TESL-EJ Teaching English as a Second or Foreign Language
The Electronic Journal for English as a Second Language

August 2017 – Volume 21, Number 2

Coding and English Language Teaching

*** * On the Internet * * ***

Vance Stevens
Higher Colleges of Technology, CERT, KBZAC, Al Ain, UAE
<vancestev@gmail.com>

Jennifer Verschoor
St George´s College North, Buenos Aires, Argentina

Vance reflects on 20th century learning, as an early adopter and early adaptor

When I was studying my MA/ESL in Hawaii in the early 1980s and trying to get my head around how people learn languages, I was struck by something Earl Stevick said in one of his books, in a chapter on games in language learning,

The quality of the learning that takes place when we focus our attention only on the items to be learned is different from (and probably inferior to) the quality of the learning that is incidental to something else that we are trying to do (Stevick, 1982; pp.131-132).

In my career as an English teacher, I've always been on the lookout for tasks with incidental language learning outcomes. In my first job after earning my MA at the University of Hawaii, Manoa, I was responsible for planning and implementing a curriculum for teaching ESL to foreign students at Hawaii Preparatory Academy, a boarding school on the Big Island of Hawaii. There were only a few ESL students, mostly Japanese and Chinese, and they tended to keep to themselves and speak together in their language rather than mix easily with the predominantly native English speaking students at the academy. I was an early *adopter* of CALL for language learning, and tried to use computers with my students to the extent possible. This was in 1983, at the cusp of the age of ubiquitous personal computing, so most student computing at HPA was done at terminals connected to a mainframe computer in a single room where all the students went when they needed to compute or print assignments. When I gave my foreign students writing assignments to be done at those computers, I noticed that while they were there in the room with the other students, sitting randomly next to them at the available terminals, they would often have questions, not usually about language, but more often about how you got the mainframe to do this or that, format or print a document, for example. My ESL students mentioned to me that such moments were rare opportunities to interact naturally but meaningfully with the native speaking students there. I had sent them to the computer lab to write sentences and paragraphs, and they

had come away having had authentic communication experiences with other English speakers that hopefully led to more such opportunities in other contexts. This could only have improved their language development in ways that I could not have arranged for them if I had tried to develop assignments to focus on that.

I was also an early *adaptor* of CALL programming at HPA. I had succeeded in budgeting my program for a lab with a few Apple II computers but I was surprised to be told when the computers arrived that there were no further funds for software on top of my request to buy hardware. Personal computers had only recently been invented, I hadn't realized I'd need a separate budget for software as well, and it had not been obvious to anyone else at HPA at the time that such hardware was of little use without software.

Fortunately, in these early days of personal computing there was a budding community of hobbyists who were writing programs on Apple II for simple games and distributing them in the public domain. My Apple Club in Honolulu held swap meets occasionally where such programs were exchanged on five-and-a-quarter inch floppy disks, and I had a trove of those I had brought over with me to the Big Island. The programs were written in Basic and the code was visible and could easily be changed. A lot of the programs were word games or cloze programs where I could simply change the text in the program interface or items in the databases, while keeping intact the subroutines that did the actual work, to produce my first CALL software programs (Stevens, 1985). This was one of my first forays into coding.

There was a popular commercial game program at the time called *Mystery House* whose developers had decided to donate it to the public domain. Armando Baltra had been promoting use of this game for language learning in conference presentations and articles at about this time (e.g, Baltra, 1984). I acquired the public domain version for my lab and encouraged my students to play it. They communicated with the game in subject/verb utterances such as "go west" or "light match." The game used stick-figure graphics but somehow managed to give the students the impression they were exploring a house and meeting people there (but then the characters in the house started turning up dead, and the object of the game was to find the killer from among those remaining in the house before the killer found you). The students had explored almost everywhere they could until one of them issued a command, quite by accident, that caused a wall to move and revealed passages behind the walls. This stirred very real excitement in my classroom, which in turn led to communication with and among the students that took on greater urgency and motivation than any I had been able to elicit in any English class before then.

This was my first palpable awareness of the power computers have to create situations in which students have genuine needs to communicate on topics not teacher-directed, but teacher-facilitated. Since then I have been exploring the benefits of serendipity in language classes, and the importance of promoting chaos as a way of getting students to discover solutions to problems they were motivated to solve, and to communicate the processes involved in those solutions to others in the target language.

CALL and LACL

At a more recent job this century, where I was teaching computing at Petroleum Institute in Abu Dhabi, we had a curriculum that focused largely on Microsoft Office applications, but I managed to convince my colleagues to allow me to create some units teaching HTML, in which the students coded their own web pages. Since the students were non-native English speaking engineering students, I called what I was doing LACL, language-assisted computer learning, a play on the better-known acronym CALL (computer-assisted language learning). My colleagues tolerated my contributions to the program for a while but eventually over-ruled me since they didn't feel they or their students needed to know the code that underpinned most of the content on the Internet. Their attitude was that we all have browsers for that, and if you need to produce code, you can usually find an HTML editor to produce it for you. However, sites such as PBWorks, Wikispaces, or Blogger, have underlying HTML code which can be used to troubleshoot problems when the WYSIWYG interfaces don't work as expected, as sometimes happens. As Dudeney, Hockly, & Pegrum put it (2013, p.162):

> *Knowing how to recognise and change elements in coding languages allows us to escape the constraints of templates, and gives us greater control over the format and appearance of some online communication. A basic familiarity with the common coding language HTML (HyperText Markup Language), which underpins most webpages, including blogs, wikis, etc., is a good place to start.*

Having recourse to the underlying code, and knowing enough about coding to be able to tame a WYSIWYG editor, can be empowering, satisfying, and save a lot of time trying to tweak an interface that will not bend to your wishes until you get under the hood and see what wires are crossed there. Just wanting to know how to tweak code can pose seemingly wicked problems that networking within a community commonly helps resolve. Knowledge gaps are venerable language teaching ploys, and as with my students at HPA, in an environment where help is available from asking others, a post on a social media site can elicit help and start conversations. If this post is from a non-native speaker in a target language, but is answered by a native speaker or even another NNS who uses the target language as the lingua franca for the discussion, then authenticity of purpose and motivation to learn converge with availability of interlocutors, all catalysts for success in learning a foreign language.

We see this dynamic play out not only in the examples above but in gaming communities where NNS players achieve fluency in English while playing with others in games such as Minecraft, all the while speaking in English. They typically engage in a host of peripheral behaviors such as researching tips and 'cheats' on wikis and YouTube (in English), recording themselves communicating in English with others while playing the game, and in posting their own tips and how-to videos on YouTube (e.g., Smolčec, Smolčec, and Stevens, 2014). So we see that this combination of ingredients for success in language learning — authenticity of purpose and motivation to learn plus an availability of interlocutors — applies not only in the 20th century language learning I was facilitating in the mid-80's at HPA, but even more so this far into our present millennium.

21st century learning

A common catch phrase for this is 21st century learning. As Rich (2010) puts it, "the term '21st-century skills' is generally used to refer to certain core competencies such as collaboration, digital literacy, critical thinking, and problem-solving that advocates believe schools need to teach to help students thrive in today's world." In an era where schools realize they are training students to excel in jobs that haven't been invented yet, employers are looking for students who can learn on the job, and who have a skill-set that includes the three C's: creativity, communication, and collaboration. These skills are needed to enable employees to adapt to dynamically shifting work environments, communicate with others not only in the immediate environment but across networks, and in so doing absorb on-the-job training through collaboration with others. To these C's others, for example, Thoughtful Learning (2017) add a 4th C for critical thinking, as well as the literacy skills of information, media, and technology literacies. World Economic Forum (2015, p. 3) characterise this in an infographic giving a comprehensive breakdown of 21st Century Skills, as quoted in Table 1.

Table 1. 21st Century Skills

Foundational literacies *How students apply core skills to everyday tasks*	**Competencies** *How students approach complex challenges*	**Character qualities** *How students approach their changing environment*
Literacy	Critical thinking / problem solving	Curiosity
Numeracy	Creativity	Initiative
Scientific literacy	Communication	Persistence / grit
ICT literacy	Collaboration	Adaptability
Financial literacy		Leadership
Cultural and civic literacy		Social and cultural awareness

Lifelong learning

One driver of the attention to these so-called 21st century learning skills is what employers say they are looking for in job applicants nowadays. Scott (2015) conducted an insightful meta analysis into 21st century job skills sought by employers. Among the studies cited is one where Wagner (2010) and the Change Leadership Group at Harvard University conducted "several hundred interviews with business, nonprofit and education leaders" which led them to conclude that, from an employer's point of view:

> *students need seven survival skills to be prepared for twenty-first century life, work and citizenship:*
>
> • *Critical thinking and problem solving*

- *Collaboration and leadership*
- *Agility and adaptability*
- *Initiative and entrepreneurialism*
- *Effective oral and written communication*
- *Accessing and analysing information*
- *Curiosity and imagination [1]*

In her meta analysis Scott also cites a study by a consortium comprising the Conference Board, et al. (2006, p.9) who jointly surveyed over 400 employers across the USA and found that the "applied skills" mentioned above:

> *trump basic knowledge and skills, such as Reading Comprehension and Mathematics. In other words, while the "three Rs" are still fundamental to any new workforce entrant's ability to do the job, employers emphasize that applied skills like Teamwork/Collaboration and Critical Thinking are "very important" to success at work.*

The Conference Board report continues, "when asked to assess new workforce entrants, employers report that many of the new entrants lack skills essential to job success" (p.10). So it is apparent that teachers who promote skills that will leverage success in the world at large in the course of teaching their subject to their students are doing their students as well as themselves a valuable service.

Coding and 21st century learning

According to Dudeney, Hockly, and Pegrum (2013) coding is a deeper skill subsumed under the four main digital literacies of language, connections, information, and (re)design. Where should English teachers start if they are planning to teach this new kind of literacy? Let´s start by understanding the concept of coding.

Coders or programmers are people who write the programmes behind everything we see and do on a computer. Most of our students spend several hours playing online games, but few know how to create a game. Learning to code encourages students to become creators, not just consumers of the technology they use. Clare Sutcliffe, co-founder of Code Club (https://www.codeclub.org.uk/), a UK network of educators running coding clubs for students aged 9-13, was quoted in Morrison (2013) as saying that teaching coding:

> *has the potential to bring about a fundamental shift in the way we view technology, turning us from passive consumers into active producers. "There is a massive difference between consuming content and being able to create it," Sutcliffe adds. "It is important to have agency over the tools you are using."*

When children, or adults for that matter, learn to code, it helps them to develop essential skills such as problem solving, logic and critical thinking. Through coding, we realize that there's often more than one way to solve a problem, and that simpler and more efficient solutions are often better. Analysing and discussing the processes of critical thinking and problem solving can result in meaningful language practice.

Coding and English teaching

Though this paradigm applies across all aspects of the curriculum, the focus of this article is on how it applies to English teaching, and more specifically, on how English teaching can be done in conjunction with encouraging students to work with code, so that between the two, they inculcate aspects of the 21st century skills mentioned here while promoting language development. More than just an idea, this is an actual business model for Bucksmore Education in UK, which offers summer coding classes where students aged 13-16 can take 15 hours per week in English and another 7.5 in coding classes, and take a Raspberry Pi home with them at the end of the course, http://www.bucksmore.com/courses/computer-coding-summer-school/.

Scratch, a free object-oriented programming language developed at MIT, is a popular tool of choice for teachers wishing to integrate coding into their language learning lessons. Writing in *Slate*, Berdik (2015) quotes ScratchEd founder Karen Brennan, who "describes Scratch as simply another way to express learning and creativity. 'Instead of writing an essay or doing a PowerPoint presentation, for a class, you can use Scratch to create your own interactive media,' she said."

Joan Brown is an English teacher who uses Scratch with her students. In Brown (2016) she presents a rationale for, as her blog post title puts it, "Coding in English Class – Perfect Pair". In her words:

> *we rarely think of coding and English going hand in hand, yet as I was instructing the students, I was amazed at the similarities I found … So similar to writing in many ways. You can't start writing and have no direction.*

Coding, she believes, teaches the concept of planning in a way that might carry over into writing. She further points out that programmers have "their own grammar rules called syntax. That syntax determines whether the next programmer will be able to read the code as well as whether the code will run correctly. Such a wonderful analogy to actual grammar."

> *teaching the logic behind conditional statements is the art of writing a great comparative essay or a great reinforcement to so many subjective decisions about literature. The If-Then-Else block requires students to weigh two possible scenarios and never leave anything out. There are so many applications to using that flow chart concept in English class.*

Alan Cohen (2017) uses Scratch in ESL classes by getting students to code short conversations. Noting that Scratch is easy to learn, and that its drag and drop interface doesn't create spelling and syntax barriers in the code they create, Cohen has his students

> *code dialogue that cartoon-like characters display in speech bubbles …. As they discuss and plan their story, the teacher listens to the groups and corrects pronunciation where needed. The students then tell the class what their story is about and run their program. The class and teacher can correct any spelling or grammar mistakes. The coding exercises provide other benefits. They teach planning and logical thinking.*

Laura Bradley (2016) uses tutorials from Code.org (https://code.org/) in her English classes. She likes the fact that she doesn't have to be an expert in code and "the tutorials differentiate the experience for my students, some of whom have been learning to code on their own via Khan Academy, and others who have never heard of coding. Best of all? The online tutorials are all free," and remain online beyond the end of the MOOC sessions for which they were created.

Having taught her students the basics of coding in Scratch, Bradley reports on a follow-on activity from an assignment to write novels in English class. The follow-on was to:

> *create a computer game based on the novel you just wrote … Having just invested over a month into that novel, they knew their characters, plots, and conflicts inside and out. I hoped that the chance to create a game from that story would honor their writing and stretch some different brain muscles, while also giving them the basis for a richer game than they might create if it didn't come from a well-developed story.*

She continues:

> *There is so much about this project that mirrors the writing process: in addition to creating a story, they brainstormed and outlined their games, drafted them, tested them, found errors to fix, drafted some more, tested some more, revised some more. And eventually they will publish their games to an audience as big as the internet (via the Scratch site), where this creative gaming community can play, rate, and give them feedback. And if they didn't proofread their game carefully? It won't run.*

Bradley (2017) reflects on this project and concludes that "although coding may not seem to fit in an English class … coding is a language with its own vocabulary, and proofreading one's work is critical."

> *So why would I take three weeks out of my English curriculum to let my students learn the basics of computer coding? Why should teachers of science, history, or math do the same? … because coding:*
>
> - *builds problem-solving skills and logical thinking*
> - *opens new avenues to creativity*
> - *gives students a foundation for success in 21st century careers*
> - *reinforces our own curriculum through a different lens*
> - *helps students understand how their own technology works*
> - *opens their eyes to potential careers*

From a student's perspective, a native Farsi speaker with an MA in English language living in Finland takes a CLIL approach to learning coding and English at the same time. Without mentioning CLIL, Omid (2014) argues that his best way to learn English is to learn it in conjunction with something else, such as programming languages. Omid points out that if you want to learn coding, then doing it in the context of ESOL is a good way to leverage the benefits of both subjects. He suggests starting with Codeacademy, https://www.codecademy.com/. In his words:

> Codecademy *courses are good for improving your reading and comprehension skills because, basically, you read and follow instructions, which are all written in English. You can also practice your writing skills by joining the discussions in the forums. Also, subscribe to their free email newsletter where you will read about people who have done something notable using what they have learned from* Codecademy *courses.*

This echoes what Smolčec, Smolčec, and Stevens (2014) discovered, as mentioned above with respect to Minecraft, when the other thing you are trying to learn has a wealth of materials and tutorials written about it in English.

Two of that article's co-authors, Marijana and (her son) Filip Smolčec, have been participating for the past three years in EVO Minecraft MOOC, an Electronic Village Online session meant to engage educators in Minecraft so that they can in turn use it with their students (Stevens, 2017). One of the moderators of that session, Mircea Patrascu, has been showing his EVO Minecraft MOOC participant peers how to build in Minecraft with script, while in Romania, he teaches children to code, and has written instructions for connecting Scratch to Minecraft and ScriptCraft (Petrascu, 2016). As Missio (2015) points out, Minecraft "is also now part of Hour of Code, a worldwide initiative that aims to teach kids computer programming" (https://code.org/minecraft).

Bringing teachers up to speed on coding

Morrison (2013) points out that "if coding is to become embedded in schools it is going to take a massive effort in terms of teacher-training." Morrison also interviews Code Club teacher Laura Kirsop, who, noting lack of such training in her own education, says "there is a long way to go before teachers feel confident enough to teach these skills."

Claire Siskin has been promoting coding for teachers of English to speakers of other languages for a long time. Her web page at http://www.edvista.com/claire/rev/ compiles dozens of links where teachers can get resources on LiveCode, a scripting language similar to Hypercard on Mac, but which runs on iOS, Android, Windows, Mac, Linux, Server & HTML5. Recently she facilitated a project whereby English teachers at Daffodil University in Bangladesh used LiveCode to create an app for smartphone called BrimmEng, "designed to provide English language practice outside the classroom and reinforce learning in the classroom," (Siskin, 2107). Her project demonstrates the willingness and ability of English teachers in developing nations to learn a coding language from the start and use it to create their own apps for language learning (Stevens, 2016).

In September 2015 New York city mayor Bill de Blasio announced a program for providing every public school student in New York City with computer science courses at every grade level by 2025, and allocated $80 million in funds to providing 5,000 teachers with computer science training in order to bring that many existing staff up to speed with the initiative. In an article in the *New Yorker* entitled "Can an English Teacher Learn to Code?" Morais (2015) introduces us to Meredith Towne, a theater and English teacher, and beneficiary of this program.

> *"The language in Scratch is very similar to theatre language," she said. … She devised an assignment in which students use Scratch to direct staging—that is,*

program their fellow-actors…. Her ninth graders have been tinkering with Scratch for a couple of months now. "They're already better at it than me," she said.

The observations above address a major concern of humanities teachers over whether they can learn coding themselves well enough to use it as a basis for class projects. Karen Brennan's ScratchED is a community of learners and teachers who help each other overcome such hurdles. She has studied the strategies of a subset of students who work on their own more than they rely on support from the community to debug their programs. She uses that knowledge to help teachers with "getting unstuck," the term Brennan (2014) uses in her talk on the HarvardEducation YouTube channel, where she assures teachers that "students don't need you in the way you think they need you. They don't need you to solve every problem." Instead teachers should "embrace the vulnerability of not knowing" and let students understand the value of learning in collaboration with the teacher.

Where can you and your students learn coding?

In the context of ELT there are several ways to start integrating coding as a new kind of literacy. Most coding websites are easy to follow and they provide clear tutorials on how to get started, so neither students nor English language teachers need to have previous knowledge of coding.

Let´s begin by exploring the apps, programs and websites to bring coding into the classroom mentioned by the English teachers whose voices emerge from earlier in this article. These teachers have converged on the following Web projects in particular: Code.org, Hour of Code, Code Club, Scratch, ScratchEd, and Codeacademy.

Code.org and Hour of Code

Getting started with https://code.org is very simple for teachers. The main objective of this website is to teach coding using blocks in a simple and entertaining way.

Teachers will find a variety of coding activities divided into levels ranging from K-5 through 6-12 and University+. The site is divided into the categories (1) Students, (2) Educators, (3) Hour of Code, and (4) Get involved. A click on "Students / Explore our courses" takes you to this page: https://studio.code.org/courses.

Here you will find a full course catalog divided into the three audience levels (K-5, 6-12, and university), plus the Hour of Code option designed to help students (or anyone, try it yourself!) start learning how to code, or improve their coding skills, in only one-hour chunks. Here, we are introduced to the fundamentals of coding in a set of over a hundred tutorials and activities created by computer science specialists. English teachers should be able to realize the potential in working with their students, not only in using the language in the modules themselves, but in talking in class around the content of these tutorials.

As given at https://code.org/hourofcode/overview, these tutorials are:

• *Minecraft – Program animals and other Minecraft creatures in your own version of Minecraft.*

• Star Wars – Learn to program droids, and create your own Star Wars game in a galaxy far, far away.

• Frozen – Use code to join Anna and Elsa as they explore the magic and beauty of ice.

• Sports – Make a basketball game or mix and match across sports.

• Flappy Code – Wanna write your own game in less than 10 minutes? Try our Flappy Code tutorial!

• Classic Maze – Try the basics of computer science. Millions have given it a shot.

• Infinity Play Lab – Use Play Lab to create a story or game starring Disney Infinity characters.

• Play Lab – Create a story or make a game with Play Lab!

• Artist – Draw cool pictures and designs with the Artist!

• Text Compression – In this lesson, students will use the Text Compression Widget to compress text by substituting patterns with symbols.

• Conditionals – Learn about algorithms and conditional statements in this "unplugged" activity using a deck of cards.

Most of these modules are broken down into 10 or 20 puzzles. Students are not writing down code. They are using Blockly as a visual programming language, to tell the computer in plain English what to do. Students must drag and drop the Blockly directions to make a game. The directions with Blockly look like this:

```
"move forward"
"turn left"
"turn right"
```

Javascript is generated under these blocks. Translated into code, it looks something like:

```
moveForward();
turnLeft();
turnRight();
```

ESL teachers should review with students the specific vocabulary of each game before working with Code.org; for example, "flap", "when clicked", "play wing sound", etc. A simple way to get started is to teach the Blockly directions needed.

Teachers can explore more tutorials at https://code.org/learn or even explore the possibility of introducing robotics at https://code.org/learn/robotics.

Code Club

If you teach children 9 to 13, you can set them up in their own Code Club. Code Club was designed to offer weekly extra-curricular coding specifically for this age group. No prior coding knowledge is needed to start or to join the clubs. Students can learn how to create games, websites and animations using Scratch, HTML and Python.

To create a Code Club go to https://www.codeclub.org.uk, or if you live outside the UK go to https://www.codeclubworld.org. Once you activate your account you will find

step-by-step instructions on how to create a Code Club, and have access to resources, materials and guidance to run your Code Club.

If your students do not fall into the correct age brackets, you can still explore this site, and the others mentioned here, and adapt the concept of teaching human languages through coding in a club setting of your own devising, designed to suit your context.

Scratch and ScratchEd

Coding impacts every career in the 21st-century, and Scratch (https://scratch.mit.edu/) is one the best websites for beginners to learn the basics of coding. Scratch was created by the MIT Media Lab as a free programming language designed to program websites, games and animations. It's good for a range of classroom activities because it's easy to use, it runs in a browser, it's drag-and-drop, and does not require students to provide correct syntax. Thus it detracts negligibly from what Stevick called, "the learning that is incidental to something else that we are trying to do" (1982, p. 132).

According to the Scratch website, "Scratch is designed especially for ages 8 to 16, but is used by people of all ages. Millions of people are creating Scratch projects in a wide variety of settings, including homes, schools, museums, libraries, and community centers" (so it's already being used in many careers in the 21st century). And there is a version called Scratch Jr. especially created for ages 5-7 enabling students to create their own interactive stories and games. Teachers can download the guide at https://www.scratchjr.org/learn/interface/ where they will find clear instructions on how to use Scratch Jr.

Teachers can join the ScratchEd community of practice at http://scratched.gse.harvard.edu/ to get help and share projects with teachers worldwide. Joining the community is free and teachers will encounter engaging ideas introducing how Scratch has been used by educators in different schools.

Teachers can also find lesson plans using Scratch at the Code.org website, https://code.org/educate/curriculum/teacher-led/. The lessons are grouped into elementary school, middle school, and high school projects, but the topics might apply to all levels, and even seed fun projects for adults. High school projects listed here range from a variety of STEM activities (e.g., "Code and animate a Solar System simulation, an interactive ecological pyramid, a working analog clock"), through various oral history and digital storytelling projects, to this one: "Your class will be be creating a 'history of computers' web page/Scratch project/video that we can share with the world. To make this web page, you and your partner will do research and write about one important event or person in computer history." Teachers well-practiced in the art of adapting lesson plans will know that any kind of history, or almost anything else, can be substituted for "computer history". Teachers of English to students of all ages should be able to find activities here suitable to their contexts.

Codeacademy

Codeacademy is a website that provides a variety of coding languages. Teachers can find simple lesson plans and class sequences ready to implement for all levels. However, of all the sites mentioned here Codeacademy might be one that teachers of English should

use with an awareness of what it does and doesn't do. There is a balanced review of it at Kite (2017), who notes that whereas this is a good starting point for learning rudimentary code for free, it has "limited depth" and it's directed at adult learners (e.g., text-based descriptions, no videos to engage digital natives). Two advantages of Codeacademy are that the coding is taught inside a browser (avoiding your having to install the language on your computer) and that it teaches APIs for hooking into apps such as YouTube and Twitter (note the digital storytelling possibilities there), but Scratch works inherently inside a browser and you might not want to go as far as working with APIs if your subject is English.

Whereas the CLIL approach to learning English while learning programming seems to have worked with Omid (2014), according to Hughes (2015), "The reason why Codecademy is successful is because it takes coding, and transforms it into addictive bite-sized pieces that are easy to accomplish, and offer instantaneous feedback. It's the candy of coding." Whereas they have introduced many to the fundamentals of programming and have helped launch countless careers, "their product – and to be more precise, their teaching methods – leave a lot to be desired, and are leaving thousands frustrated, and unsure of where to progress with their formative development skills." Furthermore, these exercises do not sufficiently recapitulate previously learned items ("blink, and you'll miss it," as he puts it), nor do they help users apply what they learn to practical projects. In particular, Hughes faults them on producing exercises in chunks that do not teach the mindset of programming. They teach syntax, but not the art of programming, and it's the art, the inculcation of a way of thinking about approaching, troubleshooting, and solving problems, that most concerns 21st century educators.

Although a Google search trawls up a surprising number of similar perspectives on Codeacademy, there are also those who have found it to be a useful experience; Omid (2014) for one, and also Rushkoff (2012) who says:

> *To build my own code literacy, I decided to take free classes through the online website Codecademy.com, and ended up liking it so much that I'm now working with them to provide free courses for kids to learn to code. The lessons I've learned along the way are of value to parents and teachers looking to grow more code literate young people.*

When Rushkoff says he is now "working with" Codeacademy, he means he has become a "resident Digital Literacy Advocate or 'Code Evangelist' at Codecademy" (https://www.edutopia.org/user/202975) but there are many proponents of Codeacademy who extol its virtues because it's free and it teaches coding. It's probably worth looking into, if only to get a better grounding in coding as a teacher, as you work through the other sites with your students.

Conclusion

Rushkoff (2012) makes some compelling points in his article aptly entitled "Code literacy: A 21st century requirement." Using the example of Facebook, he notes that whereas we (over 2 billion of us, including teachers and students) use the site superficially to share the minutiae of our lives with friends, the deeper purpose of Facebook is to sell data points on those minutiae to high bidders, and that an

understanding of the deeper mechanics of how our digital world works is a critical literacy skill for the 21st century. Learning to code, he says, is one way that we can regain some control over our world. Program or be programmed.

Learning human languages is similarly empowering, as is self-evident in the number of people willing to employ teachers to help them learn them. Learning coding in the context of language development should be doubly empowering, especially as development of either, even on its own, or ideally both, enables critical thinking skills and leverages opportunities in an uncertain future. As Rushkin says, "Code literate kids stop accepting the applications and websites they use at face value, and begin to engage critically and purposefully with them instead. Otherwise, they may as well be at the circus or a magic show."

Sometimes we as teachers look out at our students peering at their hand-held devices and wonder if they are using them productively or distractingly. Often, it's apparent that the latter is the case. Engaging our students is our key to success, for then we can discourse with them and have them describe their adventures on their learning journeys, and how these relate to their present (perhaps the present class) and long-term goals. But we ourselves must be worthy of our side of the discourse, which means we have to keep notching our own skills upward in order to bootstrap our students, or to appreciate when they bootstrap us, which can be the case when teachers learn to leverage into their teaching style "the vulnerability of not knowing" Brennan (2014).

We are all now temporally and firmly ensconced in the 21st century. We need to be focused now on moving ourselves and our students toward the 22nd. It is incumbent on us all to be coming to grips with the literacies and skill sets that will improve likelihood of success and even survival as we prepare to adapt to the world unseen not far around the corner.

Note

[1] The quote above is taken from Scott, 2015, p.3, but the URL given in her references is incorrect. The correct URL is given under Wagner (2010) in the Reference section below. Nearly identical points made by Tony Wagner were recorded in American Youth Policy Forum (2010) and are reiterated on Wagner's website at http://www.tonywagner.com/7-survival-skills/).

References

American Youth Policy Forum. (2010). Preparing students for the rapidly changing world: Implications for instruction and assessment (Forum 2). *American Youth Policy Forum*. Available: http://www.aypf.org/resources/preparing-students-for-the-rapidly-changing-world-implications-for-instruction-and-assessment-forum-2/.

Baltra, A. (1984). An EFL classroom in a Mystery House. *TESOL Newsletter 18* (6): 15.

Berdik, C. (2015). Reading, writing, 'rithmetic, 'rogramming: Should every school class be a computer coding class? *Slate* http://www.slate.com/articles/technology/future_tense/2015/04/building_coding_into_art_english_and_history_classes.html.

Bradley, L. (2016). Coding in English class? YES! and not just an #hourofcode, but a #monthofcode! *Laura Bradley*. Available: https://laurabradley.me/2016/12/15/coding-in-english-class-yes-and-not-just-an-hourofcode-but-a-monthofcode/.

Bradley. L. (2017). Coding in English class? Yes! And in your class, too! *KQED Education*. Available: https://ww2.kqed.org/education/2017/06/22/coding-in-english-class-yes-and-in-your-class-too/.

Brennan, K. (2014). Getting unstuck: Karen Brennan's '8 for 8'. *HarvardEducation* YouTube channel. Available: https://youtu.be/c_AdWB1GkRw.

Brown, J. (2016). Coding in English class – Perfect pair. *Current and Cool*. http://currentandcool.blogspot.ae/2016/01/coding-in-english-class-perfect-pair.html.

Cohen, A. (2017). Teaching ESL through coding. *LinkedIn Pulse*. https://www.linkedin.com/pulse/teaching-esl-through-coding-alan-cohen.

Conference Board, Corporate Voices for Working Families, the Partnership for 21st Century Skills and the Society for Human Resource Management. (2006). Are they really ready to work? Employers' perspectives on the basic knowledge and applied skills of new entrants to the 21st century U.S. workforce. *P21 Partnership for 21st Century Learning*. Available: http://www.p21.org/storage/documents/FINAL_REPORT_PDF09-29-06.pdf.

Dudeney, G., Hockly, N., & Pegrum, M. (2013). *Digital literacies*. Harlow: Pearson (electronic).

Hughes, M. (2015). 4 reasons why you shouldn't learn to code from Codecademy. *MUD*. Available: http://www.makeuseof.com/tag/4-reasons-shouldnt-learn-code-codeacademy/.

Kite, C. et al. (2017). Codeacademy review. *CodeConquest*. Available: http://www.codeconquest.com/reviews/codecademy/.

Missio, E. (2015). What kids learn when they play Minecraft. *Parents*. Available: http://www.cbc.ca/parents/learning/view/what-kids-learn-when-they-play-minecraft.

Morais, B. (2015). Can an English teacher learn to code? *New Yorker*. http://www.newyorker.com/tech/elements/can-an-english-teacher-learn-to-code.

Morrison, N. (2013). Teach kids how to code and you give them a skill for life. *Forbes*. https://www.forbes.com/sites/nickmorrison/2013/12/27/teach-kids-how-to-code-and-you-give-them-a-skill-for-life/#768b91d265d2.

Omid. (2014). Learning English by learning programming. *Italki*. https://www.italki.com/article/281/learning-english-by-learning-programming.

Patrascu, M. (2016). Using Scratch with Minecraft & Scriptcraft – Step by step instructions. *Kids love to code*. Available: https://kidslovetocode.wordpress.com/2016/09/27/using-scratch-with-minecraft-scriptcraft-step-by-step-instructions/.

Rich, E. (2010). How do you define 21st-century learning? One question. Eleven answers. *EdWeek*. Available: http://www.edweek.org/tsb/articles/2010/10/12/01panel.h04.html.

Rushkoff, D. (2012). Code literacy: A 21st century requirement. *Edutopia*. Available: https://www.edutopia.org/blog/code-literacy-21st-century-requirement-douglas-rushkoff.

Scott, C. (2015). The futures of learning 2: What kind of learning for the 21st century? *UNESCO Education research and foresight working papers*. Available: http://unesdoc.unesco.org/images/0024/002429/242996E.pdf.

Siskin, C. (2017). BrimmEng. *Edvista*. Available: http://www.edvista.com/claire/apps/brimmeng.html.

Smolčec, M., Smolčec, F. & Stevens, V. (2014). Using Minecraft for learning English. *TESL-EJ, 18*(2),1-15. Available: http://www.tesl-ej.org/pdf/ej70/int.pdf.

Stevens, V. (1985). You'd be surprised at how much public domain software you can adapt to ESL and language learning. *TESL Reporter 18*, 1:8-15.

Stevens, V. (2016). BrimEng: Carry English in your pocket! LiveCode app development from Bangladesh. *Learning2gether*. Available: https://learning2gether.net/2016/06/26/carry-english-in-your-pocket-brimeng-livecode-app-development-from-bangladesh/.

Stevens, V. (2017). Gamifying teacher professional development through Minecraft MOOC. In Zoghbor, W., Coombe, C., Al Alami, S. & Abu-Rmaileh, S. (Eds.). *Language culture communication: Transformations in intercultural contexts. The proceedings of the 22nd TESOL Arabia Conference* (pp. 75-92). *Dubai: TESOL Arabia*.

Stevick, E. (1982). *Teaching and learning languages*. New York: Cambridge University Press.

Thoughtful Learning team of teachers, writers, and designers. (2017). What are 21st century skills? *Thoughtful Learning*. Available: https://k12.thoughtfullearning.com/FAQ/what-are-21st-century-skills.

Wagner, T. 2010. Overcoming the global achievement gap (slides). *American Youth Policy Forum*. Available: http://www.aypf.org/documents/Wagner%20Slides%20-%20global%20achievement%20gap%20brief%205-10.pdf

World Economic Forum in collaboration with the Boston Consulting Group. (2015). New vision for education: Unlocking the potential of technology. *World Economic Forum*. Available: http://www3.weforum.org/docs/WEFUSA_NewVisionforEducation_Report2015.pdf.