Scratch: A Way to Logo and Python

Mark Dorling
Computing at School and Network of Excellence
BCS Chartered Institute for IT
Swindon UK
+44 07825 746035
mark.dorling@computingatschool.org.uk

Dave White
Computing at School and Network of Excellence
BCS Chartered Institute for IT
Swindon UK
+44 7540 096876
dave.white@computingatschool.org.uk

ABSTRACT

There is concern amongst teachers about how to support all pupils in making the transition from popular graphical languages like Scratch to text-based languages like Python. In a new subject, not taught widely before at both primary and secondary education in England, there is inevitably a lack of tuned-in pedagogical expertise. In this paper, the authors address the transition process by exploring established pedagogy in Computer Science, and other subjects including Mathematics, Science and Languages, and by sharing and testing their findings with pupils and teachers in the classroom.

Teaching the fundamentals of programming is well served by applying sequential solutions in both graphical and text-based languages. This practitioner action research paper focuses on scaffolding support for pupils when making the transition from graphical to text-based languages. In an approach which uses graphical languages in conjunction with, not in place of, text-based programming languages, the authors discuss ways to tackle the difficulties presented to pupils by text-based languages, and propose a tested strategy for teachers to enable pupils to undertake the transition successfully.

Categories and Subject Descriptors

K.3.2 [**Computers & Education**]: Computer and Information Science education - *Computer Science Education, Curriculum*
D.3.2 [**Language Classifications**]: Scratch 2.0, Logo, Python 3

General Terms

Theory, Experimentation, Human Factors

Keywords

Graphical programming language, text-based programming language, transition process, computational thinking, unplugged activity, Computer Science Education, Scratch, Logo, Python

SIGCSE '15, March 04 - 07 2015, Kansas City, MO, USA

ACM 978-1-4503-2966-8/15/03..$15.00
http://dx.doi.org/10.1145/2676723.2677256

1 BACKGROUND

From September 2014, pupils in English state-maintained schools will be expected to follow the programmes of study set out in the national curriculum document [4]. Computing at School has responded with a targeted resource [2].

The programme of study has high-level aims in terms of the introduction of computer science [4]. The following extracts illustrate learner capabilities at different stages of primary and secondary education.

At Key Stage 2 (age 7-11) pupils should be able to (amongst other things): "… solve problems by decomposing them into smaller parts." and "use sequence, selection, and repetition in programs; work with variables and various forms of input and output." and also "… detect and correct errors in algorithms and programs." ([4], p. 189).

At Key Stage 3 (ages 11-14) pupils should be able to: "… make appropriate use of data structures [for example, lists, tables or arrays]; design and develop modular programs that use procedures or functions." and also "Understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming…" ([4], p. 190).

For primary educators there is no specification in the programme of study to teach either a graphical or text-based language, instead the emphasis is placed on teaching concepts and principles. In contrast, the programme of study for secondary education, Key Stage 3 makes it explicit:

Pupils should be able to: use two or more programming languages, at least one of which is textual, to solve a variety of computational problems ([4], p. 190)

2 INTRODUCTION

There is a growing range of graphical programming environments available to teachers, such as Scratch [9], Alice [3], Kodu etc. but little research into how effective they are at supporting pupils to make the transition from graphical to text-based languages. Wolz U. et al reported that after initially learning Scratch, the students' transition to Java or C appeared to be easier [15].

After using graphical programming environments, there is a perception amongst non-specialist teachers in both primary and secondary education that text-based programming is hard. And with some reason: "Beginners need to learn to identify the structure of a problem and the core logic of a program to solve it but they are simultaneously forced to deal with technical details of the programming environment that are not related to these and this can be overwhelming and often discouraging for beginning programmers." [5]. Pupil difficulties with text-based language syntax and error messages are identified in [6], [10], [15]. As a result, some are publicly questioning whether programming is a skill for only the more able pupils, and have asked questions about the body of pedagogical research for teaching pupils with Special

Educational Needs (SEN) such as Dyslexia, and English as a Second Language (EAL).

With their experience at opposite ends of the learning spectrum the authors met at a Computing at School meeting and found their teaching paths surprisingly convergent both in content and pedagogy. One, in the Digital Schoolhouse conducting action research to teach primary and secondary pupils directly to program in Scratch and Logo, the other delivering CPD Courses in Computing to Primary and Secondary School teachers from different disciplines, mostly without a background in Computing, on how to teach Scratch and Python to pupils; and at the same time to investigate and consider how their own programming skills might enhance the delivery of their own subjects. The authors concentrated on:

1. Identifying the difficulties associated with learning to program in a text-based language available in the research literature, and their own experiences of the difficulties which arise in teaching programming in Scratch and text-based based languages.
2. Investigating the background knowledge necessary for the cross-curricula topic chosen --- geometric shapes and patterns.
3. Utilising existing pedagogy, by consulting and observing colleagues teaching in different disciplines, --- including Computer Science, Mathematics, Science and Languages, in order to make the transition process for pupils and teachers a realistic proposition.

The focus of this practitioner action research was on upper key Stage 2 and key Stage 3 pupils and their teachers.

2.1 Aims and Methodology

The authors, with their ongoing involvement with pupils and teachers, decided to pool their resources and experience: to see if they could build on pupils' and teachers' confidence and familiarity with Scratch to find a realistic pathway through the transition process to text-based languages, navigable by teachers and their pupils.

The authors experimented with a number of ideas -- some tried and tested approaches – other ideas were researched and pursued with pupils and teachers in the classroom. Those adopted are summarized here:

3 TRANSITION STRATEGY

A constructionist approach to learning programming and concepts in Computer Science developed from the many years that Logo has been in existence[8], underpinned by the elements of Computational Thinking [11]: *abstraction, algorithmic thinking, decomposition, evaluation and generalization.*

To use a methodology that all three languages shared, namely, sequential processing with a sprite/turtle and the reward of the instant feedback of sprite/turtle activity on the screen.

To reduce the number of tasks to accomplish at the screen by undertaking learning away from the computer in the form of a number of 'unplugged' activities , which are targeted to cover the background knowledge of an exciting cross-curricular topic, in this case geometric shapes and patterns and undertake some text-based programming learning experience in a relaxed play setting away from the computer, using UPL (Unplugged Programming language) --- essentially a very small subset of Logo, which drives a pet/robot.

To harness the benefits of Scratch 2.0 and its freedom from syntactical errors in developing sequential solutions to problems along a simple pathway through the topic -- this pathway is designed to take account of the other known difficulties in programming generally, of concurrency, initialization, and variables [6].

To address, in a rudimentary way, the fundamental control structures of programming: *sequence, repetition, function, selection and communication,* in the course of the pathway.

After a solution to an exercise is completed successfully in Scratch, to use the structural similarity of a Scratch program to the equivalent program in text-based languages. The program would be mapped into a Logo/Python program, with the focus now solely on attending to the environment and syntax of the text-based language.

To consult colleagues and observe existing pedagogy in the topic as well as pedagogy in other subjects, including Computer Science, Mathematics, Science and Languages. Some of the difficulties associated with learning text-based languages were tackled directly from these resources. Others were delayed until later in the transition process.

4 DISCUSSION

The authors observed and consulted specialist colleagues teaching Mathematics, Science and Languages in local primary and secondary schools to gain a greater understanding of pedagogy in other subjects, and, in the first instance, how it could be applied to computational thinking and programming in the transitional process the authors were undertaking, and further, to Computer Science in general.

They were particularly interested in Mathematics teachers' approaches to geometrical shapes, symmetry and patterns, mental processes in mental arithmetic and progressing to elementary linear algebra and linear transformations.

Science teachers methods of helping pupils to collect and analyse data, to experiment and hypothesize in Science, including key questioning techniques employed to harness enquiry-based learning, and use of scientific language and concepts for variables and constants.

Language teachers' approaches to teaching pupils learning new languages and applying rules of syntax in languages. Literacy: reading precedes writing [12].

5 THE PATHWAY

The authors have outlined the pathway as a sequential journey through a course of 10 sessions. Each session, unplugged [1], or at the computer may consist of a mixture of: a direct teaching component, an activity, a worksheet with graded exercises (*s ranging from 1-5) and individual coaching/tutoring

In the next section 5.2.1, the authors map out, in more detail than in other sections, how the unplugged session on programming takes pupils through the use of the control structures ***sequence, repetition and use of functions,*** preparing the way for interaction with the computer in subsequent sections. Apart from the two further unplugged sessions, sections up to 5.2.8 summarise how these control structures are programmed in Scratch in the first instance, before mapping to a text-based program in Logo/Python.

5.2.1 Unplugged 1

'Action Geometry' in UPL --- LOGO Unplugged

In this session, we get used to programming a pet/robot/sprite/turtle in a text-based format without a computer. In UPL (an unplugged programming language, essentially Logo, but adapted for unplugged use), a pet/robot is defined by 2 characteristics:

- Its position -- where it is standing, if we are 'walking' the talk, or a point on the paper if we are drawing the path. We start at the origin O.

- The direction in which it is facing at any moment. (We assume it is facing right → to start at O).

We consider 3 of the instructions in UP:

forward 1 step (or 2 or 3),
left turn (through 90 degrees),
right turn (through 90 degrees).

We abbreviate the instructions when writing, for example, to

fd2, lt and rt

The program to draw a straight line of length 2, and return to the starting point facing in the original direction would be:

fd2 lt lt fd2 lt lt

Notice that the pet/robot has turned through what we call a SPIN(360) in this program. Pupils can work in pairs using UP in a number of ways.

1. 'Walk the talk': one pupil 'talks' the instructions, the other 'walks' the figure
2. 'Draw the talk': one pupil 'talks' the instructions, the other 'draws' the figure
3. Use a combination of 1, 2 interactively to construct a program on paper

***Sequence*:** is setting up the instructions in a program in the right order usually set out one after the other, reading left to right and top to bottom.

- Activity: Using the instructions in UP in combination, how would you program the pet/robot to: trace a capital letter shape like L, I, T, F, E and return to the starting point and starting direction? Starting point and direction for the pet/robot is at O for the letter L.
- Activity: trace a square of side-length 2 paces and return to the starting point and starting direction.
- Activity: Ex 1. Cracking the Code: what does the pet trace out in the following program?

fd2 lt fd2 lt fd2 lt fd2 lt fd2 fd2 lt fd2 lt fd2 lt fd2 lt

fd2 fd2 lt fd2 lt fd2 lt fd2 lt lt lt fd2 fd2 lt lt

Quite often, the idea of repetition is used to exploit the occurrence of pattern in the code, which in turn reflects the pattern in the figure generated by the code.. A solution code for drawing a square might look something like this:

fd2 lt fd2 lt fd2 lt fd2 lt

With the 'pronounced' spacing it is evident that 2 instructions 'fd2 lt' are repeated 4 times corresponding to the four sides of the square.

Repetition A repeat structure is a way of repeating a set of instructions as many times as specified.We use the UPL instruction repeat as follows:

repeat4 [fd2 lt]

The repeat structure shortens the program appreciably and makes it more easily interpreted. We have another structure, a *function*, that helps to shorten the code and makes it more easily interpreted in a different way. In Ex 1, we saw a longish UP program, in which it was hard to see the wood for the trees. If you break it down into its components (***decomposition***) it is:

1. repeat the instructions 2, 3(below) sequentially three times
2. draw a square of 2 paces
3. move forward 2 paces
4. to end up with 3 squares in a row
5. then return the pet to the starting point and direction.

Function: use of a function sq to draw a square.

- Activity: we introduce a new instruction 'sq' in UPL. sq2 draws a square of side 2 paces. In the program in Ex1, identify the code that sq2 stands for. Rewrite the program in UPL using sq2 in place of code wherever you can. Rewrite the program in UPL using sq2 and the repeat structure
- ***Activity : We name a function row which stands for and names the three squares in a row, with each square of sidelength 2 paces. Use row and the repeat structure in UP, to write a program in UP to draw a 3x3 array of 9 squares made up of three rows of squares touching.

5.2.2 Unplugged 2: A worksheet

An example of 'joining dots' action geometry in Figures 1 and 2 serves to acquaint pupils with the background knowledge of polygons (a) and stars (b), (d), (e), (f).

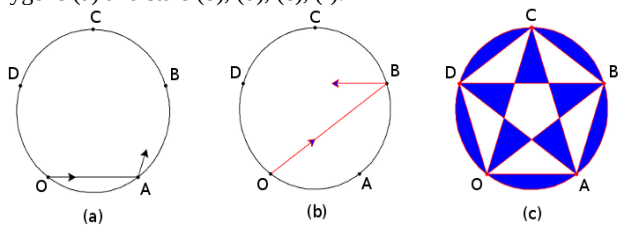
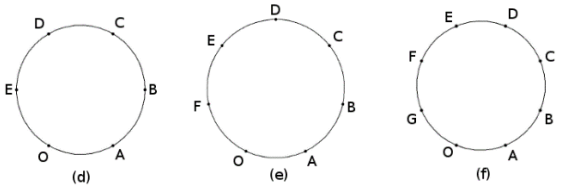


Figure 1. Joining the dots in circles with 5 equally spaced dots at OABCD.



.

Figure 2. *action geometry:* for the hexagon, heptagon and octagon.

- Activity

Polygons: (d), (e), (f) Start at O, join dots OA, AB …
Stars: (e) Start at O, skip a dot, join OB, BD …
(e) (f) Start at O, skip 2 dots, join OC, CF …
*** (d) Start at O, skip a dot, Join OB, BD … What happens? How would you complete a hexagram (star of David)?

5.2.3 Comparison

Scratch, Logo and Python

The similarity between UP and Logo is obvious in both layout and individual instructions. Logo and Python turtle instructions are closely matched. And the matching structure of a program is reflected in Scratch Logo and Python in Figure 3, where UPL instructions have been entered vertically to facilitate the comparison (similarly for Logo).

Once the pupils had spent time doing unplugged programming and were familiar with programming simple letter shapes and other exercises in UP, the next step was to develop the program to draw a square on screen in the familiar Scratch environment.

Once the program was tested and running, the pupils would attempt to map it onto a Logo/Python program, focusing solely on managing the new environment and getting the specific syntax correct of the text-based language. See Figures 3, 4.

Table 1

Turning Angle	Repeat value	Regular Shape	Shorthand	Length of side
90	4	square	(4, 360/4)	100
45	6, 7 ?			50
60	?			50
?	5	pentagon	(5, 360/5)	75
?	10			?
?	7	heptagon		50
4	90	?		5

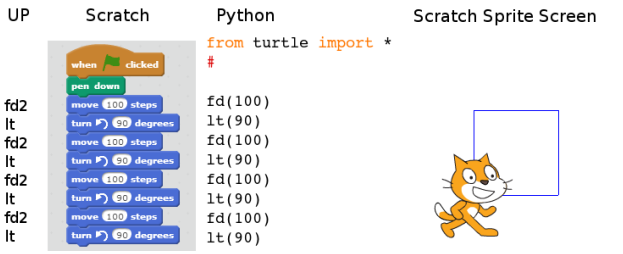
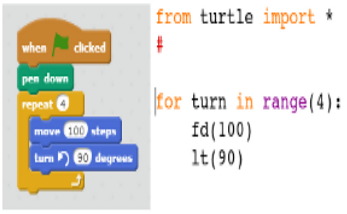


Figure 3. Matching Program structures in UP, Scratch and Python: drawing a square

5.2.4 Repetition

The repetition of the 2 instructions FD 100 LT 90 makes the introduction of the repeat loop welcome, necessary (and useful for experimenting with the values in the loop)



All pupils managed the ordering (sequence) of instructions when designing an algorithm, to draw a square. And having met the repeat structure for drawing a square in UPL, found it straightforward to do the same in Scratch.

Figure 4. The repetition structure in Scratch and Python

5.2.5 Experimentation

As a result of observations and pedagogical consultations, the authors modified their approach to teaching computational thinking [7]. By teasing out *decomposition and generalisation* in 'unplugged' sessions, as well as in coding computing lessons, and extending experimentation to include collection and interpretation of data, the authors introduced a more challenging yet supportive interactive style both oral and written.

When pupils had constructed the repeat loop in Scratch (and Python) see Figure 4, they were then able to experiment with the turning angle (90 degrees) by substituting 45, 30, 60 for 90 degrees in their repeat loop. They were asked to see if they could find another value of the repeat parameter (4) which would result in the turtle 'closing' (returning to the starting point and starting direction) the shape they were drawingSPIN(360). The trial and error process they were engaged in involved *evaluating* the effectiveness of the program they were using albeit with simple changes and was a successful introduction to semantic debugging. Octagons, hexagons, dodecagons (12-sided polygons) readily popped up on their screens, along with some excitement.

The pupils now had accumulated some data as a result of their experimentation in Table 1 below. The authors' dialogue with Science and Mathematics teachers proved invaluable. A period of guessing, hypothesizing and trial and error followed.

- Activity: Complete Table 1: What do you make of the last entry?

Some more able students were able to find the angle for the pentagon. But the heptagon would involve the use of a calculator to record a non-integer value. And the key question to those who found answers was: "Describe how you did it?

(The production of these, and similar unplugged exercise sheets make useful programming projects for pupils (and indeed teachers) when they become more adept programmers. These 'unplugged' worksheets are a good example of how the ability to program can produce learning materials which would otherwise be a difficult chore [14].

5.2.5 Unplugged 3 Enquiry-based learning

Some questions were posed interactively in the classroom --- the questions that follow were set in a CPD worksheet for teachers to adapt for their own classrooms.

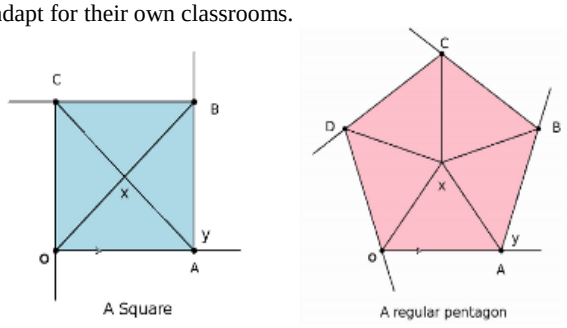


Figure 5. Tracking the square and the pentagon

- Activity: Angles and SPIN(360)

This process could be undertaken as a formal geometry exercise. Here it is introduced calling upon symmetry and 'action geometry' in the computational thinking of *decomposition and generalizing*. Devise a set of questions to form an enquiry-based exercise to help the reader calculate angles x and y in the square in Figure 5. Afterwards use exactly the same questions but substitute the words 'regular pentagon' wherever you see the word 'square' in your questions. The questions should achieve the same result when applied to the pentagon. Here are a couple of questions to start:

First the Square: (a 4-sided simple, regular, symmetric polygon)

1. What does symmetry mean for a square?
2. Hint: If you turn round completely once and face the same way as you started, how many degrees have you turned through? We will call this a SPIN.

***Generalising* from the known to the unknown**

In the last example with the square, we may have known from our experience what the values of x and y are. In other polygons, we may not be so familiar with angle sizes, hence the importance of *how we work the values of x and y out* with the square. We then have a basis for generalising to the pentagon.

- Activity: More Angles and SPIN(360)

Now the pentagon: (a 5-sided simple, regular, symmetric polygon)

After the unplugged sessions with polygons and stars and plugged-in experimentation, pupils were able to investigate, catalogue and decide which stars could not be drawn straightforwardly. They

were then able to test and verify the formal hypothesis (and *generalisation*) for drawing a star: (in shorthand notation)

(n, r * 360/n) where r > 1 and r and n are co-prime, that is, have no common divisor other than 1, and where n is the number of sides of the star, and r runs from 1 to n-1.

In physical terms:

- the r corresponds to starting at O and joining every rth vertex of the star,
- and to r SPINS of the turtle on itself as it traces the star.

For all simple regular polygons r = 1 which generalises this formula to include all regular polygons and stars.

5.2.7 Function

Definition and call, simplifying and (generalizing) with parameters.

The next step, and a crucial one, in extending pupils' interaction with, and learning of, *generalisation* and *abstraction*, is in the creation of their own user functions. In the unplugged programming sessions pupils were able to make use of the user function **'sq2'** for a square with argument 2, without needing to formally define it in a specified format. (Advantage of unplugged! Of course, pupils already make use of system functions like **'fd2'** without seeing a formal definition).

After pupils have practiced *sequence and repetition* using Scratch and transitioned to Logo/Python, they are introduced at this early stage to a function without parameters: 'start-right', which they use at the start of all their Scratch programs. It synchronizes the initialisation of a Scratch Sprite to start from the origin and pointing to the right, with the Python turtle, every time an amended program is re-run. Then they meet the *generalizing* idea of a parameter (sidelength) for a function **'sq'** they have employed in an unplugged setting. See Figures 5 and 6 Scratch came of age as a fully-fledged programming language with the arrival of the explicitly defined function facility (custom block), which arrived in Scratch version 2.0 in April 2013.

```
def sq(sidelength):
    #
    for turn in range(4):
        fd(sidelength)
        lt(90)
```

Figure 5. Function definition of sq(sidelength in Python)

The custom block in Scratch is a good environment in which to learn a difficult concept. It requires careful tuition and practice in both the concept and the process of the definition of a function.

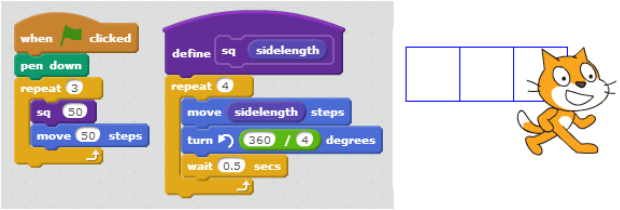


Figure 6. Function definition of sq(sidelength) in Scratch.

Figure 5 encapsulates in the function definition the shorthand form in a repeat loop for a rotation (4, 360/4). In the program the shorthand form, in a repeat loop for a translation of a square through 3 times through 50 steps(pixels), is (3, sq(50), 50). The use of the function, through the shorthand form gives rise to the physical interpretation of translating the square, and replaces the more indecipherable nested loop that would otherwise arise.

The pupils have now learned how to 'hide' the definition of a function (by capping it in a custom block definition and moving it to the side in Scratch 2.0; and by declaring it in a file (module) say 'd.py' and preloading it with 'for d import *' for Python 3. The function can then be inserted as a single instruction in code, thus simplifying the coding on both sides of the transition, (see Figure 7).

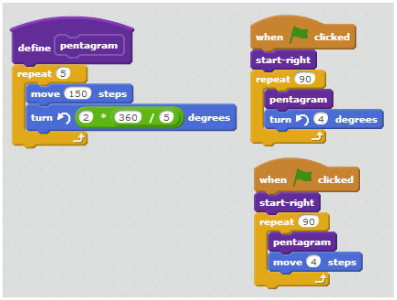


Figure 7. Function definition for a pentagram (5-star) in Scratch

Pupils are now equipped to devise and program unique patterns and in Python 3 to add colouring-in, which enhances spectacularly their adventures into patterns. The definition of the function pentagram shorthand (5, 2*360/5) illustrates how the shorthand notation and the format of entries in the repeat loop is an aid to *generalizing*. The top program in Figure 7 rotates the pentagram (90, pentagram, lt(4)) and the bottom one translates it (90, pentagram, fd(4)), see Figure 8 and 9 for more examples of rotation and translation.

5.2.8 Patterns

Using functions, rotation and translation. Colouring-in in Python

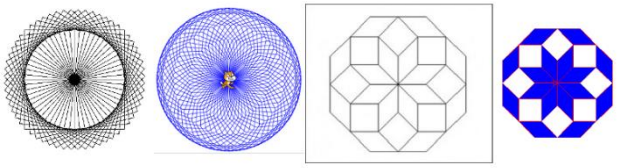
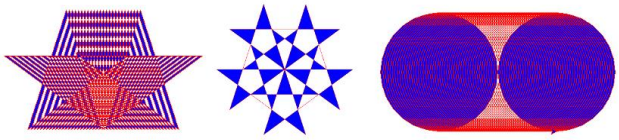


Figure 8. Rotating shapes: square, 11-polygon octagon and octagon coloured (Python)



(a) (b) (c)

Figure 9. A shooting (translating) star, a double spin of a star and translating a circle.

In Figure 9(a), translate a star through 125 pixels, 5 pixels at a time; shorthand form: (125/5, pentagram, fd(5)).

In Figure 9(b), SPIN(2 *360) a pentagram in steps of 2*360/5 degrees; shorthand form:(5, pentagram, 2 * 360/5).

In Figure 9(c) translate a circle through 200 pixels, 4 pixels at a time: shorthand form: (200/4, circle, fd(4))

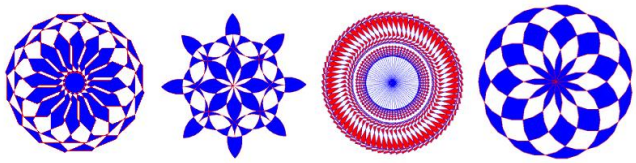


Figure 10. Fancy Rotations of regular shapes

5.2.9 Selection

Drawing a general polygon/star. (algorithmic thinking, decomposition and generalization). Selection structures are best dealt with in a number of scenarios in Scratch directly before a transition takes place to Logo/Python. An activity that arises naturally in the narrative of *generalisation* in patterns is to write an App (function) to draw a polygon/star of any number of sides). And for the drawing to fit in the space available *(selection).*

5.2.10 Communication

Talking to the user; ask and answer in Scratch and print and input in Python. If statements in both. The App above is extended to allow a user to choose the polygon/star to be drawn. In Scratch, a communication with the user may be conducted with the easy-to-use 'ask and wait' block for the program to ask a question, and the reply from the user is entered into the special variable 'answer'. The Python 3 structure can be directly paralleled in the transition.

6 CONCLUSION

The National Curriculum doesn't specify that pupils should learn a graphical language in primary school, only that the pupils should learn two languages in secondary school, one of which should be a text-based language. The National Curriculum is regarded as the minimum expectation of what should be taught to pupils. This paper suggests that a greater focus should be placed by teachers on developing and embedding good pedagogy in computing lessons with teachers paying particular attention to the transition process from their graphical to text-based language. Furthermore, that there is established pedagogy in other subjects that can be effectively used in a Computing lesson. In turn, it was also evident that programming could enhance the delivery of other curricula topics.

Despite the concerns alluded to in the opening of this paper concerning the difficulties for pupils in programming in text-based languages, the anecdotal evidence, that this paper provides, supports the practice of using graphical languages in conjunction with, (in effect using a graphical tool as a form of pseudo coding), not in place of, text-based programming languages, to improve the confidence, independence and resilience of pupils when learning to program using a text-based language, in both primary and secondary education.

Given good pedagogy, expert support for teachers and a strategy (see Section 3) for supporting all pupils with making the transition from graphical to text-based languages, the authors conclude:

- Pupils can learn a text-based programming language whilst at primary school.
- All secondary school pupils are able to master the basics of text-based programming.
- A graphics based language like Scratch would be a good introduction to programming for beginners at both primary and secondary levels.

It is the experience of the first author that this transition process has been a factor in an increased uptake of Computer Science GCSE qualification at Key Stage 4.

7 ACKNOWLEDGMENTS

The first author acknowledges the financial support by the SSAT of the Digital Schoolhouse Trust, and more recently The Digital Schoolhouse London Project funded by the Mayor of London and Department of Education through the London School's Excellence Fund.

8 REFERENCES

[1] Bell, T., Witten, I.H., Fellows, M. 2005. Computer Science Unplugged. http://csunplugged.com/ [last accessed 5 July 2014].

[2] Computer science: A Curriculum for Schools 2012. Available from: http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf

[3] Dann, W., Cooper, S. and Pausch, R. 2009. Learning to Program with Alice, Second Edition. Pearson.

[4] Department for Education. 2013. The national curriculum in England, Framework document. Available: www.education.gov.uk/nationalcurriculum [Accessed 13-08-2013].

[5] Kelleher, C. and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Survey 37, 2, 83-137.

[6] Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2010) Learning computer science concepts with Scratch. ICER '10 Proceedings of the Sixth international workshop on Computing education research. ACM, 69 – 76

[7] National Research Council. (2011) Report of a workshop of pedagogical aspects of computational thinking, Washington, DC: The National Academies Press

[8] Papert, S. and Harel, I. 1991. Constructionism. Ablex.

[9] Resnick, M., Maloney, J. Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. 2009. Scratch: Programming for all. Commun. ACM, 52, 11, 60-67.

[10] Rizvi, M., Humphries, T., Major, D., Jones, M., and Lauzun, H. (2011). A CS0 course using scratch. J. Comput. Small Coll., 26(3):19-27.

[11] Selby, C. and Woollard, J. (2014) Computational Thinking: The developing definitions. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE 2014. ACM.

[12] Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., and Prasad, C. 2006. An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. In Proceedings of the Eighth Australasian Conference Computing Education (Hobart, Australia, January 16-19, 2006), 243-252.

[13] Wing, J. (2006) Computational Thinking. Commun. ACM, 49, 33 – 35

[14] White, D.[no date]. Unplugged: [online]. Available from http://www.ispython.com/unplugged [accessed 5 July 2014]

[15] Wolz, U., Leitner, H. H., Malan, D. J., and Maloney, J. (2009). Starting with scratch in CS 1. SIGCSE Bull. 41, 1, 2-3.