Teaching Resources for Young Programmers: the use of Patterns

Kashif Amanullah
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
kashif.amanullah@pg.canterbury.ac.nz

Tim Bell
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
tim.bell@canterbury.ac.nz

***Abstract*—This Full Paper in the Research Category identifies and evaluates teaching resources suitable for teaching younger students using the Scratch and Python programming languages. Choosing suitable resources to introduce programming to children is a balance between making sure they are appropriate to their skills such as literacy and numeracy, and are motivating in their social context. Resources need to strike a balance between allow early success, but also introduce genuine programming skills, so that students can progress their programming skills rather than repeating simple tasks over and over. An important element is the choice of teaching resources used to support students' learning. We propose using elementary programming patterns as a measure of how comprehensive a teaching resource for programming is. Note that this doesn't mean advocating that students should be pressed to learn advanced patterns quickly, but it does provide a measure of how deeply a particular resource covers general programming concepts. We identify a set of patterns that are relevant to basic programming practice, and have analyzed some recommended online teaching resources and a small sample of introductory programming books for Scratch and Python against this set of patterns. The use of given set of patterns was relatively low, but some resources did introduce a range of patterns, and some new patterns emerged from the analysis that seemed to be used frequently in Scratch.**

***Index Terms*—programming patterns; Scratch; primary school students**

I. Introduction and Background

A key element that makes particular introductory programming languages attractive is the availability of good teaching resources for the language. As teachers use these resources, it would be natural that the patterns that students use are based on those they have seen in the resources (patterns are combinations of commands that occur commonly in a variety of programs [1]). If a teaching resource focuses on giving a quick introduction to students, they are likely to miss out on the depth of learning where they would encounter patterns commonly used in programming. This is not necessarily a bad thing - a short and positive experience with "coding" may inspire students to go further, and be more beneficial than a detailed and uninspired lesson that is too advanced for their capability and interest, so we don't argue that beginning students necessarily need to learn a range of patterns. However, what we propose is the use of patterns as a benchmark to evaluate the depth of resources for teaching programming — how suitable the resource is to carry students beyond an initial positive experience, and expose them to ideas that are needed in more general purpose programming contexts, particularly in the use of control structures. Resnick et al. [2] point out that introductory languages aim for a low floor (minimal barriers to starting), wide walls (useful for a range of projects) and high ceiling (supports advanced programming), but for beginners the focus is more on the first two objectives, noting that in the Scratch language they "plan to keep our primary focus on lowering the floor and widening the walls, not raising the ceiling. For some Scratchers, especially those who want to pursue a career in programming or computer science, it is important to move on to other languages." Nevertheless, Scratch and many other block-based languages are Turing-complete, so in principle could be used to implement sophisticated computations.

TABLE I: List of elementary patterns considered

Selection Patterns	Loop Patterns
Whether or Not	Process All Items
Alternative Action	Linear Search
Unrelated Choice	Guarded Linear Search
Partial Dependence	Loop and a Half
Conditional Expression	Polling Loop
Range of Possibility	Extreme Values
Sequential Choice	
Return Not Else	

Thus, a resource to support beginning programmers may not be intended to be comprehensive, and need not necessarily use all the basic patterns we discuss here, but the extent that patterns are covered gives us one measure of how comprehensive a resource is, and how appropriate it is for achieving the learning outcomes desired.

We focus here on "elementary patterns" (Table I), which are patterns that are relevant to novices who need to learn basic but fundamental programming skills [3]–[5]. We analyse a selection of published resources that are used for teaching programming to younger students (typically K-8), to establish which patterns are used in practice for education, and to identify where improvements could be made. Scratch and Python are two of the most widely used introductory programming languages in schools around the world, and are used here as representative languages for each paradigm i.e. block-based and text-based.

Several researchers have advocated the use of patterns for teaching programming. Hundley [6] points out that experience plays a pivotal role in becoming a good programmer, which comes from solving many problems over a number of years. This experience helps to build patterns which can be applied as a working solution to future problems belonging to relevant patterns. Clancy and Linn [3] argue that learning patterns greatly improves the ability to learn programming. Incorporating the use of patterns into teaching can prove to be an effective way to enable students to identify patterns to solve problems, and also build new patterns for new problems. They found evidence that patterns having promising implications for pedagogical practices, and suggested a better integration of use of patterns in course design and instruction through various means to help students better understand the concept of patterns. De Aquino and Ferreira [7] used patterns-based and games-based pedagogical approaches to teach programming to first year secondary school students (age 13-15). The patterns that they used were most of those that we have selected below, and the concept of patterns was explicitly taught to students. They reported improved performance and increased participation with the use of patterns. Cardell [8] evaluated the ability of students to understand and apply patterns in Java code by using an automated tool to identify patterns and anti-patterns. The results showed that many students were able to understand the concept of patterns, but were not able to apply it to problems other than specific examples provided. These studies support the concern that without appropriate teaching, students may see languages like Scratch as an application with a number of features which can be gradually explored (e.g. choosing sprites, and various forms of output), leading to an approach that one teacher we work with referred to as a "Spritefest," where students explore aspects of the *development environment*, but not necessarily many concepts of *programming*. Patterns take the learner beyond the idea of learning all the "features" of a language or environment, since patterns apply language elements in combination. This helps us to use the language as a means to teach programming, rather than just being another tool to learn.

The goal of the work reported here is to articulate a view of learning programming that uses the full power of computer programming (particularly variables, selection and iteration). The patterns that we have chosen as a benchmark for teaching novices are drawn from the work of Astrachan and Wallingford [4] and Bergin [5], and are listed in Table I.

Not all patterns listed in Table I, and Table II are supported by both Scratch and Python. For example, Scratch doesn't support functions, `switch` statements, `elseif` statements, and ternary operators, and therefore it is not possible to use the Return Not Else, Range of Possibility, Sequential Choice, and Conditional Expression patterns in Scratch. (We note that Snap!, which is related to Scratch, does support functions, and *can* support the Return Not Else pattern). Python doesn't provide a `switch` statement or `broadcast` as a built-in structure, therefore the use of `Range of Possibility`, `Search (broadcast)`, `Linear Search (broadcast)`, and `Guarded Linear Search (broadcast)` patterns can be ruled out in Python. We could argue that we could achieve the similar functionality of unavailable patterns using other available constructs (the `Sequential Choice` pattern achieves a very similar outcome as `Range of Possibility`), but we are not concerned with that at the moment since learning to shoehorn a pattern into a language may be unnecessarily demanding for beginning students, so we focus on the classic use of these elements in the provided format.

In choosing relevant patterns, we compared the same programs written in both Python and Scratch based on the patterns. This comparison showed how these elementary patterns are supported in both Scratch and Python, but also that because different constructs are available (particularly for controlled loops), it highlighted that the patterns sometimes need slightly different logic to implement them. For example, the `repeat until` block in Scratch uses the opposite condition to Python's `while` loop, since the use of the Boolean value is inverted in the two languages. Another difference between the languages is that the `Linear Search` pattern uses the `break` command in Python, but in Scratch the nearest equivalent is using a `stop` block. There are different views on whether one should even use a `break` command, but the key is that both are forms of a pattern that loops until a condition is met. The `Extreme Values`, `Whether or Not`, and `Alternative Action` patterns are fairly similar in both Scratch and Python, except constructing complex conditions and showing long messages involving variable values could prove to be cumbersome in Scratch due to the required use of multiple blocks with multiple join operators. Future work could distinguish the pattern best suited to solve a problem from the pattern in the implementation, since the latter is more language dependent.

Of course, many programmers use these basic patterns without being explicitly aware of them, and they are part and parcel of any programming solution to real world problems. We have taken the approach of identifying them explicitly, at least so that a teacher can measure coverage of the topic of programming. And if we incorporate the use of these patterns in our teaching resources, we will be exposing students to a wide range of commonly used programming techniques.

There is no fixed set of rules for choosing the teaching resources and materials for a programming course or most of the courses for that matter. Some people may be required to use the recommended material provided by their respective organisations or districts, other use recommendation provided online, or end up using whatever they stumble upon first or find comfortable to use. This could mean that in many cases not much thought is given to the teaching resource used, even though it plays a major role in overall learning. Considering the above points, it is not surprising that not much data is available about which teaching resources are being used around the world. There are many studies which show what kind of tools (Scratch, Blockly, python, etc.) are being used,

TABLE II: Summary of elementary patterns and their signatures

Pattern	Signature
`Whether or Not`	`if`
`Alternative Action`	`if-else`
`Unrelated Choice`	`if if ...`
`Partial Dependence`	`if/else` → `if/else`
`Process All Items`	`repeat/forever [list]`
`Search`	`repeat/forever [variable]` → `if/else` → `stop`
`Search (broadcast)`	`repeat/forever [variable]` → `if/else` → `broadcast`
`Linear Search`	`repeat/forever [list]` → `if` → `stop`
`Linear Search (no stop)`	`repeat/forever [list]` → `if`
`Linear Search (broadcast)`	`repeat/forever [list]` → `if` → `broadcast`
`Guarded Linear Search`	`repeat/forever [list]` → `if-else` → `stop`
`Guarded Linear Search (no stop)`	`repeat/forever [list]` → `if-else`
`Guarded Linear Search (broadcast)`	`repeat/forever [list]` → `broadcast`
`Loop and a Half`	`ask repeat-until [list]` → `ask` OR `forever [list]` → `ask` → `if` → `stop`
`Polling Loop`	`ask repeat-until [variable]` → `ask` OR `forever [variable]` → `ask` → `if` → `stop`

but not many show the exact resources (books, tutorials, online curriculum, etc.) that such courses use to teach these tools or programming languages [9]–[13]. Therefore, this study looks at some of the popular resources recommended across different platforms and evaluates their coverage of elementary patterns. Previous studies [14]–[16] have supported the potential of elementary patterns to be an effective approach for teaching, so if teaching resources use more patterns in the content it is expected generate improved results in learning.

II. Evaluating Online Resources using Patterns

Table II provides a summary of elementary patterns and their "signatures" that will be used as a criteria to assess the comprehensiveness of the teaching materials. This list is a refined form of the patterns given in Table I and excludes the ones not supported in Scratch. The arrows in the signature of each pattern mean that the element to the right of arrow is inside the preceding element. An example of a `Search` pattern can be seen in Figure 1, where there is a `stop` block inside an `if` block which itself, along with other blocks, is contained in a `forever` block. There is also "[variable]" in the signature of the Search pattern which means that the continuation of the loop depends on the value of a variable. We have developed "Scratch Analyzer," which takes Scratch code in JSON format (usually downloaded from Scratch "studios" in their online sharing community) and runs relevant analyses to find code matching the signatures in Table II.

Scratch's official website[1] has some useful resources for educators to get their students started, and we begin by analysing these, since they are likely to be a first port of call for educators new to Scratch. Two of these resources are CS First[2] and Code Club[3]. CS First teaches programming through of a number of activities, which are divided into different categories based on the amount of time required to complete them (one hour vs. multiple days). For this study we look at two activities under the advanced section to get a feel for what kind of concepts these activities entail. We wanted to measure how much these activities touch upon any patterns, since patterns are an important part of problem solving for programming, and these are the advance level activities in the resources. Similarly, the Scratch course in Code Club is divided into different modules and we analyse the most advanced module, module 3.

It is possible to create either a student or teacher account in CS First, and each activity in the curriculum came with a number of supporting materials for both teachers and students. We created a teacher's account, which enabled us to access the solution sheets, along with other resources (getting started guide, lesson plan, etc.). The examination of the solution sheets[4] of the first advanced activity (it was divided into eight sub activities) in the CS First Curriculum helped us to find the coverage of programming concepts used in the curriculum. There was no use of lists in this activity, so all the elementary patterns based on a list could be ruled out. Since `Whether or Not` simply checks for the use of `if`, it is no surprise that this pattern was common. `Alternative Action` was only used once in one sprite. The `Unrelated Choice` pattern was used few times in more than one sprite. The `Partial Dependence` pattern didn't occur. Out of the more sophisticated patterns, `Search`, was the only pattern that was used once (Figure 1), while none of the other patterns matched the signature of the patterns we were counting. The situation in the second advanced activity, "Game Design"[5] was similar, with slightly more coverage of elementary patterns, but list-based patterns were still non-existent as there was no use of the list data structure.

For a thorough analysis of the existence of elementary patterns we ran the Scratch Analyzer on all the projects shared by the CSFirst studio[6] on the Scratch website, which generated the results shown in the second column of Table III. There were 622 projects in total shared on the CSFirst studio. Out of 622 projects, 128 were blank, meaning they had no blocks of code inside them. This account was used in the CS First curriculum, and for each of the activities the CS First team

[1]https://scratch.mit.edu/educators
[2]https://csfirst.withgoogle.com/s/en/home
[3]https://projects.raspberrypi.org/en/codeclub
[4]http://tiny.cc/03qbmz
[5]http://tiny.cc/7bclmz
[6]https://scratch.mit.edu/users/CSFirst

TABLE III: Elementary patterns used in Scratch studios in online resources

Pattern	CS First	Scratch Encore	Code Club	CS Concepts in Scratch
`Whether or Not`	475	92	388	826
`Alternative Action`	73	7	195	467
`Unrelated Choice`	80	9	52	55
`Partial Dependence`	15	8	23	144
`Process All Items`	15	1	12	53
`Search`	2	12	1	7
`Search (broadcast)`	4	5	11	14
`Linear Search`	0	0	0	1
`Linear Search (no stop)`	0	0	4	29
`Linear Search (broadcast)`	0	0	0	1
`Guarded Linear Search`	0	0	0	0
`Guarded Linear Search (no stop)`	0	0	0	3
`Guarded Linear Search (broadcast)`	0	0	0	1
`Loop and a Half`	0	0	0	0
`Polling Loop`	0	0	0	0
`repeat`	383	33	203	94
`forever`	501	57	425	183
`repeat until`	193	24	123	151
`lists`	23	3	93	365
`variables`	243	71	367	735

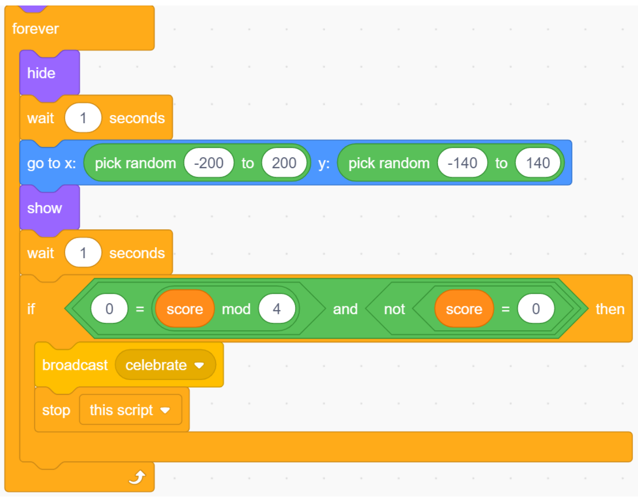

```
forever
  hide
  wait (1) seconds
  go to x: (pick random (-200) to (200)) y: (pick random (-140) to (140))
  show
  wait (1) seconds
  if <<(0) = ((score) mod (4))> and <not <(score) = (0)>>> then
    broadcast [celebrate v]
    stop [this script v]
```

Fig. 1: Search Pattern in CS First (https://scratch.mit.edu/projects/19225867)

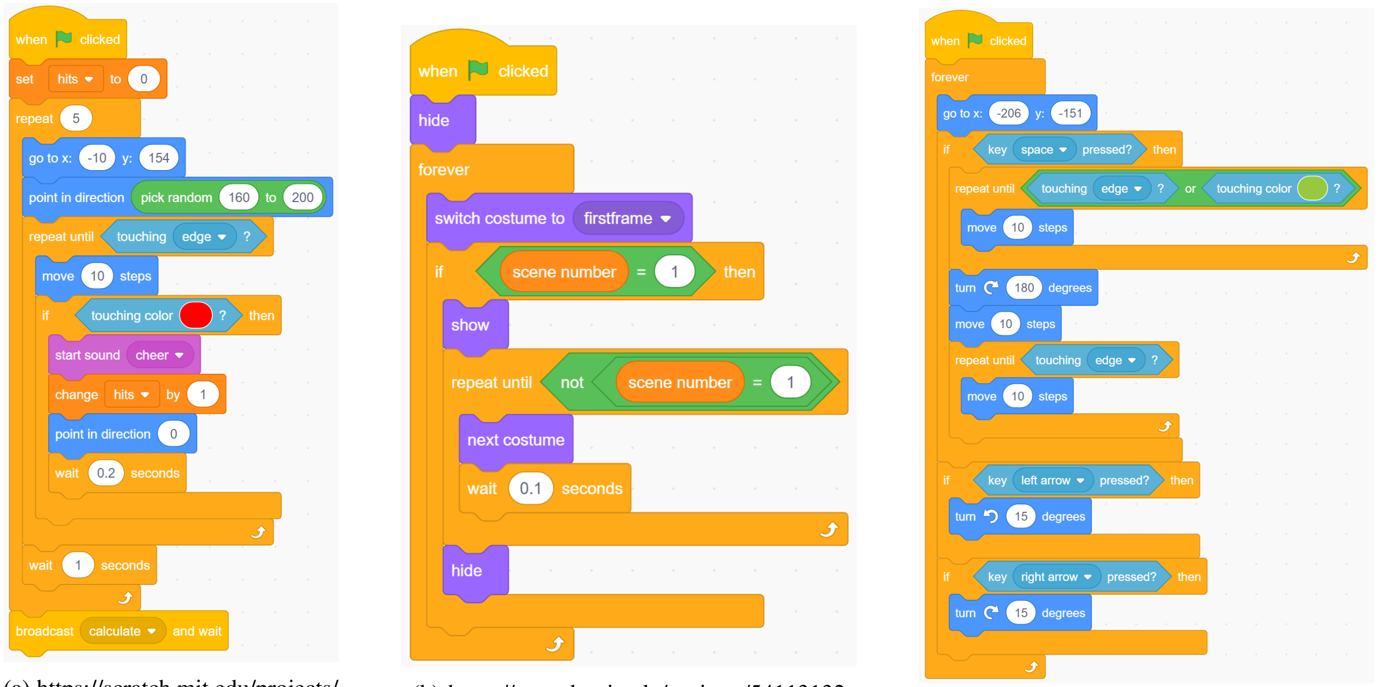
provide a starter project to help students get started with each of the activity. Most of these starter projects were blank (some contained an initial set of code) which largely explains the high number of empty projects. There were 52 projects that we were not able to process because Scratch Analyzer was written primarily for Scratch 2.0, and those 52 projects were created using Scratch 3.0. We are planning to adjust the software to be able to process Scratch 3.0 projects too, but here we report only on Scratch 2.0 projects. Table III shows the results based on 442 non empty accessible projects. A thing to note is that some projects had multiple versions due to these projects being part of multi-day activities or being remixed, so the numbers in the tables can be higher than the number of novel contexts in which students will encounter those patterns.

Table III shows a typical situation that we have observed through other analyses; there is a moderate amount of use of basic patterns (`Whether or Not`, `Unrelated Choice`, etc.) but rare (or zero) use of more sophisticated patterns (`Search`, `Loop and a Half`, etc.). We also show in the table the use of some important programming elements in the projects like loops (`repeat`, `repeat`, `repeat`), list, and variables. Simple loops (`repeat` and `repeat`) are a basic building block for many Scratch project, and have understandably high usage. The use of lists was less common, which explains why loop based patterns were rare. The use of variables was quite high, but it is surprising that variable-based patterns were still non-existent.

One observation was made though, that there were a couple of patterns that are not in the list but seemed to be used fairly often. One is `forever` $\rightarrow$ `if` $\rightarrow$ `repeat until`, and the second is `repeat` $\rightarrow$ `repeat until` $\rightarrow$ `if`. These two can be combined in a generalized form i.e. `repeat/forever` $\rightarrow$ `if` $\rightarrow$ `repeat until`. To determine whether this is a frequently used pattern, to justify its status as a separate pattern, we ran the code over all the projects under CSFirst Scratch studio. This generalized pattern was found 21 times in total in 11 different projects. A few projects were different versions or remixes of the same problem, but the occurrence of this pattern is significant enough to shortlist it as a new pattern. Figure 2 shows three examples of this pattern appearing in three different projects. From this we can see that this pattern is predominantly being used to facilitate movement or animation, which makes sense since Scratch projects focus heavily on games and animations to make it more appealing to a younger age group. This reiterates the point that there are perhaps a different set of patterns more suitable for Scratch, which need to be explored. This doesn't mean that the patterns provided in Table II are not useful, as they can still teach important programming and problem solving skills, but due to the focus on game-based teaching in Scratch, this list can be expanded.

Scratch Encore[7] is a useful resource similar to CS First,

[7]https://www.canonlab.org/scratch-encore



```
when 🏁 clicked
set hits to 0
repeat 5
    go to x: -10 y: 154
    point in direction pick random 160 to 200
    repeat until touching edge ?
        move 10 steps
        if touching color 🟥 ? then
            start sound cheer
            change hits by 1
            point in direction 0
            wait 0.2 seconds
    wait 1 seconds
broadcast calculate and wait
```

(a) https://scratch.mit.edu/projects/47448024

```
when 🏁 clicked
hide
forever
    switch costume to firstframe
    if scene number = 1 then
        show
        repeat until not scene number = 1
            next costume
            wait 0.1 seconds
        hide
```

(b) https://scratch.mit.edu/projects/54113132

```
when 🏁 clicked
forever
    go to x: -206 y: -151
    if key space pressed? then
        repeat until touching edge ? or touching color 🟩 ?
            move 10 steps
        turn ↻ 180 degrees
        move 10 steps
        repeat until touching edge ?
            move 10 steps
    if key left arrow pressed? then
        turn ↺ 15 degrees
    if key right arrow pressed? then
        turn ↻ 15 degrees
```

(c) https://scratch.mit.edu/projects/65505530

Fig. 2: Example of new pattern in CSFirst studio

designed as a curriculum[8] for school students, divided into different modules, each containing multiple lessons. Scratch Analyzer was run over the ScratchEncore studio[9] and produced the results given in the third column of Table III. Scratch Encore is a resource that is still under active development, and new materials are being added. So far it covers only the basics, like sequencing, conditionals, and loops, without using more sophisticated patterns. This is apparent from Table III. The new pattern identified in CS First was not found in the ScratchEncore studio, which is understandable, as the focus is on the basics. Also, many projects shared under the ScratchEncore studio are meant to be used as starter projects, and students are instructed to enhance the functionality of the programs. Therefore, it is quite possible that more advanced programs are not provided on purpose, hence the analysis may under-report the number of patterns that students work with.

Code Club is an international organisation that provides an online resource with example projects that have step-by-step instructions related to various technologies. The Scratch code club had three modules, and we performed an analysis on module 3, since it is the final module and covers the most sophisticated concepts and elements. There were six projects under module 3, and it did appear to cover many patterns, as across all projects there was use of lists, variables, and custom blocks. For numerical results we did an analysis of two Scratch studios that shared projects from Code Club[10,11]. These results are shown in the second last column of Table III. There were 393 (254 and 139 respectively) projects in total in both studios combined. The second studio had most of its projects created using Scratch 3.0, so 96 projects out of 139 could not be processed. There were 44 blank projects. No new patterns emerged from the review of module 3 of Code Club.

Computer Science Concepts in Scratch [17] is an online resource by Armoni and Ben-Ari (it happens to be presented in book format, but is available directly online), which explicitly mentions patterns:

> It is often the case that the person solving a problem finds that some parts of the problem have already been solved before, either by herself or by someone else. For example, if the problem involves searching in a list, the patterns for solving this problem are well known and can be used, perhaps with some adaptation. Similarly, counting and accumulating occur frequently and there are patterns for using variables to do this task. Using known patterns can significantly simplify problem solving. These patterns are found in computer science textbooks and in software libraries.

This was the only resource for younger students that we found that explicitly mentions and uses patterns. The code for all the examples used in the book was provided in the book resources. We analyzed those programs to check for the fre-

[8]https://www.canonlab.org/scratchencorematerials
[9]https://scratch.mit.edu/users/ScratchEncore
[10]https://scratch.mit.edu/users/CodeClubRik
[11]https://scratch.mit.edu/users/RPiFoundation

TABLE IV: New patterns and their signatures

Pattern	Signature
`one`	`forever/repeat` → `if` → `repeat until` OR `forever/repeat` → `repeat until` → `if`
`two`	`forever` → `repeat [variable]`
`three`	`repeat/forever` → `repeat until [key or touching]`
`four`	`forever` → `if [key or touching]`

TABLE V: Count of new found patterns in various Scratch studios

Pattern	CSFirst		ScratchEncore		Code Club		CS Concepts in Scratch	
	projects	count	project	count	projects	count	projects	count
`one`	11	21	0	0	2	2	1	2
`two`	3	7	0	0	3	3	2	2
`three`	10	12	0	0	3	3	0	0
`four`	78	148	13	27	40	84	8	10

quency of patterns. This analysis was done manually as these scripts were created using Scratch 1.4 and Scratch Analyzer runs on Scratch 2.0. The book is divided into different chapters each focusing on a specific task that become more challenging as the chapter progresses. The results confirmed the use of almost all patterns in at least one example in the book. To be thorough, there are two Scratch studios managed by each author of the book[12,13], so we also examined the scripts shared under these studios to ascertain the use of patterns (last column in Table III). The authors together have shared 93 projects between them, out of which two were blank and six were not accessible. Their projects show the use of most of the patterns listed, and so this resource appears to provide good coverage of concepts beyond just a basic introduction.

Through the manual analysis of the chapters in the book, we also discovered three new frequently occurring patterns, which brings the total of new found patterns to four. These are shown in Table IV, along with their signatures. Next, we worked backwards and ran Scratch Analyzer over all the resources (Scratch studios) discussed so far to confirm the frequent usage and to verify whether these patterns are specific to the materials they were found in, or if they are used by others too.

Table V shows the count of each of the new patterns in each studio along with the number of unique projects it appeared in. Pattern `four` is particularly popular, and arguably the simplest of the group (since the others involve a loop within a loop), while being directly relevant to games and animations.

III. Analysis of Books

For a broader analysis, we have selected six books to evaluate: three for Scratch and three for Python. The book selection criteria is based on recommendations, popularity, and citations, and is only intended as an initial sample; there are many books in this space, and their popularity varies between communities. Note that "How to Think Like a Computer Scientist" [18] is not especially aimed at younger students, but it is an open source book that has appeared in several forms and is widely used as a first programming book for students.

[12]https://scratch.mit.edu/users/MotiB/

[13]https://scratch.mit.edu/users/hhtsimpson/

We manually analyzed each book by counting occurrences of examples related to each pattern. For Python books, we analyzed only examples related to chapters appropriate for novice school students; in particular, we didn't include sections on object-oriented programming as that is not widely used in junior curricula. This might have led to reporting lower usage of patterns in our results, as most Python books spend the first few chapters introducing the basic concepts and might have the use of more sophisticated patterns in later chapters when discussing more advanced concepts. But we wanted a fair analysis between Scratch and Python and most of these advanced features of Python are not available in Scratch. We were not particularly expecting the concept of patterns to be discussed explicitly in the books, although some of them did use the word "pattern" in a very general way that doesn't refer explicitly to the idea of patterns in programming. One book, "How to Think Like a Computer Scientist" [18], mentions patterns explicitly at few places, but not as a theme for how the programming is taught.

The result of our analysis is shown in Table VI. This analysis is based only on the example programs provided within the book because we are operating under the assumption that a teaching resource using patterns will lead to the transfer of skills into the students from examples. It is possible that students go on to learn these patterns on their own, or homework exercises provided within each book might require the use of more patterns, but these scenarios are less likely. Scratch books are generally smaller and only discuss few examples, and the related chapters in Python books only form a small part of Python books, therefore the keys are chosen accordingly. For thorough analysis we also consider the patterns from Table I that were excluded in the analysis of online resources for Scratch. The patterns not supported by respective languages are marked "$\otimes$".

We observed that the `Process-All-Items` and `Alternative-Action` patterns are fairly common across all Python books. This is due to the fact that these two patterns are related to basic programming constructs i.e. loops and conditionals (`for`, `while`, and `if-else`), and whenever a program requires the use any of these, it is most likely to use these patterns. The Whether-or-Not pattern is also a very common programming construct, but in many cases it is either

TABLE VI: Analysis of introductory programming books for their use of elementary programming patterns

Pattern	The Quick Python Book [19]	How to Think Like a Computer Scientist [18]	Hello World! Computer Programming for Kids and other Beginners [20]	Super Scratch Programming Adventure! [21]	Scratch 2.0 Beginner's Guide [22]	How to Teach Primary Programming using Scratch [23]
`Whether or Not`	·	⬤	⬤	⬤	⬤	⬤
`Alternative Action`	●	⬤	⬤	•	•	⬤
`Unrelated Choice`	○	○	•	⬤	●	⬤
`Partial Dependence`	·	•	•	·	·	·
`Process All Items`	⬤	⬤	⬤	○	○	·
`Search`	·	○	•	●	•	●
`Search (broadcast)`	⊗	⊗	⊗	⬤	•	○
`Linear Search`	○	○	○	○	·	○
`Linear Search (no stop)`	•	●	•	○	○	○
`Linear Search (broadcast)`	⊗	⊗	⊗	○	○	○
`Guarded Linear Search`	○	○	○	○	○	•
`Guarded Linear Search (no stop)`	○	○	○	○	·	○
`Guarded Linear Search (broadcast)`	⊗	⊗	⊗	○	○	○
`Loop and a Half`	○	○	○	○	○	·
`Polling Loop`	○	○	•	·	○	•
`Extreme Value`	·	·	·	○	○	○
`Range of Possibility`	⊗	⊗	⊗	⊗	⊗	⊗
`Sequential Choice`	·	•	⬤	⊗	⊗	⊗
`Return Not Else`	•	○	○	⊗	⊗	⊗
`one`	○	○	○	·	○	○
`two`	○	○	○	○	·	○
`three`	○	○	○	·	○	○
`four`	○	○	○	●	●	⬤
`repeat`	⬤	●	⬤	⬤	⬤	⬤
`forever`	○	○	○	⬤	⬤	⬤
`repeat until`	●	⬤	•	●	•	●
`lists`	⬤	⬤	⬤	○	⬤	●
`variables`	⬤	⬤	⬤	⬤	⬤	⬤

Key: ⊗ Not possible ○ Possible but no example · Rare (1-2) • Few examples (3-5) ● Many examples (6-8) ⬤ A large number of examples (8+)

followed by an else or used with loops, so it becomes part of other patterns.

The `Unrelated Choice` pattern is more frequent in Scratch books compared to Python books. Scratch has no `elseif` block, so programs in Scratch tend to use independent `if` blocks when the problem is based on multiple conditions. Python programs favor the `elif` statement, which can be used for the `Sequential Choice` pattern.

`Partial-Dependence` pattern is useful when nested `if` and `else` statements are required. Although it should be a common programming problem, surprisingly few books include problems based on this pattern.

Searching is a common pattern used in programming but it is not common across the examples in the books. `Search` is the only pattern out of all eight search based pattern which is fairly frequent in both Scratch and Python. `Search (broadcast)` is often used in two Scratch books but not possible in Python. `Linear Search (no stop)` is consistently found (albeit on the low usage side) across Python books but no examples found in Scratch books.

The `Loop and a Half` pattern is often used in problems related to user input. The `Loop and a Half` and `Polling Loop` patterns are similar, the only difference is `Polling Loop` works on variables while `Loop and a Half` is based on lists. However, both of these patterns are quite rare in either Scratch or Python books.

The `Extreme Values` pattern is very specific in nature as it mainly deals with finding maximum or minimum values. It is a common programming problem so most programming languages have special functions to calculate extreme values, which means that students don't need to implement the pattern explicitly except for learning purposes. Most books cover extreme values in some way, but it is not used as frequently as a pattern.

We also measured `repeat`, `forever`, `repeat until`, lists, variables, and four new patterns identified through the analysis of online resources for Scratch. There is no `repeat`, `forever`, and `repeat until` in Python but we counted the examples using their counterparts i.e. `for`, `while (True)`, and `while (condition)` respectively. The new patterns identified are more suited to Scratch therefore it is not surprising to see zero examples of these patterns in Python books. Also, patterns `three` and `four` involve GUI based programming which is considered an advanced subject in text based programming languages, and advanced chapters were skipped for Python books. As with online resources, pattern `four` is also very common in Scratch books, but other three patterns are quite rare.

The analysis in Table VI reflects trends in the Python and Scratch books respectively. Scratch books generally take a game-based approach, since the major target audience is younger children, and the environment naturally supports interactive games. Generally, each chapter teaches how to create a game or a problem of graphical nature and introduces the programming blocks necessary to solve the problem. Along the way, an explanation is also provided about why and how this problem is solved. The focus of Scratch books can be more towards teaching Scratch as a tool and using it to create games, and not on teaching broader principles of programming; this is reflected in relatively light use of many of the patterns. On the other hand, the Python books analyzed tend to take the opposite approach, focusing more on teaching programming concepts and constructs by introducing each element and providing appropriate examples. This approach might seem a little academic to children, and might not be as engaging as the approach used by Scratch books, but it is more thorough in its coverage. Therefore, it should come as a no surprise that there are more examples covering the patterns in Python compared to Scratch.

IV. Discussion and Conclusion

Our analysis of the use of elementary patterns in online resources and a small sample of books has revealed the use of a broad range of elementary patterns in the teaching of both Scratch and Python, although Scratch resources tend to have fewer patterns. We also identified four common patterns that are used in Scratch but aren't part of the usual elementary patterns. With the exception of two resources, patterns are used implicitly rather than explicitly introducing the idea of patterns, and even one of those resources mentions it mainly in passing. However, all of the resources touched on some examples of patterns. The Python books are more oriented to giving general examples compared to Scratch, and they had more frequent examples of patterns, and more advanced patterns.

Elementary patterns have the potential to be useful for teaching introductory programming to younger students in introductory classes. Both Scratch and Python can be used as an introductory language as both provide adequate support for most commonly used elementary patterns, although a text-based language like Python requires a higher level of accuracy with typing, which needs to be supported by literacy and numeracy skills. Scratch has the advantage of being more engaging for younger children due to its appearance and interface, and it eliminates most simple typographical errors, yet it is capable of supporting most elementary patterns. On the other hand, Python (and other text based languages) would be a better choice for teaching patterns and programming in general due to more support for patterns and a traditional style of programming.

We plan to expand this exploration to other languages used for novices, and other resources for students, such as online "coding" courses. There is work to be done to further identify useful patterns for these learners, and to distinguish computational patterns associated with problems from patterns that apply to specific languages. There is also a need to match the choice of patterns to the background of the students, including their cognitive abilities, and literacy and numeracy skills; younger or less experienced students may find some patterns easier to work with than others, while more senior students should be able to work with all of the elementary patterns.

References

[1] E. Wallingford, "Elementary patterns and their role in instruction," in *OOPSLA'98*, 1998.

[2] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Others, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[3] M. J. Clancy and M. C. Linn, "Patterns and pedagogy," in *ACM SIGCSE Bulletin*, vol. 31, no. 1. ACM, 1999, pp. 37–42.

[4] O. Astrachan and E. Wallingford, "Loop patterns," in *Proc. Fifth Pattern Languages of Programs Conference*, Allerton Park, Illinois, 1998.

[5] J. Bergin, "Patterns for selection version 4," 1999.

[6] J. Hundley, "A review of using design patterns in CS1," in *Proceedings of the 46th Annual Southeast Regional Conference*. ACM, 2008, pp. 30–33.

[7] A. V. de Aquino Leal and D. J. Ferreira, "Learning programming patterns using games," *International Journal of Information and Communication Technology Education (IJICTE)*, vol. 12, no. 2, pp. 23–34, 2016.

[8] R. Cardell-Oliver, "Evaluating the application and understanding of elementary programming patterns," in *Software Engineering Conference (ASWEC), 2013 22nd Australian*. IEEE, 2013, pp. 60–67.

[9] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning computer science concepts with Scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 2013. [Online]. Available: https://doi.org/10.1080/08993408.2013.832022

[10] R. Mason and Simon, "Introductory programming courses in Australasia in 2016," *ACM International Conference Proceeding Series*, pp. 81–89, 2017.

[11] F. E. Castillo-Barrera, P. D. Arjona-Villicana, C. A. Ramirez-Gamez, F. E. Hernandez-Castro, and S. M. Sadjadi, "Turtles, robots, sheep, cats, languages what is next to teach programming? a future developer's crisis?" in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2013, p. 1.

[12] S. Grover and R. Pea, "Computational thinking in k–12: A review of the state of the field," *Educational Researcher*, vol. 42, no. 1, pp. 38–43, 2013. [Online]. Available: https://doi.org/10.3102/0013189X12463051

[13] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for k-12?" *Computers in Human Behavior*, vol. 41, pp. 51 – 61, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0747563214004634

[14] K. Amanullah and T. Bell, "Analysing students' scratch programs and addressing issues using elementary patterns," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct 2018, pp. 1–5.

[15] ——, "Analysis of progression of scratch users based on their use of elementary patterns," in *2019 14th International Conference on Computer Science Education (ICCSE)*, 2019, pp. 573–578.

[16] ——, "Evaluating the use of remixing in scratch projects based on repertoire, lines of code (loc), and elementary patterns," in *2019 IEEE Frontiers in Education Conference (FIE)*, 2019, pp. 1–8.

[17] M. Armoni and M. Ben-Ari, "Computer science concepts in scratch," *Rehovot, Israel: Weizmann Institute of Science*, 2010.

[18] A. B. Downey, J. Elkner, and C. Meyers, *How to think like a computer scientist*. Green Tea Press, 2001.

[19] V. Ceder, *The quick Python book*. Manning Publications Co., 2010.

[20] W. Sande and C. Sande, *Hello world! Computer programming for Kids and other beginners*. Manning Publications, 2009.

[21] T. Project, *Super Scratch Programming Adventure!: Learn to Program by Making Cool Games (Covers Version 2)*. No Starch Press, 2013. [Online]. Available: https://books.google.co.nz/books?id=rrAdAgAAQBAJ

[22] M. Badger, *Scratch 2.0 Beginner's Guide Second Edition*. Packt Publishing, 2014.

[23] P. Bagge, *How to Teach Primary Programming Using Scratch*. The University of Buckingham Press, 2015.