(More than) Twenty Things to Do in Turtle Blocks

by Walter Bender[1], Cynthia Solomon[2], Claudia Urrea[3]

Abstract

In the late 1960s when Logo was first used in schools, student projects took on a seemingly science-fiction ring as Logo-programmed floor turtles equipped with sensors navigated the physical world, computer music was composed in Logo, turtles were used to investigate the concepts of speed and acceleration, and keyboards were being replaced by child-friendly input devices. Today's projects are less in the realm of science fiction—computation and sensors have found their way into a myriad of consumer products—nonetheless, there has been a great expansion of what children can do with Logo and how it can be used as the underlying motivator for "improving" programming languages and programmable devices.

In this paper, we illustrate this point by discussing (more than) twenty of our favorite explorations of Turtle Blocks, a visual programming environment based on Logo. Papert and Solomon compiled a list of "Twenty Things To Do With A Computer" in their 1971 paper, where they described the interaction between the learner and the computer as a "conversation" in words or numbers. We observe that the conversation has grown to include multi-media, the Internet (both as a forum for collaboration and data collection), and a much broader collection of sensors and robots than was available 40 years ago. Also, in Turtle Blocks, there is an opportunity for the learner to expand upon the language, taking the conversation in directions unanticipated by the Turtle Block developers. In this paper, we focus on the needs of the novice user: (1) art and media; and (2) geometry; the advanced user: (3) sensors and data; (4) gamification; (5) collaboration; and (6) robotics; the user who wants to reach beyond block-based programming: (7) math and programming; and a catch-all theme: (8) "other".

Keywords Logo, Constructionism, Programming

1. Turtle Art and Turtle Blocks

Turtle Art is a programming environment with a Logo-inspired [1] graphical "turtle" that draws colorful art based on snap-together visual programming elements. Its "low floor" provides an easy entry point for beginners to programming. Turtle Art was written by Brian Silverman, the author of numerous Logo and block-based programming environments (Microworlds 2012, Lego Mindstorms [2]), Artemis Papert, and Paula Bonta. There are two versions of Turtle Art: a version that is maintained by Silverman [3] and a version that is maintained by Walter Bender [4].

Turtle Art is one in a family of block-based programming environments descendant from Fred Martin's Logo Blocks that are designed for children [5]. What distinguishes Turtle Art from some of its peer environments is its emphasis on visual expression. Other environments, such as Scratch put their emphasis on narrative. Etoys and Turtle Blocks are more general programming environments. The Turtle Art focus on art is explicit. As Papert and Silverman put it:

> Turtle Art is about art. It is a system that is relatively unsophisticated on the technological front and that is quite narrow in terms of the kind of artifacts that can be produced. Turtle Art is focused on creating static images. It is not a general programming environment or a system for exploring math, language, science, etc. [6].

[1] Executive Director, Sugar Labs walter@sugarlabs.org
[2] Consultant, cynthia@media.mit.edu
[3] Researcher, MIT, calla@mit.edu

Turtle Art aims to engage learners in personal expression. Papert and Silverman argue that by engaging in “deep exploration and produc[ing] substantive works”, children become fluent users of technology.

Like Logo in the late 1960s, Turtle Art has only one turtle. Oh, but the things you can do. Papert, an artist, has collaborated with Silverman to show us how to create beauty and artistry. Turtle Art has evolved to suit this artistic bent. For example, setxy never leaves a trace of the turtle’s path; it ignores the pen’s state. There are other differences when the Turtle Art turtle is compared with Scratch sprites: the turtle turns its body visually to point in the direction it is heading and the single turtle carries only one costume or “shell”.

Bender wrote and maintains Turtle Blocks, a fork of Turtle Art that has "high-ceiling" programming features that challenge the more adventurous student. Turtle Blocks is distributed as part of the core Sugar distribution that is being used by more than 3-million students in 40+ countries. Turtle Blocks extends Turtle Art from a compact turtle graphics environment to a full-fledged programming environment. Special blocks allow communication with sounds, sensors, and robots from the physical world. Users can also write programs for multiple users over the net. Turtle Blocks includes affordances to allow the user to make inline extensions to the language as well as plugin support for writing and distributing extensions.

2. Using Turtle Art/Blocks

Programming in both Turtle Art and Turtle Blocks is done by snapping together blocks. Each block is a command for the turtle, e.g., there is a block to tell the turtle to go forward, to turn right, etc. (See Figure 1). The blocks are organized on palettes: one for the turtle, one for the pen, etc. Each palette contains a themed collection of blocks that are combined into a program. For a more detailed introduction on how to use Turtle Art, see the on-line tutorials (Turtle Art Intro and Turtle Blocks getting started).

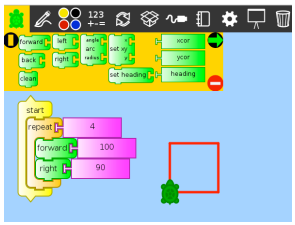


Figure 1. Programs are created by stacking blocks dragged from the blocks palettes. Shown above is a program that uses the forward, right, and repeat blocks to draw a square.

3. Debugging Aids in Turtle Blocks

Turtle Blocks provides a number of affordances for debugging. The program blocks are highlighted as the program executes, enabling the user to observe the flow of program logic in real time. Execution can be slowed down to a “snail’s pace” or run at full speed (See the “snail” and “rabbit” buttons on the upper right of Figure 1). Breakpoints can be set; variable-state monitored, print statements and inline comments inserted. There is hover help with brief descriptions of every block and contextual message displayed whenever the program encounters a syntax error. Turtle Blocks logs errors and a list of which blocks are used and provides a mechanism for users to visualize the evolution of their programming style over time.

4. Free/Libre Software and the affordances of responsibility

Perhaps the biggest difference between Turtle Art and Turtle Blocks is the facility with which the end-user can make modifications to the language itself. The initial motivation for extensibility was a request from a Uruguayan teacher for a "square root" block to use in teaching the Pythagorean theorem. We were going to respond, "Turtle Art is Free Software: you have access to the source, so add it yourself", but this was asking too much of our users: Turtle Art, although licensed to permit end-user modifications, was not written with end-user modifications in mind. We forked Turtle Art and began a series of refactoring it to make it easier to add new blocks: first through inline extensions and ultimately through a plug-in mechanism. The result has been numerous extensions (Butia, LEGO WeDo®, LEGO NxT®, ExpEyes®, Arduino®, currency, nutrition, Box2D physics, binary logic, trigonometry, etc.) by users who are pursuing goals other than those envisioned by the authors. By giving our users not just the license but also the means to make modifications, they have taken responsibility for shaping their own learning environments.

5. (more than) (More than) 20 Things to do with Turtle Art

Art	**Collaboration**
1. Make art!	25. Program with other people (sharing both turtles and code).
2. Write a paint program.	26. Create a multimedia chat.
3. Write a paint program that responds to sound.	Robotics
4. Incorporate rich media into your artwork.	27. Build an interface to an Arduino (WeDo, NXT, GoGo, etc.) robot.
5. Make ASCII art.	28. Build an interface to a collection of robots that interact with each other.
Geometry	**Math and programming**
6. Draw a square.	29. Learn to debug.
7. Draw a flag.	30. Explore recursion, procedures, variables, etc.
8. Solve a geometry puzzle.	31. Learn about Boolean and binary operations.
Sensors and data	32. Explore Cartesian and polar coordinates.
9. Build a temperature sensor.	33. Explore trigonometry by defining inline functions.
10. Build a webcam that records pictures whenever there is a loud noise.	34. Calculate and display prime factors.
11. Build a VU meter.	35. Design extension to Turtle Blocks in Python.
12. Build an oscilloscope.	36. Export your project as Logo.
13. Build a seismograph.	37. Export your project as Python.
14. Measure gravitational acceleration.	**Other**
15. Build a cattle inventory system using RFID.	38. Build worlds for the Box2D physics simulator.
16. Build a bicycle odometer.	39. Animate a sprite walking across a background image a la Scratch.
17. Record data to a journal.	40. Build a Galton box.
18. Build a web browser.	41. Explore basic electronics (ExpEyes plugin).
19. Plot global temperature data.	42. Learn about currency.
20. Plot the results of rolling a pair of dice.	43. Calculate Weight Watchers® "Points".
21. Plot data and images on a map.	44. Build an interactive text-to-speech engine.
Gamification	45. Write PowerPoint®.
22. Write a snake game.	
23. Build a digital piano.	
24. Write a multi-player network game.	

5.1. Art and media

Turtle Art is first and foremost about engaging young learners in programming through personal expression, e.g. making art. In that sense, it is very similar to earlier Logo implementations (See Figure 1). Silverman was careful to include just enough functionality to unleash the raw potential of the language without burdening them with too many high-level concepts. For example, in the original Turtle Art implementation, there were only two variables ("store in box 1" and "store in box2") and two stacks ("to stack1", "to stack2"). These constraints do not seem to hamper the ability to generate complex visual designs (Figure 2).

Turtle Blocks provides rich-media tools that enable the incorporation of sound, typography, images, and video [7]. In addition, it enables the artist to incorporate sensors into their work. Among the sensors available are the mouse button and mouse x and y position. These can be used to create a

simple paint program, as illustrated in Figure 3. Writing your own paint program is empowering: it demystifies a commonly used tool. At the same time, it places the burden of responsibility on the programmer: once we write it, it belongs to us, and we are responsible for making it cool. Figure 4 shows a variant on paint that uses microphone levels to vary the pen size as ambient sound-levels change. Once learners realize that they can make changes to the behavior of their paint program, they become deeply engaged (Figure 5).

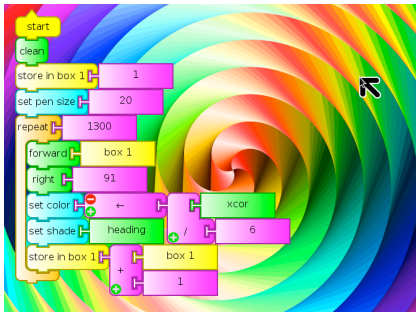


Figure 2. "Squiral" by Brian Silverman.

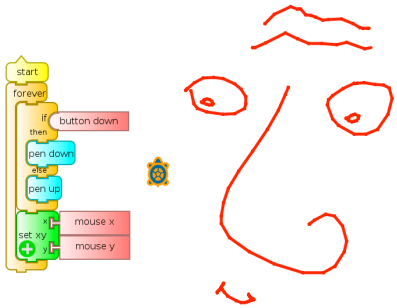


Figure 3. With nine blocks, you can write a functioning paint program.

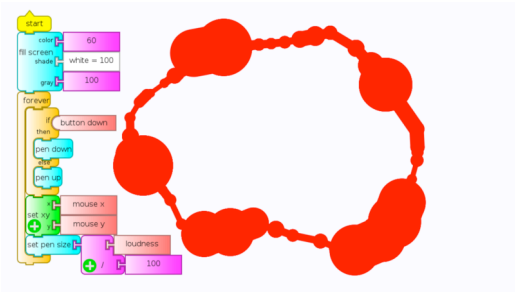


Figure 4. Sound Paint. The pen size varies in direct proportion to loudness.

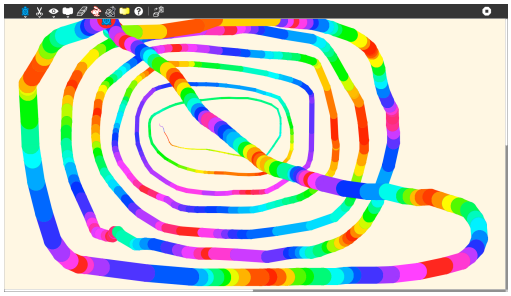


Figure 5. Time Paint. In another variant, a student from Colombia changed the pen size and color based on time.

5.2. Geometry

From the outset, geometry has been one of the more popular applications of Logo. Writing a program to draw a square and then to generalize that program to draw regular polygons is first introduction to programming that many of us who first learned to program using Logo would have encountered. Turtle Blocks expands upon this tradition by presenting programming challenges, such as the Turtle Confusion challenges designed by Barry Newell in 1988 [8]. Turtle Confusion presents 40 shape challenges to the learner that must be completed using basic Logo blocks. The students can author their own challenges as well. One of many variants, Turtle Flags challenges the learner to program flags (See Figure 6). In both cases, the user is invited to create their own challenges and share them with their peers.

5.3. Sensors and data

Seymour Papert's idea of learning through making is well supported in Turtle Blocks. According to Papert, "learning happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe" [9]. Research and development that supports and demonstrates the children's learning benefits as they interact with the physical world continues to grow [10, 11, 12, 13, 14]. In similar ways, children can communicate with the physical world using a variety of sensors in Turtle Blocks. Sensor blocks include keyboard input, sound, time, camera, mouse location, color that the

turtle sees. For example, children may want to build a burglar alarm (see Figure 7), and save photos of the thief to a file. Turtle Blocks also makes it possible to save and restore sensor data from a file. Children may use a "URL" block to import data from a web page.

Figure 6. Turtle Makes Flags presents more than 200 challenges (one for the flag of each nation) to the learner using Turtle Blocks. In the figure is the national flag of Austria.

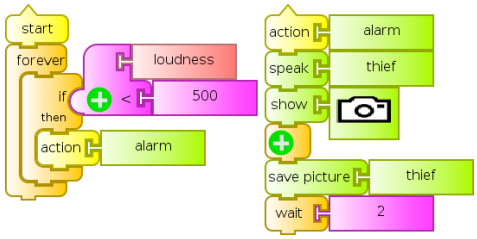


Figure 7. Using sensors. The loudness block is used to determine if there is an intruder. A loud sound triggers the alarm action: the turtle shouts "thief" and takes a picture of the intruder.

Using Turtle Blocks in the OLPC XO [15] provides additional functionality. The OLPC XO can measure external inputs with its microphone jack (resistive and voltage), and the XO 1.75 also includes an accelerometer. For example, children may want to build an odometer using a Hall-effect sensor and a magnet on a wheel spoke.

Teachers from the OLPC community have developed extensive collection of examples using Turtle Block sensors.[4] In one example, Tony Forster, an engineer from Australia, uses the XO laptop to experiments with motion, rolling a ball down a ramp. He made inexpensive sensor switches using aluminum foil (see Figure 8). Other examples can be found at [17].

5.4. Gamification

Children like playing games and creating games.[5] Turtle Blocks expands upon the idea of children as creators of games by giving them access to sensors, rich media, and the network (for multiplayer games). The Continent Game (Figure 9) written by a 3rd grade student loads a series of maps (downloaded from the Internet and then hand-colored) and a prompt by utilizing the "show" block on the Media Palette. A Snake Game (Figure 10) uses the keyboard as a sensor. Variants using other sensors to control the direction of the turtle is a popular variant. It is entertaining to watch (and listen to) a room full of children playing a racing game, where each player's turtle advances across the screen based on whistling at a pre-specified frequency. The "pitch" block is used in a simple loop: if pitch eq 800 [forward 10].

4 Guzmán Trinidad, a physics teacher from Uruguay, wrote a book, Physics of the XO, which includes a wide variety of sensors and experiments [16].

5 As Idit Harel demonstrated in 1988 [18], children like creating games (Harel had 4th graders at the Hennigan School in Boston, Massachusetts write games to teach 3rd graders fractions).

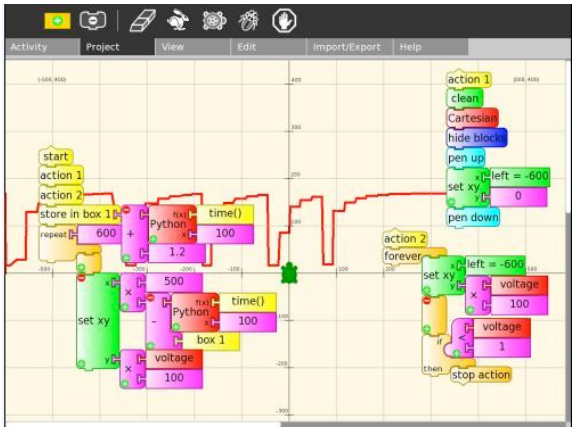


Figure 8. Measuring gravitational acceleration. Shown above is a triggerable oscilloscope with calibrated time base. "Action 1" clears the screen and draws the scale, the program waits in "action 2" until triggered by the first switch. The graphing then starts. Switches were placed at 20 cm, 60 cm, 100 cm, 140 cm, and 180 cm along a ramp of length 180 cm and height 25 cm.

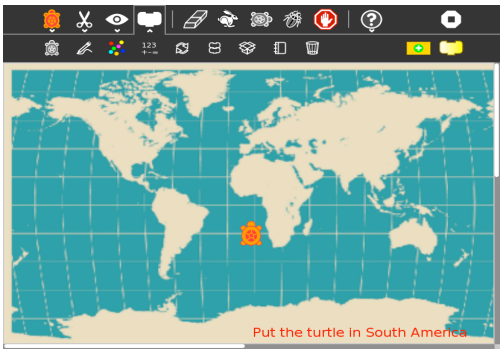


Figure 9. In the Continent Game (left), written by third-grader Jalen Basquiet, the user is prompted by the name of a continent and must move the mouse to that continent

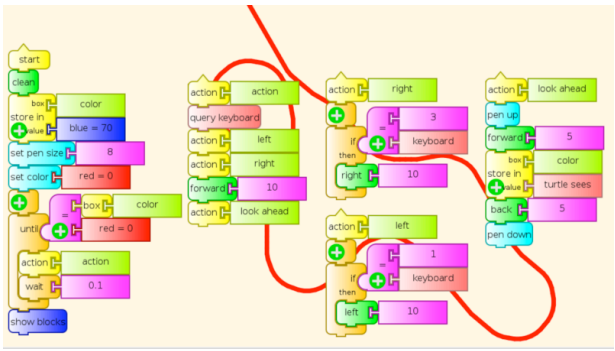


Figure 10. A Snake Game (right). The direction of the turtle is determined by keyboard input. The goal is to not cross the path that is drawn as the turtle moves forward.

5.5. Collaboration

Learning is not something that needs to be done alone: indeed there are numerous studies that demonstrate the power of children working together and learning from each other. Turtle Blocks incorporates a mechanism for sharing turtles and code over a network (either peer-to-peer or through a server) with just one mouse click that enables its users to share turtles (everything my turtle draws shows up on your canvas, everything your turtle draws, shows up on my canvas) and code (I can send you a stack of blocks). The collaboration model is used for programming multiplayer games (and multi-robot systems) but also has some more mundane applications. It takes just four blocks to program a webcam (Figure 11). Using the "show" and "speak" blocks while sharing turns Turtle Blocks into a multimedia chat program (Figure 12). A common elementary school project is to draw a picture of the solar system. Using Turtle Blocks collaboration, students can each program the orbit of a single planet (represented by a turtle used as a sprite) and collectively animate the solar system.

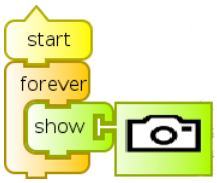


Figure 11. : A webcam (left) in four blocks.

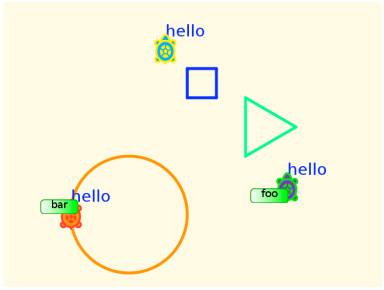


Figure 12. Multimedia chat (right). Each user sees their own turtle and the turtles of their collaborators. They see what each turtle draws and they can share stacks of blocks.

5.6. Robotics

Interacting with the physical work involves also being able to create physical artifacts and programming them to interact with the physical world. Turtle Blocks has been extended to support different robotic platforms. Plugins have been created to support robotic platforms such as Arduino, WeDo, NXT, and GoGo.

TurtleBots is a version of Turtle Blocks that includes plugins for Arduino and other robotics and sensor kits (see Figure 13). TurtleBots was written and is maintained by Alan Aguiar and his colleagues at FING, the National Engineering University in Montevideo, Uruguay. They also built a robotic platform called Butia [19] that extends the OLPC XO capabilities by adding sensors and actuators transforming it into a mobile robotic platform (see Figure 14). The goal behind Butia is to create an affordable platform that school or high school students can use to learn about programming by adding and debugging behaviors to a robot.

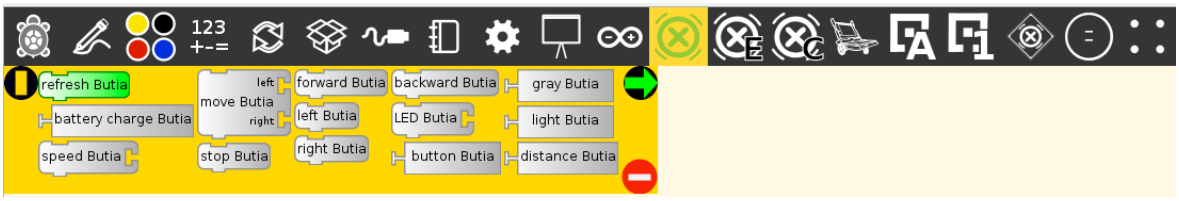


Figure 13. The TurtleBots' Palettes. From left to right: Arduino, Butia (3 palettes), Follow Me, Signs, NxT(2 palettes), Sumo, WeDo.'

5.7. Math and programming

Turtle Blocks is sufficiently powerful to use in exploring complex and powerful ideas from mathematics and computer science, e.g., recursion, fractals, databases, scientific visualization, etc. Because Turtle Blocks allows the dynamic assignment of variables and methods (similar to text-based programming languages but unusual in block-based languages) the Turtle Art programmer can explore concepts such as arrays and dictionaries. An eloquent example is shown in Figures 15 & 16: the result of throwing two dice is stored in a box. The value of that box is used to index an array of dynamically allocated bins.

Sometimes you need a new block. Recall the request for a square root block made by the teacher from Uruguay. Turtle Blocks has a "Python" block that lets you define new blocks on the fly (See Figure 17). More complex code can be loaded into a block from a file on the fly (see code). There is also a mechanism for saving a stack of blocks as a macro on a user-defined palette. The plugin mechanism is used for defining new palettes. And there is a "load" block that lets the user alter their programs on the fly, i.e., it enables the user to write a Turtle Blocks program using Turtle Blocks.

One of the goals of Turtle Blocks is to launch its users into the world of text-based programming. To facilitate that transition, we provide mechanisms for exporting projects as Logo and as Python. It is our hypothesis that by making a direct connection between block-based programming and text-based programming, the introduction to the text-based languages will be more meaningful.

Figure 14. Floor turtle 2014-style (by Cecilia Vilaró). The Butia robot powered by TurtleBots. Through the plugin mechanism, Turtle Blocks talks to an arduino that controls sensors and motors, including an actuator to implement "pen up" and "pen down" commands.

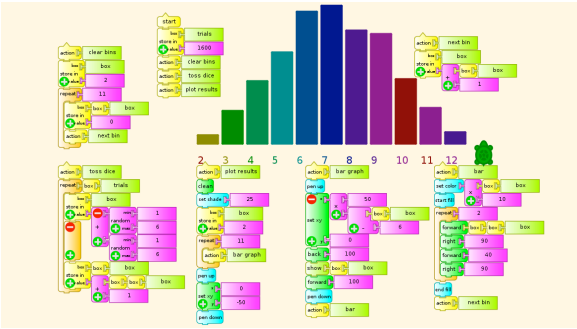


Figure 15. Plotting the results of throwing a pair of dice (example by Tony Forster).

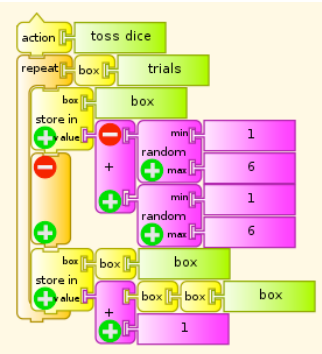


Figure 16. The value in box "box" is used to reference another box that stores the results.

5.7. Other

The plugin mechanism means Turtle Blocks can be extended. A popular extension involves currency: Blocks representing coins and bills can be used in combination with arithmetic operators or simply to move the turtle. In a similar vein, Turtle Blocks has been used as a vehicle to learn about nutrition is an extension that includes blocks to represent food and operators to extract their nutritional content. The turtle can go forward by the number of calories in a chocolate-chip cookie. Another popular plugin is the Physics Palette that enables the user to create precision machines for the Box2D physics simulator.

The final activity in our list of more than twenty things to do with Turtle Blocks is to write PowerPoint®. While we don't expect our users to write a fully featured presentation application, Turtle Blocks is more than adequate for making slide shows. (One of the authors has used Turtle Blocks exclusively for presentations since 2008.) Our mantra: "Don't just use PowerPoint, write PowerPoint."

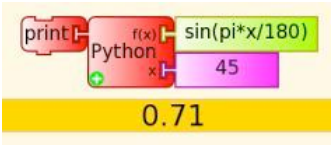


Figure 17. The Python block (top left) can be used to define new blocks on the fly simply by typing in a Python expression. Up to three arguments (x, y, z) can be used in the expression.

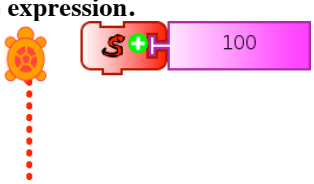
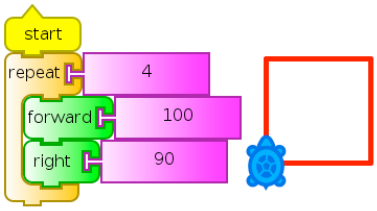


Figure 18. The Python code implements an algorithm to draw a dotted line. It is loaded into a "Python" block from a file. This user-defined Python block can be used like any other block.

```
def myblock(tw, line_length):
    ''' Draw a dotted line. '''
    try:  # make sure line_length is a number
        line_length = float(line_length)
    except ValueError:
        return

    if tw.canvas.pendown:
        dist = 0
        while dist + tw.canvas.pensize < \
              line_length:
            tw.canvas.setpen(True)
            tw.canvas.forward(1)
            tw.canvas.setpen(False)
            tw.canvas.forward(
                (tw.canvas.pensize * 2) - 1)
            dist += (tw.canvas.pensize * 2)
        # make sure we have moved exactly
        # line_length
        tw.canvas.forward(line_length - dist)
        tw.canvas.setpen(True)
    else:
        tw.canvas.forward(line_length)
    return
```



Logo code

```
window
to start
repeat 4 [forward 100.0 right 90.0]
end
```

Python code

```
#!/usr/bin/env python
from pyexported.window_setup \
    import *
tw = get_tw()
ACTION = {}
def start():
    turtles = tw.turtles
    turtle = \
    turtles.get_active_turtle()
    canvas = tw.canvas
    for i in range(int(4.0)):
        turtle.forward(100.0)
        turtle.right(90.0)
        yield True
    yield True
ACTION["start"] = start
```

Figure 19. A Turtle Blocks program to draw a square exported as Logo and as Python (Marion Zepf) [20]

6. Conclusion

> We learnt [sic] to make lots of different patterns by commanding the turtle to do things. E.g. Arc 90o and go forward. Using this we could create many different things things, including paint which you could control using your voice! I really enjoyed it because I never knew something so complicated could be really fun and quite simple as using a comande! [sic] I never thought I could be quite capable of doing something like that. —3rd grader from Singapore

Block-based programming languages have "low floors"; that is part of their appeal for novice programmers. However, most block-based languages also have a "low ceiling": by keeping things simple for their users, they preclude their users from reaching towards complexity and the ensuing powerful ideas that are inherent in Logo and other text-based languages. With Turtle Blocks, we have tried to raise the ceiling by (1) not shying away from giving our users undiluted access to powerful programming concepts such as variables and procedures; (2) providing multiple mechanisms for extensions to the language; and (3) providing explicit gateways out of blocks into text-based programming. Python export is a relatively recent addition to Turtle Blocks; it will be interesting to observe whether or not it is used.

Few children will go on to careers in computer science. But most children will live in a world that is in large part imprinted by computation. Some degree of experience and understanding of computational thinking will help children navigate and negotiate that world. The Turtle Blocks focus on replicating the day-to-day tools of computation (paint, presentation, chat, etc.) make these tools accessible: "Oh, that is all that paint is… nothing special. I can do that too." "So that is how PowerPoint works. No big deal." To Turtle Blocks programmers, the world of computing does not have to be accepted as it is found: they have the opportunity and responsibility to make it better.

We'd be remiss if, after discussing how children can program (with Turtle Blocks), we didn't discuss why children should program. Papert and Solomon discuss the exposing children to powerful ideas as a reason for programming. One powerful idea in particular, debugging, has been the focus of Solomon's work: "Debugging is the greatest opportunity for learning in the 21st century." The forgiving nature of programming provides a safe place to take intellectual risks: there is no one right way to do things, so it affords autonomy to the user. Through debugging, learners have an opportunity to refine their skills and to achieve mastery. Programming environments such as Turtle Blocks, which place few if any limits on the domains of investigation, let the learners choose projects that are personally meaningful: programming with a sense of purpose. Autonomy, mastery, and a sense of purpose together are powerful motivators to engage in constructing knowledge.

References

[1] Papert, S. and Solomon, C. (1971). "Twenty Things To Do With A Computer" *MIT AI Laboratory Memo 248.*

[2] *Lego Mindstorms,* http://en.wikipedia.org/wiki/Lego_Mindstorms

[3] *Turtle Art "Intro".* http://turtleart.org/programming/intro

[4] *Turtle Blocks "Getting Started",* http://wiki.sugarlabs.org/go/Activities/TurtleArt#Getting_Started

[5] Resnick, M., and Silverman, B. (2005). "Some Reflections on Designing Construction Kits for Kids." *Proceedings of Interaction Design and Children Conference*, Boulder, CO.

[6] Papert, A. and Silverman, B. (2011). "Art, Literature, and Turtles". *5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives*, ISSEP 2011, Bratislava, Slovakia, October 26-29, 2011.

[7] Benenson, A. (1990). *VideoLogo: Synthetic Movies in a Learning Environment*, MS Thesis in Media Arts and Sciences, MIT

[8] Newell, B. (1988) *Turtle Confusion: Logo Puzzles and Riddles*, Curriculum Development Centre, Canberra, Australia

[9] Papert, S. (1993). *The children's machine: Rethinking schools in the age of the computer*. New York: Basic Books. p 142–143.

[10] Bers, M.U., Flannery, L.P., Kazakoff, E.R, & Sullivan, A. (2014 - in press) *Computational thinking and tinkering: Exploration of an early childhood robotics curriculum*, 72, 145–157.

[11] Resnick, M., & Rosenbaum, E. (2013). Designing for Tinkerability. In Honey, M., & Kanter, D. (eds.), *Design, Make, Play: Growing the Next Generation of STEM Innovators*, pp. 163–181. Routledge.

[12] Resnick, M., Berg, R., and Eisenberg, M. (2000). "Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation." *Journal of the Learning Sciences*, vol. 9, no. 1, pp. 7–30

[13] Bers, M. & Urrea, C (2000) "Technological Prayers: Parents and Children Exploring Robotics and Values." In *Robots for Kids: Exploring New Technologies for Learning Experiences*. Edited by A. Druin & J. Hendler. NY: Morgan Kaufman, pp. 194–217.

[14] Resnick, M., Martin, F., Sargent, R., and Silverman, B. (1996). "Programmable Bricks: Toys to Think With." *IBM Systems Journal*, vol. 35, no. 3-4, pp. 443–452.

[15] *OLPC CL1 Hardware Design Specification* (2008) http://wiki.laptop.org/images/7/71/CL1A_Hdwe_Design_Spec.pdf pp 13–14

[16] Trinidad, G. (2013). *Physics on the XO*. http://www.ceibal.edu.uy/Documents/Articulos/FC3ADsica20con20XO20GuzmC3A1n20Trinidad20x10.pdf

[17] Forster, T. and Trinidad, G. (2010). *Using Turtle Art with Sensors*, http://wiki.sugarlabs.org/go/Activities/Turtle_Art/Using_Turtle_Art_Sensors

[18] Harel, I.and Papert, S. (1991). "Software design as a learning environment". *Constructionism*. Norwood, NJ: Ablex Publishing Corporation. pp. 51–52. ISBN 0-89391-785-0.

[19] *Butiá Project.* https://www.fing.edu.uy/inco/proyectos/butia/

[20] Zepf, M. (2013) *Turtle Blocks Python Export* http://www.google-melange.com/gsoc/project/google/gsoc2013/mzepf/6385413778309120