rule space: An approach for dealing with misconceptions based on item response theory". In: *Journal of educational measurement* 20.4 (1983), pp. 345–354.

[80] Allison Elliott Tew and Mark Guzdial. "The FCS1: A Language Independent Assessment of CS1 Knowledge". In: *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. Dallas, TX, USA: ACM, 2011, pp. 111–116.

[81] Anne Venables, Grace Tan, and Raymond Lister. "A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer". In: *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*. ICER '09. Berkeley, CA, USA: ACM, 2009, pp. 117–128.

[82] Jonathan M. Vitale, Elizabeth McBride, and Marcia C. Linn. "Distinguishing complex ideas about climate change: knowledge integration vs. specific guidance". In: *International Journal of Science Education* 38.9 (2016), pp. 1548–1569.

[83] Yutao Wang and Neil Heffernan. "Extending knowledge tracing to allow partial credit: Using continuous versus binary nodes". In: *International Conference on Artificial Intelligence in Education*. AIED '13. Springer. 2013, pp. 181–188.

[84] Fionán Peter Williams and Owen Conlan. "Visualizing narrative structures and learning style information in personalized E-Learning systems". In: *Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on*. IEEE. 2007, pp. 872–876.

[85] Tianyi Wu, Yuguo Chen, and Jiawei Han. "Re-examination of interestingness measures in pattern mining: a unified framework". In: *Data Mining and Knowledge Discovery* 21.3 (2010), pp. 371–397.

[86] Ying Xie and Fangyun Yang. "Solving College Calculus Problems: A Study of Two Types of Instructors Feedback". In: *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2010*. Ed. by Jaime Sanchez and Ke Zhang. Orlando, Florida, USA: Association for the Advancement of Computing in Education (AACE), 2010, pp. 1293–1299.

[87] Claus Zinn and Oliver Scheuer. "Getting to know your student in distance learning contexts". In: *Innovative Approaches for Learning and Knowledge Sharing*. Springer, 2006, pp. 437–451.

Appendix A

Constructed-Response, Code-Tracing Wrong Answer Tags

To be concrete, we wrote all descriptions to start with one of two skeleton wordings. Hence, the descriptions here follow one of two formats.

Topic: Conceptually Correct

#	Description	Code Example (Python)

1	The correct answer with a typo	`>>> True or False`
		`Treu`

Topic: Assignment

#	Description	Code Example (Python)

2	This wrong answer demonstrates the student assigns the name of function to the variable instead of the return value of call expression.	`>>> f = lambda x: x`
		`>>> a = f(5)`
		`>>> a`
		`f`

3	This wrong answer demonstrates the student confuses assignment = with ==.	`>>> a == 3`
		`Nothing`
		`>>> a = 3`
		`True`

4	This wrong answer demonstrates the student believes when assigning a value to a name it also displays something on the terminal.	`>>> a = 1`
		`a`
		`>>> b = 2`
		`2`

5	This wrong answer demonstrates the student believes the variable's value is the string of the variable name.	`>>> a = 1`
		`>>> a`
		`'a' # Or a`

Topic: Boolean

#	Description	Code Example (Python)

6	This wrong answer demonstrates the student evaluates until the end of the boolean expression without short circuiting.	`>>> ’’ and False`
		`False # instead of ’’`

7	This wrong answer demonstrates the student believes `and` boolean expressions short-circuit at the first truth-y value.	`>>> True and 13`
		`True`

8	This wrong answer demonstrates the student believes evaluating boolean expressions only returns True or False, not the evaluated value itself.	`>>> 0 or False or 2 or 1 / 0`
		`True # correct: 2`

9	This wrong answer demonstrates the student believes that the value False is represented by the integer 0 and the value True is represented by 1.	`>>> 0 or True`
		`1 #instead of True`

10	This wrong answer demonstrates the student believes that a falsy value in Python is actually truthy.	`>>> False or 0`
		`True #instead of 0`
		`# student perceived 0 as True`

11	This wrong answer demonstrates the student does not recognize that a boolean expression is evaluated to a True/False value. Instead the entire code snippet verbatim is used and “passed around” as opposed to the value it evaluates to.	`not 0` is passed around as `not 0` instead of its evaluation `True`.
12	This wrong answer demonstrates the student believes a nonzero integer is a falsey value.	`15` is a `False` value

13	This wrong answer demonstrates the student believes when using boolean `not`, it could return an error.	`>>> not 23`
		`Error`

14	This wrong answer demonstrates the student believes `not` does nothing.	`>>> not ’’`
		`’’`

15	This wrong answer demonstrates the student evaluates until the end of the boolean expression without short circuiting.	`>>> True or 1`
		`1 #instead of True`

16	This wrong answer demonstrates the student believes `or` boolean expressions short circuit at the first falsey value.	`>>> True or False or True`
		`False`

Topic: Boolean (cont.)

#	Description	Code Example (Python)

17	This wrong answer demonstrates the student believes `and` is `or` and `or` is `and`.	`>>> 0 and 1 and True`
		`1`
		`>>> True or False`
		`False`

18	This wrong answer demonstrates the student believes boolean operators are evaluated left to right despite boolean operators having precedence. This is a misunderstanding of order of operations.	`>>> False and True or True`
		`False`
		`# Precedence says: (F and T) or T`

Topic: Built-Ins

#	Description	Code Example (Python)

19	This wrong answer demonstrates the student believes when checking for equality Python returns something that is not True/False.	`>>> def g(x):`
		`...   print('foo')`
		`...   while x > 0:`
		`...     print('bar')`
		`...     yield x`
		`...     x -= 1`
		`...     print('baz')`
		`>>> a = g(3)`
		`>>> a == iter(a)`
		`Generator`

20	This wrong answer demonstrates the student believes the `in` keyword returns something that is not boolean. For example: Error or default value (0 or `None`) if the item is not there, value if it is there, or number of times it appears. If this is using `in` with a Dictionary, use Tag 26 instead.	`>>> 4 in {1,2,3,4}`
		`4`
		`>>> 4 in {1,2,3}`
		`0  # Or Error or None`
		`>>> 4 in [4,4]`
		`2`

21	This wrong answer demonstrates the student believes the Python `is` keyword returns anything but boolean values. For example: Error or default value (0, None, ()), or some answer that makes logical sense within the context such as if `is` returns true it is the value both sides of the `is` evaluate to.	`>>> l = Link(1, Link(2))`
		`>>> l.rest.rest is Link.empty`
		`0 or Error or () or Empty`

22	This wrong answer demonstrates the student believes return does not terminate the function. Often the wrong answer looks like they are trying to convey multiple lines in one line.	`>>> def f(x):`
		`...   return x`
		`...   print 'foo'`
		`>>> f(10)`
		`10 \n foo`

23	This wrong answer demonstrates the student believes the function `sorted` returns a boolean of whether the input argument is sorted or not.	`>>> sorted(set([4,2,6,7,7])`
		`False`
		`>>> sorted(set([4,6,7,7])`
		`True`

24	This wrong answer demonstrates the student believes the function `sorted` returns something that conveys it is a list with generally the right answer, however it is not in the format of a list. Often seen as a set or maybe a string of strings.	`>>> sorted(set([4,2,6,7,7])`
		`{2,4,6,7}`

Topic: Dictionary

#	Description	Code Example (Python)

25	This wrong answer demonstrates the student believes a dictionary can have duplicate keys.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> d[’a’] = 20`
		`>>> sorted(list(d.keys()))`
		`[’a’, ’a’, ’b’, ’c’]`

26	This wrong answer demonstrates the student believes something weird happens when using `in` for membership in a dictionary, e.g. Error is thrown or it is added to the dictionary and a default value is returned (.e.g. None, 0) if something cannot be found.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> ’d’ in d`
		`Error  # or 0, None`

27	This wrong answer demonstrates the student believes a dictionary is only the pairs it was originally initialized as.	`>>> d = {’b’:5, ’a’:10}`
		`>>> d[’c’] = 1`
		`>>> sorted(list(d.keys()))`
		`[’a’, ’b’]`

28	This wrong answer demonstrates the student believes when calling d.keys() it returns the dictionary.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> sorted(list(d.keys()))`
		`{’a’:10, ’b’:5, ’c’:1}`

29	This wrong answer demonstrates the student believes when calling d.keys() it returns the first key alphabetically.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> sorted(list(d.keys()))`
		`’a’`

30	This wrong answer demonstrates the student believes when calling d.keys() it returns the last key alphabetically.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> sorted(list(d.keys()))`
		`’c’`

31	This wrong answer demonstrates the student believes when calling d.keys() it returns d.values() instead.	`>>> d = {’a’:10, ’b’:5, ’c’:1}`
		`>>> sorted(list(d.keys()))`
		`[1, 5, 10]`

32	This wrong answer demonstrates the student believes calling `len` on a dictionary is not counting the number of items. Instead it could: (1) return a random value in the dictionary, (2) be the sum of all the values, (3) Error.	`>>> d = {’a’:10, ’b’:5, ’c’:1}`
		`>>> len(d)`
		`10  # or 5, 1, 10, 16, or Error`

33	This wrong answer demonstrates the student believes when reassigning a dictionaries's key's value to another key's value, the key name is set to the other key's name rather than that key's value. This could happen in either direction of setting the key.	`>>> d = {’b’:5, ’a’:10, ’c’:1}`
		`>>> d[’a’] = d[’b’]`
		`>>> sorted(list(d.keys()))`
		`[’a’, ’a’, ’c’]`
		`# or [’b’, ’b’, ’c’]`
		`# or [’a’, ’c’]`
		`# or [’b’, ’c’]`

Topic: Environment

#	Description	Code Example (Python)

34	This wrong answer demonstrates the student believes the value of a variable is not the value from the environment at that time, but the value it is set at initialization or elsewhere in the program. This is a subset of Tag 165 and therefore Tag 165 should not appear with this tag.	`>>> def f(x):`
		`...   x += 1`
		`...   def g(y):`
		`...     return x + y`
		`...   return g`
		`>>> f(2)(2)`
		`3`

Topic: Error

#	Description	Code Example (Python)

35	This wrong answer demonstrates the student thinks the error raised is the return value.	`>>> [x for x in iter(i)]`
		`[1, 2, ..., StopIteration]`

36	This wrong answer demonstrates the student believes a `for` loop (or a list comprehension) does not catch a `StopIteration` error.	`>>> [x for x in iter(i)]`
		`StopIteration`

37	This wrong answer demonstrates the student believes dividing by zero does not cause an error.	`>>> True and 1 / 0 and True`
		`True`

Topic: Evaluation

#	Description	Code Example (Python)

38	This wrong answer demonstrates the student does not recognize that you evaluate an expression to obtain its value. Instead the entire code snippet verbatim is saved and "passed around" as opposed to the value it evaluates to.	`>>> a = 1 + 2`
		`>>> a`
		`1 + 2`

39	This wrong answer demonstrates the student believes when a variable's value cannot be found it is the "empty" version of that type so as not to error out	`>>> a = 2`
		`>>> c = a + b`
		`2 # believe b == 0`

40	This wrong answer demonstrates the student believes function call expressions are called left to right, however there is precedence by parenthesis or nested calls. This is a misunderstanding of order of operations.	`>>> t = lambda f: lambda x: f(f(f(x)))`
		`>>> s = lambda x: x + 1`
		`>>> t(s)(0)`
		`Function`

Topic: Function

#	Description	Code Example (Python)
41	This wrong answer demonstrates the student believes a composed function is a single call of that function.	f(f(f(f(x)))) is just f(x)

42	This wrong answer demonstrates the student believes a function call cannot have zero arguments and would error.	`>>> f = lambda : 3`
		`>>> f()`
		`Error`

43	This wrong answer demonstrates the student believes when displaying a function's value it displays nothing at all.	`>>> f = lambda x : x`
		`>>> f`
		`Nothing`

44	This wrong answer demonstrates the student believes when the name of a function is in a line of code (but not being called) the function is being called.	`>>> f = lambda x: 1`
		`>>> f`
		`1`

45	This wrong answer demonstrates the student believes when a function call does not have the right number of arguments it will still work, when it is supposed to error out.	`>>> f = lambda x: print(x)`
		`>>> f()`
		`#no error, something happens`

46	This wrong answer demonstrates the student believes a series of evaluations for the operator of a function call is not possible and causes an error.	`>>> f=lambda: lambda: 2`
		`>>> f()()`
		`Error`

Topic: Generator

#	Description	Code Example (Python)

47	This wrong answer demonstrates the student believes when calling the `iter` function on a generator it runs the beginning of the function, most likely to the first `yield`, without necessarily yielding it.	`>>> def g(x):`
		`...   print(’foo’)`
		`...   while x > 0:`
		`...     print(’bar’)`
		`...     yield x`
		`...     x -= 1`
		`...     print(’baz’)`
		`>>> iter(g(3))`
		`foo`

48	This wrong answer demonstrates the student believes that in a generator rather than going to the top of the loop after reaching the end, Python goes to the top of the function.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`3`
		`>>> next(a)`
		`baz`
		`foo`

49	This wrong answer demonstrates the student is being sloppy in not incrementing the value being yielded and therefore subsequent calls to `next` yield the first value. This means it does not get the Tag 164 tag, even if that is why the student is getting that answer.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`3`
		`>>> next(a)`
		`baz`
		`bar`
		`3`

50	This wrong answer demonstrates the student thinks the subsequent call to the `next` method acts like the first call to `__next__` and thus starts over.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`3`
		`>>> next(a)`
		`foo`

51	This wrong answer demonstrates the student thinks the implicit generator’s `__iter__` method does not return `self`.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> a == iter(a)`
		`False`

Topic: Generator (cont.)

#	Description	Code Example (Python)

52	This wrong answer demonstrates the student confused the order of when variables are changed and when they are yielded.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`2`

53	This wrong answer demonstrates the student thinks `yield` does not return anything and subsequently halts the execution of the program.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`baz`

54	This wrong answer demonstrates the student thinks `yield` acts like `continue` instead of a `return`.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`bar`

55	This wrong answer demonstrates the student believes that for a subsequent call to `next`, instead of starting where the `yield` left off, it starts at the beginning of the loop body.	`>>> import g from Tag_47`
		`>>> a = g(3)`
		`>>> next(a)`
		`foo`
		`bar`
		`3`
		`>>> next(a)`
		`bar`

Topic: If

#	Description	Code Example (Python)

56	This wrong answer demonstrates the student believes `if..elif` or `if..else` is actually two `if` clauses. Often the wrong answer looks like they are trying to convey multiple lines in one line.	`>>> def f(x): = 5`
		`...   if x <= 5:`
		`...     print('one')`
		`...   elif x > 3:`
		`...     print('two')`
		`>>> f(5)`
		`one \n two`

57	This wrong answer demonstrates the student believes two `if`'s next to each other are actually an `if..else` clause. When the system asks for another line they do not believe will show, the student usually submits "Nothing."	`>>> def f(x): = 5`
		`...   if x <= 5:`
		`...     print('one')`
		`...   if x > 3:`
		`...     print('two')`
		`>>> f(5)`
		`one`
		`Nothing`

58	This wrong answer demonstrates the student is being sloppy in checking `if` conditions, often missing the true condition.	`>>> def func(x):`
		`...   if x == 'x':`
		`...     return 'huh'`
		`...   if x == x:`
		`...     return 'well'`
		`>>> func('s')`
		`'huh'`

Topic: Iterator

#	Description	Code Example (Python)

59	This wrong answer demonstrates the student believes that the `iter` function must return a new instance of the iterator.	`>>> odds = OddNaturalsIterator()`
		`>>> odd_iter1 = iter(odds)`
		`>>> odd_iter2 = iter(odds)`
		`>>> next(odd_iter1)`
		`1`
		`>>> next(odd_iter1)`
		`3`
		`>>> next(odd_iter1)`
		`5`
		`>>> next(odd_iter2)`
		`1 # Should be 7`

60	This wrong answer demonstrates the student believes that the `iter` function must return `self`.	`>>> evens = EvenNaturalsIterator()`
		`>>> even_iter1 = iter(evens)`
		`>>> even_iter2 = iter(evens)`
		`>>> next(even_iter1)`
		`0`
		`>>> next(even_iter1)`
		`2`
		`>>> next(even_iter1)`
		`4`
		`>>> next(even_iter2)`
		`6 # or 4 or 8; should be 0`

61	This wrong answer demonstrates the student believes when calling the `next` function again on the same iterator instance, the value returned is the first value of the iterator.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...         self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(5)`
		`>>> next(i)`
		`1`
		`>>> next(i)`
		`1 # instead of 2`

Topic: Iterator (cont.)

#	Description	Code Example (Python)

62	This wrong answer demonstrates the student is confusing the order of when variables are changed and when it is returned. Often using the initial value in their answer, when the initial value is not actually part of it.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...         self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     if self.i >= self.n:`
		`...       raise StopIteration`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`[0, 1, 2]  # correct: [1, 2]`

63	This wrong answer demonstrates the student is confusing the order of when variables are changed and when the `StopIteration` is raised. Often missing the last value in their answer.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...         self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     if self.i >= self.n:`
		`...       raise StopIteration`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`[1]  # correct: [1, 2]`

64	This wrong answer demonstrates the student thinks once `StopIteration` is raised once, when `__next__` is called again, the iterator starts over (for iterators).	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...         self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     if self.i >= self.n:`
		`...       raise StopIteration`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`[1, 2]`
		`>>> next(i)`
		`1  # correct: Error`

Topic: Iterator (cont.)

#	Description	Code Example (Python)

65	This wrong answer demonstrates the student thinks once `StopIteration` is hit once, when `__next__` is called again, the iterator resumes on the line of code after `StopIteration`.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...  self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     if self.i >= self.n:`
		`...       raise StopIteration`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`[1, 2]`
		`>>> next(i)`
		`3  # correct: Error`

66	This wrong answer demonstrates the student thinks that the `__next__` method needs to have a `StopIteration`.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...  self.i = n`
		`...   def __next__(self):`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`Error`

67	This wrong answer demonstrates the student thinks after a `StopIteration`, the `next` call returns the most recent value.	`>>> class MyIter:`
		`...   def __init__(self, n):`
		`...  self.n, self.i = n, 0`
		`...   def __next__(self):`
		`...     if self.i >= self.n:`
		`...       raise StopIteration`
		`...     self.i+=1`
		`...     return self.i`
		`...   def __iter__(self):`
		`...     return self`
		`>>> i = MyIter(2)`
		`>>> [x for x in i]`
		`[1, 2]`
		`>>> next(i)`
		`2  # correct: Error`

Topic: Lambda

#	Description	Code Example (Python)

68	This wrong answer demonstrates the student believes when a lambda is called it is not actually called/callable.	`>>> f = lambda x: x`
		`>>> f(5)`
		`# ‘Nothing‘ or ‘Function‘`

69	This wrong answer demonstrates the student does not recognize the lambda function can be self-referring without errors.	foo = lambda x: foo, foo returns `Function`.

70	This wrong answer demonstrates the student believes that when defining a lambda, as much of the return value is evaluated as possible in that moment of defining it.	`>>> x = 1`
		`>>> f = lambda y: x + y`
		`Nothing  # correct`
		`>>> x = 2`
		`>>> f(2)`
		`3`

71	This wrong answer demonstrates the student is being sloppy about identifying the portion of the lambda function that is the return value.	`>>> g = lambda x: lambda y: x + 3`
		`>>> g(2)`
		`5 #instead of Function`
		
		`>>> c = lambda x: lambda: print(’123’)`
		`>>> c(88)`
		`123 #instead of Function`

72	This wrong answer demonstrates the student believes a lambda needs the keyword `return` in it for the lambda to return anything. Therefore they believe the lambda returns nothing.	`>>> f = lambda: 3`
		`>>> f()`
		`Nothing`

73	This wrong answer demonstrates the student believes a lambda needs to have at least one parameter when defined or it will error out.	`>>> lambda: 5`
		`Error`

Topic: Link

#	Description	Code Example (Python)

74	This wrong answer demonstrates the student believes when accessing an attribute it returns some kind of `Link` accessing code	`>>> l = Link(1, Link(2, Link(3)))`
		`>>> l.rest.rest.first`
		`l.rest.first`

75	This wrong answer demonstrates the student believes that the `.rest` of an empty linked list is also `Link.empty`.	`>>> l = Link(1, Link(2))`
		`>>> l.rest.rest.rest.rest`
		`Link.empty`

76	This wrong answer demonstrates the student believes when returning the value of `first` for a link it is returned as a link object even though it is only a value.	`>>> l = Link(1, Link(2))`
		`>>> l.first`
		`Link(1)`

77	This wrong answer demonstrates the student believes a link list is only what it was originally initialized as.	`>>> l = Link(1, Link(2))`
		`>>> l.first = 5`
		`>>> print_link(l)`
		`<1 2>`

78	This wrong answer demonstrates the student believes that linked lists cannot link back to themselves, so if a line of code does this, it is as if that line was not executed.	`>>> l = Link(1, Link(2, Link(3)))`
		`>>> l.rest = l`
		`>>> l.rest.rest.first`
		`2`

79	This wrong answer demonstrates the student is being sloppy about evaluating the `first`/`rest` attributes of a link or a sequence of them.	`>>> l = Link(1, Link(2, Link(3)))`
		`>>> l.rest.rest.first`
		`2`

Topic: List

#	Description	Code Example (Python)

80	This wrong answer is wrong because the student believes when casting something to a list it is not a list.	`>>> d = {’a’:10, ’b’:5, ’c’:1}`
		`>>> sorted(list(d.keys()))`
		`{1, 5, 10}`

81	This wrong answer demonstrates the student believes a list comprehension does not return a list, but just a value.	`>>> [x for x in range(3)]`
		`0`

82	This wrong answer demonstrates the student believes that in a list comprehension only the first value is returned.	`>>> [x for x in range(3)]`
		`[0]`

83	This wrong answer demonstrates the student believes that in a list comprehension only the last value is returned.	`>>> [x for x in range(3)]`
		`[2]`

84	This wrong answer demonstrates the student believes that when concatenating two lists together they are zipped and some function is used to map the pairs into a single value.	`>>> [1, 2, 3] + [4, 5, 6]`
		`[14, 25, 36]`

85	This wrong answer is wrong because the list is missing brackets.	`>>> [x for x in range(3)]`
		`0, 1, 2`

86	This wrong answer demonstrates the student thinks a list doesn’t need commas.	`>>> [x for x in range(3)]`
		`[0 1 2]`

87	This wrong answer demonstrates the student expects that when indexing an element in a list the value is returned as a list.	`>>> lst = [0, 2, 4, 8]`
		`>>> lst[2]`
		`[4]`

88	This wrong answer demonstrates the student believes list indexing starts at 1.	`>>> lst = [0, 2, 4, 8]`
		`>>> lst[2]`
		`[2]`

89	This wrong answer demonstrates the student believes that to index into a list commas are used.	`lst[1, 2]` instead of `lst[1][2]`
90	This wrong answer demonstrates the student believes that for list indexing instead of stacking brackets the brackets are nested.	`lst[1[2]]` instead of `lst[1][2]`

91	This wrong answer demonstrates the student believes that nested lists are read as if they are flat.	`>>> l = [1, [2, 3], [4], 5, 6]`
		`>>> l[2]`
		`3`

92	This wrong answer demonstrates the student believes the Python list syntax is the same as Scheme’s (characteristics such as parentheses and no spaces).	`>>> [x for x in range(3)]`
		`(0 1 2)`

Topic: List (cont.)

#	Description	Code Example (Python)

93	This wrong answer demonstrates the student believes that the output of `range` is not 0...n-1, rather one of the following 0...n, 1...n, 1...n-1.	`>>> range(3)`
		`[1, 2, 3]`

94	This wrong answer demonstrates the student is being sloppy about reading the map expression inside the list comprehension.	`>>> [x*x for x in range(5)]`
		`[0, 1, 4, 27, 256]`

95	This wrong answer is wrong because the student is being sloppy in how they are sorting the values.	`>>> l = [’b’,’a’,’c’]`
		`>>> sorted(l)`
		`[’a’,’c’,’b’]`

96	This wrong answer demonstrates the student thinks a tuple, represented using parentheses, is the same as a list, represented using brackets.	`>>> (1,1) == [1, 1]`
		`True`

Topic: Math

#	Description	Code Example (Python)

97	This wrong answer demonstrates the student believes that floor dividing means dividing twice.	`>>> 8//2`
		`2`
		`>>> 2//2`
		`0.5`

98	This wrong answer demonstrates the student believes that negative numbers cannot happen.	`>>> i = 1`
		`>>> while i >= 0:`
		`>>>   i -= 1`
		`>>>   print(i)`
		`1`
		`0`
		`0`

99	This wrong answer demonstrates the student believes that floor dividing a number with itself acts differently than normal.	`2//2 = 0.5` or `2//2 = 2`

100	This wrong answer demonstrates the student is being sloppy in arithmetic. They are flipping + with - or / with * or any other combination of operators.	`>>> 4 * 5 + 2`
		`40 # correct: 22`

101	This wrong answer demonstrates the student is being sloppy about incrementing by the value intended, often incrementing by 1 instead.	`>>> x = 1`
		`>>> x += 2`
		`>>> x`
		`2`

102	This wrong answer demonstrates the student believes that floor div is true div and vice versa.	`>>> 8//3`
		`2.666666666`
		`>>> 8/3`
		`2`

103	This wrong answer demonstrates the student has a different belief about the order of precedence of arithmetic operators.	`>>> 1 + 13/2 * 2 * 4/3`
		`13 # correct: 17`

Topic: OOP

#	Description	Code Example (Python)

104	This wrong answer demonstrates the student believes when accessing an attribute displays the entire object.	`>>> l = Link(1, Link(2))`
		`>>> l.first`
		`Link(1, Link(2))`

105	This wrong answer demonstrates the student believes an attribute must be predefined in the class definition or `__init__` function.	`>>> class Foo:`
		`...   a = 2`
		`>>> class Bar(Foo):`
		`...   b = 3`
		`...   def baz(self):`
		`...     return self.b`
		`>>> f = Foo()`
		`>>> f.c = 7`
		`>>> f.c`
		`Error # Should be 7`

106	This wrong answer demonstrates despite specific modification to instance attributes, the student still references the original class instance code and believes the instance has not been modified.	`>>> class Foo:`
		`...   a = 2`
		`>>> class Bar(Foo):`
		`...   b = 3`
		`...   def baz(self):`
		`...     return self.b`
		`>>> f = Foo()`
		`>>> f.a = 7`
		`>>> f.a`
		`2 # Should be 7`

107	This wrong answer demonstrates the student is being sloppy about what class the current object is an instance of and is instead thinking it is a related one (base class or subclass).	`>>> class Foo:`
		`...   a = 2`
		`>>> class Bar(Foo):`
		`...   b = 3`
		`...   def baz(self):`
		`...     return self.b`
		`>>> f = Foo()`
		`>>> Bar.baz(f)`
		`3`

108	This wrong answer demonstrates the student has merged the instance and class attribute with the same name and just grabs the value from one randomly.	`>>> class Foo:`
		`...   a = 2`
		`>>> class Bar(Foo):`
		`...   a = 3`
		`...   def baz(self):`
		`...     return self.b`
		`>>> f = Foo()`
		`>>> f.a`
		`3 # Should be 2`

Topic: OOP (cont.)

#	Description	Code Example (Python)
109	This wrong answer demonstrates the student believes calls to methods do not require being bound to a particular instance of the class, but can instead be invoked with the class itself without passing through the instance as a parameter.	`Car.name()` instead of `Car.name(johns_car)`

110	This wrong answer demonstrates the student does not recognize that a call to a function that does not exist in the local class is referenced from a inherited parent class.	`>>> class Foo:`
		`...   a = 2`
		`...   def baz(self):`
		`...     return self.b`
		`>>> class Bar(Foo):`
		`...   b = 3`
		`>>> br = Bar()`
		`>>> br.baz()`
		`2 # Should be 3`

111	This wrong answer demonstrates the student believes if an attribute cannot be found an error occurs rather than going to the local class and then up to the inherited parent class.	`>>> class Foo:`
		`...   a = 2`
		`>>> class Bar(Foo):`
		`...   b = 3`
		`...   def baz(self):`
		`...     return self.b`
		`>>> br = Bar()`
		`>>> br.a`
		`Error # Should be 2`

112	This wrong answer demonstrates the student is being sloppy by incorrectly evaluating attributes of an object, but NOT the object (that is Tag 165). Not applicable if answer is Error, answer has to be reasonable within current context. Also consider Tag 160.	`>>> class Foo:`
		`...   def __init__(self, a):`
		`...     self.a = a`
		`>>> f = Foo("hello world")`
		`>>> f.a`
		`"hello"`

Topic: Print

#	Description	Code Example (Python)

113	This wrong answer demonstrates the student believes that printing causes a function to also stop executing, kind of like a `return`.	`>>> def func():`
		`...    print(’hello’)`
		`...    print(’123’)`
		`>>> func()`
		`hello`

114	This wrong answer demonstrates the student believes when printing `None` something strange must happen rather than normal code execution.	`>>> c = print("Hello")`
		`>>> print(c)`
		`Nothing  # Should be None`

115	This wrong answer demonstrates the student believes a print returns the value it just printed.	`>>> def func():`
		`...    print(’hello’)`
		`>>> x = func()`
		`>>> print(x)`
		`hello`

116	This wrong answer demonstrates the student thinks printing a string keeps the quotes.	`>>> print(’hello’)`
		`’hello’`

117	This wrong answer demonstrates the student is being sloppy by skipping one or more print statements.	`>>> n = 3`
		`>>> while n >= 0:`
		`...   n -= 1`
		`...   print(n)`
		`-1`

Topic: PyInterpreter

#	Description	Code Example (Python)

118	This wrong answer demonstrates the student believes that the return value is not displayed in the Python interpreter.	`>>> def f(x):`
		`...   return x`
		`>>> f(x)`
		`Nothing`

Topic: Scheme

#	Description	Code Example (Python)
119	This wrong answer demonstrates the student believes arguments do not need to be separated by a space and are separated in some other way.	Student believes `(cons 1, 2, 3)` is valid

120	This wrong answer demonstrates the student believes one of multiple options, but it results in him using the variable's name as the variable's value. (1) the value of the variable is its name (2) when seeing a variable, the student doesn't evaluate it for some reason.	`scm> (define a 1)`
		`a`
		`scm> (cons a nil)`
		`(a)`

121	This wrong answer demonstrates the student believes a `cdr` of a list is a element of the list it should return.	`scm> (cdr ’(2 1))`
		`1`
		`scm> (cdr ’(1 2 3))`
		`2`
		`scm> (cdr ’(1 (2 3)))`
		`(2 3)  ; Correct: ((2 3))`

122	This wrong answer demonstrates the student believes when calling `cdr` on a list it returns a subset of the `cdr` of the list, potentially flattening the list if it is a list of lists.	`scm> (cdr ’(1 2 3))`
		`(2)`
		`scm> (cdr ’(1 (2 3))`
		`(2)  ; Correct: ((2 3))`

123	This wrong answer demonstrates the student believes a `cons` that would form a malformed list returns a list.	`scm> (cons 1 2)`
		`(1 2)`

124	This wrong answer demonstrates the student believes cons-ing with a list returns a malformed list.	`scm> (cons 2 (1))`
		`(2 . 1)`

125	This wrong answer demonstrates the student believes cons-ing with a list returns a nested list.	`scm> (cons 2 (1))`
		`(2 (1))`

126	This wrong answer demonstrates the student believes Scheme returns either booleans, empty strings, or return values for when something is first defined in scheme.	`scm> (define a 4)`
		`4  ; Correct: a`

127	This wrong answer demonstrates the student believes when a variable is defined and values in it need to be evaluated, the student thinks the variable's value is a function not an evaluated value.	With `(define y (- 1 2))` student thinks `y`'s value is Function.

Table A.1: The Scheme tags are in one table because this work's primary focus was Python.

Topic: Scheme (cont.)

#	Description	Code Example (Python)

128	This wrong answer demonstrates the student believes the `nil` value in a Scheme list is written out as `nil` rather than blank.	`scm> (cons 1 nil)`
		`(1 nil) or (1 . nil)`
		`; Correct: (1)`

129	This wrong answer demonstrates the student executes the procedure elements from right to left.	`scm> (- 10 4)`
		`-6`

130	This wrong answer demonstrates the student believes nil is the same as #f.	`scm> (and #t nil)`
		`false ; should be ()`

131	This wrong answer demonstrates the student believes the list procedure behaves like cons, where the arguments create a pair.	`scm> (list 1 (cons 2 3))`
		`(1 . (2 . 3))`
		`; Correct: (1 (2 . 3))`

132	This wrong answer demonstrates the student is not reducing the displayed output of a list to its fully reduced version, where for each ’.’ next to a ’(’ the ’.’ and pair of ’()’ are removed.	`scm> '(1 . ((2 . ()) . (3 . ())))`
		`(1 . ((2) 3))`
		`; Correct: (1 (2) 3)`

133	This wrong answer demonstrates the student believes the outer parentheses for a list are not needed.	`scm> (cons 1 (cons 3 ()))`
		`1 3  ; Correct: (1 3)`

134	This wrong answer demonstrates the student believes list elements do not need to be separated by a space and are separated in some other way, such as commas or semicolons.	`scm> '(1 2 3) -> (1,2,3)`

135	This wrong answer demonstrates the student is being sloppy when indexing into a list and is one element to the left or right.	`scm> (car (cdr '(1 2 3 4))`
		`3  ; OR 1`

136	This wrong answer demonstrates the student believes a list is displayed as a nested list. This only applies if it is clear the student knows the data structure (like for the question that gives them a picture).	`'(1 1)` is given as a picture data structure and student submits `(1 (1))`

137	This wrong answer demonstrates the student believes when seeing a dot with a parenthesis, they remove one but not the other.	`scm> '(1 . (2))`
		`(1 (2)) OR (1 . 2)`

138	This wrong answer demonstrates the student believes when returning a list it is returned as code that when evaluated is that list. This does not mean just the variable, use ScmTag 5 instead.	`scm> (cdr '(1 2 3))`
		`(list 2 3)`

Table A.2: The Scheme tags are in one table because this work’s primary focus was Python.

Topic: Scheme (cont.)

#	Description	Code Example (Python)

139	This wrong answer is wrong because the student believes that a malformed list does not need spaces around the dot '.'	`scm> (cons 1 2)`
		`(1.2)`

140	This wrong answer demonstrates the student believes a malformed list is a normal list, usually by the evidence of what they think a list looks like when displaying it.	`scm> (cons 1 (cons 2 3))`
		`(1 2 3)`
		`scm> (cdr '(1 . 3))`
		`(3)`

141	This wrong answer demonstrates the student believes there can only be two arguments in a prefix notation with + * - /.	`scm> (+ 1 2 3)`
		`Error`

142	This wrong answer demonstrates the student believes when creating a list with a list(s)/pair(s) in it, the result is a flattened list with malformed pairs potentially not having parentheses around them.	`scm> (list 1 (cons 2 3))`
		`(1 2 . 3)`
		`scm> '(1 (2 3) (3 4))`
		`(1 2 3 3 4)`
		`scm> '(1 (2 . 3) 4)`
		`(1 2 . 3 4)`
		`scm> '(1 (1 (2 . 3) 3))`
		`(1 1 (2 . 3) 3)`

143	This wrong answer demonstrates the student believes `nil` is not a list and therefore believes it is a viable answer by itself.	`scm> (list? nil)`
		`False  ; Correct: True`
		`scm> (cdr (cons 1 nil))`
		`nil  ; Correct: ()`

144	This wrong answer demonstrates the student believes that quoting code evaluates the content of that quoted list.	`scm> (cons 1 '(list 2 3))`
		`(1 . (2 . 3)) or (1 . ((2 3)))`
		`; Correct: (1 list 2 3)`

145	This wrong answer demonstrates the student believes shorthand quoting for a list results in a something other than a Scheme list or a malformed scheme list.	`scm> '(1)`
		`1`

146	This wrong answer demonstrates the student is making a mistake in translating a quoted list to the underlying data structure. Potentially also they could be making a mistkae in translating from the structure to the output, but it is not clear.	`scm> '(2 . 3)`
		`(2 . (3))`

147	This wrong answer demonstrates the student believes quoting is no different than the value.	`scm> (define a 1)`
		`a`
		`scm> (define b 'a)`
		`1`

148	This wrong answer demonstrates the student believes a nested list with a single item in it is just the inner list.	`((1))` becomes `(1)`

Table A.3: The Scheme tags are in one table because this work's primary focus was Python.

Topic: Scheme (cont.)

#	Description	Code Example (Python)

149	This wrong answer demonstrates the student is swapping the implementation of `car` and `cdr`.	`scm> (car '(1 2)) -> (2) OR`
		`scm> (cdr '(1 2)) -> 1`

150	This wrong answer is wrong because the student is being sloppy in how they display the scheme list in that they are missing or adding one to three of the space or paren characters inappropriately.	`scm> (cons 1 (cons 2 3))`
		`(1 (2 . 3))`
		`; Correct with typo ans:`
		`; (1 2. 3) or (1 2 . 3`
		`scm> (cons (cons 1 ()) (cons 2`
		`...    (cons 3 ())))`
		`(1 (2 3))`
		`; Correct: ((1) (2 3))`
		`; if answer was ((1 (2 3)) gets`
		`; correct category`
		`scm> (cons 1 (cons 2 (cons 3`
		`...    ())))`
		`(1 (2 3))`

Table A.4: The Scheme tags are in one table because this work's primary focus was Python.

Topic: Set

#	Description	Code Example (Python)

151	This wrong answer demonstrates the student believes using `&` on two or more sets raises an error.	`>>> {1,2,3} & {2,3,4}`
		`Error`

152	This wrong answer demonstrates the student believes the `|` operates like a diff.	`>>> {1,2,3} | {2,3,4}`
		`{1}`

153	This wrong answer demonstrates the student believes that `|` is XOR.	`>>> {1,2,3} | {2,3,4}`
		`{1,4}`

154	This wrong answer demonstrates the student believes the `intersect` function or the `&` of two sets is actually unioning them.	`>>> {1,2,3}.intersect({2,3,4})`
		`{1,2,3,4}`

155	This wrong answer demonstrates the student believes that duplicate elements are possible in sets.	`>>> s = set([1,1,2])`
		`>>> len(s)`
		`3`
		`>>> s = set([1])`
		`>>> s.add(1)`
		`>>> s.remove(1)`
		`>>> 1 in s`
		`True`

156	This wrong answer demonstrates the student believes a difference `-` operation between two or more sets is not itself, such as xor, Error, union, or intersect.	`>>> {1,2,3} - {2,3,4}`
		`{1,4}`
		`# or Error or {1,2,3,4} or {2,3}`

157	This wrong answer demonstrates the student is being sloppy in evaluating the outcome of an intersection operation. In that they have up to 2 extra elements or are missing no more than 2 element. If more than that do not apply this tag.	`>>> {1,2,3}.intersect({2,3,4})`
		`{1,2,3}`
		`>>> {1,2,3}.intersect({2,3,4})`
		`{2}`
		`>>> {1,2,3}.intersect({3,4,5,6})`
		`{1,2,3,4} # Do NOT use this tag`

158	This wrong answer demonstrates the student believes a `union` function call or the `|` operator on a set with another set as the input for the function is an intersection operation between the two sets.	`>>> {1,2,3}.union({2,3,4})`
		`{2,3}`

Topic: Sloppy

#	Description	Code Example (Python)

159	The wrong answer demonstrates the student is swapping when a variable is incremented versus checked in a condition.	`>>> x = -9`
		`>>> y = -12`
		`>>> while x:`
		`...    if y:`
		`...        print(y)`
		`...    y += 3`
		`...    x += 3`
		`-9; Should be -12, -9, -6`

160	This wrong answer demonstrates the student is being sloppy where they are using code that has not yet been executed to answer the current prompt.	`>>> class Foo:`
		`...   def __init__(self):`
		`...     self.a = 5`
		`>>> f = Foo()`
		`>>> f.a`
		`10`
		`# Sees future code and`
		`# uses that value`
		`>>> f.a = 10`

161	This wrong answer demonstrates the student is being sloppy about taking into account all of the code lines for the entire case. Commonly this is because there are multiple prompts/questions in the case and they forget of prior code that affects current code.	`>>> d = {'a':0}`
		`>>> len(d)`
		`1`
		`>>> d['b'] = 1`
		`>>> len(d)`
		`1`
		`# Ignores the prior prompt's code`

162	This wrong answer demonstrates the student is being sloppy in checking loop condition, often off by one.	`>>> x = -9`
		`>>> y = -12`
		`>>> while x:`
		`...    if y:`
		`...        print(y)`
		`...    y += 3`
		`...    x += 3`
		`Infinite Loop`
		`# Should be -12, -9, -6`

163	This wrong answer demonstrates the student is not reading the instructions properly. Specifically "Error" when there is an error, "Nothing" for no output, "Function" for printing out functions.	`>>> f = lambda: "hello"`
		`>>> f`
		`<function <lambda> at 0x000>`
		`# correct: Function`

Topic: Sloppy (cont.)

#	Description	Code Example (Python)

164	This wrong answer demonstrates the student is being sloppy by skipping a line of code.	`>>> n = 3`
		`>>> while n >= 0:`
		`...   n -= 1`
		`...   print(n)`
		`3`

165	This wrong answer demonstrates the student is being sloppy in evaluating the value for a variable.	`>>> a = 1`
		`>>> b = 2`
		`>>> a`
		`2`

Topic: String

#	Description	Code Example (Python)

166	This wrong answer demonstrates the student believes `None` in concatenation with another string results in just the other string (like an identity property).	`>>> None + 'Whoo'`
		`'Whoo'`

167	This wrong answer demonstrates the student does not include quotes for a string object.	`>>> f = lambda: "hello"`
		`>>> f()`
		`hello # correct: 'hello'`

Appendix B

Reteaching Hints

Tag: 1
Concept: Conceptually Correct

Typo?

——
Tags: 2, 3, 4, 5
Concept: Assignment

An assignment statement first evaluates the expression on the right side of the |=| operator to get a value. This value is then bound to the name on the left side of the |=| in the first frame in the current environment. For example, while |a = 0| displays nothing, |a == 0| displays |True|. Notice that Python distinguishes assignment |=| from equality comparison |==|.

——
Tags: 3, 19
Concept: Equality with ==

In Python, we use |==| to compare the equality of two values. For example if |foo = 0|, |foo == 0| checks to see if |foo| has the value |0| which would result in a boolean evaluation.

——
Tags: 5, 39
Concept: Variable evaluation

A variable is evaluated to the value it was assigned. If it was not assigned prior to being evaluated a NameError is raised.

Tags: 6, 7, 8, 17
Concept: Boolean `and`

`|and| either returns the first false value evaluated in an expression or the last value of that expression if all the operands are true.`

Tags: 6, 7, 15, 16, 17
Concept: Shortcircuit

`In short-circuiting Python only cares about false-y values while evaluating an |and| expression, and truth-y values while evaluating an |or| expression. A |FALSE-Y_VAL and VAL| will always evaluate to |False|, and a |TRUTH-Y_VAL or VAL| will always evaluate to |True| so as soon as one of these values is known, the other subexpressions is superfluous, and therefore Python does NOT evaluate them.`

Tags: 8, 15, 16, 17
Concept: Boolean `or`

`|or| either returns the first true value evaluated in an expression or the last value of that expression if all the operands are false.`

Tag: 9
Concept: Boolean representation

`In Python there are boolean primitives |True| and |False| as opposed to prior languages that use |1| and |0|.`

Tags: 10, 12
Concept: Boolean falsey values

`In Python, there are several values that are considered false-y such that in their interactions with other values in a boolean expression, they functionally evaluate as False. These values are: |False|, |None|, |0| (all other integers are True), |\'\'|, |()|, |[]|, and |{}| (or any other empty Data Structure).`

Tag: 11
Concept: Boolean evaluation

When evaluating a boolean expression, it is completely evaluated such that there are no boolean operators left. For example, |False and not True or 3| becomes |False and False or 3| then |False or 3| and finally |3|.

Tags: 13, 14
Concept: Boolean not

The boolean |not| operator is used to negate the boolean value of its operand. It can also be paired with operators that perform comparisons and return boolean values, such as the keyword |is|. Note that when using |not| in conjunction with a value, the resulting evaluation will always be |True| or |False|. For example, |not 0| is |True|, |0 is not 1| is |True|.

Tag: 18
Concept: Boolean order of operations

In Python, just like with arithmetic operators, there is a specific order of operations for boolean operators. Evaluating from left to right, expressions within parentheses have the highest priority, followed by |not|, then |and|, and False or finally by |or|. For example, |True and False or not False| is evaluated as |(True and False) or (not False))| which is |True|.

Tag: 20
Concept: in operator

A usage of the |in| keyword in Python is to check for item membership in a collection. This always returns a boolean value. For example, |4 in {2, 3, 4}| will return |True| and |'hello' in [4, 'hi']| will return |False|.

Tag: 21
Concept: is operator

The |is| operator returns a boolean of whether two values are exactly the same object instance. For example, if |a = [1, 2]|, |b = a|, and |c = [1, 2]|, |a is c| will return |False|, while |a is b| will return |True|.

Tags: 23, 24
Concept: `sorted` function

The |sorted()| function takes in a collection (e.g. list, set, dictionary) and returns a list of those items in alphanumeric order. For example, |sorted({3, 9, 55, 28, -19})| returns the list |[-19, 3, 9, 28, 55]|.

Tag: 25
Concept: Dictionary

Dictionaries are unordered sets of key-value pairs that map the key in the pair to its value. Since the only way to get a value is with its key, if there are two key-value pairs with the same key, it is not clear which value should be returned. Therefore dictionary’s keys must be unique.

Tags: 25, 27, 33
Concept: Setting a value in a dictionary

A dictionary maps keys to values. So if a key is not in the dictionary, a new entry is created for that key-value pair. If a key is present, its value is updated. For example, with |d = {’John’: ’Google’}|, when |d[’foo’] = ’bar’| and |d[’John’] = ’UCB’| are executed, |d|’s value is |{’foo’: ’bar’, ’John’: UCB’}|.

Tag: 26
Concept: `in` with dictionaries

Python uses |in| to check whether a dictionary has a key and returns a boolean value. For example, with |d = {’a’: 1, ’b’: 2, ’c’: 3}|, the expression |’a’ in d| returns |True|, while |’z’ in d| returns |False|.

Tags: 28, 29, 30, 31
Concept: Dictionary `keys` function

The |keys()| function returns a sequence of a dictionary’s keys. The call |sorted(list(DICTIONARY.keys()))| first creates a list from the sequence of keys and then sorts it. For example, with |d = {’c’: 1, ’a’: 2, ’b’: 3}|, |sorted(list(d.keys()))| returns |[’a’,’b’,’c’]|.

Tag: 32
Concept: Dictionary with `len` function

The length of a dictionary is the number of entries in that dictionary. For example, |{'one': 1, 2: two, 'three': '[1, 2]'}| has a length of 3.

Tag: 33
Concept: Getting a value from a dictionary

Python uses a key to get the value from a dictionary. For example, with |d = {'a': 1, 'b': 2, 'c': 3}|, in the call |d['b']|, the key |'b'| gets the corresponding value |2|.

Tag: 34
Concept: Parent frame following for scope

To find the value bound to a variable name |x| Python first looks in the current local frame environment. If no |x| exists, Python checks next in the parent frame, the next parent's frame, so forth until it reaches the Global Frame. If |x| is not defined in the Global Frame, a NameError is raised.

Tag: 36
Concept: `StopIteration` error handling and use

The |StopIteration| error is raised when the iterator has no more return values. A Python |for| loop automatically catches this error. However, if a |for| loop is not used, an explicit |try...except| is needed.

Tag: 37
Concept: Divide by zero erroring

When dividing by zero, Python raises an error, displays the error, and stops running the program. This is because dividing by zero mathematically leads to an undefined value. So Python cannot compute the value needed to complete its task.

Tag: 38
Concept: Expression evaluation

An expression is a combination of values, variables, operators, and functions that a programming language interprets and evaluates to the simplest value. Expression are evaluated upon hitting \"Enter\" or \"Return\". Primitive values (e.g. |1| or |\"hello\"|) evaluate to themselves, while expressions are evaluated based on the values, variables, operators, and functions it is made up of.

Tags: 40, 41, 42, 44, 45
Concept: Function call evaluation

A function is structured as |OPERATOR(OPERANDS)|, where OPERANDS is 0 or more OPERANDs. To evaluate the function: (1) evaluate the OPERATOR, (2) evaluate each OPERAND, and (3) apply the OPERATOR to the OPERANDs.

For example, in |pow(a, 2)|, where |a = 3|, (1) |pow| is evaluated to get the function value, (2) |a| evaluates to |3| and |2| is |2|, (3) apply the value of |pow| to the two arguments |3| and |2| to get the value |9|.

Tag: 41
Concept: Composed function evaluation

A function is structured as |OPERATOR(OPERANDS)|. Each OPERAND in OPERANDS must be fully evaluated before applying the OPERATOR to them. When the OPERAND is another function call, this is referred to as function composition. For example, to evaluate |fizz(foo(bar(1)))| we first evaluate |bar(1)| as an argument for |foo(...)| which in turn is evaluated to become the argument for |fizz(...)|.

Tag: 43
Concept: Displaying a function

A function in Python can be passed around as a value without executing the body. When the Python interpreter displays a function it uses the format: |<function FUNCTION_NAME at LOCATION>|. Additionally, Lambda functions use the format |<function <lambda> at LOCATION>|.

——

Tag: 46
Concept: Stacked function call evaluation

When a function returns another function, a quick way to call the returned function is to use another set of parentheses. These parentheses have the returned function's arguments. For example, |foo(23)(1, 4)| where |(1, 4)| is passed through as the arguments for the function that |foo()| returns.

——

Tags: 47, 51
Concept: iterfunction for generators

Like all iterators, generator objects have an |__iter__| method. This implicit method always returns the current generator, a.k.a. |self|.

——

Tags: 48, 52, 53, 54, 55
Concept: yield keyword for generator

A generator is a function that uses |yield|. Upon reaching a |yield| during an iteration, Python returns the evaluation of the statement and stores the state of the current frame for when the generator needs to return the next value.

——

Tags: 49, 50
Concept: next function for generators

A generator remembers its environment frame and its previous location. The initial values are an empty frame and the \"previous location\" is the function beginning. When |next| is called, it loads the saved frame, starts evaluating from its previous location, and runs until it hits the next |yield|.

——

Tags: 49, 52, 56, 57, 58, 62, 63, 71, 94, 95, 100, 101, 107, 112, 117, 135, 150, 157, 159, 160, 161, 162, 163, 164, 165
Concept: Sloppy

Try:
1) Reading the code or instructions more carefully,
2) Writing out the variable's values, or
3) Drawing an environment diagram.

Tags: 56, 57
Concept: If condictionals

A conditional statement consists of a series of headers and suites: a required |if| clause, an optional sequence of |elif| clauses, and finally an optional |else| clause. To execute a conditional clause, each clause is considered in order: (1) Evaluate the header's expression. (2) If it is a true value, execute the suite. Then, skip over all subsequent clauses in the conditional statement. (3) If the |else| clause is reached, only its suite is executed.

Tags: 59, 60
Concept: Iterator `iter` function

The Python iterator interface requires an |__iter__| method. This must return an iterable object, usually the current object |self|. |__iter__| is special and is called with |iter(ITER)|.

Tags: 61, 62, 63, 66, 67
Concept: Iterator `next` function

The Python iterator interface requires a |__next__| method. It returns the subsequent element of the sequence by executing the code in the method. |__next__| is special and is called with |next(ITER)|.

Tags: 64, 65
Concept: Calling `next` after a `StopIteration` was raised

Once a StopIteration is raised, each subsequent call to the |__next__| method conventionally raises a StopIteration again (technically it depends on the code in the |__next__| method).

Tags: 68, 69, 71, 72
Concept: Evaluating a lambda call

A lambda is defined as follows: |lambda PARAMETERS: RETURN_VAL|. When a lambda is called, the arguments are assigned to the PARAMETERS in the lambda's frame. Then the RETURN_VAL is evaluated and returned.

——

Tags: 70, 73
Concept: Defining a lambda function

A lambda is defined as follows: |lambda PARAMETERS: RETURN_VAL|. PARAMETERS is 0 or more parameters and RETURN_VAL is the expression that is evaluated and returned when the lambda is called.

——

Tags: 74, 78
Concept: Retreiving a link's first and rest attributes

The |first| attribute of a |Link| object has the link's value, while the |rest| attribute is a pointer to the next link or |Link.empty|. Keep in mind the value of any attribute can be anything (even a pointer to itself).

——

Tag: 75
Concept: `Link.empty` meaning

|Link.empty| represents an empty linked list and is used to mark the end a list. Thus it does not have any attributes of a link object. For example, if
|l = Link(1, Link(2))| then |l.rest.rest == Link.empty| is true.

——

Tags: 76, 79
Concept: Defining a link

A linked list is made up of Link objects that each have two attributes, |first| and |rest|. |first| contains the element inside the Link, while |rest| refers to the next Link object in the list.

——

Tags: 76, 104, 106, 111
Concept: OOP instance attributes

The value of an attribute is based on the current environment, first looking in the instance, then the instance's class. To access an attribute, Python uses the notation |OBJECT.ATTRIBUTE|.

———

Tag: 77
Concept: Setting the values in a link

A Link object is mutable, therefore the instance attributes |first| and |rest| can be modified. For example, if |l = Link(1)|, with |l.first = 9| and |l.rest = Link(3)|, |l| becomes |Link(9, Link(3))|.

———

Tags: 81, 82, 83, 94
Concept: List comprehension evaluation

The list comprehension format is |[EXPRESSION for ELEMENT in SEQUENCE if CONDITIONAL]|, where the |if CONDITIONAL| is optional. For every ELEMENT in SEQUENCE where CONDITIONAL is true, evaluate EXPRESSION with ELEMENT and append the result to a list. Return the list when through with SEQUENCE.

———

Tag: 84
Concept: List concatenation

Concatenation is creating a new list from multiple lists, ordering the elements by the order of the lists that were summed together. For example: |[3, ’two’, 1] + [] + [4, 3]| results in |[3, ’two’, 1, 4, 3]|

———

Tags: 85, 86, 92, 96
Concept: Python list syntax

A list in Python is contained within square brackets and separated by commas. The elements in the list can be be of any type, including sub-lists. For example: |[1, 2.0, [3, 4], ’5’, True]|.

———

Tags: 87, 88, 89, 90, 91
Concept: List indexing

When indexing into a list, we start counting at 0. This means to get the first element we must use the index zero. For example, in this list |x = [1, 2, 3]| to get |1| use |x[0]|. Notice that the value returned is only the element at that index.

——

Tag: 93
Concept: Range function

In Python, |range(START, STOP)| is used as a means of creating a sequence of integers that begins on START and ends right before STOP, incrementing by 1. If a START value is not given, START is set to 0. For example, |range(-1, 3)| iterates through -1, 0, 1, and 2, while |range(3)| iterates through 0, 1, and 2.

——

Tags: 97, 99, 102
Concept: Floating vs floor division

Python has two division operators: |/| and |//|. |/| is traditional division, it results in a decimal value. For example, |8 / 4| evaluates to |2.0|. |//| floors the result down to an integer. Also, if one of the operands is negative, the result is rounded *towards* negative infinity. For example, |5 // 4| evaluates to |1| and |-5 // 4| evaluates to |-2|.

——

Tag: 98
Concept: Negative number representation

In Python negative numbers are represented with a negative sign in front of the number. Therefore the negative value of |10| is represented as |-10|.

——

Tag: 103
Concept: Arithmetic order of operations

In Mathematics, and thus in Python, arithmetic evaluation follows the order of operations commonly known as PEMDAS, which stands for "Parentheses, Exponents, Multiplication/Division, and Addition/Subtraction" and is evaluated from left to right. For example the expression |4 + 2 - 4 * 3 / 4 + 2**2| will be evaluated as |((4 + 2) - ((4 * 3) / 4)) + (2**2)| or |7.0|.

——

Tag: 105
Concept: Defining class attributes

A class attribute can be defined either within the |def| of the class or by assigning a value to a new attribute with dot notation after the creation of the class, |CLASS.ATTR = VAL|.

——

Tag: 108

Concept: Class versus instance attribute evaluation

When Python is evaluating an attribute it follows this chain (instance -> class -> base-class -> ...). Attribute evaluation can start at either the instance or the class, whichever is on the left side of the dot notation. Python returns the attribute value the first time it finds it in the chain.

——

Tag: 109

Concept: OOP instance methods

When a class instance method is called on an instance, |INST.METHOD(...)|, Python fills in |self| with a pointer to that instance. When it is called using the class, |CLASS.METHOD(INST, ...)|, |self| must be provided.

——

Tag: 110

Concept: OOP attribute inheritance

If a subclass does not explicitly have a method or attribute, Python follows the chain of base classes until it is found or otherwise raises an error.

——

Tag: 113

Concept: Printing versus returning

When a function has a |print()| the arguments are always displayed in the interpreter, but Python continues executing the rest of the function. This contrasts with a |return| statement which ends the function and potentially displays the return value, depending on how the function was called.

——

Tags: 113, 116

Concept: Print function

|print()| takes its arguments' values and displays them separated by spaces. Note that a string's value is the text between the quotes, thus when printing the quotes are not displayed. For example, printing the string \"Hello World\" will result in the interpreter displaying Hello World (no quotes).

——

Tag: 114
Concept: Printing None

Python will only display |None| when the |print()| displays it. This is because the print function converts the value of its argument into a string and displays the content of that string.

——

Tag: 115
Concept: Print function's return value

The function |print()| implicitly returns the value of |None|. The Python interpretor does not display |None| unless it is explicitly forced to do so.

——

Tag: 118
Concept: When the Python interpreter displays nothing

The Python interpreter evaluates an expression and displays its output immediately. Evaluations that results in |None| and variable assignments do not have an output, and thus for these statements nothing is displayed in the interpreter.

——

Tag: 120
Concept: Scheme variable evaluation

In Scheme, a variable is evaluated to the value it was assigned using |define|. If it was not assigned prior to being evaluated, an error is raised.

——

Tags: 120, 126, 127
Concept: Scheme output when a symbol is defined

|define| has two formats: |define (PROC_NAME PARAMETERS) PROC_BODY| and |define VAR VAL|. The first creates a procedure called PROC_NAME that runs PROC_BODY using PARAMETERS. The second assigns VAL to the name VAR. When a procedure or variable is defined, its name is displayed in the interpreter.

Tags: 121, 122, 149
Concept: Scheme `cdr`

|cdr| returns the second value of a pair. With a linked list, |cdr| returns a pointer to the next pair but with a malformed list, |cdr| returns a value. For example, |(cdr ’(1 2 3))| returns |(2 3)| and |(cdr ’(1 . 2))| returns |2|.

Tags: 123, 124, 125, 142
Concept: Scheme `cons`

|cons| takes two arguments to form a pair, like a link in link lists. Also like link lists, conventionally the second value of the pair is another pair. If the second value is not a list, a malformed list is created. For example:

```
scm> (cons 1 (cons 2 ()))   +-------+  +-------+  |  scm> (cons 1 2)   +-------+
(1 2)                       | 1 | x -->| 2 | \ |  |  (1 . 2)           | 1 | 2 |
                            +-------+  +-------+  |                    +-------+
```

Tags: 123, 140
Concept: Scheme malformed lists

A list is malformed if the second value of a pair is not another pair. A |.| is used to separate the two values of the pair. For example, |(cons 1 (cons 2 3))| evaluates to the malformed list |(1 (2 . 3))|.

Tags: 128, 132, 133, 136, 137, 138, 139, 150
Concept: Scheme list display

To display a list, surround each pair with |()| and separate its values with a |.| surrounded by spaces. For each |.| immediately followed by a |(|, remove the |.| and matching |()|. For example, |(cons 1 (cons (cons 2 3) (cons 4 ())))| becomes |(1 . ((2 . 3) . (4 . ())))| and reduces to |(1 (2 . 3) 4)|.

Tags: 128, 143
Concept: How scheme empty lists are displayed

The empty list in Scheme, |()| or |nil|, denotes the end of a list, like Python link lists. |null?| returns whether a value is the empty list. The empty list is considered a list, but not a pair so using |car| or |cdr| on it causes an error.

Tags: 131, 134, 142, 148
Concept: Scheme `list` function

|list| takes zero or more arguments and returns a list with each argument as an element in the list. For example:

```
scm> (list 1 2 3)    +-------+  +-------+  +-------+
(1 2 3)              | 1 | x -->| 2 | x -->| 3 | \ |
                     +-------+  +-------+  +-------+
```

Tags: 142, 144, 145, 146
Concept: Scheme using quote to create lists

|quote| or |’| acts as shorthand to create a list with the list display syntax. Anything in the list becomes symbolic and is therefore not evaluated. For example, |(quote (1 2 3))| creates |(1 2 3)| and |’(1 . 2)| creates |(1 . 2)|, while |’(cons 1 2)| creates |(cons 1 2)| where |cons| here is the symbol |cons|.

Tag: 149
Concept: Scheme `car`

|car| returns the first value of a pair from either a regular or malformed list. This value can be of any type. For example, |(car ’(1 2 3))| returns |1|, |(car ’(1 . 2))| returns |1|, and |(car ’((1 2) 3))| returns |(1 2)|.

Tags: 151, 152, 153, 154, 157
Concept: Intersect function

The intersection operator (shorthand |&|) is used to determine the common elements in 2 or more sets. For example, where |s = {1, 3, 5}|, |t = {1, 4, 9}|, and |u = {1, 2}|, |s.intersection(t, u)| will return |{1}|.

Tag: 155
Concept: Sets

A set is a collection of elements that are enclosed by curly braces |{}| and adhere to certain properties: elements cannot have duplicates and elements are unordered. For example, |{1, 2, 1, ’hello’, 4}| results in |{4, 1, ’hello’, 2}|.

—

Tag: 156
Concept: Set differencing

The difference operation (shorthand |-|) is used to determine elements that are in the first set but not in the other set. For example, where |s = {1, 3, 5}| and |t = {1, 2}|, |s.difference(t)| will return |{3, 5}|.

—

Tag: 158
Concept: Union function

The union operation (shorthand |||) is used to create a set of all the elements present in 2 or more sets. For example, where |s = {1, 3, 5}|, |t = {1, 4, 9}|, and |u = {1, 2}|, |s.union(t, u)| will return {1, 2, 3, 4, 5, 9}.

—

Tag: 167
Concept: String syntax

In Python, a string is a series of characters. We declare something as a string by surrounding it with either single |’’| or double quotes |\"\"|. We can recognize strings in the Python interpreter by these quotes. For example: ’This is a string’ and \"This string’s contents has an apostrophe\".

Appendix C

Knowledge Integration Hints

Tag: 1

`Typo?`

Tag: 2

`Look closely at the assignment statement. What did the function call return?`

Tag: 4

`Should this assignment statement display a value?`

Tag: 7

`When does |and| short circuit?`

Tag: 8

`What are the values a boolean expression can evaluate to?`

Tag: 9

`How are boolean values represented in Python?`

Tag: 10

`What values are false-y in Python?`

Tags: 11, 38

Can’t the expression be evaluated more?

Tag: 12

Is a nonzero value truth-y or false-y?

Tag: 14

What about the |not|?

Tag: 15

What about short circuiting?

Tag: 16

When does |or| short circuit?

Tag: 17

Remember the difference between |and| and |or|.

Tag: 18

In what order are boolean expressions evaluated?

Tag: 19

What type does an |==| return?

Tag: 20

What does |in| do? What type would it return?

Tag: 21

What does |is| do? What type would it return?

Tag: 24

`What does |sorted()| return?`

Tag: 25

`Can dictionaries have duplicate keys. If so, how would you access the values?`

Tag: 26

`How does |in| work with dictionaries?`

Tag: 27

`Can dictionaries be updated?`

Tag: 29

`Remember what |keys()| returns?`

Tag: 32

`Remember what |len| does with a dictionary.`

Tag: 33

```
Remember the steps to evaluate an assignment. What is the difference between
dictionary lookup and adding/updating a key-value pair?
```

Tag: 34

`Double check the variable's value for each frame?`

Tag: 36

`How do |for| loops handle StopIteration errors?`

Tag: 37

`Can Python divide by |0|?`

Tag: 39

If the value of a variable cannot be found, what happens?

Tag: 41

Be careful about what each function call receives as an argument, what each function call returns, and in what order this process happens.

Tag: 42

Look carefully at the parameters for the function. What kind of arguments does the function call need?

Tag: 42

Look carefully at the parameters for each function. What kind of arguments does each function call need?

Tag: 43

Is the variable really bound to |None|?

Tag: 43

What does the interpreter display when a function is created?

Tag: 44

If f is a function, what is the difference between |f| and |f()|?

Tag: 46

What are the latter parentheses doing? Remember the steps to evaluate a function call.

Tag: 47

What does a generator object's |__iter__| function do?

Tag: 48

Would Python go to the top of the function when it hits the end of a loop?

Tag: 49

Double check the generator's state at this point?

Tag: 50

This is a subsequent call to |next|, so where should Python continue from in the generator?

Tag: 52

Look closely at when the code increments versus yields in the generator.

Tag: 53

What happened to the |yield|? Remember what Python does with |yield|.

Tag: 56

Hmm... Are there more than just |if| clauses there?

Tag: 57

Hmm... Are there |elif| or |else| clauses here?

Tag: 58

Double check the if condition?

Tag: 59

Read |__iter__| carefully. What is it returning?

Tag: 62

Look closely at when the code increments versus returns in the |__next__| function.

Tag: 63

Look closely at when the code increments versus raises the StopIteration error.

Tags: 64, 61

What is the state of the iterator at this point? Specifically, what are the values of its attributes when the |__next__| function is called?

Tag: 68

Isn’t this calling the returned function?

Tag: 68

What does this lambda function return when called?

Tag: 68

Isn’t this calling the function?

Tag: 68

Make sure to evaluate each function fully.

Tag: 70

Does defining a function evaluate it?

Tag: 72

Look more closely at the lambda’s return value.

Tag: 74

That output looks like it can be executed to get a value. What about answering with a value instead?

Tag: 75

Double check |Link.empty|'s implementation, especially its attributes. Can it be used like a |Link| instance?

Tag: 77

Double check if the object has changed since it was initialized?

Tag: 78

As the code executes, draw the box and pointer diagram and think about what are valid |rest| values.

Tag: 81

Hmm...isn't this a list comprehension? What is the type of its return value?

Tags: 82, 83

Is that the only value the list comprehension returns?

Tag: 84

Remember the result when concatenating lists together with |+|.

Tags: 85, 86

Is that a list? Python can't tell that it is.

Tag: 88

What index is the first element of a list?

Tag: 89

`Remember how Python indexes into nested lists.`

Tag: 90

`When indexing into a nested list, where do the brackets go?`

Tag: 91

`How many elements are in that list? What type are they?`

Tag: 93

`Remember what the start and end values are for |range|, given its argument(s).`

Tag: 95

`Hmm... is that completely sorted alphanumerically?`

Tag: 98

`What about negative numbers?`

Tag: 99

`What is the difference between |\| and |\\|?`

Tag: 100

`Hmm... The math doesn't seem to add up.`

Tag: 101

`Look more closely at the variable's initial value and how it is being changed.`

Tags: 106, 165

`Double check the variable's value?`

Tag: 107

`Double check the instance’s class.`

Tag: 108

`Check whether the right attribute/method is being evaluated. Is that from the instance or the class?`

Tag: 109

`What kind of method is this? How does Python handle calling a method with the class on the dot’s left versus calling with the instance on the dot’s left?`

Tag: 110

`What happens when Python can’t find an attribute on a class?`

Tag: 111

`What happens when Python can’t find an attribute on an instance?`

Tags: 112, 76

`Double check the attribute’s value?`

Tag: 113

`What effect does |print()| have in a function?`

Tag: 114

`What is the difference between printing and displaying |None|?`

Tag: 115

`What is |print()|’s return value?`

Tags: 116, 167

Is this printing or returning the string?

Tag: 117

Hmm... Missed a |print()|?

Tag: 118

What about displaying the return value?

Tag: 120

Hmm... When the list was made, what were the variables bound to?

Tags: 120, 5

Hmm... What is the variable bound to?

Tag: 121

Is that the second value of the pair? Is it a value or a list?

Tag: 121

Try drawing out the box and pointer plot. What is the variable pointing to? What would |cdr| return?

Tags: 123, 124

Remember how |cons| works. What does it do with its arguments and how should the result be displayed?

Tag: 126

Remember what |define| returns.

Tag: 128

Recall what |nil| represents. How is it displayed?

Tag: 131

Remember how |list| works. What does it do with its arguments?

Tag: 131

Remember how |list| works. What does it do with its arguments and how should the result be displayed?

Tags: 132, 150

Is that how Scheme would display the list? Recall the rule for displaying pairs.

Tag: 133

Is that a list? Scheme can’t tell that it is.

Tag: 135

Follow the |car|’s and |cdr|’s more carefully. Is that the correct element in the list?

Tag: 137

Double check the list structure and if this list is displayed correctly.

Tag: 138

Hmm... That looks like code. Would the Scheme interpreter output code here?

Tag: 139

Is that a decimal or a malformed list?

Tag: 142

`Double check how that list was made and if this list is displayed correctly.`

Tag: 144

`Looks like there is a quote in there. What does that do?`

Tags: 145, 146

```
Remember what quoting does with lists. Double check the list structure and how
to display it.
```

Tag: 148

```
Try drawing out the box and pointer plot and double check the exact structure of
the answer.
```

Tag: 149

`Remember the difference between |car| and |cdr|.`

Tag: 151

`Double check how |&| works with sets.`

Tag: 153

`Double check how ||| works with sets.`

Tag: 154

`Remember the difference between intersect, union, and difference.`

Tag: 155

`Can a set have duplicates?`

Tag: 157

`Double check the set intersection?`

Tag: 160

`When was the value set to that? Double check how far into the execution this is.`

Tag: 161

`Remember that prior questions may affect this one.`

Tag: 162

`When does the loop end?`

Tag: 163

`Double check the instructions?`

Tag: 164

`Hmm... Did you miss a line of code?`

Tag: 167

`How does Python know whether a value is an integer, variable, or string?`