Quickly Producing "Isomorphic" Exercises: Quantifying the Impact of Programming Question Permutations

Max Fowler
University of Illinois
Urbana, IL, USA
mfowler5@illinois.edu

David H. Smith IV
University of Illinois
Urbana, IL, USA
dhsmith2@illinois.edu

Craig Zilles
University of Illinois
Urbana, IL, USA
zilles@illinois.edu

ABSTRACT

Small, auto-gradable programming exercises provide a useful tool with which to assess students' programming skills in introductory computer science. To reduce the time needed to produce programming exercises of similar difficulty, previous research has applied a permutation strategy to existing questions. Prior work has left several open questions: is prior exposure to a question typically indicative of higher student performance? Are observed changes in difficulty due to the specific surface feature permutations applied? How is student performance impacted by the first version of a question to which they may be exposed?

In this work, we pursue this permutation strategy in multiple semesters of an introductory Python course to investigate these open questions. We use linear regression models to tease out the impacts of different surface feature changes and usage conditions. Our analysis finds similar tendencies as in prior work: question versions available in study materials tend to have 5 – 11 percentage point higher scores than novel permutations and more "substantial" surface feature changes tend to produce harder questions. Our results suggest this last finding is sensitive to how evenly permutations are applied across existing questions, as the precise impact of individual permutations changes between semesters.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; **Student assessment**.

KEYWORDS

CS1, introductory computer science, programming surface features, assessment, programming exercises

ACM Reference Format:
Max Fowler, David H. Smith IV, and Craig Zilles. 2024. Quickly Producing "Isomorphic" Exercises: Quantifying the Impact of Programming Question Permutations. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653617

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ITiCSE 2024, July 8–10, 2024, Milan, Italy
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0600-4/24/07
https://doi.org/10.1145/3649217.3653617

1 INTRODUCTION

One key way to assess students' ability to program is through programming exercises. However, assessing programming ability faces a few hurdles. First, enrollment in computer science departments has grown in both the major and non-major serving courses each year [2, 6, 9, 16]. This adds a strain to assessing programming skill in the form of grading overhead. Computer-based assessment partially mitigates this overhead by leveraging automated grading [1, 14, 40].

Mitigating grading time does not totally overcome the second hurdle: *question authoring time*. It takes time to come up with new programming exercises, write the prompts, and write and test the autograding scripts for them. Towards addressing this issue, we previously developed a permutation strategy to rapidly author new programming questions based on existing prompts [17]. Our goal was to produce questions with similar topic coverage and similar difficulty for use on versioned exams (exams that are generated from question pools for each student) to mitigate cheating. We produced a "taxonomy" of question surface features (discussed in detail in Section 3.1) that could be used to generate the question permutations. The permutations were applied manually, but permitted a "copy and modify" strategy that reduced question development time relative to authoring brand new questions.

This approach showed promise based on the one semester of data we collected. The questions were of relatively similar difficulty: permuting base questions changed the average likelihood of students answering the new variants correctly by at most 11%. In addition, a notable feature of the previous work's data collection is that many of the permuted exam questions were generated from *base* questions — preexisting questions to which permutations were applied — that were on the students' homework, in an effort to align the content of formative and summative assessments [11, 13, 25]. We had included these base versions on some of the versioned exams to compare the performance of the base version to the permutations.

In this work, we aim to expand upon the original work with five semesters worth of data where in each semester we vary which version of the questions are included on the homework and/or practice exams. This experimental design allows us to more precisely characterize this permutation strategy. Specifically, we investigate the following research questions:

RQ1: How much easier is a question when that version of the question was on homework?
RQ2: Which permutations consistently impact question difficulty across multiple semesters?
RQ3: Do base questions tend to be easier than their permutations (that is, do we as question authors write the most obvious/easiest questions first)?

2 RELATED WORK

2.1 Isomorphic questions

The term "isomorphic question" refers to variant exercises for which a number of elements are kept the same between versions [3]. In our work, we are most interested in *item isomorphs*, for which the underlying solution does not significantly change, therefore testing largely the same concepts at a similar level of difficulty as the original question [23, 24].

There is growing work in computing focusing on the creation of these isomorphic questions. Butler et al. [5] manually wrote variants of questions students would receive on exams in order to test whether they could be used for fair assessments (that is, each variant was isomorphic). They found not all variants were the same with respect to student scores or time-on-task. For example, variants which required testing for odd numbers appeared less intuitive and harder than variants testing for even numbers. Parker et al. [32] built a series of isomorphic questions in order to extend the available pool of high quality, validated questions for their Assessment of Computing for Elementary Students (ACES). Their changes included both changes to question prompts and changes to the display of answers and found that most of their changes produced significantly different outcomes, seemingly due to changes made in prompts involving spatial layouts [30]. Zingaro and Porter [43] wrote isomorphic questions based on class exercises to use on final exams, with each set of isomorphic questions having two in-class exercises (Q1 and Q2), then a final exam exercise (Q3). They found students who answered both Q1 and Q2 correctly were 81% likely to answer Q3 on the final and also found positive learning gains among students who missed Q1 initially but succeeded on Q2, having a 70% chance of getting Q3 correctly. Isomorphs have additionally been used to replicate the Foundational CS1 (FCS1) assessment [31], assess peer learning outcomes [21, 35], and facilitate smaller, more enjoyable, repeat testing through automatically generated test forms [24].

The permutation strategy work has currently focused on the goal of creating isomorphic questions. These variants are created by identifying what constitutes surface features of programming prompts, regardless of contextualization. In brief, the permutation strategy permutes what we currently believe to be the smallest unit of change in a question's prompt which *should* keep the question in the same category of problem and with a similarly difficult solution. As this strategy focuses on changing "surface features," we next briefly review some of the work on the features of question prompts.

2.2 The impact of question prompt features on problem solving

The impact of changing question features on intended isomorphs is well studied. For example, creating isomorphic versions of the classic Tower of Hanoi problem where towers are replaced by aliens or even more mundane things like flagpoles can increase problem solving time by at least three times [18, 28]. While the permutation strategy work is intended to create isomorphs, how novices recognize problems may complicate this matter [12]. In this section, we briefly review other work which looks at how question features impact student performance.

Differential focus on problems' features between experts and novices is a well-documented phenomena in a number of domains, including physics, mathematics, biology, and computer science [7, 8, 20, 27, 34, 38]. From a cognitive science lens, this differential focus and perception appears to impact novices' ability to successfully utilize key information about problems and produce correct solutions across multiple fields [36, 42]. Chi et al. [7] suggest students use this information to categorize problems and it is this categorization which may inform how students approach solving problems. Much of the work investigating these surface feature changes come from the physical sciences [19, 26, 41].

There is a growing body of work on how programming novices handle question contexts and features, with mixed results. Bouvier et al. [4]. conducted a multi-institutional study using a "Satellite problem", in a similar style to the classic Rainfall problem, in comparison to the same problem with all context removed, finding no difference in performance. Craig et al. [10] compared students' performance on longer form, context-rich programming prompts versus the same prompts divorced of context. Despite the hope that adding context would make questions easier, only one contextualized question saw better performance by students. Leinonen et al. [22] had contradicting results: they divided students into three groups (two with context, one without) and had them complete two function programming exercises each. While the contextless group performed similarly to the context groups when given an expected result from the function, they performed significantly worse when not given sample starting arguments to the function. Finnie-Ansley et al. [15] presented a group of 32 computing students in differing parts of their undergraduate education with a series of 30 programming question prompts to group into clusters in a card-sorting exercise. They found that students tended to focus on surface level details—question context, input and return types, and the need for error correction in the prompt—when asked to compare the prompts, but could be encouraged to group by common programming patterns when explicitly asked to consider the *solutions* to a problem. In summary, the impact of context and question features on computing novices appears to depend heavily on their familiarity with that context and whether the context can provide guidance.

3 METHODS AND DATA

3.1 Question variants using surface feature permutations

For our study, we apply a surface feature permutation strategy introduced in previous work [17]. The process starts by writing a *base* version of a question that addresses a given set of learning objectives. *Variants* are written from that base, by manually applying one or more permutation strategies to the base question. This involves changing the question prompt, any given code, and test cases (when automatic scoring is being performed). The permutation strategies are listed in Table 1, with the less impactful permutations at the beginning and the most impactful permutations at the end, as per the previous work [17].

Below, we illustrate the variant writing process using a Python programming question from our data set. We will start with the question *progListEmpty*, with the following prompt:

Permutation	Description
Base	The pre-existing question from which variants were generated (i.e., the first question written in a set of variants)
Var Name Change	Change the name of parameters used in a question prompt.
Function Name Change	Change the name of the function to be written in a question prompt.
Prototype Added	Add a function prototype to a prompt without one.
Prototype Removal	Remove the provided function prototype from a prompt.
Order Swap	Swap the order of parameters in a function.
Constant Change	Change constants involved in a prompt, e.g., "last 3 elements …" to "last 5 elements …".
Polarity Reverse	Change question "polarity" e.g., "positive" to "negative", "greater than" to "less than".
Data Type Change	Change the data types of one or more parameters, e.g., lists to dictionaries or integers to floats.

Table 1: The informal surface feature taxonomy from prior work [17]. We group "Order Swap" with the perceived more "complex" permutations.

> Create a function called is_list_empty that takes a single argument of type list. Your function should return True if the list is empty (i.e., has no elements). Otherwise, return False.

One new variant in Fall 21 used *Polarity Reverse* and *Function Name Change*, which we present below with changes bolded:

> F21-1: Create a function called **determine_content** that takes a single argument of type list. Your function should return **False** if the list is empty (i.e., has no elements). Otherwise, return **True**.

3.2 Course structure

Our data comes from five semesters worth of exams from a large enrollment, introductory Python course for non-majors. The number of students varies by semester, as well as by exam (due to attrition). The maximum number of students per semester is as follows: 583 from Spring 2020, 645 from Fall 2020, 697 from Spring 2021, 720 from Fall 2021, and 576 from Fall 2022. Each semester had the same exam structure: three 50-minute exams with approximately 30 questions each and one 180-minute final exam with approximately 46 questions. All exams and practice exams are hosted using a web-based exam platform, PrairieLearn [40]. Exams were all taken on a computer with Python interpreter access, but the testing environment differs slightly between semesters. For Exam 1 in Spring 2020 and for all of Fall 2021 and Fall 2022, students took their exams in a proctored computer lab. In the rest of Spring 2020 and for Fall 2020 and Spring 2021, exams were instead taken synchronously online with Zoom proctoring due to the COVID-19 pandemic. Spring 2022 featured an exam structure that differed from the course's typical structure and is omitted from the paper's analysis.

PrairieLearn supports randomly-versioned exams, where programming questions are randomly selected from question pools [40]. Each pool of questions is intended to test a particular set of learning objectives at a particular difficulty level. The questions that were involved in the surface feature permutation were short programming question prompts whose answers require less than 10 lines of code and that use test cases for automatic grading. Our data analysis includes those questions involved in variant permutation and the other programming questions from the exam (prompts that were not permuted) to improve model fit.

3.3 Homework, practice-first, and hidden

A significant number of our permutation groups included one (or more) of their versions on the homework and/or practice exams. This was intended both to motivate students to understand the homework and practice exam material (rather than plagiarize), because doing so would prepare them for the exam, and to align the formative and summative assessment in the course. For this research, we included these homework and practice exam questions in the question pools on exams in order to evaluate how much having previously seen a version of the question would impact the score on that question on the exam. Questions that the students could see first on homework are called *homework* questions. Those questions which appear first on practice exams are *practice-first* questions. Finally, any questions which appear only on exams are referred to as *hidden* questions.

Each semester, we varied which questions from a permutation group were on the homework. In this way, we could decouple the effect of having seen a question before from the inherent difficulty of the question. In Spring 2020, only base[1] questions were homework questions; in later semesters, a mix of bases and variants were homework questions. Further, exam construction saw more difficult questions rotated out as the semesters progressed, which had an impact on the ranges of difficulties present.

3.4 Data Collection and Analysis

With IRB approval, we retrieved the scores students received on these questions during exams. Students were given multiple attempts with a point being deducted from the maximum score for each incorrect attempt. Intermediary partial credit is given based on the number of tests cases passed in between each attempt. In total, we have 11,433 scores for Spring 2020, 13,538 scores for Fall 2020, 13,830 scores for Spring 2021, 13,713 scores from Fall 2021, and 12,189 scores from Fall 2022.

In order to isolate the impact of permutations (and whether the question was on homework or practice exams), we used an ordinary least squares regression with the *statsmodel* Python package [37]. We ran the same model on all of the data combined, as well as for each semester individually. The general form of the model is as follows:

$$score_{ij} = \left(\sum_{k=0}^{z} \alpha_k V_{ki}\right) + \beta_j S_j + \gamma_i B_i + \sigma_i H_i + \theta_i P_i \qquad (1)$$

In the above formula, i refers to a specific question, j refers to a specific student, k refers to a specific permutation type, and z is the total number of permutations that could be applied. Our

[1]As mentioned in the introduction, we use the term "base" to refer to the first version of a permutation group that was written; the version the permutations were "based" on.

dependent variable, *score*, ranges continuously from 0 to 1 to capture the possible partial credit students can receive on a question. Our independent variables and coefficients are enumerated below:

- α_k is the coefficient associated with a given permutation type. V_{ki} is an indicator variable representing whether permutation k was applied to question i.
- β_j is the coefficient for the ability of student j and S_j identifies the student.
- γ_i is the coefficient describing the difficulty of i's base question B_i.
- σ_i is the coefficient capturing the added difficulty of a question being hidden, where H_i is an indicator variable for whether i was hidden.
- θ_i is the coefficient capturing the added difficulty of a question being practice-first, where P_i is an indicator variable for whether i first appeared on practice exams.

We chose our independent variables such that we could tease out the impacts of each permutation strategy and the impact of being a hidden question, while controlling for individual student ability and the overall difficulty of the questions in a set of variants as indicated by their shared base's difficulty. The reference category here would be a generic question with no permutations applied and no other variants that appeared on homework first.

4 RESULTS

The results from all runs of the regression are provided in Table 2. All adjusted R^2 values are between 0.2 and 0.5, which recent work considers acceptable for examining correlations with respect to human behaviors and performance *so long as* some predictors are statistically significant [29]. Our regression coefficients serve as unstandardized effect sizes [33]. As a reminder for interpretability, our dependent variable is a continuous score ranging from 0 to 1. A coefficient of 0.05 would indicate a 5 percentage point change in score. Given we have no coefficients with magnitude over 0.11, we find our effect sizes reasonable for question level score differences.[2]

4.1 Hidden and practice-first questions always have lower predicted scores

As one might expect, the first result from the regression' coefficients is that questions being hidden — that is, appearing only on exams — and appearing on practice-first predict a lower score than questions which are previously provided to students on homework. All semesters have significant p values, with most under $p < 0.001$. In general, questions being hidden predicted a score difference of 11 to 7 percentage points on programming prompts. On average, we would expect students to get 8 percentage points less on hidden questions accounting for permutation and student differences. Practice-first questions were similar, although not always statistically significant, with an average of 7 percentage points worse score outcomes. The coefficients are presented in Table 2.

Spring 2020 saw the largest impact from questions being hidden, predicting a 11 percentage point lower score than with non-hidden questions. Spring 2020 was also the semester where only the base versions of the questions appeared on homework. We expand on the idea of base questions always being easier in Section 5.4.

4.2 Permutations have a relative ordering

Next, we address the coefficients for the permutations applied to questions. The results in Table 2 here are similar to the results from prior work [17]. That is, the "complex" permutations (*OrderSwap* through *DataTypeChange*) tend to have highly significant ($p < 0.001$), comparatively large (often from a 3 to nearly 10 percentage point difference in score) negative coefficients, while the "less complex" (*VarNameChange* through *PrototypeAdded*) are not as significant or as large. However, there are outlier semesters, notably Fall 2021 with respect to the *DataTypeChange* permutation, where the questions were predicted as significantly *easier* with a relatively large effect (6 percentage points). This may be due to deliberate removal of known difficult questions during exam writing, as Fall 2022 sees the trend generally restored, which we expand upon in Section 5.3.

5 DISCUSSION

5.1 RQ1: Prior exposure to questions does lead to consistently higher performance

Our answer to **RQ1** is expected but helpful to confirm: regardless of the semester or permutations applied to questions, questions that appear only on an exam as opposed to being reused from homework have a lower score. The hidden coefficient is always significant and almost always at the $p < 0.001$ level. In practice, this means that we would expect the same question used as homework in one semester and as a hidden question in another semester to have a 5 - 11 percentage point lower score when used in the hidden condition.

Whether a question is hidden or not has more impact on question scores than most permutations. In some semesters, hidden is the largest coefficient; in others, only "complex" permutations have similar or larger coefficients. This general trend is shared by practice-first questions. In fact, in some semesters, practice-first can be *harder* than *hidden* questions. We attribute this to deliberate exam construction: one of the motivations for creating course practice exams was to have some of the hardest questions appear on practice exams as study motivators and to reduce anecdotal student complaints of exams being strictly harder than practice exams.[3]

5.2 RQ2: In general, the "complex" permutations lead to larger score changes

The answer to **RQ2** appears to be as follows: the more "complex" permutations of *DataTypeChange, PolarityReverse, ConstantChange,* and *OrderSwap* almost always produce more difficult versions of questions than the "less complex" permutations. The one outlier is *DataTypeChange* in Fall 2021. However, if we consider the absolute value of the change as what matters, this still fits the trend, as *DataTypeChange* produced consistently easier variants in that semester.

If we consider complexity as a measure of how consequential the change is to a question, these results are reasonable. Changing

[2]Unstandardized effect sizes are sensitive to the units they measure. An 11 percentage point difference in *exam score* would be a full letter grade. As our programming prompts range from 7 to 9 points, average differences are less than a full point.

[3]This did not work: students still claimed that the exams were harder than practice exams.

Variable	S20			F20			S21			F21			F22			Combined		
	coef	p	std err	coef	p	std err	coef	p	std err	coef	p	std err	coef	p	std err	coef	p	std err
Var Name Change	0.013	0.438	0.016	0.016	0.218	0.013	-0.009	0.477	0.013	0.007	0.474	0.010	0.016	0.159	0.011	-0.013	**0.007****	0.005
Function Name Change	-0.000	0.981	0.014	-0.034	**0.003****	0.011	-0.019	0.151	0.013	0.002	0.883	0.012	0.020	0.082	0.012	-0.007	0.138	0.005
Prototype Added	0.070	**0.002****	0.022	0.009	0.632	0.019	-0.080	**0.001****	0.023	-0.040	**0.039***	0.020	-0.058	**0.007****	0.021	-0.019	**0.033***	0.009
Prototype Removal	0.030	0.062	0.016	0.016	0.181	0.012	-0.038	**0.003****	0.013	-0.007	0.559	0.012	-0.011	0.324	0.011	-0.015	**0.004****	0.005
Order Swap	-0.069	**0.000*****	0.016	-0.075	**0.000*****	0.016	-0.104	**0.000*****	0.019	0.017	0.296	0.016	-0.017	0.264	0.015	-0.068	**0.000*****	0.007
Constant Change	-0.026	**0.035***	0.013	-0.077	**0.000*****	0.011	-0.029	**0.013***	0.012	-0.008	0.361	0.009	-0.023	**0.019***	0.010	-0.035	**0.000*****	0.004
Polarity Reverse	-0.072	**0.000*****	0.012	-0.083	**0.000*****	0.011	-0.082	**0.000*****	0.011	-0.013	0.146	0.009	-0.021	**0.041***	0.010	-0.057	**0.000*****	0.004
Data Type Change	-0.067	**0.000*****	0.015	-0.119	**0.000*****	0.014	-0.097	**0.000*****	0.018	0.064	**0.000*****	0.017	-0.024	0.107	0.015	-0.065	**0.000*****	0.007
Hidden	-0.114	**0.000*****	0.016	-0.097	**0.000*****	0.011	-0.085	**0.000*****	0.016	-0.083	**0.000*****	0.013	-0.069	**0.000*****	0.011	-0.088	**0.000*****	0.004
Practice	-0.054	**0.017***	0.023	-0.066	**0.000*****	0.016	-0.110	**0.000*****	0.017	-0.046	0.095	0.027	-0.081	**0.000*****	0.021	-0.106	**0.000*****	0.007
adj R^2	0.290			0.329			0.362			0.487			0.467			0.382		

Table 2: Relevant regression coefficients, presented as the predicted impact on question scores from 0 to 1. In general, the more "complex" permutations, later in the table, have larger and more significant results. *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

variable names, function names, or the presence of a function header does not substantively change a question's solution. Changing the data types involved, constants involved, or the direction/polarity of a prompt all have the possibility of creating more substantial, and more difficult, changes. For example, changing a question that compared the magnitudes of integers to compare the lengths of strings adds additional operations to the code. Likewise, consider a base question where students should *print every element in a list.* By changing the constant *every* to *every other*, the solution becomes one step more complicated, requiring students to include some sort of step-size in their loop.

5.3 RQ2: Variance due to exam construction

The most significant difference in impact across semesters is the notable positive coefficient for *DataTypeChange* in Fall 2021. Why might these differences exist between semesters? Part of this may be attributable to exam construction.

Starting in Fall 2020, but particularly in the 2021 semesters, members of the course staff put in explicit efforts to better balance the question pools on exams. This balancing involved removing variants that were notably easier or notably harder than others and reorganizing questions such that all the questions in a pool had similar historical averages. The impact this tended to have was removing more "complex" permutations, such as those using *DataTypeChange*, where the permutation had a notable score difference from their peers. These modifications changed the distribution of permutations across exams. By 2021, it is likely that remaining questions from "complex" permutations trended easier than the class' whole set of permuted questions.

In the future, any study looking at the impact of permutations should ensure that the permutations are applied evenly across the semester and across different difficulty base questions. Efforts to balance question pools are valuable for exam fairness, but complicate the analysis.

5.4 RQ3: Negative coefficients: We tend to write the most obvious exercise first

What is perhaps surprising is that not only are the magnitudes of the "complex" permutation coefficients large, they are pretty consistently negative. This suggests that the variants tend to be more complex than the base version even though, intuitively, permutation isn't inherently directional. That is, the base version can also be viewed as a permutation of any of the variants.

We believe that the base version tends to be easier precisely because it was the first one written. That is, we are likely to write the most obvious, straightforward version of a question first. This is clearly true in cases where we permute a question like *print every element of a list* to become *print every other element* (or, even more so, for a second *ConstantChange* variant *print every third element*). It may be less obvious, however, for permutations that substitute negative numbers for positive numbers, odd numbers for even numbers, or floats for integers, but these superficial differences can trip up novices.

We expect that "complex" permutations have a large effect on difficulty precisely because of the impact they have on the length and complexity of the solution code. Future work could look at the changes a permutation necessitates in solutions to programming exercises as a metric of permutation complexity.

5.5 RQ3: Why is OrderSwap so negative?

A surprising result is the large, significant, and negative coefficient for *OrderSwap*. Unlike the other consistently difficult permutation strategies, *OrderSwap* should not be making a consequential change in the solution. We came up with the following hypothesis: *OrderSwap* is difficult because the swapped parameters contradict more natural ordering for arguments. For example, if *OrderSwap* consistently swaps the start of a number range for the end of a number range, students may use these arguments in left-to-right order out of convention, regardless of the question prompt. As an informal check of this claim, we quickly investigated the 11 variants using *OrderSwap* in Fall 2021. We identified five questions worthy of discussion.

Three questions clearly defied a more typical order by swapping the start and ending point of numerical ranges. They were all members of the same variant set and were among the lowest in that set, with the worst having the lowest score of 39% versus the set's high of 80% and the others having scores of 53% and 55%, the third and fourth lowest performing variants in that set respectively. These results provide some support for our hypothesis as defying the natural ordering for these arguments produced the low scores.

The fourth question that looked like it might be impacted by unnatural ordering was focused on printing a function's parameters. The parameters *first,second,third* were swapped to *third,first,second* in a prompt to print a formatted string "{first} ! {second} _ {third}. Similarly to the number range statements, we expected that these arguments may be processed in a left-to-right fashion out of habit. However, this was not the case: this variant's average score was

approximately the same as other variants. In this case, defying a more natural ordering was not impactful.

Finally, there is a variant that swaps x1 and x2 in a prompt to "return the smallest of x1 and x2", which should not have an impact on correct solution code. Whether students first check "x1 < x2" or "x2 < x1" is irrelevant: a correct solution will work regardless of the parameter order. In spite of this, this order swap variant was *tied* for the second lowest average score from that group that semester (82% vs 79%, the lowest). For some reason we cannot yet discern, this swap appears to have had a meaningful result despite not requiring different solution code.

5.6 Implications for Practice: How isomorphic do you need your isomorphs?

The creation of isomorphs through surface feature permutation, ideally, wants to produce *actual* isomorphs. However, our results leave an open question: which permutations produce isomorphic questions? Our answer to that question is that all of the strategies produce reasonably isomorphic questions, given the impact on scores is not much different than the impact Zingaro and Porter [43] found when using isomorphs to investigate in-class learning with an 87% correctness rate on isomorphic questions for students who learned the material in class. 87% correctness is not that much different than an 11 percentage point lower score (or, put another way, 89% of the points out of 100%).

Instructors and researchers who wish to quickly write directly equivalent questions should use the "less complex" permutations - that is, *PrototypeAdded and Removal, VarNameChange*, and *FunctionNameChange*. Given the results from this and prior work, these are the strategies most likely to construct questions with isomorphic solutions and equivalent difficulty. Meanwhile, those interested in investigating transfer or producing slightly more novel but still relatively isomorphic questions should use the "complex" permutations - *DataTypeChange, ConstantChange,* and *PolarityReverse*. These changes have the most consequential impact on student performance and, thus, may be best for assessing if students have learned underlying patterns. We personally recommend against *OrderSwap*, if only to avoid subjecting students to unnatural orderings of arguments for questions which have more natural orderings.

As a final note, it may be best to produce formative materials that are slightly more difficult than planned summative material. In particular, our results looking at hidden and practice-first question difficulty suggest that students perform better on content when they complete more challenging questions on the same content first. This benefit may be attributable to theories such as Vygotsky's Zone of Proximal Development, where harder variants fulfill the need for "tasks that students cannot do on their own, but which they can do with assistance... [39]"

6 LIMITATIONS

There remain some limitations with this work. As noted in Section 5.3, exam construction during latter semesters in our analysis appears to have harmed the consistency of coefficients' impacts. Some of this construction was reversed in Fall 2022 and saw the coefficients trend back in similar directions as those semesters prior to Fall 2021. While the general existence of trends among the coefficients gives us confidence that there is a substantive difference between "complex" permutations and the other permutations, our answer to **RQ2** is hampered by variance between semesters. It appears that consistency between semesters is sensitive to permutations being applied consistently across a range of base difficulties. A rigorous accounting where base questions are selected with an explicit range of difficulty, permuted similarly, and results are analyzed would provide the strongest answer to this question.

Our results are not and cannot be strictly causal as they are presented. We merely have reasonably strong correlations showing that certain permutations *tend to* produce questions of higher or lower difficulty. A true, randomly controlled experiment where permutation types are tested one at a time on isomorphs would be necessary to establish causality. Additionally, future work could investigate more deeply the shared patterns between questions to judge whether these isomorphs are truly isomorphic.

Finally, the data we use is limited in that it comes from multiple semesters of the same course, an introductory Python course. An exciting direction for future work would be to explore the applicability of our permutation strategy to other class contexts and other programming languages. It is likely the impacts of the permutation strategy will differ based on the affordances of the programming language used. Further, different student populations may be more prepared to transfer solution approaches between questions, thereby broadening the number of questions that can serve as suitable "isomorphs."

7 CONCLUSION

We present the results of applying a question permutation strategy on multiple semesters worth of questions and exams in an introductory Python course. We find that, typically, the most significant and impactful predictor of a question having a lower score is not being repeated from homework assignments. Additionally, the more "complex" permutations tend to have both a larger magnitude impact on difficulty and tend to make the questions harder. This latter point seems to be because we naturally tend to write the most straightforward, and therefore among the easiest, version of the question first.

We identify multiple avenues of future work. First, it should be explored to what degree the observed question difficulties can be explained by some measure of the complexity of the solution code. Second, a proper randomized control experiment can be run to investigate casual links between single permutations and isomorph difficulty. Lastly, we welcome future researchers to apply the permutation strategy in their own studies and classrooms such that greater generalizability of these trends and findings may be established.

REFERENCES

[1] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102. https://doi.org/10.1080/08993400500150747

[2] Betsy Bizot and Stu Zweben. 2019. *Generation CS, Three Years Later.* Technical Report. Computing Research Association. https://cra.org/generation-cs-three-years-later/

[3] Paul D. Bliese, David Chan, and Robert E. Ployhart. 2007. Multilevel Methods: Future Directions in Measurement, Longitudinal Analyses, and Nonnormal Outcomes. *Organizational Research Methods* 10, 4 (2007), 551–563. https:

//doi.org/10.1177/1094428107301102

[4] Dennis Bouvier, Ellie Lovellette, John Matta, Bedour Alshaigy, Brett A. Becker, Michelle Craig, Jana Jackova, Robert McCartney, Kate Sanders, and Mark Zarb. 2016. Novice Programmers and the Problem Description Effect. In *Proceedings of the 2016 ITiCSE Working Group Reports (ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 103–118. https://doi.org/10.1145/3024906.3024912

[5] Liia Butler, Geoffrey Challen, and Tao Xie. 2020. Data-Driven Investigation into Variants of Code Writing Questions. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*. 1–10. https://doi.org/10.1109/CSEET49119.2020.9206195 ISSN: 2377-570X.

[6] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: The Mixed News on Diversity and the Enrollment Surge. *ACM Inroads* 8, 3 (July 2017), 36–42. https://doi.org/10.1145/3103175

[7] Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. 1981. Categorization and Representation of Physics Problems by Experts and Novices*. *Cognitive Science* 5, 2 (1981), 121–152. https://doi.org/10.1207/s15516709cog0502_2

[8] Michael J. Clancy and Marcia C. Linn. 1999. Patterns and Pedagogy. In *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (New Orleans, Louisiana, USA) *(SIGCSE '99)*. ACM, New York, NY, USA, 37–42. https://doi.org/10.1145/299649.299673

[9] Computing Research Association. 2017. *Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006*. Technical Report. https://cra.org/data/generation-cs/

[10] Michelle Craig, Jacqueline Smith, and Andrew Petersen. 2017. Familiar contexts and the difficulty of programming problems. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, 123–127. https://doi.org/10.1145/3141880.3141898

[11] Raquel M Crespo, Jad Najjar, Michael Derntl, Derick Leony, Susanne Neumann, Petra Oberhuemer, Michael Totschnig, Bernd Simon, Israel Gutierrez, and Carlos Delgado Kloos. 2010. Aligning assessment with learning outcomes in outcome-based education. In *IEEE EDUCON 2010 Conference*. IEEE, 1239–1246.

[12] Janet E Davidson and Robert J Sternberg. 2003. *The psychology of problem solving*. Cambridge university press.

[13] Jens Dolin, Paul Black, Wynne Harlen, and Andrée Tiberghien. 2018. Exploring relations between formative and summative assessment. In *Transforming assessment*. Springer, 53–80.

[14] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 328. https://doi.org/10.1145/1384271.1384371

[15] James Finnie-Ansley, Paul Denny, and Andrew Luxton-Reilly. 2021. A Semblance of Similarity: Student Categorisation of Simple Algorithmic Problem Statements. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021)*. Association for Computing Machinery, New York, NY, USA, 198–212. https://doi.org/10.1145/3446871.3469745

[16] Daniel T. Fokum, Daniel N. Coore, Eyton Ferguson, Gunjan Mansingh, and Carl Beckford. 2019. Student Performance in Computing Courses in the Face of Growing Enrollments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/3287324.3287354

[17] Max Fowler and Craig Zilles. 2021. Superficial Code-guise: Investigating the Impact of Surface Feature Changes on Students' Programming Question Scores. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 3–9. https://doi.org/10.1145/3408877.3432413

[18] John R Hayes. 1977. Psychological differences among problem isomorphs. *Cognitive theory* 2 (1977), 21–41.

[19] Eder Hernandez, Esmeralda Campos, Pablo Barniol, and Genaro Zavala. 2019. The effect of similar surface features on students' understanding of the interaction of charges with electric and magnetic fields. In *Physics Education Research Conference, PERC 2019 (Physics Education Research Conference Proceedings)*, Ying Cao, Steven Wolf, and Michael Bennett (Eds.). American Association of Physics Teachers, United States, 220–225. https://doi.org/10.1119/perc.2019.pr.Hernandez

[20] Matthew Inglis and Lara Alcock. 2012. Expert and Novice Approaches to Reading Mathematical Proofs. *Journal for Research in Mathematics Education* 43, 4 (2012), 358–390. https://doi.org/10.5951/jresematheduc.43.4.0358

[21] Eric Kjolsing and Lelli Van Den Einde. 2016. Peer Instruction: Using Isomorphic Questions to Document Learning Gains in a Small Statics Class. *Journal of Professional Issues in Engineering Education and Practice* 142, 4 (2016).

[22] Juho Leinonen, Paul Denny, and Jacqueline Whalley. 2021. Exploring the Effects of Contextualized Problem Descriptions on Problem Solving. In *Australasian Computing Education Conference (ACE '21)*. Association for Computing Machinery, New York, NY, USA, 30–39. https://doi.org/10.1145/3441636.3442302

[23] Filip Lievens and Paul R. Sackett. 2007. Situational judgment tests in high-stakes settings: Issues and strategies with generating alternate forms. *Journal of Applied Psychology* 92 (2007), 1043–1055. https://doi.org/10.1037/0021-9010.92.4.1043

[24] Russell Millar and Sathiamoorthy Manoharan. 2021. Repeat individualized assessment using isomorphic questions: a novel approach to increase peer discussion and learning. *International Journal of Educational Technology in Higher Education* 18, 1 (2021), 1–15.

[25] Nona Muldoon and Chrisann Lee. 2007. Formative and summative assessment and the notion of constructive alignment. *Enhancing teaching and learning through assessment* (2007), 98–108.

[26] Ross H. Nehm and Minsu Ha. 2011. Item feature effects in evolution assessment. *Journal of Research in Science Teaching* 48, 3 (2011), 237–256. https://doi.org/10.1002/tea.20400

[27] Ross H Nehm and Judith Ridgway. 2011. What do experts and novices "see" in evolutionary problems? *Evolution: Education and Outreach* 4, 4 (2011), 666–679.

[28] Allen Newell and Herbert A. Simon. 1972. *Human problem solving*. Prentice-Hall, Oxford, England. xiv, 920–xiv, 920 pages.

[29] Peterson K Ozili. 2022. The acceptable R-square in empirical modelling for social science research. *Available at SSRN 4128165* (2022).

[30] Miranda C. Parker, Leiny Garcia, Yvonne S. Kao, Diana Franklin, Susan Krause, and Mark Warschauer. 2022. A Pair of ACES: An Analysis of Isomorphic Questions on an Elementary Computing Assessment. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. Association for Computing Machinery, New York, NY, USA, 2–14. https://doi.org/10.1145/3501385.3543979

[31] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 93–101. https://doi.org/10.1145/2960310.2960316

[32] Miranda C. Parker, Yvonne S. Kao, Dana Saito-Stehberger, Diana Franklin, Susan Krause, Debra Richardson, and Mark Warschauer. 2021. Development and Preliminary Validation of the Assessment of Computing for Elementary Students (ACES). In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 10–16. https://doi.org/10.1145/3408877.3432376

[33] Jolynn Pek and David B. Flora. 2018. Reporting effect sizes in original psychological research: A discussion and tutorial. *Psychological Methods* 23 (2018), 208–225. https://doi.org/10.1037/met0000126

[34] Nancy Pennington. 1987. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology* 19, 3 (1987), 295–341.

[35] Leo Porter, Cynthia Bailey Lee, Beth Simon, and Daniel Zingaro. 2011. Peer Instruction: Do Students Really Learn from Peer Discussion in Computing?. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 45–52. https://doi.org/10.1145/2016911.2016923

[36] Frederick Reif. 2008. *Applying Cognitive Science to Education: Thinking and Learning in Scientific and Other Complex Domains*. MIT Press.

[37] Skipper Seabold and Josef Perktold. 2010. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

[38] Nicole A Shea, Ravit Golan Duncan, and Celeste Stephenson. 2015. A tri-part model for genetics literacy: Exploring undergraduate student reasoning about authentic genetics dilemmas. *Research in Science Education* 45, 4 (2015), 485–507.

[39] Rob Wass and Clinton Golding. 2014. Sharpening a tool for teaching: the zone of proximal development. *Teaching in Higher Education* 19, 6 (2014), 671–684. https://doi.org/10.1080/13562517.2014.901958

[40] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. 26.1238.1–26.1238.14. https://peer.asee.org/prairielearn-mastery-based-online-problem-solving-with-adaptive-scoring-and-recommendations-driven-by-machine-learning

[41] Michele Weston, Kevin C. Haudek, Luanna Prevost, Mark Urban-Lurain, and John Merrill. 2015. Examining the Impact of Question Surface Features on Students' Answers to Constructed-Response Questions on Photosynthesis. *CBE Life Sciences Education* 14, 2 (June 2015). https://doi.org/10.1187/cbe.14-07-0110

[42] Aaron S. Yarlas and Vladimir M. Sloutsky. 2000. Problem Representation in Experts and Novices: Part 1. Differences in the Content Of Representation. In *In.* Erlbaum, 475–480.

[43] Daniel Zingaro and Leo Porter. 2015. Tracking Student Learning from Class to Exam Using Isomorphic Questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 356–361. https://doi.org/10.1145/2676723.2677239