Elicast: Embedding Interactive Exercises in Instructional Programming Screencasts

Jungkook Park[1*], **Yeong Hoon Park**[2*], **Jinhan Kim**[1], **Jeongmin Cha**[1], **Suin Kim**[1], **Alice Oh**[1]
[1]School of Computing, KAIST, Republic of Korea
[2]Department of Computer Science and Engineering, University of Minnesota, USA
[1]{pjknkda, jinhankim, jeongmin.cha, suin.kim}@kaist.ac.kr, [2]park1799@umn.edu, [1]alice.oh@kaist.edu

ABSTRACT
In programming education, instructors often supplement lectures with active learning experiences by offering programming lab sessions where learners themselves practice writing code. However, widely accessed instructional programming screencasts are not equipped with assessment format that encourages such hands-on programming activities. We introduce Elicast, a screencast tool for recording and viewing programming lectures with embedded programming exercises, to provide hands-on programming experiences in the screencast. In Elicast, instructors embed multiple programming exercises while creating a screencast, and learners engage in the exercises by writing code within the screencast, receiving auto-graded results immediately. We conducted an exploratory study of Elicast with five experienced instructors and 63 undergraduate students. We found that instructors structured the lectures into small learning units using embedded exercises as checkpoints. Also, learners more actively engaged in the screencast lectures, checked their understanding of the content through the embedded exercises, and more frequently modified and executed the code during the lectures.

Author Keywords
screencast; instructional screencast; programming exercises; embedded quizzes

INTRODUCTION
Instructional programming screencast, where an instructor records a video of their screen while writing code to teach programming, has become a popular medium for massive open online courses (MOOCs) [14], software development tutorials on the web [10, 21], and flipped classroom lectures [4, 12]. For learners, watching the process of designing and implementing the code can lead to better understanding of the logical workflow of writing a program [21]. However, these screencasts generally do not provide rich support for learners' interaction with the content, and they are thus limited in promoting learners' active engagement.

*Both authors contributed equally to this work.

L@S 2018, June 26–28, 2018, London, United Kingdom
© 2018 ACM. ISBN 978-1-4503-5886-6/18/06…$15.00
DOI: https://doi.org/10.1145/3231644.3231657

Active learning, where learners engage and participate in various activities in addition to listening to the lecturer, is particularly effective for programming education, as learners can apply abstract programming knowledge and obtain practical skills by writing their own code. In offline programming education settings, a common way to facilitate active learning is to offer programming lab sessions [6] where students work on a programming task while instruction and feedback are given by the instructor. However, it is difficult to emulate such educational environments in large-scale online lectures without a proper system to support them. Embedded (or in-video) quizzes, common in MOOC lectures [8, 9], can be regarded as an example of such support that promotes active engagement of the learners by providing frequent formative assessments [7]. However, existing online education platforms provide only a limited set of modalities for embedded quizzes, such as multiple-choice and short-answer questions [5, 22], which are not suitable for giving students hands-on programming practice.

We present Elicast, a screencast tool for recording and viewing programming lectures with embedded programming exercises. To provide continuous formative assessments with hands-on programming practices, we explore the idea of embedding programming exercises within instructional programming screencasts. Our approach goes one step further from placing simple quizzes within lectures, as we integrate programming exercises with various contents being presented in the screencast, namely the *code* written by the instructor. This approach differs from common programming assignments or notebook-based programming handouts (e.g. Jupyter Notebook) [25] that are given separately from the learning content. By integrating the assessment and the learning content together, the embedded programming exercises can be presented "just-in-time" and reduce the learners' cognitive burden of linking information from different sources together [24].

In this paper, we describe Elicast system focusing on how it embeds programming exercises in screencasts and how the exercises are automatically assessed, and explain how instructors can embed hands-on exercises in programming screencasts using the Elicast screencast recorder. Then, we report our exploratory study with five experienced instructors and 63 undergraduate students. Findings from the study include:

(i) instructors tend to structure the lectures into small learning units using programming exercises as checkpoints, and (ii) embedded programming exercises help learners to check their understanding and maintain their attention throughout the screencast.

RELATED WORK

Our research is related to three major research areas: screencasts for instructional contents, active learning in programming, and embedded quizzes in video lectures.

Instructional Programming Screencast

Screencasts, which are used in a variety of educational contexts, are becoming increasingly popular as a medium for programming education. In a programming screencast, an instructor typically demonstrates the process of writing code with explanatory narration. Screencasts are one of the common production styles for video lectures in MOOCs [14]. Moreover, there is a large and growing number of instructional screencasts in online video sharing platforms (e.g., YouTube) that demonstrate software development activities [10, 21]. Several studies [4, 12] also show the use of screencasts as flipped classroom lectures in computer programming courses.

There are several instructional screencast platforms supporting text-based screencasts that inspired Elicast. Khan Academy uses text-based screencast system in computer science courses and lets students modify and run the code during a lecture. Asciinema[1] is a screencast platform for recording and sharing terminal sessions, and Scrimba[2] is an interactive coding screencast platform mainly focused on the tutorials for web development technologies. Elicast begins with the screencast but goes beyond the prior work to seamlessly integrate hands-on programming exercises into the screencast, encouraging instructors to add checkpoints and learners to engage in active learning.

Active Learning in Programming

Literature on programming teaching distinguishes the ability to write program from being able to comprehend one. Study suggests there is little correspondence between the two abilities [33], and the ability to read and comprehend code does not directly translate to the ability to produce program code [11]. Thus, many educators give students problem-solving tasks with hands-on programming opportunities such as writing a program or debugging a given code [30]. Furthermore, a survey of university teachers and students revealed that they both agree that practical experience was most useful in understanding the programming concepts [20]. Elicast enables this active learning to occur in tight integration with the screencast lecture, which would enable delivering the hands-on programming experience to a wide audience of online learners.

Embedded Quizzes in Video Lectures

Embedded quizzes have been actively used in MOOCs as well as other online video lectures [5]. There are also a number of instances of using embedded quizzes [9, 4] in lecture videos for flipped classrooms [3]. The study shows the embedded quizzes significantly increase the learning gain of the lecture content [15]. Embedded quizzes are mostly used as a formative assessment in videos, and they can strengthen the understanding [22, 29] by providing frequent and continuous feedback [7].

In popular online learning platforms, embedded quizzes are mostly comprised of multiple-choice or short-answer questions. However, some platforms do allow embedding quizzes using other types of modalities. Several MOOC platforms support taking free-form text answers, and studies devised a way to provide automatic grading of the answers for open-ended questions [2] and essays [23]. Some studies applied different types of modalities used in embedded quizzes. L.IVE [24] embeds in-place assessment quiz in interactive videos, and RIMES [17] explores the idea of embedding more expressive quizzes that learners can record their answers with a webcam, microphone and drawings. As far as we know, Elicast is the first system to integrate hands-on programming exercises directly into the screencasts.

ELICAST

In this section, we describe Elicast, a web-based tool for recording and viewing programming screencasts with embedded programming exercises. First, we introduce how an instructor can use Elicast to embed programming exercises into the screencast (See Figure 1). Second, we explain how multiple embedded exercises can be automatically assessed using *assert* statements. Third, we show a usage scenario of Elicast when an instructor records a screencast and embeds multiple exercises in the screencast. Lastly, we show how the embedded exercises are displayed to learners and how learners can interact with the embedded exercises.

Embedding Programming Exercises

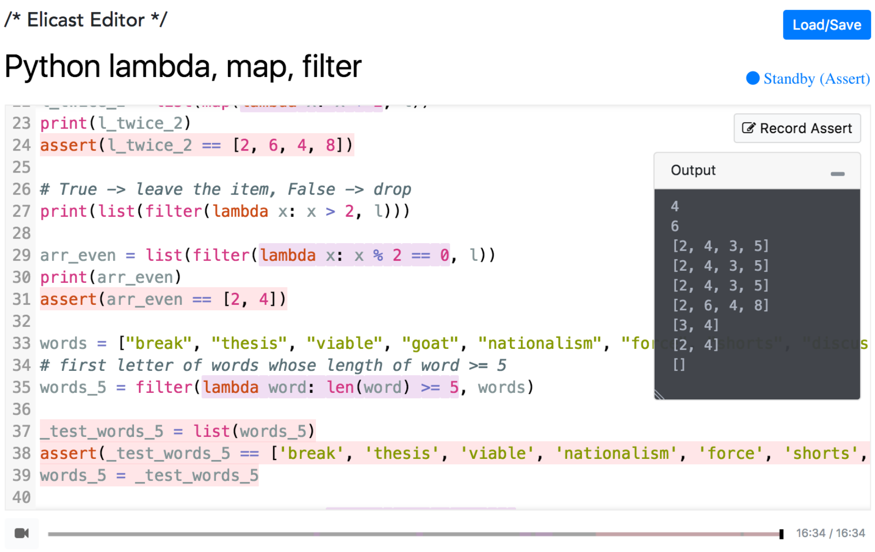
To provide hands-on programming experience for learners, Elicast allows instructors to embed programming exercises in the screencast. Then the embedded exercises are displayed to learners in a similar manner to conventional in-video quizzes; when an exercise appears during the screencast, the screencast is paused until the learner completes or explicitly skips the exercise. When a programming exercise appears, the instructor's code splits into two regions: a non-editable "skeleton region", and an editable "quiz region" where the learners insert their solution for the exercise (See Figure 2).

To integrate embedded programming exercises with instructor's code shown in the screencast, Elicast records screencasts in a text-based format, by capturing the instructor's screen as the stream of text change events [28] instead of a video consisting of image frames.

Assertion-based Automated Assessment

Elicast gives immediate assessment results upon learners' submission to the exercises. Namely, Elicast evaluates the functionality of the submitted code by testing whether the student's code performs the same functionality as the instructor's model solution that appears later in the screencast. While there are numerous automated assessment systems for evaluating program functionality, they are mostly based on the assumption

[1] https://asciinema.org/
[2] https://scrimba.com/

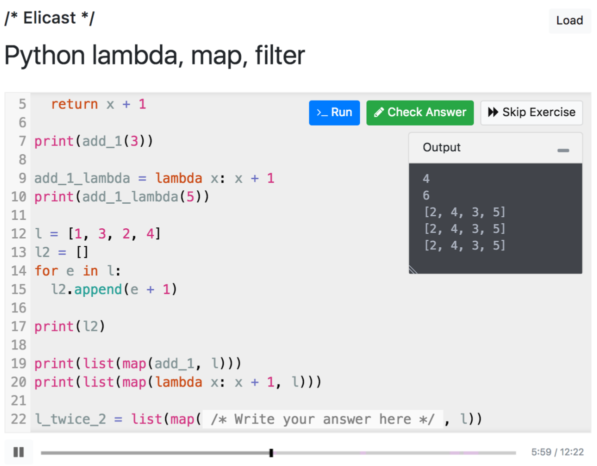


Python lambda, map, filter

```
print(l_twice_2)
assert(l_twice_2 == [2, 6, 4, 8])

# True -> leave the item, False -> drop
print(list(filter(lambda x: x > 2, l)))

arr_even = list(filter(lambda x: x % 2 == 0, l))
print(arr_even)
assert(arr_even == [2, 4])

words = ["break", "thesis", "viable", "goat", "nationalism", "fo
# first letter of words whose length of word >= 5
words_5 = filter(lambda word: len(word) >= 5, words)

_test_words_5 = list(words_5)
assert(_test_words_5 == ['break', 'thesis', 'viable', 'nationalism', 'force', 'shorts',
words_5 = _test_words_5
```

```
4
6
[2, 4, 3, 5]
[2, 4, 3, 5]
[2, 4, 3, 5]
[2, 6, 4, 8]
[3, 4]
[2, 4]
[]
```

Figure 1. The interface of screencast recorder for instructors. Instructor can record screencast, embed programming exercises, and add assertions for automated assessing. The text highlighted in purple shows the embedded programming exercise, and the text highlighted in red shows the assertions.

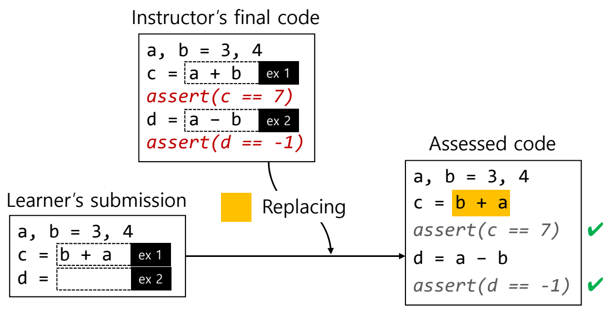


Python lambda, map, filter

```
  return x + 1

print(add_1(3))

add_1_lambda = lambda x: x + 1
print(add_1_lambda(5))

l = [1, 3, 2, 4]
l2 = []
for e in l:
  l2.append(e + 1)

print(l2)

print(list(map(add_1, l)))
print(list(map(lambda x: x + 1, l)))

l_twice_2 = list(map( /* Write your answer here */ , l))
```

```
4
6
[2, 4, 3, 5]
[2, 4, 3, 5]
[2, 4, 3, 5]
```

Figure 2. The interface of the screencast viewer for learners when an exercise appears. The code editor splits into two regions; a non-editable "skeleton region" and an editable "quiz region". The skeleton region is colored in grey, whereas the quiz region is colored in white with the place-holder text "Write your answer here". Learners can write arbitrary code in the quiz region. Learners can also run the code, submit the code for checking the answer, or skip the exercise.



```
a, b = 3, 4
c = a + b
assert(c == 7)
d = a - b
assert(d == -1)
```

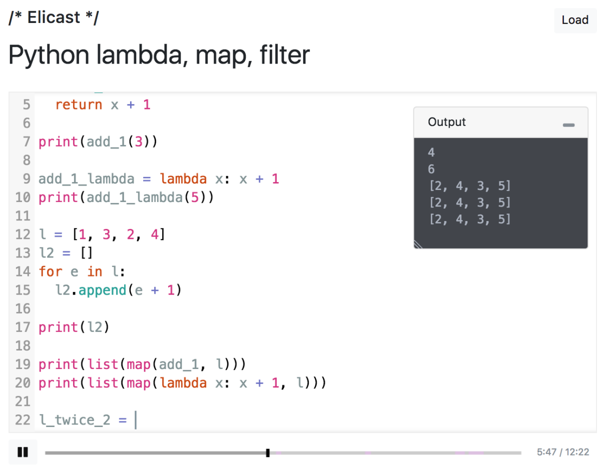
```
a, b = 3, 4
c = b + a
d =
```

```
a, b = 3, 4
c = b + a
assert(c == 7)
d = a - b
assert(d == -1)
```

Figure 3. The process of assertion-based automated assessment. Elicast constructs a temporary "Assessed code" behind the scene, by replacing the part of the instructor's final code that corresponds to the "quiz region" of the exercise with learner's response to the exercise. Then the "Assessed code" is checked if it passes all the assert statements.

that the submitted assignments are executable programs [1]. However, embedded exercises in Elicast can be created in any part of the code at any specific moment in the screencast where overall program structure may not be complete, thus a syntactically incomplete, unexecutable code may as well be a valid solution for some exercises. For example, as illustrated in Figure 3, suppose a screencast contains the code 'c=[exercise 1] d=[exercise 2]' at a specific moment and asks learners to answer the region '[exercise 1]'. In this case, whatever the learner submits as answer, the code cannot be executed because the code is incomplete due to the unanswered region 'd=[exercise 2]'. Thus, to make immediate grading possible even for incomplete codes, Elicast introduces a novel inline assertion-based automated assessing method which mixes two different codes – learner's submission code and instructor's code in the end of the screencast.

When students submit an answer to an exercise, Elicast selectively replaces a specific part of the instructor's final code of the screencast with the code written by the learner. For example, assume that the learner writes some code snippet to the first exercise. Figure 3 shows the process of assessing this example situation. Elicast system first calculates the region in the final code which corresponds to the "quiz region" of the first exercise. Then, Elicast constructs a temporary "Assessed code" by replacing the calculated region with learner's code snippet using the non-linear editing techniques for text-based screencasts [26]. Lastly, Elicast executes the "Assessed code" and checks the assertions. If there is an error, it can be seen as an error in the learner's code because other parts of the code are written by instructors to be correct. This process can



```
  return x + 1

print(add_1(3))

add_1_lambda = lambda x: x + 1
print(add_1_lambda(5))

l = [1, 3, 2, 4]
l2 = []
for e in l:
  l2.append(e + 1)

print(l2)

print(list(map(add_1, l)))
print(list(map(lambda x: x + 1, l)))

l_twice_2 = 
```

```
4
6
[2, 4, 3, 5]
[2, 4, 3, 5]
[2, 4, 3, 5]
```

Figure 4. The interface of the screencast viewer for learners. Learners can watch the screencast, run and modify the code in the editor, and engage in embedded programming exercises.

be applied to the exercises in the screencast one by one, thus Elicast can assess all exercises in the screencast. With this procedure, instructors can automatically assess all embedded exercises at once by writing assertions only at the end of the screencast.

Screencast Recording for Instructors

Lecture Content. An instructor starts recording by clicking the "Record" button. Then, she writes the code in the editor, and at the same time, narrates the explanation through the microphone. During the recording, the instructor can execute the code by clicking the "Run" button, and the running output of the executed code is shown in the "Output window", which is also recorded as part of the screencast. The instructor can pause and un-pause recording by pressing the "Pause" button. During the pause, the instructor can "Save", "Load", and "Cut" with the recorded screencast. "Cut" discards the latter part of the recording, allowing the instructor to re-record the screencast from the paused point.

Programming Exercise. The instructor embeds programming exercises by clicking the "Make exercise" button. Same as recording the lecture, the instructor can write code with voice narration; the difference from recording the lecture is that the area where the code is being written becomes an input field for the learners to fill in, and the instructor's code in that area will be hidden from the learners until they solve the exercise or skip it. After the instructor finishes writing code for the programming exercise, the instructor can click the "End making exercise" button to go back to recording the lecture.

Assertion. After recording the whole lecture, the instructor can click "Pause" and start making assertions by clicking "Record Assert". The instructor can write assertions around the input fields of the programming exercises. The instructor can also click "Run" while writing the assertions to check whether the assertions work correctly.

Screencast Viewer for Learners

A learner can start the screencast by clicking the "Start" button. Figure 4 shows the interface for the learners while playing the screencast. The learner can pause the screencast anytime by clicking "Pause", and when paused, they can change and run the code in the editor. However, the changes made by the learner is temporary, and it gets discarded when the screencast resumes.

When a learner encounters the programming exercise, as predefined in the lecture, Elicast pauses the screencast and displays an input field to ask the learner to solve the programming exercise (See Figure 2). The learner would listen to the explanation about the exercise with the instructor's voice narration, and solve the exercise by writing their own code. While solving the exercise, they can click "Run" to test the code, then click "Check Answer" to check if their answer is correct. Elicast automatically runs the learner's code with the assertions written by the instructor and gives immediate feedback {passed, failed} to the learner. The learner can then resume the screencast if the code has been assessed as correct, or they can retry or skip the exercise.

EVALUATION

We conducted exploratory studies with experienced screencast instructors and undergraduate students to investigate the efficacy of embedded programming exercises. We designed two studies: Study 1 investigates how instructors make use of embedded exercises in creating screencast lectures, and Study 2 examines how learners engage with the exercise-embedded screencasts.

STUDY 1. INSTRUCTORS RECORD EXERCISE EMBEDDED SCREENCAST

The first study focuses on the instructors' experience of making instructional programming screencasts with embedded programming exercises.

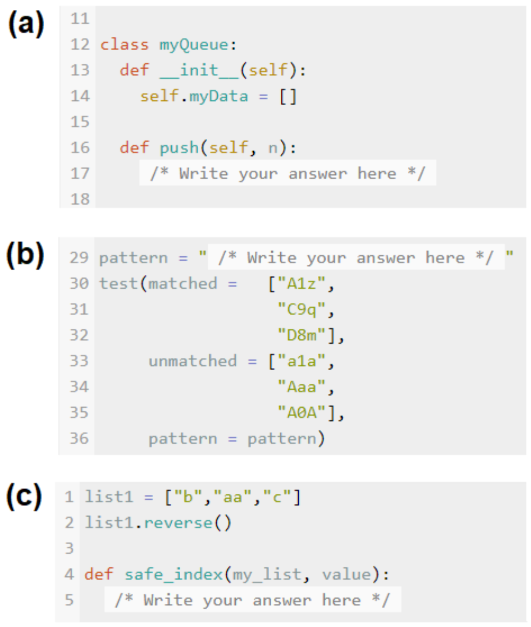
Participants

We recruited five instructors who have experience in teaching computer programming using screencasts on the web. Among them, four instructors had experience in teaching programming via live video lectures, and three of them also had experience in offline classroom lectures. All instructors stated that their teaching ability is sufficiently good to record instructional programming screencasts.

Procedure

We first provided a brief introduction of the experiment to the instructors with a tutorial video that shows how to use Elicast, with a focus on how to embed programming exercises and how the resulting screencast will be shown to students. After instructors watched the tutorial video, we provided information about the learners including their demographics and the level of prior knowledge in computer programming.

After the introduction, we asked the instructors to record at least two screencasts in different topics, each with more than three programming exercises. The topics were not restricted



```
class myQueue:
  def __init__(self):
    self.myData = []

  def push(self, n):
    /* Write your answer here */
```

```
pattern = " /* Write your answer here */ "
test(matched =   ["A1z",
                  "C9q",
                  "D8m"],
     unmatched = ["a1a",
                  "Aaa",
                  "A0A"],
     pattern = pattern)
```

```
list1 = ["b","aa","c"]
list1.reverse()

def safe_index(my_list, value):
  /* Write your answer here */
```

Figure 5. Examples of the embedded programming exercises created by the instructors. (a) Implement a method of Queue class (b) Write a regular expression that satisfies the test case below (c) Write a function described by the instructor.

but the instructors were asked to record a lecture at a level appropriate for second-year undergraduate students. They were given up to seven days to complete the screencasts.

Finally, we conducted a semi-structured interview with the instructors. For the analysis of the interview data, other two authors independently performed thematic analysis with the interview notes and transcripts.

Results

Five instructors recorded ten screencasts with topics on Python programming language and introductory data structures; the average duration of the recorded screencasts is 15.1 minutes ($s = 3.07$). The instructors created 36 embedded programming exercises in the screencasts (examples are in Figure 5), with an average of 3.6 exercises ($s = 1.07$) per screencast. The median time instructors spent in creating a 15-minute-long screencast was 24.24 minutes ($\bar{x} = 52.10$, $s = 79.50$).

During the interview, we asked the instructors a range of questions regarding their experience of creating screencast with Elicast compared to conventional screencast lectures. In the following sections, we describe the analysis results from the interviews.

Modularized, Checkpoint-style Learning Units

We first found that instructors designed the flow of the lectures differently from their past online screencasts; they tended to organize each screencast lecture into smaller learning units. Most instructors responded that with Elicast they felt they can insert checkpoints at the end of each sub-unit of the lecture.

> "With Elicast, what I felt different from the conventional lecture style was that I could define finer-grained goals of the lecture. For instance, I made programming exercises to teach `push` and `pop` instead of writing and testing the code for the `queue` as a whole." (Instructor 1)

> "A step-by-step lecture. Each step is comprised of an explanation of the concept and a programming exercise, and the exercises serve as checkpoints." (Instructor 5)

Instructor 2 pointed out that before recording the lecture he made a whole schedule to make his lecture modular, to gradually increase the difficulty of each module from easy to difficult.

> "When I was asked to record a 15 minute lecture with four exercises, I felt I needed to schedule well to evenly distribute time and the level of difficulty among the exercises. This was the difference from my past online lecture – at that time I didn't feel the need to do it this way. I designed each of the exercises to be easy, but the last one should incorporate all the concepts in the lecture video." (Instructor 2)

Assertions Are Easy-to-Create Yet Limited

All instructors felt assertion-based automated assessment is easy to create. Instructors 1, 2, 3, and 4 stated that they had no technical difficulty when writing assertions to test student's code as it felt similar to writing test cases in software development. Data from log analysis also corresponded; the time instructors spent on writing assertions has a median of 1.82 minutes ($\bar{x} = 1.81$, $s = 0.99$).

However, some instructors worried that the assertion-based assessment would not be applicable to certain types of exercises. Instructor 1 and 4 commented there would be certain cases where it was difficult to design the code with assert statements for their embedded exercises:

> "If I wanted to test a condition in an `if` statement, then it would be... quite difficult. There are certain places I can set as an input field, for instance, I won't set a class member declaration as an input field and let students to fill out – it could affect the whole code." (Instructor 1)

> "Some things cannot be tested with assertions. For example, class structure and constructors, especially when assessing based on how well the student formed the code structure. This is essential when we teach novice students, but it was hard to capture that using assertions." (Instructor 4)

Additionally, Instructor 3 commented that he had difficulty predicting how students would perceive the grading process:

> "Programming assertions was not a problem, but it was difficult to simulate how students would feel about the programming exercise. For instance, it took a while for me to design assertion logic and decide when and where to put them." (Instructor 3)

Instructors' Expectation of Pedagogical Benefits

While some instructors felt recording Elicast lecture took more time and effort, all expected that Elicast would be pedagogically beneficial to students. Instructor 1, who is well experienced delivering online live streaming lectures, stated that

Elicast would give students a feeling of participation, similar to pair programming. Instructor 2 felt interactive, embedded programming exercise would re-engage students losing focus.

> "I like the fact that students would feel they are writing code with me, rather than repeating after me – they usually just copy the code in the video. I like how students would feel they're learning together." (Instructor 1)

Instructor 4 emphasized that runnable programming exercises would bridge the gap in student's perception between watching and actually participating.

> "Based on my previous experiences, I feel there is a big difference between how students think they understand, just by watching the lecture, and what they actually know, which is revealed only when they type and run the program. What I like about Elicast is that this tool allows the instructor to quickly create small-sized exercises, which is intuitive to both instructor and students, and students can check if they actually understood the lecture – by doing." (Instructor 4)

One negative comment on embedding programming exercise came from Instructor 2, who worried that the exercise would interrupt the student's cognitive flow as it stops the lecture in several places.

STUDY 2. LEARNERS WATCH SCREENCAST AND ENGAGE IN EXERCISES

We studied how learners engage in the programming exercises embedded within the screencasts with a user study consisting of pre- and post-tests, pre- and post-surveys, and observations of their behavior while they watched the screencasts and solved the embedded exercises.

Participants

We recruited 63 undergraduate students who have taken an introductory computer science course using the Python programming language. The majority of students ($N = 46/63$) had taken only one CS course before the experiment. In 5-point Likert scale questionnaires from the pre-study survey, students responded they have moderate expectancy in programming ability compared to other students in their academic year ($\bar{x} = 3.03$, $s = 0.95$, $Md = 3$), yet they are interested in learning programming ($\bar{x} = 4.14$, $s = 0.84$, $Md = 4$).

Procedure

The experiment for the students consisted of three steps: taking the pre-study survey, watching two lectures and taking pre-test and post-test for each lecture, and taking the post-study survey. Each lecture was 15 to 20 minutes long and the overall process took about one hour.

The experiment started with a pre-study survey asking how many courses the students had taken and how proficient they were in programming. The questions in the pre-study survey were designed based on Expectancy-Value Theory of Achievement Motivation [31].

To analyze the effectiveness of the embedded exercises, students were asked to watch two lectures–one with embedded exercises and the other without, in a random order. For the screencasts without embedded exercises, instructor's explanation about the exercise was played without a pause. We chose three screencast lectures about (L1) *Introduction to Python class*, (L2) *Data structure: Queue*, and (L3) *Regular expressions*. We expected these lectures would be new to students who had not yet taken the data structures course. The summary of the three lectures are in Table 1.

	Title	Duration	# of exercises
L1	Max Machine	15:51	5
L2	Queue	14:21	4
L3	Python RE	20:38	4

Table 1. The summary of the three lectures selected for Study 2. The lectures are expected to be new to students who had not taken the data structure course.

We asked the students to answer five multiple-choice pre-test questions before each lecture. Every question in the pre-test had an option "I don't know" to see whether the concepts covered in the lecture were new to the students.

Students also answered post-test questions after each lecture. To avoid testing effect and pre-test effect, we designed the pre-post study followed the procedure used in [16, 32]; the post-test consisted of the same questions as the pre-test in a shuffled order; the learning gain was measured by the score difference between the pre-test and the post-test.

After the students finished watching the two lectures, they were asked to answer the post-study survey. The post-study survey contained five 5-point Likert scale questions which were designed to identify students' self-reported effectiveness and satisfaction on the embedded exercises. Lastly, we asked a free-form question in the post-study survey, asking "How did the embedded exercises help you learn the content? If it did not, why?".

Results

For the analysis of the result, we employed a hybrid method of quantitative and qualitative analysis on different sources of data: pre-test and post-test scores from each lecture, usage logs from the Elicast platform, and responses from the post-study survey.

For qualitative analysis of students' responses to the free-form questions, two authors independently coded and grouped the comments into recurring topics, then discussed to reach a consensus on the groupings.

Learners Engage in Lectures More Actively

We observed that students actively engaged in lectures when the lectures have embedded programming exercises. To measure the engagement during the lecture, we collected three navigation events (click play button, click pause button, and seek through timeline bar) which were always available during the lecture and used them as the proxy of video engagement [19]. As a result, we collected a total of 2,612 video navigation events from 63 students. From unequal variances t-test on the number of navigation events per student, we identified that the

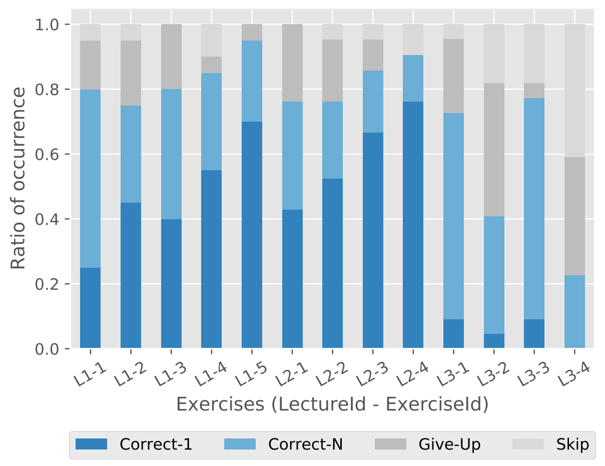


Figure 6. The ratio of exercise solving patterns for each lecture measured in Study 2. CORRECT-1: correctly answered in one try, CORRECT-N: correctly answered in multiple tries, GIVE-UP: tried but never got the answer, and SKIP: never tried.

navigation is significantly more frequent (unequal variance t-test, $t = 2.99$, $p < 0.005$) in the lectures with the embedded programming exercises ($\bar{x} = 25.16$, $s = 18.14$, $n = 63$) than the lectures without the exercises ($\bar{x} = 16.30$, $s = 14.72$, $n = 63$).

The majority of students actively participated in solving the exercises when they encountered them. Each of the students received an average of 4.33 exercises during the lecture, and, as summarized in Figure 6, 90.44% ($N = 247/272$) of the exercises were tried at least once, and 73.16% ($N = 199/272$) of the exercises were correctly answered either in a single try or after multiple tries.

The answers to the questions from the post-study survey showed consistent results of active engagement of students. In the post-study survey, the majority of the students responded they agreed that the embedded exercises positively affected their learning experience. When asked with 5-point Likert scale questions (1: strongly disagree, 5: strongly agree), students responded that the lecture with embedded exercises was more interesting ($\bar{x} = 4.17$, $s = 0.93$, $Md = 4$), easier to focus ($\bar{x} = 4.31$, $s = 0.88$, $Md = 5$), and easier to understand ($\bar{x} = 3.92$, $s = 1.02$, $Md = 4$).

We also qualitatively analyzed the responses to the free-form question in the post-study survey. In the responses, 13 students mentioned that they were able to stay focused and be engaged throughout the lecture because of the embedded exercises.

> "Online lectures are usually disengaging, but I stayed focused this time in order to solve the problems." (Student 17)

> "... and because we're forced to pay attention to the lecture I think it'll help us learn better." (Student 22)

> "It made me take time to write code and apply things that I might have overlooked otherwise." (Student 56)

On the other hand, a few students felt disengaged from the lecture because there were too many things to do.

	Avg. score (SD)		Avg. learning gain (SD)	
	Pre-test	Post-test	w/ Exercise	w/o Exercise
L1	4.17 (1.32)	5.00 (1.02)	0.95 (1.28)	0.70 (1.52)
L2	3.16 (1.91)	4.95 (0.80)	1.71 (1.96)	1.88 (1.80)
L3	0.98 (1.36)	4.78 (1.04)	*4.59 (0.94)	3.14 (1.74)

Table 2. The summary of pre- and post-tests scores and learning gain of the three lectures in Study 2. The unequal variance t-test is used to measure the significance of the average learning gain between the group with embedded exercises and the group without embedded exercises. $^*p < 0.001$

> "It was nice to be able to use what I learned immediately, but I lost interest if there was too much to learn." (Student 12)

> "There were too many small problems." (Student 38)

> "It was nice to be able to think and review the lecture to solve the problem. However, the feeling of interruption interfered with the concentration." (Student 61)

Preliminary Evidence on Higher Learning Gains

From the pre-test and post-test experiment designed to measure the learning gain from each lecture, we observed that the embedded exercises promote learning gains most when the lecture introduces concepts that are unfamiliar to the learners. The topic of L3 was observed to be new to most students (pre-test score $\bar{x} = 0.98$, $s = 1.36$, out of 6), and the learning gain of the group who had seen the lecture with the embedded exercises ($\bar{x} = 4.59$, $s = 0.94$, $n = 22$) was significantly higher (unequal variance t-test, $t = 3.42$, $p < 0.001$) than the group without the exercises ($\bar{x} = 3.15$, $s = 1.74$, $n = 27$). However, there was no evidence for significant difference of learning gains between the two groups for L1 and L2. We conjecture this is because there was not enough room to measure the learning gain. For both lectures, the majority of the students already scored more than half in the pre-test (Table 2), and answered the embedded exercises correctly in the first try (Figure 6), suggesting the topics were already familiar to most of the participating students.

Furthermore, responses to the free-form question in the post-study survey hint that embedded exercises have positive effects on learning gains. We grouped the responses into the three following categories:

Checking for Understanding: Eight students reported that they could check whether they were following the instructor's explanation up to the point of the embedded exercises.

> "I was able to clearly confirm that I understood the lecture." (Student 6)

> "... It gave me a chance to think twice about the contents that I was going to go through in confusion." (Student 58)

Memorizing: Six students mentioned that they could better memorize what they had learned in the lecture.

> "... It helped me to reinforce my knowledge because solving the exercises while watching the lecture led me to retrieve what I had learned before. ..." (Student 1)

> "Solving the exercises during the lecture, I was able to take control of my own learning, and I will probably remember longer through repetition of the concept." (Student 11)

Applying What I just Learned: 11 students said that they could apply what they learned during or right after the lecture.

> "... The exercises made me think about which parts of the lecture were really important. Also, they made me think about applying what I learned." (Student 10)

> "I realized that understanding something conceptually is quite different from applying it in practice." (Student 43)

On the other hand, a few students reported that there was nothing to learn because the lecture was too easy or too difficult.

> "I rarely understood the content of the second lecture, so I had to skip all the problems. To be honest, I did not get much help." (Student 48)

> "I think I would have been able to watch to the lecture more interestingly if I had it a bit more difficult topic." (Student 53)

Elicast Promotes Learning by Doing

We found that Elicast promotes "Learning by Doing"; students tended to run the code more frequently during lecture when there were embedded exercises. Although students in the group without the exercises were still able to modify and run the code after pausing the text-based screencasts, the number of code snippets executed per student during a lecture was observed to be significantly greater (unequal variance t-test, $t = 4.67$, $p < 0.001$) in the group who had seen the lecture with the embedded exercises ($\bar{x} = 5.14$, $s = 7.27$, $n = 63$) than the group without the exercises ($\bar{x} = 0.71$, $s = 1.69$, $n = 63$).

Also, not only the increment of students' execution activities, we observed positive learning effect from "Learning by Doing". From correlation analysis, we found a weak positive correlation between the number of code executions and the learning gain (Pearson's $r = 0.26$), in both the group without the embedded exercises ($r = 0.30$) and the group with the embedded exercises ($r = 0.27$).

Moreover, students responded in the post-study survey that they executed their code to understand the code written by the instructor, and tried different ways of problem-solving to further understand the lecture.

Trying Different Solutions: Several students reported that they got an opportunity to think for themselves and attempted implementations different from the solution suggested by the instructor.

> "... It made me think a little harder about implementing actually and thinking of all the possibilities for myself, even if the instructor solves it a certain way. ..." (Student 3)

> "... The embedded exercises from Elicast let me think about writing code for new examples." (Student 32)

DISCUSSION

Different Styles of Exercise-Embedded Screencasts

By analyzing the exercise-embedded screencasts created by the instructors from Study 1, we identified two types of instruction styles closely related to the different usage patterns of embedded programming exercises.

Writing Structured Code: In half of the screencasts ($N = 5/10$), the instructor demonstrates a process of writing structured code (i.e. python class, tic-tac-toe game) and let the learners write some of its components as exercises. For example, in a screencast titled "Queue" by Instructor 1, he scaffolds a small amount of code for a class for the queue data structure, and lets the learners implement some of the class methods such as `pop()` as embedded exercises.

Giving Independent Examples: The other half of the screencasts ($N = 5/10$) cover the topics that do not involve designing a structured code. These include Python list comprehension and regular expressions. In this type of screencasts, the instructor shows multiple examples while teaching the concept, and lets the learners try out writing new variations of the given examples as embedded exercises.

We also expect that more diverse instruction styles using embedded exercise will emerge as instructors get more familiar with Elicast. As of the design of our study, instructors were not given enough opportunities to get familiar with Elicast. In addition, since the screencasts in the study are targeted for second-year undergraduate students, the topics were limited to introductory programming topics (i.e. List comprehension) or basic data structures (i.e. queue). Thus, to further explore more instructional styles possible with exercise-embedded screencasts, further work is needed to test Elicast deployed in a more natural setting where instructors can use Elicast to produce a series of instructional screencasts, and cover topics with more diverse range of difficulty levels.

Learners Need More Feedback from Exercises

Elicast gives an automated assessment result immediately upon learners' code submission. However, it does not provide any information on why the submission is graded as incorrect. Some students brought up this issue in the post-study survey, expressing the need for more detailed feedback on why their solution was wrong. In particular, they said they needed some hints for further direction when their code was syntactically correct but semantically incorrect.

> "... but I thought I needed some hints that would guide me in solving problems and lead me to the intended direction when my code didn't have any grammatical error but didn't pass the test." (Student 31)

Overall, students expressed frustration about the lack of assistance when they were not able to make progress on the exercises. This can cause the learner to be de-motivated, potentially resulting in disengagement from learning [18]. Therefore, providing feedback with automated grading result is one of the major directions for further improvement of Elicast. Conventional test-cased feedback generation technique would be easily applied to assertion-based automated assessing, by labeling each assert statement with a feedback message. Applying more advanced approaches, such as using error models [27] or clustering code variations [13] of the learners' solutions, would be worth exploring in future work.

Exercise Engagement Data as Lecture Feedback

During the interview, we asked instructors for the possibility of using Elicast as a potential communication medium between students and instructors. Specifically, we asked what they would like to see if they can access students' problem-solving data collected from Elicast. All instructors responded that they would check the number of passes, fails, and skips for each exercise.

There were two potential use cases suggested by the instructors, of how to utilize student data. First, Instructors 1, 2, and 5 commented that they would regard pass, fail, and skip rate as a quantitative feedback from students. Higher fail rate would imply that the programming exercise was too difficult, and higher skip rate would mean the exercise was not interesting or too difficult.

> "I would consider myself a good instructor if most student pass all programming exercises." (Instructor 1)

> "The most skipped exercise would be my primary interest. Then I would improve my lecture based on that data." (Instructor 2)

Second, Instructor 3 and 4 wanted to see how students perform well by observing their problem-solving attempts. Both instructors said Elicast could provide finer-grained data to assess students' level of understanding, which cannot be obtained in conventional lecture settings.

> "It would be interesting if I could observe each student's status in finer granularity. Lecturers could know who did not understand which part of the lecture. Lecture can be split into smaller parts. This could not be done in my past lecture experiences." (Instructor 4)

Limitations and Future Work

From pre-test scores of lectures in Study 2, we found that two of the screencast lectures used in Study 2 (L1 and L2) covered the topics that were already familiar to the majority of participants, which made it difficult to determine the effect size of learning gain with the pre-test post-test experiment. For through validation of the benefits of embedded exercises on learning gains, further evaluation will be needed on the learning contents with more varying difficulty levels. Future work will deploy Elicast in real-world environments, and investigate the efficacy of exercise embedded screencast in the long term. We open-sourced Elicast under MIT license to be freely available for public use. The source code is available at `https://github.com/elicast-research/elicast`.

CONCLUSION

We present Elicast, a screencast tool for recording and viewing programming lectures with embedded programming exercises. Elicast employs just-in-time programming exercise within the screencast, which is designed to easily be embedded and make learners to engage in the lecture by writing their own code into the screencast. The exploratory study of Elicast with five experienced instructors and 63 undergraduate students showed that Elicast positively influenced the behaviors of both instructors and learners. In making screencast, instructors tended to organize each lecture into smaller learning units using exercises as checkpoints, and they expected that Elicast would bring benefits to learners by giving students a feeling of participation. We found that learners actively engaged in the lectures when they encountered the embedded programming exercises, and they tended to run the code more frequently when there are embedded exercises.

ACKNOWLEDGEMENTS

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7065962).

REFERENCES

1. Kirsti M Ala-Mutka. 2005. A survey of automated assessment approaches for programming assignments. *Computer science education* 15, 2 (2005), 83–102.

2. Sumit Basu, Chuck Jacobs, and Lucy Vanderwende. 2013. Powergrading: a clustering approach to amplify human effort for short answer grading. *Transactions of the Association for Computational Linguistics* 1 (2013), 391–402.

3. Jacob Lowell Bishop and Matthew A Verleger. 2013. The flipped classroom: A survey of the research. In *ASEE National Conference Proceedings, Atlanta, GA*, Vol. 30. 1–18.

4. Jennifer Campbell, Diane Horton, Michelle Craig, and Paul Gries. 2014. Evaluating an inverted CS1. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 307–312.

5. Jyoti Chauhan and Anita Goel. 2016. An analysis of quiz in MOOC. In *Contemporary Computing (IC3), 2016 Ninth International Conference on*. IEEE, 1–6.

6. Michael Clancy, Nate Titterton, Clint Ryan, Jim Slotta, and Marcia Linn. 2003. New roles for students, instructors, and computers in a lab-based introductory programming course. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 132–136.

7. National Research Council and others. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press.

8. Coursera. 2017. Answer in-video questions — Coursera Help Center. (2017). `https://learner.coursera.help/hc/en-us/articles/209818663-Answer-in-video-questions` [Online; accessed 18-September-2017].

9. Stephen Cummins, Alastair R Beresford, and Andrew Rice. 2016. Investigating Engagement with In-Video Quiz Questions in a Programming Course. *IEEE Transactions on Learning Technologies* 9, 1 (2016), 57–66.

10. Mathias Ellmann, Alexander Oeser, Davide Fucci, and Walid Maalej. 2017. Find, understand, and extend development screencasts on YouTube. *arXiv preprint arXiv:1707.08824* (2017).

11. Ursula Fuller, Colin G Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L Lewis, Donna McGee Thompson, Charles Riedesel, and others. 2007. Developing a computer science-specific learning taxonomy. In *ACM SIGCSE Bulletin*, Vol. 39. ACM, 152–170.

12. Gerald C Gannod, Janet E Burge, and Michael T Helmick. 2008. Using the inverted classroom to teach software engineering. In *Proceedings of the 30th international conference on Software engineering*. ACM, 777–786.

13. Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 7.

14. Philip J Guo, Juho Kim, and Rob Rubin. 2014. How video production affects student engagement: An empirical study of mooc videos. In *Proceedings of the first ACM conference on Learning@ scale conference*. ACM, 41–50.

15. Mina C Johnson-Glenberg. 2010. Embedded formative e-assessment: who benefits, who falters. *Educational Media International* 47, 2 (2010), 153–171.

16. Juho Kim and others. 2015a. *Learnersourcing: improving learning with collective learner activity*. Ph.D. Dissertation. Massachusetts Institute of Technology.

17. Juho Kim, Elena L Glassman, Andrés Monroy-Hernández, and Meredith Ringel Morris. 2015b. RIMES: Embedding interactive multimedia exercises in lecture videos. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1535–1544.

18. René F Kizilcec, Chris Piech, and Emily Schneider. 2013. Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the third international conference on learning analytics and knowledge*. ACM, 170–179.

19. Geza Kovacs. 2016. Effects of in-video Quizzes on MOOC lecture viewing. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. ACM, 31–40.

20. Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. In *Acm Sigcse Bulletin*, Vol. 37. ACM, 14–18.

21. Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, camera, action: How software developers document and share program knowledge using YouTube. In *Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on*. IEEE, 104–114.

22. Nehal Mamgain, Arjun Sharma, and Puneet Goyal. 2014. Learner's perspective on video-viewing features offered by MOOC providers: Coursera and edX. In *MOOC, Innovation and Technology in Education (MITE), 2014 IEEE International Conference on*. IEEE, 331–336.

23. John Markoff. 2013. Essay-grading software offers professors a break. (2013).

24. Toni-Jan Keith Palma Monserrat, Yawen Li, Shengdong Zhao, and Xiang Cao. 2014. L. IVE: an integrated interactive video-based learning environment. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 3399–3402.

25. Keith O'Hara, Douglas Blank, and James Marshall. 2015. Computational Notebooks for AI Education. In *The Twenty-Eighth International Flairs Conference*.

26. Jungkook Park, Yeong Hoon Park, and Alice Oh. 2017. Non-Linear Editor for Text-Based Screencast. In *Adjunct Publication of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 183–185.

27. Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices* 48, 6 (2013), 15–26.

28. Chengzheng Sun and Clarence Ellis. 1998. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. ACM, 59–68.

29. Karl K Szpunar, Novall Y Khan, and Daniel L Schacter. 2013. Interpolated memory tests reduce mind wandering and improve learning of online lectures. *Proceedings of the National Academy of Sciences* 110, 16 (2013), 6313–6317.

30. Mark J Van Gorp and Scott Grissom. 2001. An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education* 11, 3 (2001), 247–260.

31. Allan Wigfield and Jacquelynne S Eccles. 2000. Expectancy–value theory of achievement motivation. *Contemporary educational psychology* 25, 1 (2000), 68–81.

32. Joseph Jay Williams, Tania Lombrozo, Anne Hsu, Bernd Huber, and Juho Kim. 2016. Revising Learner Misconceptions Without Feedback: Prompting for Reflection on Anomalies. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 470–474.

33. Leon E Winslow. 1996. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin* 28, 3 (1996), 17–22.