INTRODUCTION: WHAT AND WHY OF FORMATIVE FEEDBACK?

Formative feedback refers to formal and informal assessment moves or procedures that teachers employ in an effort to make inferences about what their students know and can do during their routine classroom learning. This is seen as assessment for learning (as opposed to assessment of learning, which is the more summative view of assessment). The overarching objective of the formative assessment process is not to assign a performance grade to a student but rather to supply reliable evidence to the teacher and student that could be used to enhance students’ learning.

Computer science teachers can informally assess students in several ways, for example, a show of hands in response to a question; students’ expressions of frustration, disengagement, or joy during a coding task; and informal conversations with students as they code and debug their programs. However, education literature makes the case for formal methods of feedback collection as well. Groundbreaking classroom research in the late 1990s by Paul Black and Dylan Wiliam showed that formative assessment in the classroom improves student learning.

Formative assessment is a process that involves both teachers and learners, and is characterized by the following:

1. When teachers implement formative assessment as a process in collaboration with their students, it can result in powerful learning,

2. Formative assessment as a process operates as a feedback loop and involves teachers making adjustments to their instruction based on evidence they collect while student learning is developing and providing students with feedback that helps them advance their learning, and

3. Students are equal stakeholders in the process and participate through peer and self-assessment.

Extending the idea elucidated in the third point above, a key part of the formative assessment process is the involvement of learners in the assessment process. In addition to peer and self-assessment, formative feedback to the learner (distinct from feedback to the teacher) is essential.

Formative feedback benefits both the teacher and the student by making a student aware of what they need to work on and by making the teacher aware of adjustments (if necessary) to teaching activities and pacing. It alerts teachers to students’ prior knowledge,
as well as possible misconceptions. It lets teachers know which students have understood the ideas well and those who may need extra help or support in their learning. The teacher may infer from a quick autogradable quiz that most students did not understand how to construct nested conditionals, or they may discover through a short open-ended programming exercise what the students are passionate about outside of school. Formative assessments may also serve to reveal that students may have weak prior preparation in a topic in a related subject like mathematics that she had taken for granted and that may be impacting their ability to construct Boolean expressions that use arithmetic and relational operators.

Brigid Barron and Linda Darling-Hammond assert in their book on project-based learning that robust assessments for deeper and meaningful learning must include “intellectually ambitious performance assessments” that define the tasks students will undertake in ways that allow them to learn and apply the desired concepts and skills in disciplined ways; “create guidance for students’ efforts in the form of evaluation tools such as assignment guidelines and rubrics that define what constitutes good work”; and frequently use formative assessments to guide feedback to students and teachers’ instructional decisions throughout the process.

**TYPES OF FORMATIVE ASSESSMENT**

Grover’s work on assessments in introductory programming suggests that there needs to be multiple forms—or “systems”—of assessment to get a holistic and multifaceted view of student learning. Which assessments should be used when is a matter of choice for the teacher, where the class is in the learning schedule, what students might enjoy, the time available in a period or unit, the time taken (some forms are time-consuming to score whereas others provide near-instant feedback), the time available for the teacher (to grade and such), and modality (pen and paper, online, or in a programming environment). Table 1 details types of formative assessment with examples of various forms of each type as well as features and pros and cons of each type. The remainder of this chapter provides illustrative examples useful examples of each type.

---

**Table 1. Types of formative assessment with examples, features, pros and cons**

<table>
<thead>
<tr>
<th>Assessment Type</th>
<th>Examples / Details</th>
</tr>
</thead>
</table>
| Programming Assignments                | • Open-ended programming assignment  
  • Open project with specific criteria  
  • **Example 1**: My Project has at least two sprites engaging in a conversation.  
  • **Example 2**: My Project uses blocks to change the appearance to match different backdrops in a story.  
  • Closed-ended programming assignment with a desired end goal  
  • Complete a partially coded programming project |
| • Engaging and motivating               |                                                                                     |
| • Usually time-consuming to score (and subjective) |                                                                                     |
| • Do not provide quick feedback        |                                                                                     |
| • Must be accompanied with rubrics for teachers and students |                                                                                     |
| Modality: programming environment      |                                                                                     |

---

Peter Drucker famously said, “**What’s measured improves.**” Formative assessment allows teachers to see what students are thinking; this in turn helps them identify student difficulties. The ability to recognize and address errors in students’ thinking through the use of formative assessment is extremely important in that one can improve only what one measures.

---

**Feedback through Formative Check-Ins**

Peter Drucker famously said, “**What’s measured improves.**” Formative assessment allows teachers to see what students are thinking; this in turn helps them identify student difficulties. The ability to recognize and address errors in students’ thinking through the use of formative assessment is extremely important in that one can improve only what one measures.
Giving students an assignment to code a project of choice comes with its own set of considerations as does administering a quiz at the start or end of a class period. For example, projects of choice are undoubtedly an authentic form of assessment that is well-suited to project-based learning. Many chapters in this book advocate programming to engage students’ creativity through programming projects, and rightly so—they provide students an opportunity to be motivated and engaged and express their creativity in a way a multiple-choice quiz could never do. However, they are time-consuming as a formative assessment activity and even more so to score. Furthermore, scoring is subjective. Additionally, a finished project does not reveal whether a student got the program to work through “trial and error” or from borrowing a code snippet from a partner or some other program without deep understanding. However, a well-designed multiple-choice question can very quickly reveal whether a student understands a certain concept or not.

### Table 1. Types of formative assessment with examples, features, pros and cons (continued)

<table>
<thead>
<tr>
<th>Assessment Type</th>
<th>Examples / Details</th>
</tr>
</thead>
</table>
| **Quizzes: Multiple Choice (MC) & Fixed Answer** | • Present code snippets that require students to demonstrate code-comprehension skills. For example,  
**•** A program with “fill-in-the-blank” slots (fixed response or options to choose from for the blanks)  
**•** Analyzing and comparing programs  
**•** Determining whether a piece of code meets its goal  
**•** Multiple-choice options to fix a piece of buggy code  
**•** Multiple-choice options for an expression for a conditional/loop  
**•** Present a programming requirement in text  
**•** Multiple-choice options to pick the correct coding solution or aspects of the solution |
| **Quizzes: Innovative Item Types** | • Parson’s problems or puzzles (rearranging provided code blocks or commands in correct sequence)  
**•** Hotspot items  
**•** Unit-tested coding (autograded)  
**•** Match options in two columns |
| **Quizzes: Open Response Types** | • Quizlike prompts involving code snippets that require open-ended responses probing for explanations or descriptions of what a code snippet does |
| **Showcasing | Peer & Self Assessment** | • Explanations - these could be written or oral or audio/video recorded that accompany students’ code about their programming project  
**•** Show & Tell: project presentations to share various aspects of the project |
| **Video/Audio Self Explanation & Reflection** | • Reflective journals to track progress on a large project  
**•** Reflective prompts that reveal learner experience—thrills, frustrations & difficulties, collaboration; other aspects of learning |
| **Artifact-based Interviews** | • Conversation with teacher about a project |
Ideally, a teacher will balance programming assignments with more objective assessment instruments that illuminate student understanding of specific computing concepts and other computational thinking (CT) skills such as debugging, code tracing, problem decomposition, and pattern generalization. Short, high-frequency, low-stakes autogradable quizzes help to keep learners aligned with the content and understanding goals of the course. The classroom culture should be such that they are seen not as “tests” for grading the student but as a means to provide feedback to both the student and the teacher.

Summative Feedback

Although this chapter is focused mainly on formative feedback, summative assessments should also be used to get a sense for how well students are able to coalesce all their learning over the course of a term. Many of the types of assessment described next can also serve the purpose of summative assessment. Final open-ended projects, artifact-based interviews, student reflections, student portfolios, and tests with a mix of multiple-choice and open-response items are other instruments for summatively measuring student learning.

EXAMPLES OF FORMATIVE ASSESSMENTS

A. Informal Teacher Observation and Conversations With Students

A reliable way for a teacher to ascertain what students do and do not understand is through observation while they work on an assignment. It is often easy to spot their struggle with a concept, gauge their understanding and implementation. Teachers can also see whether students are developing sound debugging strategies. Paying attention to student conversations with peers is invaluable in providing insight into their understanding and ability to pose good questions and arguments. (See Chapter 17 on what questions you can ask and how to respond to student questions.)

As teachers observe, they may pause to have short conversations with students. Having students show their code for a certain concept or explain a problem they just solved provides excellent opportunities for feedback. The observation and conversation time may be as short as, say 3–5 minutes, to check the pulse of the students.

It’s important to have a way to record observations and conversations quickly, either on paper or digitally. You may then use the data you collected to inform future instruction both for students who need additional help and for those who are ready for additional challenges.

B. Entry and Exit Tickets

Time is often a scarce resource in computer science classes. Learning a concept may extend over multiple classes with a week between classes. Using quick check-ins at the beginning or end of classes can be helpful. With primary students (grades K–2), exit tickets may be as simple as choosing an emoji or a picture of faces showing emotions to indicate if they think they understood the lesson (smile, neutral, frown). For older students, an exit ticket typically includes a question that will help to identify if the student understood the content for the day and may also include their opinion of how the day went for them. For the former, ideas from Section C—Quizzes—can be used to keep the exit ticket to about 5 minutes at the end of class and are
also useful for quick feedback. Google Forms or assessment platforms (such as Edfinity.com) are good ways to collect this information. The information collected needs to be minimal, so it is easy to sort the answers into three main categories—didn’t get it, sort of got it, got it—so students needing help can get further instruction.

C. Quizzes: Easy to Grade; Provide Quick Feedback

Multiple-Choice Quizzes

Well-designed multiple-choice quizzes can be powerful mechanisms that provide quick feedback and can also probe student understanding. Teachers can measure different types of knowledge and understanding through multiple-choice questions: knowledge of terminology and syntax, code comprehension, conceptual understanding of how foundational constructs (like loops, conditionals, or variables) work, identifying and proposing a fix for a bug, and code comparison for accuracy and efficiency.

1. Multiple-choice questions that don’t measure deep understanding but check to see if students have learned the terminology or syntax you just introduced.

   (1) All Java applications must have a method _______.
   - public static Main(String args[])
   - public void main(String[] args)
   - public static main(String[] args)
   - public static Main(String[] args)
   - public static void main(String[] args)

   (2) Which one of the following is NOT a correct variable name?
   - myName
   - 2Bad
   - zero
   - theLastValueButOne
   - year2000

   (3) In the code block to the left, what is the term for what is being done in the first four instructions (the ones just before “Repeat 24”).?
   - Iteration
   - Looping
   - Conditional
   - Initialization
   - Nesting

2. Which one of the following is NOT a correct variable name?

   When the code on the left is executed, for which pair of inputs will the sprite say “Hello”?
   - num1 = 5, num2 = 8
   - num1 = 3, num2 = 9
   - num1 = 8, num2 = 8
   - num1 = 7, num2 = 6

3. The following multiple-choice question revealed misconceptions about students’ understanding of how variables work, and when they are (or are not) updated. In Grover’s research (2014), a majority of students answered C.

   What is the value of the variable steps after these two blocks are executed?
   - A. 0
   - B. 10
   - C. 20
4. Multiple-choice question to locate bug and debug code.

Raul wants to make a timer that will count down from 30 to 0. Raul has written the following code using a time variable:

(1) Will Raul’s code work as desired? Yes / No
(2) In Raul’s code, will the Repeat Until loop ever stop (i.e., will the “time=0” condition ever be satisfied)? Yes / No
(3) If you had to change just one thing to fix the bug, what would you change?
   - The Set time block
   - The Repeat Until “time=0” condition
   - The wait block
   - The change time by block
   - The stop all block

Innovative Autogradable Technology-Enhanced Assessments

Thanks to the affordances of technology, new types of assessment can also be autograded for quick feedback. Emergent assessment platforms such as Edfinity (edfinity.com) allow for a growing number of new assessment types beyond multiple choice or multiple answer or fixed response that make assessment more interesting but at the same time provide quick feedback to the teacher and learners. Such platforms also allow for explanations or detailed descriptions of solutions that help the learners understand why their response was incorrect (or even if it was correct) and allow for multiple attempts to a problem by the student.

1. Parsons Problems: Rearranging Given Blocks of Code

A Parsons problem is where code statements or algorithm steps is given to the pupil, and they have to rearrange it into the right order. These can be used early in the learning process when students are uncomfortable starting with a blank slate to write code. In Figure 1a, students are given a Parsons puzzle of an algorithm that helps them come to grips with assigning values to variables and interacting with multiple variables. Parsons puzzles are also popular in interactive textbooks involving text-based programming used in high school and postsecondary classrooms (Figure 1b).

Parsons problems can also be used within block-based programming languages, asking pupils to rearrange blocks of code to complete a project.

Figure 1 (a). Algorithmic Parsons problem for younger learners; b) Parsons problem in Python (source: Runestone Interactive)
2. Hotspot and Point-and-Click Items
Figures 2a and 2b show examples of problems that measure code comprehension through requiring students to click or select parts of code that answer the question.

![Figure 2a & b. Innovative assessments types: hotspot and point-and-click items that measure students code comprehension](image)

3. Unit-Test Coding Assessments
These assessment types require students to enter code snippets that are then tested and autograded (Figure 3). They also show students the correct solution.

![Figure 3. Unit-test coding example](image)
4. Quizzes With Open-Response Items
These are similar to multiple-choice quizzes but require an open response that may include explanations for their answer, as shown in Figure 4. Such items are time-consuming and subjective to score, even with a rubric.

![Figure 4. Example of an open-response item](image)

D. Programming Assignments and Projects
Programming assignments require actual programming, but they can come in various forms, as described in Table 1. They could be completely open ended or open projects that combine choice but with constraints, such as requiring the use of a specific construct (“Code a game in Scratch that uses one or more variables and a repeat until loop”). They could have an end goal that is specified but provide flexibility in terms of added features. Students could be given partially coded programs to complete or asked to debug buggy programs. See the fairy assessment in Alice (Figure 5) for examples of these two types. Next, we discuss ways to guide and assess student work in an open-ended project with constraints.

![Figure 5. Fairy assessment in Alice requiring code completion and bug fixing](image)

Scaffolds and Assessment Measures
This is an example that Powers and colleagues designed and used in New York City schools. The goal was to use everything students had been learning to create an integrated project about a story that they enjoyed reading during the past year. Students designed the next chapter of their story with a partner to infuse their
own creativity in this project. They planned on paper, chose sprites, created sequences using code blocks, and created code using events and parallelism. Students then shared their animations with teachers and parents during a project showcase. Students completed checklists (Table 2) and student reflection questions, received peer feedback (Table 3), and used a rubric to guide their performance (Table 4). Projects were shared to the Class Studio in Scratch for students to comment on and share with the community.

Table 2. Project phases and planning checklist for students

<table>
<thead>
<tr>
<th>Unit 5: ELA Integration With Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part 1: Decomposition</strong></td>
</tr>
<tr>
<td><strong>Day (Planning)</strong></td>
</tr>
<tr>
<td>- Pick a book</td>
</tr>
<tr>
<td>- Outline characters</td>
</tr>
<tr>
<td>- Choose setting</td>
</tr>
<tr>
<td>- Sequence of events</td>
</tr>
<tr>
<td><strong>Scene 1: Building</strong></td>
</tr>
<tr>
<td>- Choose main sprites</td>
</tr>
<tr>
<td>- Choose your first background</td>
</tr>
<tr>
<td>- Code sprites to start on your first background and first costume</td>
</tr>
<tr>
<td>- Code changing sprites costumes</td>
</tr>
<tr>
<td>- Add messaging (broadcasting and receiving messages, wait blocks)</td>
</tr>
<tr>
<td><strong>Scene 2: Building</strong></td>
</tr>
<tr>
<td>- Choose any new sprites you need in this scene</td>
</tr>
<tr>
<td>- Choose your second background</td>
</tr>
<tr>
<td>- Continue to add messaging (broadcasting and receiving messages, wait blocks)</td>
</tr>
<tr>
<td><strong>Self-Evaluation</strong></td>
</tr>
<tr>
<td>- Run your project to look for bugs!</td>
</tr>
<tr>
<td>- Use the hide/show blocks for any sprites that do not need to be in all scenes</td>
</tr>
<tr>
<td>- &quot;Clean up&quot; your code—trash unused blocks</td>
</tr>
<tr>
<td>- Debug as you work!</td>
</tr>
<tr>
<td><strong>Peer Feedback</strong></td>
</tr>
<tr>
<td>- Give 2 stars and a wish to at least one other project</td>
</tr>
<tr>
<td><strong>Revising/Editing</strong></td>
</tr>
<tr>
<td>- Use what your partner said to improve your project</td>
</tr>
<tr>
<td><strong>Catch-Up or Scene 3 (Spicy!)</strong></td>
</tr>
<tr>
<td>- Finish your project</td>
</tr>
<tr>
<td>- Add another scene</td>
</tr>
<tr>
<td><strong>Showcase</strong></td>
</tr>
<tr>
<td>- Present your project</td>
</tr>
<tr>
<td><strong>Self-Reflection</strong></td>
</tr>
<tr>
<td>- Respond to prompts about your project and CT</td>
</tr>
</tbody>
</table>

Questions for Student Reflection

1. What are you most **proud** of about **creating** your project?
2. What was the most challenging part of building your project? How did you **persevere** and overcome this challenge?
3. Did you have to **debug** any issues in your project? How did you spot the bug, and what did you do to fix it?
4. How was telling a story in **Scratch** similar to writing a story on paper? How was it different?
5. Did you like using **Scratch** to publish your next chapter or scene? Why or why not?
6. How were you able to use **peer feedback** to help you modify your project?

Which one of these thinking skills did you use the most in your project: **logic**, **evaluation**, **algorithms**, **patterns**, **decomposition**, **abstraction**? How?
Table 3. Peer Feedback Form

<table>
<thead>
<tr>
<th>Scene 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>❑ It is clear what my partner’s book is.</td>
</tr>
<tr>
<td>❑ There are two main sprites (characters).</td>
</tr>
<tr>
<td>❑ The backdrop matches the setting.</td>
</tr>
<tr>
<td>❑ There is dialogue between the characters.</td>
</tr>
<tr>
<td>❑ The code runs smoothly (no bugs).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scene 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>❑ The backdrop changes in Scene 2.</td>
</tr>
<tr>
<td>❑ The sprites (characters) continue their dialogue.</td>
</tr>
<tr>
<td>❑ The program switches backdrops and sprite costumes without bugs.</td>
</tr>
</tbody>
</table>

**Evaluation**

| ❑ The sequence of events makes sense. |
| ❑ There are no bugs when I run the program. |
| ❑ Hide/show blocks are used for sprites that don’t belong in the scenes. |
| ❑ The code is “cleaned up”— there are no unused blocks. |

Written Feedback: *Then you can add your feedback on their project page!

Two Stars

Wish

Table 4. Student performance rubric

<table>
<thead>
<tr>
<th>Project</th>
<th>4 Exceptional</th>
<th>3 Proficient</th>
<th>2 Developing</th>
<th>1 Beginning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprites (*ELA 3.3)</td>
<td>I used at least 3 sprites in my program and had multiple costume changes and detailed dialogue.</td>
<td>I used at least 2 sprites and had costume changes and dialogue.</td>
<td>I used at least 2 sprites and attempted costume changes and dialogue.</td>
<td>I am still working on adding sprites that change costumes and have dialogue.</td>
</tr>
<tr>
<td>Backdrops</td>
<td>I used more than 2 backdrops and coded them to change efficiently during the program.</td>
<td>I used at least 2 backdrops and coded them to change during the program.</td>
<td>I used 2 backdrops and attempted to code them to change during the program.</td>
<td>I have 1 or no backdrops and am working on coding them to change during my program.</td>
</tr>
<tr>
<td>Sequence of Events (*ELA 3.3)</td>
<td>I continued a story from third grade using Scratch. My story has a logical and creative sequence of events.</td>
<td>I continued a story from third grade using Scratch. My story has a logical sequence of events.</td>
<td>I attempted to continue a story from third grade and tried to create a logical sequence of events.</td>
<td>I am still working on creating a logical sequence of events based on a book I’ve read.</td>
</tr>
<tr>
<td>Time Management</td>
<td>I finished all of my project before the end of class time and used the extra time to make improvements.</td>
<td>I used project time well and met all deadlines.</td>
<td>Sometimes I was able to meet deadlines.</td>
<td>I need to find new ways to complete my tasks to meet deadlines.</td>
</tr>
<tr>
<td>Reflection</td>
<td>In my reflection and showcase, I clearly express my thoughts in different ways about the questions I am asked.</td>
<td>In my reflection and showcase, I express my thoughts about the questions I am asked.</td>
<td>In my reflection and showcase, I answer the questions I am asked.</td>
<td>With help, I answer the questions I am asked in my reflection and showcase.</td>
</tr>
</tbody>
</table>
E. Code Reviews and Project Showcases

Programming classrooms are collaborative spaces where students work individually or in pairs or small groups to create computational artifacts that are often driven by student choice and interest. Being able to “show and tell” or share their creations to the rest of the class is a great opportunity for teachers to get a sense of not only student engagement, motivation, and creativity but also how well students collaborated and how they communicate their programming efforts to other students. Such a code review is an opportunity for students to explain the goals of their project and explain to their peers how they accomplished their goal. Students should be encouraged to step into the code to show key parts of their programs and also share what bugs they encountered and what moves fixed the code. Such an activity can be guided by checklists and prompts (Tables 2 and 3; Figure 6) to encourage students to pay attention and learn from the experience.

![Figure 6. Prompts for making project showcases a productive learning experience](Image)

While code reviews are beneficial, they are not imperative for every programming project as they are time-consuming (especially compared to quizzes and entry/exit tickets). Although demos and code reviews could assume the form of one-on-one reviews with teachers, there are other options that Sedgwick has successfully used.

- **Video:** Students can record a video of their programming project that includes them showing and explaining their code or a portion of their code. Students could also create tutorials for other students.

- **Code Documentation:** Students can copy their code into Google Slides or other presentation software, and then document what it does (Figure 7).
Artifact-based interviews, first used by Brigid Barron and her colleagues in 2002, have been invaluable in eliciting student explanations by cueing student memory and perspectives using students’ own projects. Teachers can use ABIs to ask students to explain the goal of the project and how they got there or to justify their choices by asking questions, such as “How do you know?” “How did you decide?” and “Why do you believe that?” ABIs questions can also be part of the process of summative reflection on the entire class and experience. For example,

- How did you decide on your project?
- What does it do? How did you go about coding your project?
- What was it like working on this project? (Offer suggestions such as fun, challenging, difficult, creative, or boring.)
- Overall, how did you feel about this course?
- How does it feel to know more about computer science?
- Thinking back on the course, what did you enjoy the most? What was the hardest part?

Peer and Self-Assessment

Students in a programming classroom should be encouraged to provide feedback to each other and also assess each other’s as well as their own work. However, it is important that they be guided by well-designed rubrics to support them in the process of peer and self-assessment. Tables 2 and 3 are examples of these. Figure 8 is an example from Sedgwick’s work of a rubric that can be used of self or peer-assessment for a Scratch mathematics game programming project. Rubric elements can also be tied to standards such as CSTA standards for computer science.

F. Artifact-Based Interviews (ABIs)

Artifact-based interviews, first used by Brigid Barron and her colleagues in 2002, have been invaluable in eliciting student explanations by cueing student memory and perspectives using students’ own projects. Teachers can use ABIs to ask students to explain the goal of the project and they got there or to justify their choices by asking questions, such as “How do you know?” “How did you decide?” and “Why do you believe that?” ABIs questions can also be part of the process of summative reflection on the entire class and experience. For example,

- How did you decide on your project?
- What does it do? How did you go about coding your project?
- What was it like working on this project? (Offer suggestions such as fun, challenging, difficult, creative, or boring.)
- Overall, how did you feel about this course?
- How does it feel to know more about computer science?
- Thinking back on the course, what did you enjoy the most? What was the hardest part?
H. Reflection Journals in Project-Based Programming

Classrooms where learning of programming is centered on creating projects over several days or weeks (such as the one described in Chapter 8 on project-based constructionist learning), formative feedback tools include

- A checklist to understand a student’s status on a project - this allows teachers to note if progress has stalled.
- Checkpoints that allow the teacher to review projects at designated junctures.
- Design notebooks, where students record plans, document the progress of a project, and reflect on things that went well or problems that arose in making their projects. Getting students to record their progress and reflect in short (5-minute) daily entries at the end of the day (like an exit ticket) in response to prompts such as “What went well today?” “What problem(s) came up?” or “What is a tip or trick you would suggest to another student doing this project?” are useful.

Reflection questions around code are ipsative or “assessment as learning.” Students can also be prompted to add summative reflections about their final project. For example

- How is this project similar to or different from previous projects?
- What new code or tools did I add to this project that I hadn’t used before?
- How can I use what I learned in this class in future projects?
- What questions do I have about coding that I’d like to explore next time
READINGS AND RESEARCH

Ever since the Black and Wiliam’s (1998) landmark study on the benefits of formative assessment and feedback for learning, the emphasis on the integration of assessment and instruction has grown, with the goal of seamlessly combining teaching with an ongoing analysis of student progress toward instructional goals (Heritage, 2007). Bloom’s taxonomy (Figure 9) provides a useful framework to think about various forms of activities and assessments in a classroom and how to aim activities at various levels of skill and thinking. The SOLO taxonomy by Biggs and Collins has also been found to be useful in designing assessments for programming (Lister et al., 2006).

![Bloom's taxonomy](Source: The Derek Bok Center for Teaching & Learning, Harvard University)

Well-designed multiple-choice assessments can be used to further learners’ understanding, provide feedback, and keep the learners engaged in activities such as reading code that are as crucial a skill as writing code in building understanding of algorithmic thinking (Lister, Fidge, & Teague, 2009). Many of the multiple-choice assessment examples described in this chapter are drawn from Grover’s dissertation research (Grover, 2014). These assessments aimed to help learners develop familiarity with code tracing and the ability to understand an algorithm (Lopez, Whalley, Robbins, & Lister, 2008). Some formative assessments also involved Parsons problems (Parsons & Haden, 2006; Denny, Luxton-Reilly, & Simon, 2008) created in Scratch where students were presented jumbled blocks required for a program that they were required to snap together in the correct order. In addition, they were given closed- and open-ended projects in Scratch. The latter provided students authentic means of showcasing their learning through projects that were personally meaningful. Open-ended projects can also provide several insights—in the aggregate—to the teacher on the types of projects students tend to create and what basic and advanced constructs they are comfortable using (Grover, Basu, & Schank, 2018).

You can read about artifact-based interviews in Barron et al. (2002), Grover (2017), and O’Leary (2019). O’Leary (2019) also describes how to implement ipsative assessments, or assessments as learning.

Assessment platforms such as Edfinity (edfinity.com), which Grover is using in her current research, provide features for autograding assessments, adding explanations (in addition to the correct solution) that students could read after submitting their answer, and providing multiple attempts for an incorrect answer. These platforms also enable the designing of innovative assessments such as Parsons problems, which are becoming increasingly popular in introductory programming settings, thanks to research about their benefits (e.g., Ericson et al., 2017).
BIBLIOGRAPHY


