A Pedagogical Analysis of Online Coding Tutorials

Ada S. Kim
The Information School
Mary Gates Hall 015
University of Washington
+1 206-498-1216
kimsk@uw.edu

Andrew J. Ko
The Information School
Mary Gates Hall 015E
University of Washington
+1 206-221-0352
ajko@uw.edu

ABSTRACT
Online coding tutorials are increasingly popular among learners, but we still have little knowledge of their quality. To address this gap, we derived several dimensions of pedagogical effectiveness from the learning sciences and education literature and analyzed a large sample of tutorials against these dimensions. We sampled 30 popular and diverse online coding tutorials, and analyzed what and how they taught learners. We found that tutorials largely taught similar content, organized content bottom-up, and provided goal-directed practices with immediate feedback. However, few were tailored to learners' prior coding knowledge and only a few informed learners how to transfer and apply learned knowledge. Based on these results, we discuss strengths and weaknesses of online coding tutorials, opportunities for improvement, and recommend that educators point their students to educational games and interactive tutorials over other tutorial genres.

Keywords
Online learning; coding tutorials; curriculum; pedagogy

1. INTRODUCTION
In recent decades, desire to learn programming has increased dramatically, while major government and non-policy efforts such as the Hour of Code, CS Education Week and CS For All have begun to create infrastructure for broad scale learning of computing and coding. To meet this high demand, a variety of online resources for learning how to code have emerged. Some of these tutorials are open-ended, creative platforms such as Scratch [24] and Alice [8]. Others are lecture-style courses provided by massively open online courses (MOOCs) like Coursera (coursera.org) and edX (*edx.org*). Some are tutorial-style curricula such as Khan Academy (*khanacademy.org*) and Codecademy (*codecademy.com*), which offer a range of content to teach popular programming languages and platforms. There are also many evidence-based educational programming games like Gidget [19], Lightbot [15], and Code Hunt [2], which aim to teach coding by gamifying some form of programming activity. There are of course also many reference guides with substantial example code, including W3 Schools (*w3schools.com*), Tutorials Point (*tutorialspoint.com*), and more social forums such as Stack Overflow (*stackoverflow.com*) that provide significant reference resources for learners. Popular tools such as the online Python Tutor even allow learners to visualize program execution [16].

SIGCSE'17, March 8–11, 2017, Seattle, WA, USA.

DOI: http://dx.doi.org/10.1145/3017680.3017728

Millions of people are using these resources every day to learn independently, but we have only just begun to understand their effectiveness. Recent work, for example, has explored the learning outcomes of open-ended creative environments and MOOCs, finding that while many learners use sophisticated programming language constructs [9, 11], there is still little evidence that they produce robust programming knowledge [18, 20, 34]. There is some evidence that explicit instruction and guidance through tutorials can improve learning [17], and more recent evidence-based that while e-books for CS teacher training can engage, learning is a continued challenge [33].

This evidence has several limitations. First, the evidence is *sparse*, only investigating a few types of tutorials; most of them are research prototypes [2, 15, 16, 19]. This means that we still know little about the current content of the popular tutorials that learners are using. Second, most of the evidence is *narrow*, in that it focuses on specific measurements of learning and engagement, overlooking many important factors in learning that are more difficult to measure and control for. The result is that teachers have little holistic guidance about how to choose effective tutorials and researchers have little insight into the broader set of online materials and how they differ.

To address these problems, we took an analytical approach to evaluating online coding tutorials, investigating what online tutorials currently teach and how they teach it by analyzing tutorials against a set of *curriculum design dimensions*. The benefit of an analytical approach is that we could assess a large set of tutorials and we could assess aspects of tutorials that are difficult to measure quantitatively. This approach is inspired by a long history of curriculum evaluation frameworks, which offer principles and rubrics grounded in theories of learning [10, 28, 29]. To improve the actionability of our results, we generated pedagogical principles specific to coding tutorials, deriving them from more general pedagogical design principles.

In the rest of this paper, we discuss our sampling and assessment approach in detail, describing how we derived our assessment model. We then discuss our results and their implications in detail.

2. METHOD

2.1 Selecting Tutorials
To begin, we generated a list of tutorials to evaluate (Table 1). One of our criteria for selecting tutorials was *popularity*. Using the Google search engine with two query terms "online coding tutorial" and "coding tutorial," we sampled and reviewed active coding tutorial websites appeared in the first 10 pages. We ensured Google's personalized search was turned off to prevent any effects from the search history in the browser. We excluded the websites that simply aggregated content from other sites.

In addition to search result popularity, we also estimated the amount of web traffic of each tutorial by using Alexa (*alexa.com*)

on July 29th 2016. We used Alexa's global rank, an estimate of a site's popularity relative to all other sites over the past 3 months, updated daily. The rank was based on a combined measure of unique visitors, the number of unique Alexa users who visited a site on a given day, and page views, the total number of Alexa user URL requests for a site. The site with the highest combination of unique visitors and page views was ranked the first. Based on the global rank provided by Alexa system, we included tutorial websites that ranked below 100,000.

We also considered popularity in educational settings. For example, Scratch [24] and Alice [8] are broadly used in classrooms but had relatively high Alexa rankings of 4,397,390 and 212,300, respectively. Educational games such as Lightbot, powered by Hour of Code, ranked 214,897. As this paper aimed to compare pedagogical approach across genres of online coding tutorials, we also included these tutorials.

Next, because many tutorial sites taught multiple programming languages, we also focused our assessments on the tutorials for popular languages. To do this, we referred to four online sources of programming language popularity: GitHub, tag rankings in Stack Overflow, TIOBE programming community index (*www.tiobe.com/tiobe-index*), and PopularitY of Programming Language index (*pypl.github.io/PYPL.html*). We chose courses and curricula that taught one of the six most popular languages (Java, Python, PHP, JavaScript, C#, C) overlapping across all four sources.

Our resulting sample included 30 tutorials, shown in Table 1. To help compare the tutorials, we also categorized them under one of five genres of resources:

- *Interactive tutorials* required learners to interact with command window, text editor, or equivalent in order to pass successive stages. This genre included sites such as Codecademy, Khan Academy programming, and *Codeschool.com*. Some of these tutorials focused on specific functionality such as Regex Golf (*regex.alf.nu*) and *Regex 101.com*.
- *Web references* played the role of a "dictionary." Tutorials under this genre, such as Tutorials Point, help learners properly code against a library, API, or platform. Some web references such as W3Schools or *Learnpython.org* provided code editors or command windows for learners who might want extra practice for reference code.
- *MOOCs* had a hierarchical structure with step-by-step stages, incorporating text-based quizzes and exams after a sequence of instruction. This genre included popular lectures in *Lynda.com*, edX, and Coursera.
- *Educational games* provided goals, story, and immediate feedback and often provided a more visually rich graphical environment. They often provided scores based on achievement or game items which can be consumed within a system. This genre included games such as Gidget [19], Code Combat, and Code Hunt.
- *Creative platforms* provided learners with an editor and content, but little instruction other than a reference guide and no explicit goals. This included Scratch [24] and Alice [8].

2.2 Dimensions for analysis

Here we describe our process for obtaining dimensions for analyzing the tutorials. First, we needed a framework against learning science principles. We based our evaluation on findings from learning sciences, focusing on the nine groups of 24 dimensions, shown in Table 1. These groups spanned four core pedagogical principles: 1) *connecting to learners' prior knowledge* [22, 23], 2) *organizing declarative knowledge* [3], 3) *practice and feedback* [1, 13], and 4) *encouraging meta-cognitive learning* [14, 21]. We adapted these four principles from the major effort over a decade ago to synthesize the seminal theoretical and empirical discoveries in learning sciences and education research into actionable principles for teaching and learning [3]. We decided to exclude principles related to collaborative learning, as most of the coding tutorials in our sample are not explicitly social experiences.

To assess the degree to which the tutorials in our sample followed the four principles, we generated 24 pedagogical dimensions specific to individual learning in coding tutorials. Each of the 24 dimensions we derived had significant nuance, but to simplify analysis and reporting, we reduced all but one dimension to a binary yes or no, where "yes" constituted satisfying a particular pedagogical design dimension, as defined by a written criterion. For example, the criterion for the *utilization* dimension (Table 1.2) was "*The instruction of the new stage explicitly requires to use at least one command or one function taught in the previous stage.*" If a tutorial met this criterion, we marked a "yes", and a "no" otherwise.

After analyzing the first few tutorials with a prototype of several dimensions we initially created, we evaluated them through discussions. We refined the detail of each dimension to see its necessity for tutorial analysis and removed unhelpful or uninformative dimensions. We iterated until all dimension criteria were sufficiently described and assessable.

With the final dimensions and criteria, we accessed each tutorial online and went through the course to checked the criteria for each dimension. In case that the category of answers was more than binary (Table 1.4), we recorded all answers. Also we marked "NA" in case that a tutorial was not applicable for a particular dimension (Table 1.3). We analyzed at least one module of each tutorial and in some cases analyzed an entire tutorial. It took approximately 2 hours per tutorial to check all criteria for dimensions and 60 hours overall.

3. RESULTS

Our final data set appears in Table 1, showing that tutorials varied widely in their compliance with our pedagogical principles, though some genres were more principled than others. In this section, we discuss each of the dimensions we evaluated in detail, organizing our discussion by our four core principles.

3.1 Connecting to learners' prior knowledge

Our first set of dimensions concerned tutorials' approach to learners' prior knowledge. It is now widely accepted in learning sciences that people construct new knowledge based on what they already know and believe [6, 7, 22, 23, 31, 32]. Regardless of age [23], learners bring prior knowledge in the form of facts, perceptions, beliefs, values, and attitudes to the new learning context [6, 7, 22]. While accurate and complete prior knowledge facilitates learning new knowledge, inaccurate and incomplete preconceptions hinders it. Therefore, in any learning context, it is important to understand learners' prior knowledge and deeply connect instruction to this prior knowledge.

To evaluate how the coding tutorials in our sample connected to learners' prior knowledge, we analyzed two groups of dimensions: *personalization* (Table 1.1) and *utilization* (Table 1.2) of knowledge. First, personalization of knowledge represented whether the tutorials customized teaching materials to meet prior knowledge along three dimensions: whether tutorials

Table 1. Thirty tutorials analyzed across 24 dimensions. Each check mark represents satisfaction of a pedagogical principle.

Tutorials	1. Personalization			2. Utilization	3. Contents								4. Organization		5. Context			6. Actionability	7. Feedback		8. Transfer learning			9. Support
	a. Age	b.Educational status	c. Coding experiences	Subsequent knowledge	a. Variables	b.Arithmatic	c. Logical	d.Conditional	e. Loops	f. Arrays	g. Function	h. Object	a. Bottom-up	b. Hierarchial structure	a. Lecture-based	b. Project-based	c. Storyline	Learners write code	a. Feedback	b. Immediate feedback	a. How to use	b. When to use	c. Why to use	Additional materials for self-monitoring
Codecademy Python		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔			✔
CodeSchool				✔	✔	✔	✔	✔					✔	✔	✔	✔	✔	✔	✔	✔	✔			✔
Khan Academy		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	both	✔	✔	✔		✔	✔	✔	✔			✔
Online Python Tutor													top-down					✔	✔	✔				
Codingbat Python				✔	✔		✔	✔	✔	✔			✔	✔				✔	✔	✔				✔
Regex Golf													✔					✔	✔	✔				
Regex 101		✔											✔					✔	✔	✔	✔			✔
W3School JS					✔	✔	✔	✔	✔	✔	✔	✔	✔					✔	✔	✔	✔			✔
LearnPython					✔	✔	✔	✔	✔	✔	✔	✔	✔					✔	✔	✔				
Learnjavascript					✔	✔	✔	✔	✔	✔	✔	✔	✔					✔	✔	✔				
Funprogramming			✔	✔				✔	✔	✔	✔		✔					✔		✔	✔	✔		✔
TutorialsPoint Python		✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔								✔			✔
Cprogramming		✔			✔	✔	✔	✔	✔	✔	✔		✔								✔			✔
Codingunit C tutorial		✔	✔		✔	✔	✔	✔	✔	✔	✔		✔								✔	✔		✔
Wired.com Javascript			✔		✔			✔	✔	✔	✔	✔	✔	✔							✔			✔
Pythonprogramming					✔	✔		✔	✔	✔	✔		both						✔		✔			✔
Stack Overflow					N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	top-down						✔		✔	✔	✔	✔
edX Microsoft JS				✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔				✔		✔			✔
Lynda.com		✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	top-down	✔	✔				✔		✔	✔	✔	
Coursera Javascript				✔	✔	✔	✔	✔	✔	✔	✔	✔	top-down	✔	✔			✔	✔	✔	✔			✔
Coursera Python				✔	✔	✔	✔	✔	✔	✔	✔	✔	both	✔	✔			✔	✔	✔	✔			✔
Code.org	✔	✔	✔	✔	✔	✔	✔	✔	✔		✔		top-down	✔	✔	✔	✔	✔	✔	✔	✔			✔
Gidget				✔	✔	✔	✔	✔	✔	✔	✔		✔	✔		✔	✔	✔	✔	✔	✔			✔
CodeCombat		✔		✔	✔	✔	✔	✔	✔	✔	✔		both	✔			✔	✔	✔	✔	✔	✔		✔
LightBot	✔	✔	✔	✔	✔				✔				both	✔		✔	✔	✔	✔	✔				
CodeHunt			✔	✔	✔	✔	✔	✔	✔	✔			both	✔		✔	✔	✔	✔	✔				✔
Code Avengers		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔			✔	✔	✔	✔			✔
Grok Learning		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔		✔	✔	✔			✔	✔	✔	✔		✔	✔
Scratch	✔	✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	top-down			✔		✔	✔	✔	✔			✔
Alice		✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	top-down			✔		✔	✔	✔	✔			✔

helped learners select an appropriate learning material based on their age range (Table 1.1.a), educational status (Table 1.1.b), or prior coding knowledge (Table 1.1.c). Among many ways to personalize learning materials, we chose these three because they are common factors that curricula use to differentiate instruction in formal educational systems. Many coding tutorials did not personalize what they teach for their learners when we evaluated the level of personalization of three dimensions above. Only Code.org, Lightbot, and Scratch explicitly indicated appropriate learners' age range for tutorial selection. Fourteen out of 30 tutorials considered learners' education status, but it was rather superficial such as vaguely separating beginner, intermediate, and advanced levels to indicate difficulty. None of the tutorials recommended specific stages or modules, based on learners' prior coding experience.

For the "utilization" dimension, we analyzed how the tutorials helped learners leverage the knowledge they accumulated throughout the tutorial (Table 1.2). For example, some tutorials summarized the knowledge from prior lessons and showed learners how to apply it to new concepts; others taught material once, and never mentioned it again. All educational games required learners to apply knowledge from stage to subsequent stages, which helped learners' better connection of knowledge than other genres of tutorials. A few interactive tutorials like Code School and Codingbat Python (*codingbat.com/python*), and MOOCs like Coursera and edX, also had structural stages that utilized the information taught in a stage to the new ones. Only one web reference tutorial, *FunProgramming.org*, had the similar form of structure. Only Codecademy, Khan Academy, and Code Avengers (*codeavengers.com*) required knowledge from previous stages to pass the overview stage at the end of the module.

3.2 Organizing declarative knowledge

Transforming factual information into robust declarative knowledge is another key principle for effective learning [3]. For successful transformation, binding a large set of disconnected facts is important as well as connecting prior knowledge to new knowledge [1,23,31,32]. Providing a conceptual framework for organizing information into meaningful knowledge helps learners to gain a deeper understanding of learning material [4,5].

To apply these principles to our evaluation, we analyzed the *content* of what tutorials taught (Table 1.3), focusing on the eight learning objectives in the FCS1 assessment [28], which included basic programming language concepts such as variables, arrays, loops, and functions. All five genres of tutorials taught all eight learning objectives except a few tutorials that focused on specific abstractions (namely Regex 101 and Regex Golf). Most of the educational games did not teach, or at least not explicitly, the concept of objects or object-orientation.

How information is organized can influence application of declarative knowledge [1]. Experts often organize information hierarchically, indicating their deeper understanding of how various pieces of information fit within a complex structure. In that sense, we analyzed the *organization* of information (Table 1.4), noting whether the tutorials structured information "bottom-up" (starting with basic concepts and building up to complex ones) or "top-down" (successively breaking down complex concepts into smaller ones) (Table 1.4.a), and whether the structured information was in a hierarchical form or not (Table 1.4.b).

Tutorials organized the information differently across the genres. For example, web references and interactive tutorials organized content bottom-up, starting with the most elementary concepts, using one or two commands or functions at a time to solve a problem. The most common case was teaching how to print "Hello World" using a certain programming language; in order to do this, they explained what kinds of command (e.g. print() in python3) should be typed in the text editor or interpreter, and displayed how it worked.

In contrast, educational games, MOOCs, and creative platforms combined both bottom-up and top-down approach or mainly used top-down approach more than web references and interactive tutorials. For example, Scratch suggested a goal ("Make the cat in the screen dance") that learners can model and provided step-by-step instruction to reach the goal, but also allowed high level of freedom for users to apply the instruction to design one's own code.

Organizing information hierarchically can help learners connect scattered facts [1,5]. All games structured information hierarchically, which included many simple stages teaching one command or function at a time under particular programming topics. MOOCs and interactive tutorials with high Alexa ranking like Codecademy and Khan Academy did the same. For example, a module teaching conditional statements often included many sub-stages about how to correctly write `if` and `while` structures.

Finally, we analyzed the *context* of how the information was organized (Table 1.5), judging the story, background, and other concrete details in which content was presented [1, 3, 4]. We considered three dimensions of context. The first was the use of lectures, presenting content authoritatively (Table 1.5.a). MOOCs and the popular interactive tutorials used lecture-based contexts heavily (Table 1.5.a). We also considered the use of goal-driven project contexts, in which learners were given a high level goal to achieve by learning lower-level content (Table 1.5.b). Such goals can help learners' active engagement in goal-based practice [13]. Only a few tutorials, primarily educational games and creative platforms, offered project-based contexts that provided an explicit goal of a stage or a module. For example, one of the goals in Gidget's stage was to operate a small robot, named Gidget, to carry a kitten to the basket. To achieve the goal, learners should think about not only what functions to be written, but also how to arrange them. Finally, we considered the use of story-based contexts (Table 1.5.c), which were used to connect learning goals. For example, Code Hunt supposed learners as "hunters" who perform a secret project by fixing fragments in codes.

Most of the web references did not establish a specific learning context for what they taught, whether an authoritative lecture based context, a goal-driven context, or a story-based context.

3.3 Practice and feedback

Evidence is clear that deliberate practice helps learners achieve mastery in a particular domain [12, 25]. Clearly structured and articulated goals are critical to enhancing the effectiveness of deliberate practice [13]. Deliberate practice, however, must be coupled with appropriately targeted feedback, including information about learners' progress to guide them toward goals [1]. To evaluate whether the tutorials supported deliberate practice, we analyzed two groups of three dimensions: *learner actionability* (Table 1.6) and *feedback* (Table 1.7).

Deliberate practice becomes effective when learners actively engage in it; the best way to practice coding is to write code. Therefore, our *learner actionability* (Table 1.6) dimension measured whether the tutorial required learners to actually write programs of some kind to learn. We found that all genres of tutorials offered some kind of interactive editor requiring learners to provide input, with the exception of a few web references that provided read-only information. We also found interesting diversity in the type of editors across the interactive tutorials and education games. For example, Gidget equipped a sophisticated editor panel so that learners even could see the error messages and syntax errors in the editor, which was more instructive than just providing a text guideline for practices. Khan Academy provided visualized walkthrough with the editor panel so that learners could modify and run the code to see how their editing changed contents in the walkthrough.

Prior work has shown that immediate, targeted feedback is critical for meta-cognitive monitoring [1,3,5]. Therefore, to analyze feedback, we judged two dimensions: whether tutorials provided feedback at all (Table 1.7.a) and whether that feedback was immediate (Table 1.7.b). All interactive tutorials and educational games with a code editor provided some form of immediate feedback, but much of these was shallow. For example, almost half of the tutorials did not provide feedback when learners made errors. These tutorials fell into two cases: 1) some tutorials like Scratch provided open-ended practice, but did not provide feedback about right or wrong code relative to a goal or 2) a tutorial's code editor did not produce feedback about error messages. These latter tutorials were usually web references that allowed learners to test functionality, but did not explain failures.

Some tutorials provided feedback through instructor or peer communication. For example, MOOCs provided some opportunities to communicate with instructors or peers, and some resources had online communities in which learners could ask questions. While this feedback was available, none of it was immediate and learners had no guarantee of receiving answers to their questions.

3.4 Encouraging meta-cognitive learning

Two key ideas of meta-cognitive learning are learners' ability to predict the outcomes of their learning tasks and monitoring their understanding [4, 5]. Focusing self-reflection on what worked and what needs improving helps learners transfer what they learned to the new settings and events [14, 21, 26, 27].

To evaluate whether tutorials encouraged metacognitive learning, we analyzed whether they taught *how, when,* and *why* learners should use a particular command or a function to help learners transfer or apply knowledge learned from the tutorials (Table 1.8). Few tutorials guided learners in transferring and applying knowledge to further learning contexts outside of the curricular provided by tutorials. Most of the tutorials strongly emphasized how to use particular functions and commands in coding. Only

five tutorials across three genres, web references, educational games, and MOOCs attempted to explain when and why learners should use a particular command or a function.

We also analyzed whether the tutorials provided *support* by providing additional materials outside the curriculum so that learners monitoring their understanding could seek answers to their own questions beyond the tutorial content (Table 1.9). Almost all genres of tutorials provided some form of additional support, whether it was a discussion form or additional references or resources. Four tutorials attempted to indicate where a learner's performance was ranked and how high it was, which might encourage learners to self-monitor their level of progress in learning. For example, Code Hunt provided information related how fast and accurate the learner performance was after passing every stage, which enhance learners' engagement in playing and level completion speed [20]. Five tutorials proactively helped learners recognize errors in their actions. Gidget was a good example: The tutorial notified its learners when they omitted a required expression at the end of the function (i.e. "When I try to access an object in the world, I need to terminate its name with a "/" character.")

3.5 Tutorial Recommendations

Despite their limitations, interactive tutorials and educational games satisfied the majority of the pedagogical principles reflected in our dimension's criteria. All tutorials in both genres required learners' active engagement in writing code, and most of them provided a structured hierarchy including several stages of goal-directed practice with subsequent applying of learned knowledge. The educational games in particular offered many forms of context, which may help learners actively engage in deliberate practice. The educational games also provided the most immediate and personalized feedback, likely improving the gains from deliberate practice. Therefore, from a pedagogical perspective, we recommend games such as Gidget, Lightbot, Code Hunt, and tutorials provided by Code.org as the tutorials most likely to be effective at producing learning.

4. DISCUSSION

Our results reveal several trends in coding tutorial pedagogy:

- They largely teach the same content.
- Most teach content bottom-up, starting with low-level programming concepts, and building up to high-level goals.
- Most require learners to write programs.
- Most provide some form of immediate feedback in response to learner actions, but this feedback is shallow.
- Few explain when and why a particular concept is useful in programming.
- Few provide guidance for common errors.
- None provide personalization based on prior coding experience or learner goals, other than rudimentary age-based differentiation.

Despite the diversity of languages and content, most of the coding tutorials shared a similar paradigm. They dissected coding into the most detailed, elemental level. This bottom-up approach in organizing information enabled the tutorials provide goal-based practices with one simple task for each stage. For example, most of the tutorials gave instruction about how to write a few simple lines of code (e.g., `var1 = 1, var2 = 2`, then what would be `var1 + var2`?) and test it by typing the answer to the command window. At this low level, the goal was clear (use this function to clear the stage) and feedback was also clear (clear the stage or not). In that sense, the tutorials might fulfill one important criterion of effective learning: providing clearly structured, and articulated goals for practice, in the beginning stages.

This paradigm has several limitations. First, coding tutorials gave more attention to emphasizing *how* to practice particular commands and functions rather than to provide contextual information like *when* and *why* to use them. More generally, none of the tutorials provided a detailed and systemized problem solving instruction other than one- or two-sentence hints when learners made errors in each stage. These pedagogical choices might limit tutorials' ability to teach learners' to apply skills to broader learning contexts outside of the curriculum.

Lacking a personalized instruction might also limit effectiveness. As our first learning principle emphasizes, it is important to connect existing knowledge to the new knowledge, and to consider learners' incomplete understandings and the false beliefs in that connecting process. However, most of the tutorials did not provide access to any sort of agent or instructor to give personalized feedback to guide deliberate practice. Second, we found that while tutorial feedback was immediate, it was rarely precise enough to improve learners' conceptions of the material, and it was not customized at all to learners' prior knowledge. This is a major area for future work that has yet to be deeply explored.

5. LIMITATIONS

There are many limitations in our study to address in future work. Although our sample was diverse, it is not necessarily representative of all of the tutorials used around the world, particularly those in languages other than English. Although we tried to measure popularity by using Google search engine and Alexa, these methods could provide only general information about how many learners visited the website per day, not about their actual progress. Moreover, online coding tutorials are constantly evolving as companies seek ways to improve learning and engagement.

Our analysis also has limitations. Most of our criteria were binary judgments, even though many of the dimensions have substantially more nuance. The first author was also the only one who assessed all of the tutorials, and so there may have been systematic bias in her evaluations that was not eliminated through redundant coding.

Another major limitation of our study is that we analytically assessed tutorials, rather than measuring learning outcomes directly. It may be possible that many of the tutorials are effective despite failing to satisfy many of the learning principles we identified in prior work, as those principles might have been met in subtle ways not observed in this study.

6. CONCLUSION

Our results suggest that most online coding tutorials are still immature and do not yet achieve many key principles in learning sciences. Future research and commercial development needs to better emphasize personalized support and precise, contextualized feedback and explore ways of explaining to learners why and when to use particular coding concepts. Based on our sampled tutorials, we recommend that teachers be very selective in their use of materials, focusing on the more evidence-based tutorials, particularly the educational games. All educational games in the list provide hierarchical structure, immediate feedback, and opportunities that learners actively write code and use subsequent knowledge for coding throughout the tutorial. With future

research, these tutorials and potentially future commercial tutorials will become much better teaching supplements, as well as resources for independent learning.

7. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants 1314399, 1240786, and 0952733 and by Microsoft.

8. REFERENCES

[1] Ambrose, S.A., Bridges, M.W., DiPietro, M., Lovett, M.C. and Norman, M.K. 2010. *How learning works*. John Wiley & Sons.

[2] Bishop, J., Horspool, R.N., Xie, T., Tillmann, N. and de Halleux, J. 2015. Code Hunt: Experience with coding contests at scale. 398–407.

[3] Bransford, J., Brown, A.L., Cocking, R.R. National Research Council (U.S.) Committee on Developments in the Science of Learning 2000. *How people learn: Brain, mind, experience, and school*.

[4] Brown, A.L. 1975. The development of memory: Knowing, knowing about knowing, and knowing how to know. Advances in Child Development and Behavior. Elsevier. 103–152.

[5] Chase, W.G. and Simon, H.A. 1973. Perception in chess. *Cognitive psychology*. 4, 1, 55–81.

[6] Cobb, P. 1994. *Theories of mathematical learning and constructivism: A personal view*. Symposium on Trends and Perspectives in Mathematics Education.

[7] Cobb, P., Yackel, E. and Wood, T. 1992. A Constructivist alternative to the representational view of mind in mathematics education. *Journal for Research in Mathematics education*. 23, 1, 2.

[8] Cooper, S., Dann, W. and Pausch, R. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*. 15, 5, 107–116.

[9] Dasgupta, S., Hale, W., Monroy-Hernández, A. and Hill, B.M. 2016. Remixing as a pathway to computational thinking. 1438–1449.

[10] Dewey, J. 1959. The child and the curriculum.

[11] Ericson, B.J., Guzdial, M.J. and Morrison, B.B. 2015. Analysis of interactive features designed to enhance learning in an Ebook. 169–178.

[12] Ericsson, A.K. and Charness, N. 1994. Expert performance: Its structure and acquisition. *American Psychologist*. 49, 8, 725–747.

[13] Ericsson, A.K., Krampe, R.T. and Tesch-Römer, C. 1993. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*. 100, 3, 363–406.

[14] Flavell, J.H. 1976. *Metacognitive aspects of problem solving*. The nature of intelligence.

[15] Gouws, L.A., Bradshaw, K. and Wentworth, P. 2013. Computational thinking in educational activities: an evaluation of the educational game light-bot. 10–15.

[16] Guo, P.J. 2013. Online python tutor: embeddable web-based program visualization for cs education. 579–584.

[17] G, Y. and K, F. 2014. The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education - An International Journal*. 131, 33–50.

[18] Lee, M.J. and Ko, A.J. 2015. Comparing the effectiveness of online learning approaches on CS1 earning utcomes. 237–246.

[19] Lee, M.J. and Ko, A.J. 2011. Personifying programming tool feedback improves novice programmers' learning. 109–116.

[20] Lee, M.J., Ko, A.J. and Kwan, I. 2013. In-game assessments increase novice programmers' engagement and level completion speed. 153–8.

[21] Palincsar, A.S. and Brown, A.L. 1983. *Reciprocal teaching of comprehension-monitoring activities*. Technical Report #269.

[22] Piaget, J. 1978. Success and understanding.

[23] Piaget, J. 1952. The origins of intelligence in children.

[24] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. 2009. Scratch: programming for all. *Communications of the ACM*. 52, 11, 60–67.

[25] Rothkopf, E.Z. and Billington, M.J. 1979. Goal-guided learning from text: Inferring a descriptive processing model from inspection times and eye movements. *Journal of educational psychology*. 71, 3, 310–327.

[26] Scardamalia, M., Bereiter, C. and Steinbach, R. 1984. Teachability of reflective processes in written composition. *Cognitive science*. 8, 2, 173–190.

[27] Schoenfeld, A.H. 1983. Problem solving in the mathematics curriculum: A report, recommendations, and an annotated bibliography.

[28] Tew, A.E. 2010. Assessing fundamental introductory computing concept knowledge in a language independent manner.

[29] Tyler, R.W. 1959. *Basic principles of curriculum and instruction*. University of Chicago Press.

[30] Tyler, R.W. 1967. Changing concepts of educational valuation. *Perspective of Curriculum Evaluation*. Rand McNally & Company 13–18.

[31] Vygotsky, L.S. 1980. Mind in society: The development of higher psychological processes. Harvard University Press.

[32] Vygotsky, L.S. 1962. Thought and language. MIT Press.

[33] Werner, L., Campe, S. and Denner, J. 2012. Children learning computer science concepts via Alice game-programming. 93–98.

[34] Zhu, J., Warner, J., Gordon, M., White, J., Zanelatto, R., & Guo, P. J. (2015, October). Toward a domain-specific visual discussion forum for learning computer programming: An empirical study of a popular MOOC forum. In Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on (pp. 101-109). IEEE.