Distractors Make You Pay Attention: Investigating the Learning Outcomes of Including Distractor Blocks in Parsons Problems

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

Seth Poulsen
Utah State University
Logan, UT, USA
seth.poulsen@usu.edu

Chinedu Emeka
University of Illinois
Urbana, IL, USA
cemeka2@illinois.edu

Zihan Wu
University of Michigan
Ann Arbor, MI, USA
ziwu@umich.edu

Carl Haynes-Magyar
Carnegie Mellon University
Pittsburgh, PA, USA
chaynesm@cs.cmu.edu

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

ABSTRACT

Background: In CS1 courses, Parsons problems are a popular activity in which students are given blocks of code and asked to rearrange them into the correct order. Parsons problems often include incorrect blocks of code referred to as distractor blocks. Despite their widespread use, there have been few investigations into how distractor blocks impact student learning.
Objectives: Our goals are to understand (1) the impact that including distractor blocks in Parsons problems has on learning and (2) the causality underlying that learning, if any.
Methods: In this paper, we present the results of an explanatory sequential mixed methods study investigating the impact of distractor blocks on student learning. For the initial, quantitative stage, we use a randomized control trial to quantify the learning outcomes from practice with Parsons problems that include distractor blocks, as measured via post-test taken immediately after the practice activity and a retention test taken a week later. This study is followed by think-aloud interviews with 10 students practicing using a mix of Parsons problems that do and do not contain distractors to understand differences in how students approach those problems.
Findings: Our findings show that students who practiced using Parsons problems that contained distractors performed 11 percentage points better on the immediate post-test (statistically significant) and 10 percentage points better on the retention test (approaching significance). The results of the think-aloud interviews indicate that grouping distractors with blocks of correct code causes students to more closely attend to the details of the code within those blocks.
Implications: The results of this study indicate that distractors are essential when Parsons problems are used in a formative context. When they are not included, students may be able to successfully place blocks of code without attending to details of the code. This in turn limits their ability to learn new concepts or reinforce existing knowledge from those code blocks.

ICER '24 Vol. 1, August 13–15, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0475-8/24/08
https://doi.org/10.1145/3632620.3671114

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

Parsons problems, CS1, learning, distractors, formative

ACM Reference Format:
David H. Smith IV, Seth Poulsen, Chinedu Emeka, Zihan Wu, Carl Haynes-Magyar, and Craig Zilles. 2024. Distractors Make You Pay Attention: Investigating the Learning Outcomes of Including Distractor Blocks in Parsons Problems. In *ACM Conference on International Computing Education Research V.1 (ICER '24 Vol. 1), August 13–15, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3632620.3671114

1 INTRODUCTION

Programming is a complex task that requires proficiency in a variety of skills. This includes: 1) the mastery of a language's basic syntax and semantics [2, 53], 2) the ability to read, write, and comprehend code [27, 52, 87], 3) higher-level skills such as writing high-quality code, and 4) debugging [8, 14, 58]. Given the diversity and complexity of these skills, many educators have developed tools and pedagogical approaches to ease their learning curve.

Among these, Parsons problems were first introduced by Parsons and Haden [59] as "Parsons Programming Puzzles." Their intention was to scaffold the process of learning to write code by providing students with a set of blocks of code that they must rearrange to solve a given task. Parsons problems have since garnered significant interest from educators and researchers alike due to their educational affordances [16, 20]. Parsons problems have been shown to improve learning efficiency [21, 35], reduce cognitive load [3, 25, 56], and improve student engagement with study materials [22, 55, 59].

Since their introduction, Parsons problems have included incorrect blocks of code referred to as "distractors" or "distractor blocks". The purpose of including these blocks is to highlight and correct common errors in formative settings [59] and increase problem difficulty in summative contexts [15, 72, 74, 86]. However, the body of work investigating the role and impact of distractors on learning is small and somewhat inconsistent in its findings. Harms et al. [34] found that students practicing with Parsons problems that included distractors took longer in completing practice activities but did not show improved performance compared to students practicing with problems that did not include distractors. A recent ITiCSE working group found that students who practiced with Parsons problems that included distractors performed better on code fixing questions and made fewer errors when completing code writing questions [24]. Similarly, Poulsen et al. [60] studied a related exercise "Proof Blocks" where students arrange blocks to form an inductive proof and found that, although students practicing with distractors performed better on a post-test, the difference was not statistically significant.

Given the mixed results of prior work, we perform a conceptual replication [33] of the studies conducted by Ericson et al. [24] and Poulsen et al. [60]. In doing so, we use an explanatory mixed-methods approach [12]. This begins with a randomized control trial aimed at comparing the learning gains of students practicing under one of two conditions: Parsons problems containing distractors (intervention) to those that do not (control). The research questions this approach addresses are as follows:

RQ1: What are the learning differences between students practicing under each condition in an immediate post test?

RQ2: What are the learning differences between students practicing under each condition in a retention test taken a week later?

To offer deeper insight into the results of **RQ1** and **RQ2**, we designed a subsequent qualitative study using think-aloud methods where we observed students practicing a new topic using a mix of Parsons problems that did and did not contain distractor blocks. Our goal is to understand differences in how students interact with Parsons problems that contain distractor blocks versus those that did not. This was done to address the final two research questions:

RQ3: How do students' problem solving procedures differ when solving Parsons problems that include distractors versus those that do not?

RQ4: What are students' perceptions of practicing with Parsons problems that include distractor blocks versus those that do not?

Using both lenses, we hope to provide a fuller picture of the impact of distractors on learning in addition to shedding light on the underlying reasons for those results.

2 RELATED WORK

We organize our related work into three areas. First, in Section 2.1 we cover the origins of Parsons problems, developments made since their introduction, and their adoption into classrooms. Next, in Section 2.2, we cover literature relating to the design and utility of distractors drawing on investigations into the effectiveness of multiple-choice questions and Parsons problems. We conclude in Section 2.3 by covering learning theory relevant to Parsons problems and the use of distractors.

2.1 Parsons Problems - Designs and Affordances

Parsons problems began as a mechanism for providing students with a drilling exercise that improved engagement and could provide immediate feedback [59]. The design goals of this question type were to:

(1) **Maximize engagement** through a more puzzle like interface which Parsons and Haden theorized would be more fun than traditional CS exercises.
(2) **Constrain logic** by only allowing students to solve the problem with the provided code.
(3) **Permit common errors** through the inclusion of "distractor blocks" containing errors.
(4) **Model good code** by letting them interactively construct an instructor's solution.
(5) **Provide immediate feedback** indicating which blocks are not in the correct order.

Parsons problems have since become a popular tool in teaching introductory computer science, gaining a large and steadily increasing body of work investigating and informing their use [16, 20]. A recent 2022 ITiCSE working group by Ericson et al. [20] found that these investigations have included: 1) identifying the learning gains Parsons problems provide, 2) improving students' engagement and motivation, 3) scaffolding students' ability to learn how to write code, and 4) investigating students' perceptions of the problem type. Ericson et al. [20] also identified a variety of gaps in the existing literature, among them the need to understand the role and impact of distractor blocks in Parsons problems.

Many variants of Parsons problems have emerged to improve the original problem type and to address other learning goals. Ihantola and Karavirta [39] introduced 2D-Parsons problems, which require students not only to order the blocks vertically but also place blocks at the correct levels of indentation. Adaptive Parsons problems extend the scaffolding ability of the original problem by merging correct blocks and removing distractor blocks interactively through the use of a hint button [18, 19, 21]. Students are permitted by the system to use the hint feature after encountering several errors in an attempt to match the problem's difficulty to the student's ability. Faded Parsons problems present students with fill-in-the-blank code blocks that they must both complete and place in the correct order [28, 80, 81]. Micro Parsons problems, sometimes called "horizontal Parsons problems," present students with segments of a single line of code and require them to arrange the blocks horizontally [84, 85]. The problem format at the core of Parsons problems have also been extended to the domain of math education with "Proof Blocks," where students arrange elements of a proof rather than code [60–64].

Among the reasons for the popularity of Parsons problems, and their variants, is their effectiveness with respect to students' learning. Ericson et al. [23] found that students practicing two-dimensional, non-adaptive Parsons problems completed their practice problems more quickly than those students practicing code-writing problems. They also found that the two groups performed similarly on a post-test. Poulsen et al. found similar results in the context of Proof Blocks, though measuring students proof writing knowledge proved more difficult [60–62]. Ericson et al. [21] completed a follow-up study that included a comparison between adaptive and non-adaptive Parsons problems. They found that the groups practicing with each implementation did not differ in terms of learning gains or problem-solving efficiency.

2.2 Distractors in Parsons Problems versus Multiple-Choice Questions

The term "distractor," as it pertains to its use in Parsons problems, originates in the domain of multiple-choice questions. There, their core role is to obfuscate the correct response by presenting it alongside plausible but incorrect alternatives (i.e., the distractors). In a summative context, this is important as the quality of distractors is one of the main contributors to question quality [29]. If distractors are poor, then the correct response will be immediately obvious which, in turn, limits the ability of that item to provide information on the knowledge of that respondent [77]. By extension, this can also limit the learning potential of the item. If students are not forced to engage with plausible alternatives, this limits the retrieval and encoding of information related to the problem to which they are responding [5, 51].

Compared to the large body of work that exists with respect to multiple-choice questions to inform the creation of distractors [9, 31], the number of distractors to include [10, 65, 78], and the evaluation of distractors [30, 77], there is a relative dearth informing their use in Parsons problems. In summative contexts, Denny et al. [15] showed that the use of distractor blocks that are jumbled in amongst the response options (*jumbled distractors*) appear to produce extraneous cognitive load compared to those that are paired with their correct alternatives (*visually-paired/grouped*). A series of papers by Smith IV et al. showed that including distractors did not lead to improved item quality and greatly increased the amount of time students spent on the problem [72–75]. In formative contexts, Harms et al. [34] showed that students who practiced using Parsons problems that included distractor blocks took more time to solve the problems and did not lead to learning gains compared to practice that did not include distractor blocks. Conversely, Ericson et al. [24] showed that students who practiced a new concept with Parsons problems performed better on problems where they were required to fix a segment of code that contained errors. They also showed that students made fewer syntax errors when completing code writing problems. Haynes-Magyar [36] found that learners with attention-deficit/hyperactivity disorder (ADHD) had positive perceptions of *visually-paired* distractors, feeling that the presence of the groups of blocks helped focus their attention when solving said problems in an informal learning context. Poulsen et al. [60] showed a small but not statistically significant improvement in learning gains for students who used Proof Blocks problems with distractor blocks over those who used problems without distractor blocks.

2.3 Theory Informing the use of Parsons Problems and Distractor Blocks

Research on Parsons problems has been tightly coupled with theories of learning and memory. In this section we cover a variety of those theories including: 1) Cognitive Load theory (Section 2.3.1), 2) Selective Attention (Section 2.3.2), 3) Desirable Difficulties (Section 2.3.3), the relationship between learning from errors and feedback (Section 2.3.4), and 4) Vygotsky's Zone of Proximal Development (Section 2.3.5). In particular, we relate the implications of these theories and their findings to the inclusion of distractor blocks in Parsons problems as well as the design of Parsons problems more generally.

2.3.1 Cognitive Load Theory. Cognitive load theory (CLT) was first introduced by Sweller [76] with the following goal:

> "... to suggest that contrary to current practice and many cognitive theories, some forms of problem solving interfere with learning."

To help distinguish between the "forms of problem solving" and its relevance to learning, cognitive load is often categorized as being: 1) *intrinsic* which is related to the task itself, 2) *extraneous* which is related to processing task-irrelevant information, and 3) *germane* which is cognitive load dedicated to the act of learning itself [47]. To mitigate extraneous and intrinsic cognitive load, many suggest the use of guided instruction (e.g., step-by-step walkthroughs) to minimize the high cognitive load associated with the alternative of learning through self-guided inquiry and discovery [45, 46].

CLT has been applied to understand the impact of including distractor blocks in Parsons problems. Denny et al. [15] compared students solving problems which included *jumbled distractors* to problems which included *visually-paired* distractors. They found that jumbled distractors lead to extranous cognitive load since the task of sorting through the blocks itself was an extraneous measure which had no apparent learning benefits. Similarly, Harms et al. [34] found that the inclusion of a *partial suboptimal path* (that is, distractors that lead students down a suboptimal solution path) increased cognitive load relative to problems that did not contain distractors, with no benefit to student learning. These studies identified a method of including distractors and a type of distractor that may be at odds with and lead to less efficient learning, respectively.

2.3.2 Selective Attention. The issue of "selective attention," raised by Sweller [76], is another issue related to CLT. It refers to the situation where problem solving approaches and schema acquisition are entirely disjoint tasks, in which students can successfully apply a problem solving procedure without engaging in schema acquisition. Through the use of eyetracking, Rehder and Hoffman [66] investigated how learners focus visual attention when learning to categorize entities. Their results show that individuals early on in the category learning process pay attention to all attributes of the entity. As those learners progress, they are able to identify dimensions on which to focus their attention, in order to more efficiently categorize.

Though selective attention has yet to be studied in the context of Parsons problems, it does raise important questions with respect to designing problems that accomplish their given learning objectives. If students can solve Parsons problems without attending to the elements relevant to their learning objectives, that, in turn, limits their ability to meet those objectives. More research is needed to understand the processes by which students solve Parsons problems and how this evolves as students acquire experience both in programming and solving Parsons problems. In particular, this motivates our work to understand how the inclusion of distractor blocks in Parsons problems impacts students' problem solving processes and attention when solving problems that include them.

2.3.3 Desirable Difficulties. The term "desirable difficulty" was first introduced by Bjork [6] in the context of how difficulties that align with a given learning objective can enhance learning. In this context, *desirability* is defined by a given *difficulty's* contribution to a student's encoding and recall process, as well as its contribution to long-term learning [7]. The desirability of a given difficulty is not fixed in any way and can vary depending on a student's ability to understand and respond to the difficulty [4]. Related to CLT, the desirability of a difficulty is therefore defined by how well it is aligned with a given learning objective and the degree to which it contributes to the learning of that objective.

In Parsons problems, distractor blocks are often cited as a feature used to present students with desirable difficulties [21, 23]. In this context, the difficulties associated with Parsons problems (e.g., indentation, placing blocks in the correct order) are considered desirable, as they have been shown to contribute to students ability to write code independent of Parsons problems. Distractors add an additional dimension of difficulty that prior work has indicated may facilitate students' ability to recognize and fix errors [24].

2.3.4 Learning from Errors and Feedback. The term "error" often has a strong negative connotation, particularly in the context of learning. This may lead some to be concerned that allowing students, in particular novices, to make errors may lead to confusion, frustration, or potentially reinforce misconceptions [54, 67, 88]. Despite these concerns and intuitions, many studies suggest that the presence of errors can aid learning [48, 49, 68]. In particular, permitting and correcting errors related to misconceptions can help individuals remediate those misconceptions through a process known as cognitive conflict [41, 57]. Similarly, allowing students to engage in a process known as *productive failure*, during which students first make (typically) unsuccessful attempts to complete a new task independently in a "problem solving" first teaching approach, has been shown to lead to improved learning during subsequent instruction they receive [42–44]. Central to all of these ideas is the key point that allowing for students to make errors both allows for those errors to be corrected and makes any subsequent learning potentially more potent by contextualizing an error which a student may have previously encountered.

Central to the learning benefits of encountering errors is the ability for students to see how to correct those errors, particularly through feedback. This raises the obvious questions of when and how to deliver that feedback. A meta-analysis by Kulik and Kulik [50] found that the results of lab studies on the topic often suggest delayed feedback, whereas classroom studies often suggest immediate feedback. They suggest that the differential findings about learning is more likely due to the mechanisms by which students are incentivized to engage with the feedback rather than when the feedback happens to take place. They note that in classroom contexts, students are likely more incentivized to engage with immediate feedback as it can help them get closer to a correct solution.

With respect to how the feedback should be given, there is a general consensus that dichotomized (right or wrong) feedback is ineffective and that students should, at minimum, receive a correct answer that they can compare their unsuccessful attempt against [54]. A meta-analysis by Bangert-Drowns et al. [1] found that students who received some higher level form of feedback (e.g., correct answer, repeat until correct, explanation) had higher effect sizes with respect to learning than studies that included only right/wrong feedback. Finn and Metcalfe [26] found that feedback that contained hints lead to learning gains on both a post and retention test. Similarly, Hao et al. [32] found that students who were given dicotomized feedback were less efficient in arriving at a correct solution and showed signs of engaging in trial-and-error problem solving. These studies highlight the importance of feedback design in learning activities that allow students to commit errors and the importance of requiring them to independently remediate those errors.

The design principles of Parsons problems closely align with the best practices that can be derived from work on errors and feedback. Parsons problems often use immediate feedback which, as suggested by Metcalfe [54], may be preferred in instructional settings. Similarly, they permit and provide feedback on errors related to incorrect ordering and indentation. By adding distractors, the problem includes an additional dimension by which an error might occur. However, to date, there has been limited research investigating the impact of these errors and the feedback associated with them on learning.

2.3.5 Zone of Proximal Development and Scaffolding. What has come to be known as the *Zone of Proximal Development* (ZPD) was first introduced by Vygotsky and Cole [79] under the following definition,

> "It is the distance between the **actual development level** as determined by independent problem solving and the **level of potential development** as determined through problem solving under adult guidance or in collaboration with more capable peers." (p. 86)

This zone is defined as coming beyond the *Zone of Actual Development* (ZAD), where an individual has already developed a given skill or set of skills and therefore does not require assistance in successfully exercising that skill set. The ZPD model of learning sits at the heart of *scaffolding*. Though the use of the scaffolding metaphor has a long and complex history [71] a commonly cited early definition was provided by Wood et al. [83].

> "...that enables a child or novice to solve a task or achieve a goal that would be beyond his unassisted efforts" (p. 90)

Contrasting this definition with the one for ZPD, its similarity to Vygotskian concepts of learning are clear. Scaffolding seeks to support students who are capable of completing a task with assistance (i.e., in their ZPD) with the goal of helping them acquire the skills needed to complete the task independently (i.e., in their ZAD) [70].

Although ZPD and scaffolding were not explicitly mentioned by Parsons and Haden [59] when they introduced the Parsons problems, one of the stated goals of their design was to "constrain the logic" of a program's solution as a method of easing the difficulty of students arriving at a correct solution. ZPD has been used more explicitly in the design of *adaptive Parsons problems*, which aim to match the problem's difficulty with the students' ability by removing distractors, correcting indentation, or merging blocks [17, 21, 38].

The use and utility of distractors in Parsons problems can be viewed through the lenses of ZPD and scaffolding. Parsons and Haden [59] stated that the intention of including distractors is to "permit common errors." Students who select distractors are required to engage in a form of debugging where they would ideally compare the block they select to the alternative(s) and identify 1) the differences and 2) why their initial selection was incorrect. Similar to "constrain the logic," this design decision allows students to make mistakes and engage in debugging processes in a scaffolded environment that reduces the chances of them getting stuck.

3 METHODOLOGY

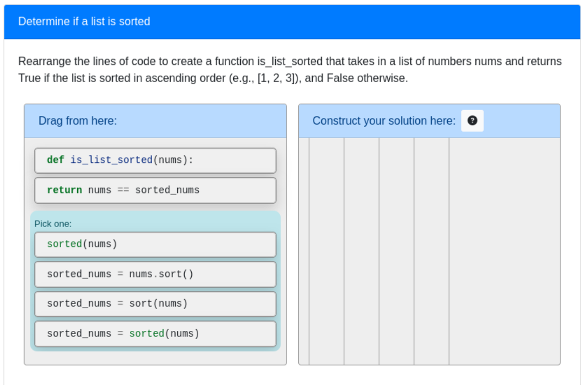
This study employs an explanatory sequential mixed-methods design to investigate both the magnitude and the source of impact of practice with distractors on students' learning [12]. We first ran a randomized control trial to measure the impact of distractors on student learning, and then followed that up with interviews to gain a greater understanding of why the distractors had the effect that they did. Section 3.1 provides details on the participants, course from which they were recruited, and how that relates to the design of the activities students completed. Section 3.2 describes the design of the Parsons problems used in both the quantitative and qualitative portions of this study. Section 3.3 describes the randomized control trial used to compare learning gains between students who practiced a new concept using Parsons problems which contained distractors and students practicing without distractors. Section 3.4 describes the qualitative study which used think-aloud interviews to gain insight into the underlying attributes of including distractors in Parsons problems that may have contributed to the observed learning gains. The results of the qualitative studies are also used to shed light on students' perceptions on the presence of distractors and further inform their design and use.

3.1 Participants and Activity Designs

Participation in the study was solicited in a large introductory programming course, taught in Python, at the University of Illinois Urbana-Champaign. The Parsons problems used in both the quantitative and qualitative studies were designed with the goal of introducing students to a new concept and providing them with practice integrating that concept into problems that align with the concepts they had covered thus far in the course. In the case of the *quantitative* randomized control trial, we introduce the use of sorting functions in Python. For the *qualitative* think-aloud interviews, we introduce the use of iterating through both the key and values within a dictionary through the use of the `dict.items()` function. Each of these studies were approved by the ethics reviews board at the institution where the studies occured.

3.2 Selection and Design of Distractor Blocks

In the case of problems that contained distractor blocks, only those blocks containing a call to topic being introduced were associated with distractors. The intention of this was to reduce cognitive load by only including distractors that were aligned with the learning objective. To further reduce extraneous cognitive load, we use visually grouped distractors as suggested by Denny et al. [15] (Figure 1). To select our distractors, we used the static analysis suggested by Smith IV and Zilles [75] to analyze common errors when using sorting functions on code writing activities in the course from which students were sampled. We select the top errors for each function (Table 1) and design distractors based on them.



Determine if a list is sorted

Rearrange the lines of code to create a function is_list_sorted that takes in a list of numbers nums and returns True if the list is sorted in ascending order (e.g., [1, 2, 3]), and False otherwise.

Drag from here:

```
def is_list_sorted(nums):
```

```
return nums == sorted_nums
```

Pick one:

```
sorted(nums)
```

```
sorted_nums = nums.sort()
```

```
sorted_nums = sort(nums)
```

```
sorted_nums = sorted(nums)
```

Construct your solution here:

Figure 1: The Parsons problems interface used in this study. This is an example of a questions which includes visually grouped distractors.

Table 1: Common errors in code writing questions for sorting functions.

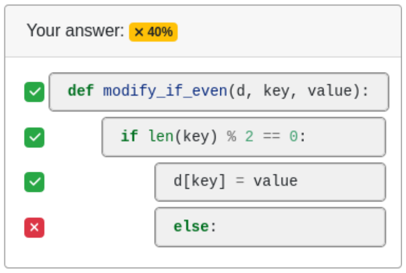
Correct Statement	Errors
list.sort()	sort(list)
	sorted(list)
	list.sorted()
	list.sort
new_list = sorted(list)	new_list = list.sort()
	new_list = sort(list)
	new_list = list.sorted()
	list.sorted()
for k, v in dict.items():	for k, v in dict.items:
	for k, v in dict:
	for v, k in dict.items():

For general feedback, we use a version of the line-based feedback used by Helminen et al. [37]. However, whereas their system highlights all blocks that are incorrectly placed or lack indentation, our system uses a top-down approach whereby blocks are highlighted as correct until an incorrectly placed block is encountered in the solution. Once such a block is encountered, it is marked with a red cross indicating that block is incorrect (Figure 2a). All subsequent blocks do not receive feedback.

As noted by Parsons and Haden [59], the original intention of including distractors in Parsons problems was to further scaffold the debugging experience for students. In reference to the benefits of a drag-and-drop interface, Parsons and Haden [59] notes,

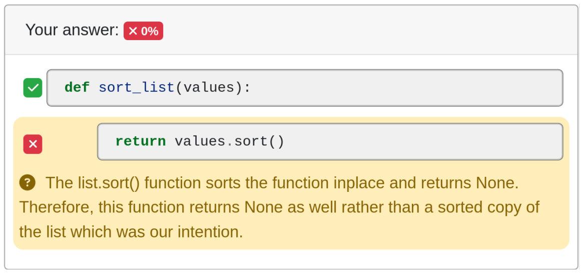
> *"Thus we can narrow a very large space of possible errors to only those that can be used to illustrate a particular point"*

To further this proposed benefit, we designed our distractors to include feedback which is provided to students in the event a distractor is selected (an example is shown in Figure 2b). The intention of this is to draw students attention to the fact they have selected a distractor and to elucidate the nature of the error in question.


Your answer: ✕ 40%

```
✓ def modify_if_even(d, key, value):
✓     if len(key) % 2 == 0:
✓         d[key] = value
✗         else:
```

(a) The general feedback provided to students when their first error does not include a distractor.


Your answer: ✕ 0%

```
✓ def sort_list(values):
✗     return values.sort()
```

The list.sort() function sorts the function inplace and returns None. Therefore, this function returns None as well rather than a sorted copy of the list which was our intention.

(b) The "distractor feedback" presented to students when the first errors in their solution contains a distractor.

Figure 2: The two modes of feedback presented to students in the event of an incorrect submission. Responses are graded top to bottom with the first incorrect block being marked with a red cross. If this block is meant to be included in the correct solution no other feedback is given (Figure 2a). If it is a distractor, students receive a description of why that block is incorrect (Figure 2b).

3.3 Quantitative Study Design

We perform a randomized control trial to evaluate the effectiveness of distractors in teaching students how to fix and write code. The topic selected for this study was one participants had yet to cover in class, in this case using the built-in sorting functions in Python. The study was constructed in two parts each of which was conducted in a proctored testing environment. During the part one, students were asked to complete the following sections in an hour and fifty minute period:

- **Introduction and Pre-survey:** A short section introducing students to the structure of the study and asking them to complete a short survey on their familiarity with the topic.
- **Lesson and Practice:** A short reading on the topic followed by a series of 14 Parsons problems to solve.
- **Post-test:** A test consisting of a variety of questions aimed at assessing students ability to write code, fix buggy code, as well as identify and explain errors in existing code on the topic just practiced.

As for part two, a week later, students were asked to complete a retention test consisting of the same question types as the post-test. Participation in either part of the study was voluntary and students were compensated for their time in the form of 0.5% extra credit added to their final grade for each part of the study they completed. The structure of each of these sections is described in more detail below.

3.3.1 *Introduction and Pre-survey.*

The study begins with a short introduction to the study and a pre-survey. The introduction provided students with an overview of the study's components, the Parsons problems interface, and an informed consent document. From there, students were given a pre-survey to gauge their familiarity with the topic they would be practicing and ultimately be tested on, sorting lists in Python. Given students have not previously been exposed to the topic, a pre-test would likely suffer from a significant floor effect. Therefore, we use a study design that does not use a pre-test and instead use a series of questions on familiarity with the topic to gauge prior knowledge on the topic. This approach is similar to that used by Ericson et al. [24] and Poulsen et al. [60] who conducted similar studies. This survey showed students a code snippet that contained an example of a function covered in the lesson and asked them the following series of 5-point Likert scale questions. The following pair of questions were asked with respect to both the `list.sort()` and `sorted()` functions.

(1) How would you rate your experience, if any, with using the function in Python?
(2) How would you rate your experience, if any, with reading code that includes the function in Python?

3.3.2 *Lesson and Practice.*

Students in both conditions began with a short reading on the topic of using both the `list.sort()` and `sorted()` functions in Python. This lesson showed a variety of examples of each usage and focused on highlighting the difference between the in-place nature of `list.sort()` and the fact that `sorted()` returns a new list leaving the original list unaltered.

Following the reading, students are then randomly assigned to either the *no distractors* (control) or *distractors* (intervention) groups. Students in the *distractors* group were given distractors for blocks of code that contained either `list.sort()` or `sorted()`. Practice in both conditions consisted of a series of fourteen Parsons problems, seven of which used `list.sort()` and seven of which used `sorted()`. These questions were designed to guide students from simple use cases (solutions with 2 blocks of code) to their use in more complex functions (solutions with upwards of 7 blocks of code).

3.3.3 *Post-test and Retention Test.*

Students were given a post-test immediately following the practice section and a retention test a week later. Each of these tests consisted of the following four question types: explain error, statement, fix code, and write code (Table 2). The statement, fix, and write code questions were autograded using a series of unit tests. For each incorrect attempt, students were given the standard Python *unittest* output for each failed test case. Additionally, each incorrect attempt reduced students score by a percentage proportional to the number of attempts they had made with some intermediary partial credit given for passing some but not all test cases.

The explain error questions were graded by two graders who were blind to the condition of the student responding. The questions

Table 2: The question types used in the post test and retention test and their respective grading schemes.

Question Name	Description	Attempts	Incorrect Attempt Penalty
Explain Error	Students are given a static block of code and asked to: 1) identify if a bug exists and, if a bug exists, 2) explain the bug or how to correct it.	(manually graded)	NA
Fix Error	Students are given a block of buggy code and asked to correct it until it successfully passes a battery of unit tests.	5	-20%
Statement Question	Students are asked to construct an individual line of code.	4	-25%
Write Code	Students are given a problem statement and asked to write a function that solves the problem.	7	-14.25%

were graded in a binary manner with students receiving full credit if they identified if an error exists and the source of that error. The graders independently graded 70 responses and met to resolve any discrepancies. Following that meeting, the graders coded the remaining responses independently. The inter-rater reliability was calculated using Cohen's Kappa [11] and was found to be 0.93 indicating a very high level of agreement between the two graders. The graders then met to resolve any disagreements that occurred in the final set of responses to assign final scores.

3.4 Qualitative Study Design

To evaluate students' problem solving approaches and perceptions of questions that include distractors, we designed another learning activity similar to that used for the randomized control trial. Here, students were asked to complete a short reading about the `dict.items()` function. Specifically it's usage in for loops (i.e., `for k, v in dict.items()`). They then completed fourteen Parsons problems as a practice activity with seven containing distractors and the other seven not. These problems were interleaved so students alternated between the two throughout the practice activity. Students were asked to think-aloud while solving the Parsons problems and, upon completing the exercises, were asked the following reflection questions:

(1) What are your general thoughts in comparing the questions that did and did not include distractors?
(2) How would you compare the two types of Parsons problems in terms of difficulty?
(3) How would you compare the two types of Parsons problems in terms of their value for learning? For instance, which would you prefer to practice with when encountering a new concept and why?
(4) What are your thoughts on the feedback you encountered when you selected a distractor?

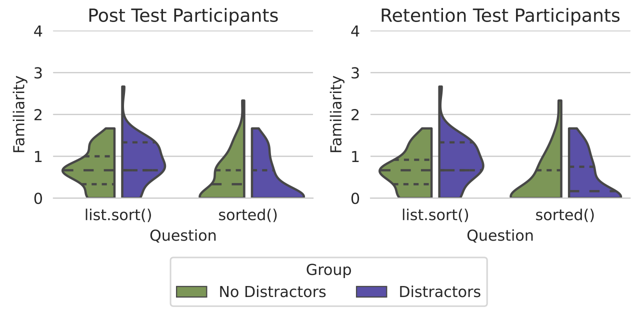
Once these interviews were completed, they were then automatically transcribed, manually reviewed, and corrected. Students were recruited from the same introductory programming course as the quantitative study and participation was entirely voluntary. The interviews lasted approximately one hour and students were compensated fifteen dollars in the form of an Amazon gift card.

4 QUANTITATIVE STUDY - ANALYSIS AND RESULTS

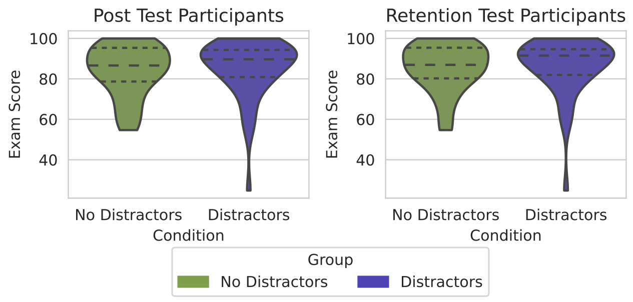
In total, 88 participants completed the first phase of the study with 46 from the *distractors* group and 42 from the *no distractors* group. A total of 74 students returned for the retention test the following week with 40 from the *distractors* group and 34 from the *no distractors* group. To ensure the equivalence of these two groups we compare them in terms of: 1) their self-reported familiarity with the sorting functions and 2) their scores on an exam taken recently in the course. To ensure that students had sufficient time to complete all parts of each study, we look at the time spent on each part by students in each practice condition.

Familiarity Comparison: Responses to the familiarity survey questions asked for each of the sorting functions, `list.sort()` and `sorted()`, had an extremely high internal reliability as calculated using Cronbach's alpha [13] (α = 0.94 and α = 0.88 respectively). As such, we average them to compute an average familiarity score for each of the functions. Overall, students reported being unfamiliar with both functions prior to the practice activity (Figure 3a). A series of Shapiro-Wilks tests indicate that these distributions deviate from a normal distribution (Table 4, Appendix A). As such we select the non-parametric Mann-Whitney U Test to compare the groups in terms of their self-reported familiarity with each of the functions. For students taking the post test, our results indicate there is no significant difference between the groups in terms of their familiarity with the `list.sort()` (U=944.5, z=21.49, p=0.85) or `sorted()` (U=1152.5, z=26.22, p=0.11) functions. Similarly, there is no significant difference between the groups for students that returned for the retention test for either the `list.sort()` (U=820.0, z=22.24, p=0.12) or `sorted()` (U=719.5, z=19.51, p=0.65) functions.

Comparison in Exam Score: In place of a pre-test, we use students scores on an recent exam given in the course from which the students were sampled. This exam was given the same week during which the study took place. This exam consists of: 1) multiple-choice and true-false (10 questions), 2) single line code writing (5 questions) 3) code writing (5 questions), 4) code reading (1 question), 5) code tracing (5 questions), and 6) Parsons problems (2 questions). The test is a 50-minute exam given on the PrairieLearn online, assessment platform [82] and in the same proctored computer based testing environment [89] in which this study took place. A series of



(a) Average familiarity with the sorting functions for each group on each of the tests



(b) Average course exam score for each group on each of the tests.

Figure 3: Comparing the similarity of the participants in each practice condition in terms of their prior familiarity with the topic and their more general programming skill.

Shapiro-Wilks tests indicate that these distributions deviate from a normal distribution (Table 5, Appendix A). Therefore, we select the non-parametric Mann-Whitney U Test to compare the groups in terms of their exam performance. The results indicate that there is no significant difference between the *distractors* and *no distractors* for participants who completed the post-test (U=1136.0, z=23.57, p=0.56). Similarly, for the retention test, there was no significant difference in exam performance between the *distractors* and *no distractors* groups (U=711.5, z=19.29, p=0.74). As a result, students in both groups, for both the post and retention tests, were similar in general programming ability at the time of the study.

Time Spent on Studies: The vast majority of students were successful in completing all of the practice activities in both the *distractor group* (M=96%, SD=11%) and the *no distractor* group (M=97%, SD=11%). Similarly, the average time taken on part one of the study (presurvey, reading, practice, and post test) was 47.38 minutes (SD=16.94) for the *distractor* condition and 46.61 minutes (SD=19.49) for the *no distractor* condition, with the longest duration being 96.55 minutes.This indicates that students had ample time to complete everything within the one-hour and fifty minute time limit. Similarly, for the 50-minute retention test, students on average took 24.79 minutes (SD=9.85), with the longest duration being 49.32 minutes.

Relationship between Test Scores and Time-on-Task: Students in the practice condition with distractors (M=20.67, SD=6.48) spent considerably more time on questions than those in the practice condition without distractors (M=12.89, SD=5.55). However, in looking more closely at the relationship between practice time and performance on the post and retention test, we see the trends differ between the two conditions (Figure 4). In the condition with distractors, practice time had a slight, though not significant, negative correlation with performance for both the post and retention tests (r=-0.19, p=0.189 and r=-0.23, p=0.148 respectively). However, in the condition containing no distractors, practice time was uncorrelated with performance on both the post test (r=0.01, p=0.960) and retention test (r=0.08, p=0.635).

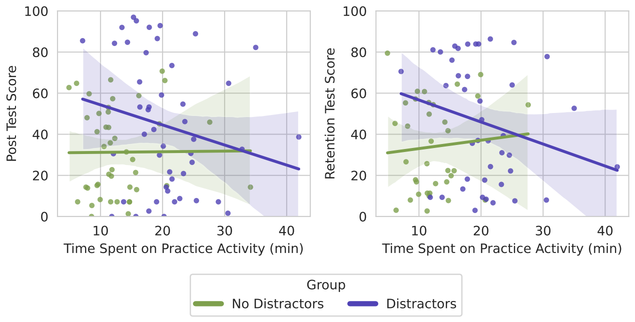


Figure 4: Students in the practice condition with distractors spent roughly double the amount of time on the practice activity than those in the practice

4.1 Analysis

To address the research questions relating to differences in performance between the two conditions on the post and retention tests (**RQ1**, **RQ2**), we perform an Ordinary Least Squares (OLS) regression.

$$\begin{aligned}\text{Score} = \beta_0 + \beta_1 \cdot \text{Distractors}\\ + \gamma_2 \cdot \text{FamiliaritySort} + \gamma_3 \cdot \text{FamiliaritySorted}\\ + \gamma_4 \cdot \text{Exam}\end{aligned}$$

This regression is used twice, where the dependent variable (Performance) is the score on each of the tests: post and retention. The *Distractors* predictor is a binary variable indicating whether that student was in the *no distractors* (0) or the *distractors* (1) practice condition. We include the students' self reported familiarity score (0-4) for each of the Python functions the activities covered (FamiliaritySort and FamiliaritySorted). To account for their general programming ability at the time of the study, we use their performance (0-100) on a an exam taken in the course from which they were recruited which took place in the same week the study was conducted. We discuss this further in Section 6.2.

4.2 Results

4.2.1 Differences in Performance on the Post Test (RQ1). Looking first at the raw scores on the posttest, we see that students in the practice condition that included distractors (M=43.89, SD=32.44) outperformed those in the condition without distractors (M=31.28, SD=21.51) by a large margin (Figure 5a). The results of the regression analysis confirm this difference is statistically significant and indicates students in the *distractors* condition performed approximately 12 percentage points better than those in the *no distractors* condition on the overall test (Table 3).

4.2.2 Differences in Performance on the Retention Test (RQ2). Looking first at the raw scores on the posttest, we see that students in the practice condition that included distractors (M=45.41, SD=29.43) still outperformed those in the condition without distractors (M=34.01, SD=22.44) by a large margin (Figure 5a). The results of the regression indicate that that difference was marginally insignificant. In more closely looking at the performance differences by question type we see that students in the *distractors* condition still outperformed their *no distractors* counterparts by a large margin on "Explain Error" questions. This may indicate that, though students ability to use and correct incorrect usage of the functions they practiced had waned, their ability to recognize incorrect usage of those functions was retained.

5 QUALITATIVE STUDY - ANALYSIS AND RESULTS

In total, ten think aloud interviews were conducted. The audio recordings from the interviews were transcribed. Researchers then performed thematic analysis on the resulting transcripts. Following the method presented by Jones et al. [40], researchers independently coded the first three interviews and met to discuss and consolidate a set of common themes. The remaining seven interviews were then independently coded by the two researchers using the consolidated themes. The researchers met to reconcile disagreements and integrate any new themes that emerged.

The results of the thematic analysis are presented as follows. Section 5.1 covers themes relating to how students solved Parsons problems and how the inclusion of distractors influenced it. Section 5.2 covers themes that emerged relating to students' perceptions of difficulty, distractor feedback mechanism, and learning potential of the problems they encountered.

- Brackets ([]) are used to add explanatory text to clarify what the student in the quote is referring to or add context on what the student is doing. (e.g., this [block] is, this [clicks on block]).
- A dash (—) indicates a hard stop in a sentence.
- Ellipsis (...) is used to indicate the removal of text.

Beyond the inclusion of these notations, the quotes had *minimal* changes made to them. The only change made to the text was the removal of excessive filler words that may hinder the reading and interpretation of the quote.

5.1 Solving Parsons Problems with Distractor Blocks (RQ3)

In this section, we highlight two key themes: 1) students' approaches to placing blocks that were not associated with distractors (Section 5.1.1) and 2) how the inclusion of distractors altered their behaviour (Section 5.1.2).

5.1.1 Students often place blocks not grouped with distractors without attending to the details of the code block: When interacting with blocks of code that were not grouped with distractors, students often placed blocks without closely attending to the code within the block. Rather, when placing these blocks, they referred to the common format for these code segments, e.g., function definition, for loop, conditional.

> *Interview 9:* "So starts with the **definition** defining it — and **indent** the **for loop** — and then again for **the print** because it'll iterate through okay, yeah, that was pretty easy. "
>
> *Interview 2:* " **I know how the code would look typically**, like the structure of it. **You define the function, then there's an indent and there's a for [loop]**..."

In extreme examples, some students were able to solve problems using these heuristics to solve all, or part of a problem, without reading the problem description.

> Interview 2: "**Definition first... return values last probably... Have to initialize this for loop conditional, and then what it does in the conditional.**" [Gets the answer correct].
> "Yeah, kinda **even without reading the directions** I kind of – Yeah, I kind of knew it was this without reading the directions just because of how it's formatted."

This shallow interaction with the code contained within a given block is particularly at odds when exercises are introducing new concepts within those blocks.

5.1.2 Distractors focus attention on blocks that they are associated with: The presence of distractors appear to be an antidote to the formerly raised issue of students placing blocks without attending to the details of the code within said blocks. This was made evident by many students stopping to carefully read through and compare the various blocks within the distractor group:

> *Interview 2:* "... then I need to pick one of these [blocks in distractor group] to go in between these two [blocks in solution]. It's kind of tricky to know **if it's value and key or key and then value**. I think you need these **parentheses by the items** so I think it's one of these ones [blocks ending with parentheses]."
>
> *Interview 1:* "And **it's name [key] first and grade [value] second** so we're going to choose from this [blocks with name, grade]."
>
> *Interview 8:* "So — I know this one [for v, k in d.items()] is **supposed to be key comma value**. I know this

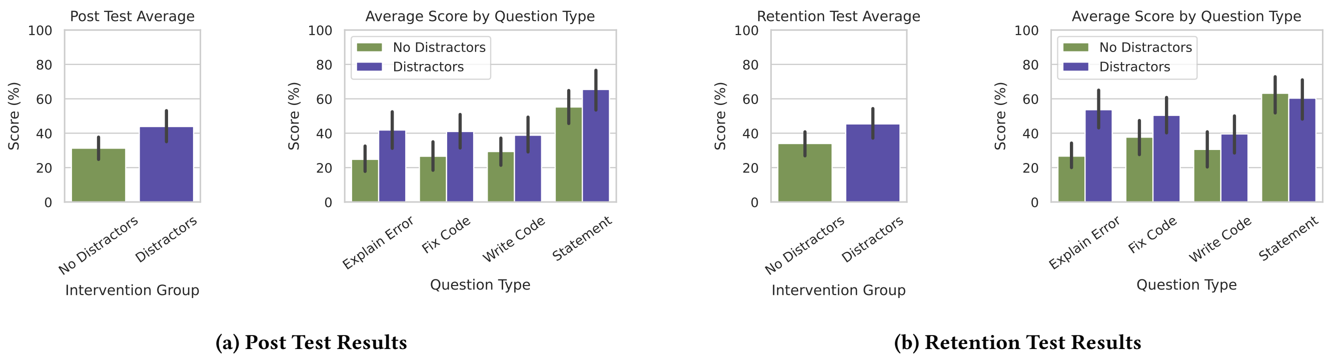


Figure 5: Performance differences between those practicing with distractors and those practicing without both in terms of overall performance and on each question type for the post and retention tests.

Table 3: OLS Regression Results. The group who solved Parsons Problems with distractors performed 12% better on the post test and 10% better on the retention test, even when controlling for prior student ability.

(a) Post Test

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-0.85	0.16	-5.19	0.000***	-1.17	-0.52
Distractors	0.12	0.05	2.43	0.017*	0.02	0.21
ExamScore	0.01	0.00	7.56	0.000***	0.01	0.02
FamiliaritySort	0.01	0.04	0.16	0.874	-0.07	0.08
FamiliaritySorted	0.01	0.04	0.15	0.884	-0.07	0.08
R^2_{adj}: 0.42	**F-statistic:** 16.56		**N-obs:** 88			

* p<0.05, ** p<0.01, *** p<0.001

(b) Retention Test

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Intercept	-0.96	0.18	-5.21	0.000***	-1.32	-0.59
Distractors	0.10	0.05	1.95	0.055	-0.00	0.20
ExamScore	0.02	0.00	7.51	0.000***	0.01	0.02
FamiliaritySort	-0.01	0.04	-0.25	0.801	-0.09	0.07
FamiliaritySorted	-0.00	0.04	-0.00	0.999	-0.08	0.08
R^2_{adj}: 0.44	**F-statistic:** 15.53		**N-obs:** 74			

* p<0.05, ** p<0.01, *** p<0.001

one is incorrect. So if I had to guess [it would] probably be this [for k, v in d.items()]."

In reflecting on differences in their experiences solving Parsons problems both with and without distractors, several students explicitly stated: 1) that they often place blocks that are not grouped with distractors without attending to the details of the code within the block and 2) that distractors forced them out of this behavior.

> *Interview 4:* "And the ones [questions] without the distractions were a lot quicker to solve because I know – **I know how the code would look typically. Like the structure of it.** You define the function then there's an indent and there's a for [loop]. So those were a lot quicker to solve. But **I don't think I really paid attention to what the code was about.** I just kind of dragged it [and], I was like, I know this is gonna be right. **But the ones with distractors I think made me think a lot more.**"

> *Interview 9:* "**You got to make sure you're really paying attention to the layout** of that [code block] and making sure you're doing it all correctly [with distractors], rather than when there aren't the distractors."

An objective of both the quantitative study and qualitative studies was to introduce a new concept and have students practice that concept with Parsons problems. Given that goal, many of the problems students encountered in both activities contained a mixture of simple problems (e.g., 2-4 blocks of code) and recognizable patterns (e.g., counting, filtering, categorizing) which may have exacerbated the degree to which students could successfully place blocks without closely attending to the details within. Though there may be arguments for grouping distractors with all blocks in such a condition so students are forced to attend to all the blocks, it appears from our results that *at minimum* those blocks which contain the concept being introduced should be coupled with some number of distractors.

5.2 Perceptions of Distractor Blocks (RQ4)

We highlight two thematic groups that emerged which pertain to students' perceptions of practicing with Parsons problems that contain distractors.

5.2.1 Relationship Between Difficulty and Learning. When asked to compare the two types of questions in terms of difficulty, the majority of students reported finding those questions that contained distractors as being more difficult than those without distractors.

> *Interview 4:* "So the ones without distractors are definitely the easier ones."

> *Interview 6:* "I mean, obviously, the one with like the distractors has a higher difficulty"

Despite this added difficulty students, by in large, still responded more positively to questions that contained distractors when queried on the matter. In particular, they cite the perception that the added level of difficulty contributed to their learning of the concept they were practicing.

> *Interview 9:* "I would say the ones with the distractor because, just because **they were harder questions and they required more thought**..."

> *Interview 8:* "Yeah. So I feel like the blue blocks [distractor blocks], **even though they were more difficult, I feel like I learned more**. I learned that the format for it was key, comma value, and you had to put the d dot items with the parentheses. . . But with the blue blocks, I still understood the question better. **So it was difficult, but I still understood it better. So it taught me more.**"

Related to the findings of Section 5.1.2, several students explicitly stated their perceived learning benifits of distractors was due to the distractor blocks causing them to more closely attend to the implementation details of the code blocks with which the distractor blocks were grouped.

> *Interview 5:* "I think it definitely helped because having a [problem] that have **distractors really makes you more attentive on how it should be**, because later on, I'll be writing these functions by hand without a structure already given to me. And I think that having those distractors are very important, because **those are very common errors that I might make while writing functions.**"

> *Interview 2:* "As someone that wants to learn, the one with the distractors is definitely better. Because you have to think harder about it. **With no distractors, it's definitely easier to just kind of get in the groove of the format and not really worry about the actual line of code.**"

However, perceptions regarding the learning value of the difficulty provided by the distractors was not universal. Two students indicated they found the presence of distractor blocks overwhelming and unhelpful.

> *Interview 3:* "Yeah, I think it's way easier to do it when there aren't a bunch of like – codes, and [distractor] blocks. **Because then even when you know the answer, you kind of overthink it and it takes longer.**"

> *Interview 10:* "**When you have distractor, there's a lot going on, right?** And so you don't really know — Okay, so now I have to think of, you know, which line to use, right? ... **For me, as a beginner, not having any Python experience before the class, I would prefer no distractors.** And then maybe as we get more advanced, towards the end of the semester in [course], then you will have more distractors..."

The primary takeaway of the interviews is that, though students found Parsons problems that contained distractor blocks to be more difficult, they both valued this difficulty and perceived it as contributing to their learning. Students, by and large, had a sense of the value of "desirable difficulties" which lead those difficulties that they did encounter to be valuable for their learning rather than merely obstacles. However, instructors should be aware that the presence of distractors may increase the difficulty of problems to be beyond what some students are capable (i.e., outside their zone of proximal development). This may reinforce the need for Parsons problems that are adaptive or investigations into other methods of scaffolding students' interactions with distractors. Providing a large number of attempts for each Parson problem may also help to address concerns about question difficulty.

5.2.2 The Perceptions and Role of Distractor Feedback. All students who encountered the feedback associated with distractors reflected on it positively. In particular, students noted the feedback both elucidated the origin of the error and reinforced the proper syntax of the block:

> *Interview 4:* "**I think the yellow boxes [distractor feedback, Figure 2b] really helped me understand why it was wrong.** So the first one, when iterating over a list of tuples, it tells — it told me that the order is key then value instead of value then key. Which helped me reinforce that my brain, especially from the reading, too. And then I think it just helped reinforce proper code format and what code should typically look like."

> *Interview 5:* "The first one [error] was the first question or just that usually do the key first before the value. And I was like, oh, okay, that makes a lot more sense... **Once I had that feedback, I felt a lot more clear about how it's supposed to be structured.** And that helped me a lot"

One student noted that the immediacy of the detailed feedback was useful and contrasted it with cases where feedback was not available.

> *Interview 3:* "I think a typo may tell you why something is wrong, because then that's something you're gonna remember. **If they just tell you what's wrong with no reason, then it's – it's a task to go and look it up or go back to the textbook.** So I would have probably just let it go. **But when someone tells you why something's wrong... you're gonna remember that and not make that mistake again.**"

However, one student did note that their interaction with the feedback was superficial, and, though it did help them arrive at the correct solution, it did not lead to a deeper understanding of the function they were learning.

> *Interview 2:* "The one that I really remember was when I picked the incorrect distractor for v comma k. And it told me that it should have been key comma value. **Honestly, I don't really know if it explained why or if I just kind of skimmed over it to understand what the right answer was.** But it was helpful in me getting the right answer. Although I don't really

understand why should be k and then v instead of the other way. "

As the original goal of distractors was to aid in correcting common errors and misconceptions in a scaffolded environment, these results provide indications that coupling distractors with feedback may aid in that reinforcement.

6 DISCUSSION

In the spirit of the explanatory mixed-methods approach used, we highlight two key takeaways using the results of the quantitative and qualitative results. In Section 6.1 we explore how differences in how students interact with problems containing distractors compared to those that do not may explain the learning differences observed between those conditions in the quantitative results. Similarly, in Section 6.2 we explore how the relationship between practice time and performance on the retention test differed between the two conditions.

6.1 Distractor Blocks and Attention

Sweller [76] provides selective attention as an example of how certain problem solving strategies can be at odds with learning.

> "Solving a problem and acquiring schemas [learning] may require largely unrelated cognitive processes... Previously used problem-solving operators and the relations between problem states and operators can be totally ignored by problem solvers using this strategy under most conditions."

Similarly, Rehder and Hoffman [66] showed that as individuals gain experience with categorizing various entities they optimize their attention to attend to the minimal set of features needed for correct categorization. The results of the qualitative analysis suggest that each of these are true, to some degree, for students experienced with solving Parsons problems.

Students in this study where midway through their semester and had encountered Parsons problems throughout their weekly homeworks as well as their exams. They were very familiar with the problem type and had experience solving them. As such, many of these students showed evidence of categorizing the blocks based on their purpose (e.g., for loop over list, if statement, add item to the list). Similarly, students showed evidence of correctly identifying relationships between blocks they categorized (e.g., if statement usually goes in for loop). However, in many cases students seemingly placed these blocks without closely attending to their implementation details. It was only when blocks were grouped with distractors that students closely read and compared the code within those grouped blocks.

The goal of the activity was to provide students practice using a function they had not previously encountered. If students are able to place blocks of code without closely attending to the syntax and semantics of the new function this appears to limit their ability to remember how to use that function in subsequent code fixing and code writing activities. Distractors offer the opportunity for instructors to draw students attention to lines of code that align with the learning objectives of the practice problems. Similarly, the inclusion of descriptive feedback on why a distractor is incorrect can further elucidate the nature of the error.

6.2 Distractor Blocks, Practice Time, and Learning

There was one notable exception, the relationship between practice time and performance on both the post and retention tests. Students practicing with distractors had a weak negative correlation between the time they spent on the practice activity and their performance on those tests. It is worth noting that this correlation was statistically insignificant and may be the result of a few outliers who took a great deal of time on the practice activity and performed poorly on the subsequent tests. Nevertheless, this result stands contrary to the intuition that increased time-on-task leads to more time spent learning and therefore better performance. It may be the case that, for a number of students, the inclusion of distractors pushed them out of their Zone of Proximal Development and lead to more time spent struggling rather than learning. This was reflected in one instance during the think-aloud interviews where one student showed signs of struggling with the distractors and appeared to disengage from the activity, simply trying the distractors one after the other rather than attending to their implementations. Future work should further consider this relationship, its causes, and further scaffolding that can be used to support students who may struggle with distractors such as adaptive Parsons problems.

7 LIMITATIONS AND FUTURE WORK

This work suffers from three core limitations that result from the experimental design. The first is that the familiarity survey was only given prior to the post-test. As such, in the analysis of the retention test, we are unable to account for any additional familiarity they may have gained from outside resources in the week between taking the post-test and the retention test. The risk of participants further encountering these concepts is small, as they were not taught in the course from which participants were sampled until well after the study's conclusion, but we cannot negate the possibility of participants engaging in independent study or reading ahead in their text book.

Second, and again with respect to the retention test, it is difficult to disentangle the impact of the practice from the impact of the post-test on the participants' performance on the retention test. The testing effect is a well understood phenomenon where the act of engaging in retrieval can improve long-term memory of the concept the test taker is being tested on [69]. The results relating to the retention test should be taken as the aggregate impact of study under a particular condition (i.e., distractors or no distractors) and any learning that occurred during the post-test.

Finally, both the qualitative and quantitative studies were focused on teaching participants functions that relate to Python's built-in data structures. Though the results of the qualitative study seem clear that distractors are required to focus participants on the implementation details of a given code block, it is unclear how the magnitude of the learning gains seen in this study would translate when using Parsons problems to teach other concepts (e.g., logic, conditionals, loops). Future work should investigate the impact of distractors on learning in a wider variety of topics.

8 CONCLUSION

Overall, we find strong evidence in support of including distractor blocks in Parsons problems when they are used in formative activities. The result of our randomized control trials show that students perform better on a post-test when practicing a new topic with Parsons problems that include distractors. The results of our think-aloud interviews indicate that this performance gain may largely be explained by the distractors causing students to more closely attend to the implementation of the code within the blocks with which distractors are grouped. The recommendation that follows from these results is that, if Parsons problems are to be used as a tool for introducing students to new concepts, distractors play an essential role in ensuring student success when they attempt to transfer those skills to code writing and fixing activities.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. DUE 21-21424.

REFERENCES

[1] Robert L Bangert-Drowns, Chen-Lin C Kulik, James A Kulik, and MaryTeresa Morgan. 1991. The instructional effect of feedback in test-like events. *Review of educational research* 61, 2 (1991), 213–238.
[2] Brett A Becker and Thomas Fitzpatrick. 2019. What do cs1 syllabi reveal about our expectations of introductory programming students?. In *Proceedings of the 50th ACM technical symposium on computer science education.* 1011–1017.
[3] Jeff Bender, Bingpu Zhao, Alex Dziena, and Gail Kaiser. 2023. Integrating Parsons puzzles within Scratch enables efficient computational thinking learning. *Research and Practice in Technology Enhanced Learning* 18 (2023), 022–022.
[4] Elizabeth L Bjork, Robert A Bjork, et al. 2011. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *Psychology and the real world: Essays illustrating fundamental contributions to society* 2, 59-68 (2011).
[5] Elizabeth Ligon Bjork, Nicholas C Soderstrom, and Jeri L Little. 2015. Can multiple-choice testing induce desirable difficulties? Evidence from the laboratory and the classroom. *The American Journal of Psychology* 128, 2 (2015), 229–239.
[6] Robert A Bjork. 1994. Memory and metamemory considerations in the training of human beings. (1994).
[7] Robert A Bjork. 2017. Creating desirable difficulties to enhance learning. *Carmarthen: Crown House Publishing* (2017).
[8] Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle Van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. 2018. " I know it when I see it" Perceptions of Code Quality: ITiCSE'17 Working Group Report. In *Proceedings of the 2017 iticse conference on working group reports.* 70–85.
[9] Derek C Briggs, Alicia C Alonzo, Cheryl Schwab, and Mark Wilson. 2006. Diagnostic assessment with ordered multiple-choice items. *Educational Assessment* 11, 1 (2006), 33–63.
[10] James E Bruno and Arie Dirkzwager. 1995. Determining the optimal number of alternatives to a multiple-choice test item: An information theoretic perspective. *Educational and Psychological Measurement* 55, 6 (1995), 959–966.
[11] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.
[12] John W Creswell and Vicki L Plano Clark. 2017. *Designing and conducting mixed methods research.* Sage publications.
[13] Lee J Cronbach. 1951. Coefficient alpha and the internal structure of tests. *psychometrika* 16, 3 (1951), 297–334.
[14] JHII Cross, T Dean Hendrix, and Larry A Barowski. 2002. Using the debugger as an integral part of teaching CS1. In *32nd Annual Frontiers in Education*, Vol. 2. IEEE, F1G–F1G.
[15] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research.* 113–124.
[16] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A review of research on Parsons problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference.* 195–202.
[17] Barbara Ericson, Austin McCall, and Kathryn Cunningham. 2019. Investigating the affect and effect of adaptive Parsons problems. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research.* 1–10.
[18] Barbara J Ericson. 2014. Adaptive Parsons problems with discourse rules. In *Proceedings of the tenth annual conference on International computing education research.* 145–146.
[19] Barbara J Ericson. 2016. Dynamically adaptive Parsons problems. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* 269–270.
[20] Barbara J Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S Miller, Briana B Morrison, Janice L Pearce, and Susan H Rodger. 2022. Parsons problems and beyond: Systematic literature review and empirical study designs. *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (2022), 191–234.
[21] Barbara J Ericson, James D Foley, and Jochen Rick. 2018. Evaluating the efficiency and effectiveness of adaptive Parsons problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research.* 60–68.
[22] Barbara J Ericson, Mark J Guzdial, and Briana B Morrison. 2015. Analysis of interactive features designed to enhance learning in an ebook. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research.* 169–178.
[23] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. 2017. Solving Parsons problems versus fixing and writing code. In *Proceedings of the 17th koli calling international conference on computing education research.* 20–29.
[24] Barbara J Ericson, Janice L Pearce, Susan H Rodger, Andrew Csizmadia, Rita Garcia, Francisco J Gutierrez, Konstantinos Liaskos, Aadarsh Padiyath, Michael James Scott, David H Smith IV, et al. 2023. Multi-Institutional Multi-National Studies of Parsons Problems. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education.* 57–107.
[25] Geela Venise Firmalo Fabic, Antonija Mitrovic, and Kourosh Neshatian. 2019. Evaluation of Parsons problems with menu-based self-explanation prompts in a mobile python tutor. *International Journal of Artificial Intelligence in Education* 29 (2019), 507–535.
[26] Bridgid Finn and Janet Metcalfe. 2010. Scaffolding feedback to maximize long-term error correction. *Memory & cognition* 38, 7 (2010), 951–961.
[27] Max Fowler, David H Smith IV, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education* (2022), 1–29.
[28] Flynn Fromont, Hiruna Jayamanne, and Paul Denny. 2023. Exploring the Difficulty of Faded Parsons Problems for Programming Education. In *Proceedings of the 25th Australasian Computing Education Conference.* 113–122.
[29] Mark J Gierl, Okan Bulut, Qi Guo, and Xinxin Zhang. 2017. Developing, analyzing, and using distractors for multiple-choice tests in education: A comprehensive review. *Review of Educational Research* 87, 6 (2017), 1082–1116.
[30] Bert F Green, Carolyn R Crone, and Valerie Greaud Folk. 1989. A method for studying differential distractor functioning. *Journal of Educational Measurement* 26, 2 (1989), 147–160.
[31] Louis Guttman and Izchak M Schlesinger. 1967. Systematic construction of distractors for ability and achievement test items. *Educational and Psychological Measurement* 27, 3 (1967), 569–580.
[32] Qiang Hao, David H Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. 2022. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education* 32, 1 (2022), 105–127.
[33] Qiang Hao, David H Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Amy J Ko. 2019. A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education (TOCE)* 19, 4 (2019), 1–18.
[34] Kyle James Harms, Jason Chen, and Caitlin L Kelleher. 2016. Distractors in Parsons problems decrease learning efficiency for young novice programmers. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* 241–250.
[35] Carl C Haynes and Barbara J Ericson. 2021. Problem-Solving Efficiency and Cognitive Load for Adaptive Parsons Problems vs. Writing the Equivalent Code. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–15.
[36] Carl Haynes-Magyar. 2024. Neurodiverse Programmers and the Accessibility of Parsons Problems: An Exploratory Multiple-Case Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 491–497.
[37] Juha Helminen, Petri Ihantola, Ville Karavirta, and Satu Alaoutinen. 2013. How do students solve Parsons programming problems?–execution-based vs. line-based feedback. In *2013 Learning and Teaching in Computing and Engineering.* IEEE, 55–61.
[38] Xinying Hou, Barbara Jane Ericson, and Xu Wang. 2022. Using adaptive Parsons problems to scaffold write-code problems. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1.* 15–26.
[39] Petri Ihantola and Ville Karavirta. 2011. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education* 10, 2 (2011), 119–132.
[40] Susan R Jones, Vasti Torres, and Jan Arminio. 2013. *Negotiating the complexities of qualitative research in higher education: Fundamental elements and issues.* Routledge.

[41] Sukjin Kang, Lawrence C Scharmann, and Taehee Noh. 2004. Reexamining the role of cognitive conflict in science concept learning. *Research in science education* 34 (2004), 71–96.

[42] Manu Kapur. 2008. Productive failure. *Cognition and instruction* 26, 3 (2008), 379–424.

[43] Manu Kapur. 2014. Productive failure in learning math. *Cognitive science* 38, 5 (2014), 1008–1022.

[44] Manu Kapur and Katerine Bielaczyc. 2012. Designing for productive failure. *Journal of the Learning Sciences* 21, 1 (2012), 45–83.

[45] Paul Kirschner. 2018. Inquiry Learning Isn't–A Call For Direct Explicit Instruction. *The researchED Magazine* 1, 1 (2018), 9–11.

[46] Paul Kirschner, John Sweller, and Richard E Clark. 2006. Why unguided learning does not work: An analysis of the failure of discovery learning, problem-based learning, experiential learning and inquiry-based learning. *Educational psychologist* 41, 2 (2006), 75–86.

[47] Melina Klepsch and Tina Seufert. 2020. Understanding instructional design effects by differentiated measurement of intrinsic, extraneous, and germane cognitive load. *Instructional Science* 48, 1 (2020), 45–77.

[48] Nate Kornell, Matthew Jensen Hays, and Robert A Bjork. 2009. Unsuccessful retrieval attempts enhance subsequent learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 35, 4 (2009), 989.

[49] Nate Kornell, Patricia Jacobs Klein, and Katherine A Rawson. 2015. Retrieval attempts enhance learning, but retrieval success (versus failure) does not matter. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 41, 1 (2015), 283.

[50] James A Kulik and Chen-Lin C Kulik. 1988. Timing of feedback and verbal learning. *Review of educational research* 58, 1 (1988), 79–97.

[51] Jeri L Little and Elizabeth Ligon Bjork. 2015. Optimizing multiple-choice tests as tools for learning. *Memory & Cognition* 43 (2015), 14–26.

[52] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research.* 101–112.

[53] Anna Ly, John Edwards, Michael Liut, and Andrew Petersen. 2021. Revisiting Syntax Exercises in CS1. In *Proceedings of the 22nd Annual Conference on Information Technology Education.* 9–14.

[54] Janet Metcalfe. 2017. Learning from errors. *Annual review of psychology* 68 (2017), 465–489.

[55] Brooke C Morin, Krista M Kecskemety, Kathleen A Harper, and Paul Alan Clingan. 2020. Work in Progress: Parsons Problems as a Tool in the First-Year Engineering Classroom. In *2020 ASEE Virtual Annual Conference Content Access.*

[56] Briana B Morrison, Lauren E Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals help students solve Parsons problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* 42–47.

[57] Fatni Mufit, Fauzan Festiyed, Ahmad Fauzan, and Lufri Lufri. 2018. Impact of learning model based on cognitive conflict toward student's conceptual understanding. In *IOP Conference Series: Materials Science and Engineering*, Vol. 335. IOP Publishing, 012072.

[58] Linus Östlund, Niklas Wicklund, and Richard Glassey. 2023. It's Never too Early to Learn About Code Quality: A Longitudinal Study of Code Quality in First-year Computer Science Students. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 792–798.

[59] Dale Parsons and Patricia Haden. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52.* 157–163.

[60] Seth Poulsen, Hongxuan Chen, Yael Gertner, Benjamin Cosman, Matthew West, and Geoffrey L Herman. 2023. Measuring the Impact of Distractors on Student Learning Gains while Using Proof Blocks. *arXiv preprint arXiv:2311.00792* (2023).

[61] Seth Poulsen, Yael Gertner, Hongxuan Chen, Benjamin Cosman, Matthew West, and Geoffrey L Herman. 2024. Disentangling the learning gains from reading a book chapter and completing proof blocks problems. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1.* 1056–1062.

[62] Seth Poulsen, Yael Gertner, Benjamin Cosman, Matthew West, and Geoffrey L Herman. 2023. Efficiency of learning from proof blocks versus writing proofs. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 472–478.

[63] Seth Poulsen, Mahesh Viswanathan, Geoffrey L. Herman, and Matthew West. 2021. Evaluating Proof Blocks Problems as Exam Questions. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA) *(ICER 2021)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/3446871.3469741

[64] Seth Poulsen, Mahesh Viswanathan, Geoffrey L Herman, and Matthew West. 2022. Proof blocks: autogradable scaffolding activities for learning to write proofs. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1.* 428–434.

[65] Mark R Raymond, Craig Stevens, and S Deniz Bucak. 2019. The optimal number of options for multiple-choice questions on high-stakes tests: application of a revised index for detecting nonfunctional distractors. *Advances in Health Sciences Education* 24 (2019), 141–150.

[66] Bob Rehder and Aaron B Hoffman. 2005. Eyetracking and selective attention in category learning. *Cognitive psychology* 51, 1 (2005), 1–41.

[67] J Elizabeth Richey, Juan Miguel L Andres-Bray, Michael Mogessie, Richard Scruggs, Juliana MAL Andres, Jon R Star, Ryan S Baker, and Bruce M McLaren. 2019. More confusion and frustration, better learning: The impact of erroneous examples. *Computers & Education* 139 (2019), 173–190.

[68] Lindsey E Richland, Nate Kornell, and Liche Sean Kao. 2009. The pretesting effect: Do unsuccessful retrieval attempts enhance learning? *Journal of Experimental Psychology: Applied* 15, 3 (2009), 243.

[69] Christopher A Rowland. 2014. The effect of testing versus restudy on retention: a meta-analytic review of the testing effect. *Psychological bulletin* 140, 6 (2014), 1432.

[70] Karim Shabani, Mohamad Khatib, and Saman Ebadi. 2010. Vygotsky's zone of proximal development: Instructional implications and teachers' professional development. *English language teaching* 3, 4 (2010), 237–248.

[71] Anna Shvarts and Arthur Bakker. 2019. The early history of the scaffolding metaphor: Bernstein, Luria, Vygotsky, and before. *Mind, Culture, and Activity* 26, 1 (2019), 4–23.

[72] David H Smith, IV, Seth Poulsen, Max Fowler, and Craig Zilles. 2023. Comparing the Impacts of Visually Grouped and Jumbled Distractors on Parsons Problems in CS1 Assessments. In *Proceedings of the ACM Conference on Global Computing Education Vol 1.* 154–160.

[73] David H Smith IV. 2023. Useful Distractions? Investigating the Utility of Distractors in Parsons Problems. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 2.* 62–63.

[74] David H Smith IV, Max Fowler, and Craig Zilles. 2023. Investigating the Role and Impact of Distractors on Parsons Problems in CS1 Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1.* 417–423.

[75] David H Smith IV and Craig Zilles. 2023. Discovering, autogenerating, and evaluating distractors for python Parsons problems in cs1. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 924–930.

[76] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[77] David Thissen, Lynne Steinberg, and Anne R Fitzpatrick. 1989. Multiple-choice models: The distractors are also part of the item. *Journal of Educational Measurement* 26, 2 (1989), 161–176.

[78] Rashmi Vyas and Avinash Supe. 2008. Multiple choice questions: a literature review on the optimal number of options. *Natl Med J India* 21, 3 (2008), 130–3.

[79] Lev Semenovich Vygotsky and Michael Cole. 1978. *Mind in society: Development of higher psychological processes.* Harvard university press.

[80] Nathaniel Weinman, Armando Fox, and Marti Hearst. 2020. Exploring challenging variations of Parsons problems. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* 1349–1349.

[81] Nathaniel Weinman, Armando Fox, and Marti A Hearst. 2021. Improving instruction of programming patterns with faded Parsons problems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–4.

[82] Matthew West, Geoffrey L Herman, and Craig Zilles. 2015. Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. In *2015 ASEE Annual Conference & Exposition.* 26–1238.

[83] David Wood, Jerome S Bruner, and Gail Ross. 1976. The role of tutoring in problem solving. *Journal of child psychology and psychiatry* 17, 2 (1976), 89–100.

[84] Zihan Wu. 2023. Investigating the Effectiveness of Variations of Micro Parsons Problems. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 2.* 120–122.

[85] Zihan Wu, Barbara Ericson, and Christopher Brooks. 2021. Regex Parsons: Using Horizontal Parsons Problems to Scaffold Learning Regex. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research.* 1–3.

[86] Zihan Wu and David H Smith IV. 2024. Evaluating Micro Parsons Problems as Exam Questions. *arXiv preprint arXiv:2405.19460* (2024).

[87] Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253.

[88] Bin Zhao. 2011. Learning from errors: The role of context, emotion, and personality. *Journal of organizational Behavior* 32, 3 (2011), 435–463.

[89] Craig B Zilles, Matthew West, Geoffrey L Herman, and Timothy Bretl. 2019. Every University Should Have a Computer-Based Testing Facility.. In *CSEDU (1).* 414–420.

A NORMALITY TESTS

Table 4: Shapiro-Wilk Normality Test Results for Familiarity Scores

Group	Test	Function Survey	W	p
Distractors	Post-Test	Sorted	0.794	5.13e-06***
		Sort	0.899	0.00177**
	Retention Test	Sorted	0.787	1.00e-06***
		Sort	0.898	0.00070***
No Distractors	Post-Test	Sorted	0.747	2.81e-06***
		Sort	0.898	0.00417**
	Retention Test	Sorted	0.796	3.66e-06***
		Sort	0.896	0.00113**

Table 5: Shapiro-Wilk Normality Test Results for Exam Scores

Group	Test	W	p
Distractors	Post-Test	0.801	2.05e10-06***
	Retention Test	0.764	1.32e10-06***
No Distractors	Post-Test	0.914	0.00385**
	Retention Test	0.915	0.0119***