Learning Geometry using Computational Thinking, Scratch and Python Turtle

Paolo Musmarra, *Universita' di Salerno*, pmusmarra@gmail.com

Maria Rosaria Del Sorbo *L. Da Vinci High school, Poggiomarino (NA)*,marodeldue@gmail.com

Abstract. Algorithmic thinking and computer programming are mandatory student skills that are to be implemented in all types of education, and the mathematics education especially. Although over the past decades Curricula have been updated, many mathematics teachers still lack competence and experience in computer programming and they often have to integrate their skills with new technologies and elements of computer science as well as coding. Therefore, there is a need for specific teacher resources, such as simple but efficient readily developed education programs that can easily be adapted and applied, either as whole or as extension for teachers in all types of classrooms.

The contribution of this paper is twofold: firstly, we present an education program that aims to introduce basic programming concepts, such as loops and logical conditions and usage of logical operator, as and/ or to lower middle school students through their preexisting knowledge of mathematics and geometry. Secondly, we report a pilot study of an educational program carried out by two teachers in their respective classrooms over a period of two months in a Italian lower middle school and in the first year of High School. Students were guided to develop their problem solving skills using algorithmic strategies with the main functions of Scratch and to increase their spatial awareness by tackling Python Turtle programming to draw plane geometric figures.

Keywords: Block programming, Programming, Computational Thinking, Mathematics Education, Scratch, Python Turtle, Geometry.

1 Introduction

This paper investigates the use of Computational Thinking and programming to teach geometry concepts using Scratch and Python Turtle. Computational Thinking is an educational practice that focuses on developing students' skills in problem-solving, abstract thinking, pattern recognition and logical reasoning through programming and algorithmic patterns. This area of learning is being supported by various initiatives such as "CS for All" and ISTE's Standards for Students in Computational Thinking, as well as the use of tools such as robotics, 3D printing, microprocessors and intuitive programming languages (Angeli, 2020). Computational thinking is often related to problem solving in an algorithmic manner, i.e., defining problems and breaking them up into smaller solvable stepwise parts, much like the structure and purpose of a computer code (Barr, V. and C. Stephenson, 2011). Several studies argue that programming can motivate students to study mathematics and increase their problem solving skills (Barak, M. and M. Assal, 2016; Sinclair, N. and M. Patterson, 2018).
It is important for students, since primary levels, to understand and mastering the concepts of geometry, including angle, line, shape, translation and transformations. However, traditional methods of teaching geometry can be difficult for current students to understand. By using computational thinking and programming, students can gain a deeper understanding of geometry concepts. Scratch and Python Turtle are two programming languages that are often used to teach computational thinking and programming (Iskrenovic-Momcilovic, O. (2020); Rahim, 1997).

Scratch is a block-based programming language designed for beginners. It allows students to easily create interactive stories, animations, and games since their first years. Python Turtle is a text-based programming language that is more advanced than Scratch. It allows students to create complex programs and graphics.
The experimental part of this work is referred to a pilot study of an educational program carried out by two teachers in their respective classrooms over a period of two months in a Italian lower middle school and in the first year of High School. 45 students were challenged, individually and in small groups, to solve geometric problems exploiting algorithmic strategies with the main functions of Scratch and Python Turtle programming. This paper finally discusses the benefits and challenges of geometry teaching by computational thinking and programming to students between 12 to 15 years, comparing student's results before and after teaching activities with the results of a control group.

2 Theoretical Framework

Seymour Papert is an outstanding figure in the field of educational technology (Goldberg, 1991). His seminal works (among others Papert, 1972; Papert, 1980; Papert, 1993) have been widely cited and applied to various aspects of education. Papert's theories of computational thinking have been particularly influential in the development of educational applications that seek to promote problem solving and critical thinking skills. This paper is based on the theoretical framework of Papert's theories of computational thinking in education.
First, it is important to understand what Papert meant by computational thinking. Papert (Papert, 1980) defined computational thinking as "the ability to use abstractions, algorithms, and models to solve problems and create solutions." He proposed that in order to effectively apply computational thinking to any problem or situation, there are four core components: abstraction, algorithms, models and problem solving. Abstraction involves breaking down complex tasks into smaller, more manageable components. Algorithms are the step-by-step instructions that can be used to solve a problem. Models are the representations of data or concepts that can be used to explain a concept or analyze a problem. Problem solving requires the ability to use all of the previous components with the aim of developing a solution to a problem.
In order to apply Papert's theories of computational thinking to education, it is important to plan activities and tasks that allow students to practice each of the four core components. Abstraction activities should involve breaking down a complex problem into smaller, easier to understand pieces. Algorithmic activities should teach students to develop step-by-step instructions to solve a problem. Modeling activities should inspire to use representations of data or concepts to explain or analyze a problem. Lastly, problem-solving activities should require to apply all of the previously mentioned components in order to develop a solution to a problem.
In addition to activities that allow students to practice computational thinking, it is important for teachers to use instructional strategies that promote the development of computational thinking skills. For example, teachers can use inquiry-based learning to encourage students to explore complex problems. This type of learning requires students to identify and analyze problems, develop solutions, and evaluate their solutions. Teachers can also use project-based learning, which requires students to complete projects involving real-world problems. This type of learning encourages students to use their computational thinking skills to develop and evaluate solutions.
It is essential for educators to create a learning environment that is conducive to the development of computational thinking skills. This environment should encourage students to take risks and experiment with new ideas and solutions. It should also provide students with opportunities to collaborate and engage in meaningful dialogue regarding their solutions.
In conclusion, Papert's theories of computational thinking provide an important framework for the application of educational technology in the classroom. Through the use of activities and instructional strategies that promote the development of computational thinking skills, as well as a

learning environment that encourages experimentation and collaboration, educators can effectively apply Papert's theories to create an engaging and meaningful learning experience for their students.

3 Method

Seymour Papert introduced a new way of working with geometry in the 1980s, which was called Turtle Geometry (Kynigos, C. and M. Grizioti,2016; Papert, S.A.,1980). Papert and his colleagues developed the programming language Logo, which has been used frequently and widespread in schools since the 1980s (Papert, S.,1993). Logo is adapted to the use of children, as a mean for young students to explore advanced geometrical figures and patterns, through the control of a physical turtle robot, which could be programmed to move around on a piece of paper to create drawings. Students could relate their body motion to the motion of the robot and transfer this knowledge into the work of learning formal geometry, including the conceptualization of angles.
In this paper, we propose a teaching strategy for computational thinking linked to geometric understanding and drawing of geometric figures via block-based programming. The developed education program is inspired by Scratch (K. Brennan, M. Resnick, 2012) and Turtle geometry.

In other words, we proposed both the possibility of programming to fully understand how geometric figures are drawn from a computational thinking point of view, and conversely, we will try to propose the use of simple geometric figures to understand the principles of programming and computational thinking. Rather than using programming as a mean to explore geometry, we propose to use familiar geometric figures to learn the concepts using an algorithmic procedure. Teachers may apply the education program as an entry level to basic level programming concepts through geometrical understanding already owned by the students, before introducing formal, block-based programming. The programming activities were performed by Scratch and Python Turtle.

This article reports on a pilot study in which the two researchers who authored this article carried out teaching activities in the last grades of middle school and the first year of high school (12-15 years old). The classes were conducted over a period of about 4 weeks with two and a half lecturing hours per week. The activities carried out in each week were evaluated at the end of the same week, so that the activities proposed in the following week's lessons could be improved based on the students' reaction. The students' foreknowledge of geometry and mathematics was fairly homogeneous and in general none of them had ever used a programming language in a systematic way.

3.1 Activities

After brainstorming in the groups and equipping the students with all the beginning information about the learning path, the first part of the lessons was devoted to the presentation of the working environments. Scratch and Python Turtle programming was presented at the initial part of the course.
The research questions this study attempt to answer, are:
- RQ1: How can learning geometry be fruitful to learn basic programming concepts?

- RQ2: Can computational thinking and engagement in programming simplify learning to solve geometric problems and make teaching geometry to 12-15 years old more effective?

- RQ3: How were the basic concepts of computational thinking grasped by the students after they have attended an introductory course in programming?

Related to RQ3, we use the definition of computational thinking proposed by (Selby, C. and J. Woollard, 2013), namely that it is a focused approach to problem solving, incorporating thought processes that utilize decomposition, abstraction, evaluation, algorithmic design, and generalizations.

3.2 Scratch

Brennan and Resnick's study focuses on CT in the Scratch programming environment (K. Brennan, M. Resnick, 2012). The framework is organized along three dimensions: concepts, practices, and perspectives. Their seven concepts resonate specifically with programming environments. Their computational practices section is strongly focused on students learning to work actively with code. The computational perspectives category focuses on students expressing themselves creatively through computation, enriching computational experiences through interaction with others, and questioning the complicated nature of technology.
This study successfully contextualized CT to math and geometry contexts. Thereafter, teachers explained the Turtle library, and that this library contains commands to move a robot object (an arrow by default) around the screen, and that a set of consecutive commands would eventually constitute a computer code, which could draw a desired geometrical figure on a virtual canvas when executed.
This laboratory activity is designed to help K-8 and K-9 students to understand the basics of geometry and black-based programming by an easy-to-use software. In Scratch environment an action can be considered and executed by assembling blocks of different colors based on their features and functions. The activity started by introducing Scratch and how to use the blocks. Students then began drawing simple geometrical elements such as points and lines. Once they have drawn an initial example of lines moving randomly, students were challenged to create simple geometric figures. Students first obtained a kind of rainbow of free lines, and then from the blocks used with Scratch, they tried to make first a triangle and then figures with a larger number of sides.

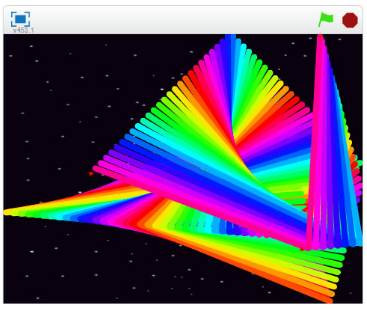


Fig.1

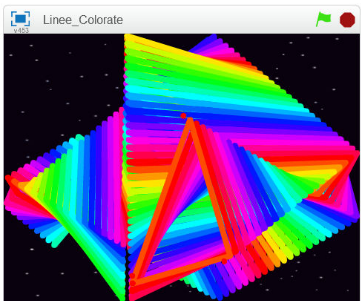
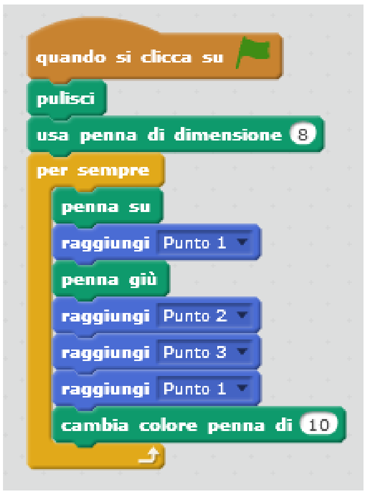


Fig.2



```
quando si clicca su
pulisci
usa penna di dimensione 8
per sempre
    penna su
    raggiungi Punto 1
    penna giù
    raggiungi Punto 2
    raggiungi Punto 3
    raggiungi Punto 1
    cambia colore penna di 10
```

Fig.3

At the end, students tried to make different geometric figures from the triangle, and then figures with a larger number of sides. A twofold effect was observed, which is a noteworthy result of this work. The use of Scratch, helped students becoming familiar with the construction of geometric figures with an increasing number of sides, and the use of geometry helped students learning the basic concepts of programming and understanding the main elements of computational thinking.
A fragment of the code used with Scratch is given in fig. 3: each point represents a vertex of a geometric figure, and using the reach Point block, the line joining two vertices is drawn, thus obtaining a geometric figure with n-sides.

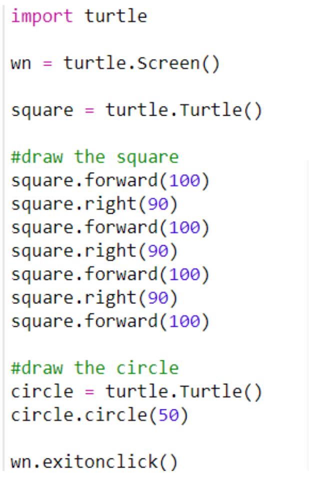
The 'Reach Point' instruction allows students to draw a line from one vertex to another in the figure. Students drew the 3 sides of triangles using code similar to the one fig. 3 one after. By Iterating this process they can obtain more complex n-sided figures. Practising geometric concepts, students also learned to understand the principles of Scratch operation and computational thinking.

3.3 Python Turtle

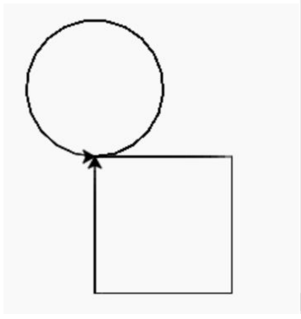
Python Turtle (https://docs.python.org/3/library/turtle.html) is a library that is part of the Python programming language. It allows users to create graphical applications using a simple and small set of commands. Using Turtle, users can draw shapes and patterns, create animations, and even play games. Turtle is a great way for beginners to learn the basics of computer programming. Turtle is based on the Logo programming language, which was developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967. Logo was designed to help teaching children the basics of programming. The Turtle library was created to give Python users access to this same set of tools. Users can exploit Turtle to create shapes, patterns, and even characters (. The library provides a set of commands that allow users to control the turtle's movements. This includes commands such as forward, back, left, and right. In addition, Turtle provides commands for drawing lines, circles, rectangles, and other shapes. Turtle also provides a set of commands for creating animations. For example, users can make the turtle move around the screen in a specific pattern or make it move in a particular direction. They can also create a loop to make the turtle repeat the same action multiple times. In addition to shapes and animations. Turtle includes a set of commands for creating simple games. These commands allow users to control the turtle's movements to create the game. Using Turtle is easy for beginners. The library has extensive documentation and tutorials that provide step-by-step instructions for creating shapes, patterns, and games. Furthermore, the library is open source, so users can access the source code and customize it to meet their needs.

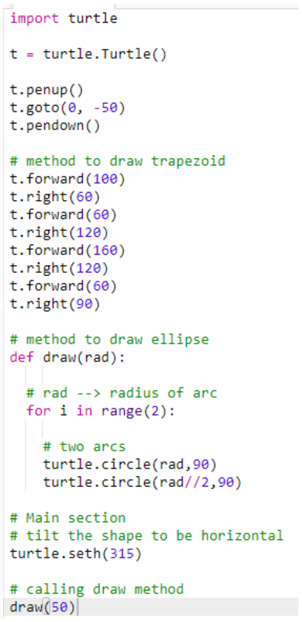
Overall, Python Turtle is a great way for beginners to learn the basics of computer programming, it is easy to use and provides users with a powerful set of tools for geometry learning (Laura-Ochoa, 2022; Svistkov, 2021; Nanavati, 2020; Svistkov, 2021)

This lab is designed to help K-8 and K-9 students learn the basics of geometry using the Python Turtle library. Students can learn how to draw simple shapes with the Turtle and use its commands to create designs of increasing complexity. The lab begins by introducing students to the Turtle graphics library and the basic commands used to draw shapes. After the basics are covered, the students used Turtle commands to draw a square and a circle. The lab will then move on to more complex shapes such as trapezoid and ellipse. The lab will then move on to draw a polygon with a variable number of sides and then to more complex concepts such as drawing patterns, creating complex designs, and drawing 3D shapes. At the end of the lab, the students were able to use the Turtle commands to draw any shape they can imagine, and create complex designs. They gained an understanding of the basic principles of geometry and be able to apply the concepts to real-life problems (Fig. 4a, 4b, 4c)



```
import turtle

wn = turtle.Screen()

square = turtle.Turtle()

#draw the square
square.forward(100)
square.right(90)
square.forward(100)
square.right(90)
square.forward(100)
square.right(90)
square.forward(100)

#draw the circle
circle = turtle.Turtle()
circle.circle(50)

wn.exitonclick()
```





```
import turtle

t = turtle.Turtle()

t.penup()
t.goto(0, -50)
t.pendown()

# method to draw trapezoid
t.forward(100)
t.right(60)
t.forward(60)
t.right(120)
t.forward(160)
t.right(120)
t.forward(60)
t.right(90)

# method to draw ellipse
def draw(rad):

  # rad --> radius of arc
  for i in range(2):

    # two arcs
    turtle.circle(rad,90)
    turtle.circle(rad//2,90)

# Main section
# tilt the shape to be horizontal
turtle.seth(315)

# calling draw method
draw(50)
```

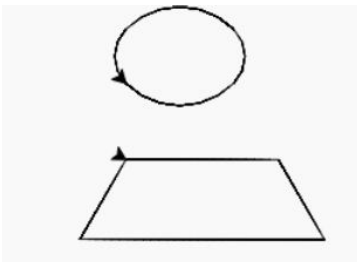
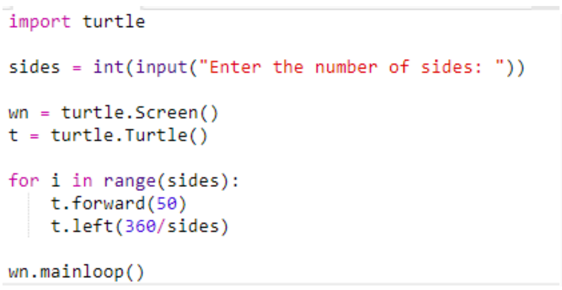


Fig.4 a **Fig.4 b**



```
import turtle

sides = int(input("Enter the number of sides: "))

wn = turtle.Screen()
t = turtle.Turtle()

for i in range(sides):
    t.forward(50)
    t.left(360/sides)

wn.mainloop()
```

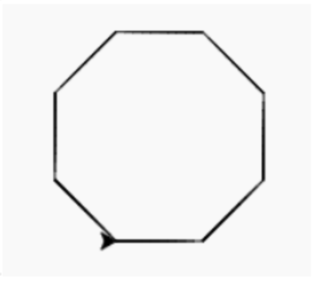


Fig.4 c

4. Results

At the end of the activities, a questionnaire was administered to the students with a twofold purpose. On the one hand, it was intended to assess the improvement of students' skills in plane Euclidean geometry; on the other hand, their reactions and enjoyment of this educational proposal were collected. The following are the most significant comments from some students that describe the general sentiment very well: Student A: "It was really interesting to learn through geometry how to be able to use Scratch and Python Turtle Graphics and their fundamental instructions.", Student B: "We did not imagine that we were able to learn so quickly to use the fundamental functions of programs like Scratch and Turtle Library and to be able to write programs. So geometry seems easier!" Student C: "I found that it is super fast to draw geometric figures, with very few lines of code," Student D: "At first I had problems, I always got figures wrong, but from the mistakes I learned a lot." Student E: "I would like to keep learning this way, it's more fun and I see the results right away." Student F: "I don't like geometry and I didn't know that coding could be fun for me. I would like to do homework in class using coding."

The group of 45 students (20 K8 and 25 K9), at the beginning of the activities (pre-test), had, in geometry, equal score on average, to the control group. Comparing student's results after teaching activities (post-test), we report the most important result of our study: K8 group reported an increase on average of 18%, and K9 an increase on average of 21%.

5. Conclusions and future work

The results of the pilot study show that the program was successful in introducing to k-8 and k-9 level students some basic programming elements in Scratch and that they were able to apply them to solve problems, besides to achieving a deeper awareness in geometric concepts. The students also developed their spatial awareness and problem solving skills by using Python Turtle programming. The teachers reported that they found the program to be effective in teaching the students and that it was easy to adapt and apply to their classrooms. Overall, this paper provides a valuable resource for teachers in all types of classrooms, as it presents a simple but effective education program that can easily be adapted and applied to introduce basic programming concepts to students. The results of the pilot study demonstrate the effectiveness of the program in teaching students and increasing their problem solving skills and their motivation towards geometry with respect a control group. This study is a starting point to research about computational thinking and the interaction between computer science and maths in education from the youngest pupils to college level.

References

Angeli, C., & Giannakos, M. (2020). Computational thinking education: Issues and challenges. Computers in human behavior, 105, 106185.

Barak, M. and M. Assal, Robotics and STEM learning: students' achievements in assignments according to the P3 Task Taxonomy—practice, problem solving, and projects. International journal of technology and design education, 2016. 28(1): p. 121-144.

Barr, V. and C. Stephenson, Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? Acm Inroads, 2011. 2(1): p. 48-54.

Brennan K. and M. Resnick, New frameworks for studying and assessing the development of computational thinking,in Proceedings of the 2012 Annual Meeting of the American Educational Research Association (2012), Vol. 1, p. 135.

Calder, N. S. (2018). Using scratch to facilitate mathematical thinking. Waikato Journal of Education, 23(2), 43–58. https://doi.org/10.15663/wje.v23i2.654.

Grover, S. and R. Pea, Computational Thinking in K—12: A Review of the State of the Field. Educational researcher, 2013. 42(1): p. 38-43.

https://www.media.mit.edu/publications/newframeworks- for-studying-and-assessing-the-development-of computational-thinking/.

Iskrenovic-Momcilovic, O. (2020). Improving geometry teaching with scratch. International Electronic Journal of Mathematics Education, 15(2), em0582.

Jackson, J. L., Stenger, C. L., Jerkins, J. A., & Terwilliger, M. G. (2019). Improving abstraction through python programming in undergraduate computer science and math classes. *Journal of Computing Sciences in Colleges*, 35(2), 39-47.

Kaufmann, O.T. and B. Stenseth, *Programming in mathematics education.* International journal of mathematical education in science and technology, 2021. **52**(7): p. 1029-1048.

Kynigos, C. and M. Grizioti, *Programming Approaches to Computational Thinking: Integrating Turtle Geometry, Dynamic Manipulation and 3D Space.* Informatics in Education An International Journal, 2018. **17**(2): p. 321-340.

Laura-Ochoa, L., & Bedregal-Alpaca, N. (2022). Incorporation of computational thinking practices to enhance learning in a programming course. *International Journal of Advanced Computer Science and Applications*, 13(2).

MacFerrin, M. (2012). Turtle Geometry on the Sphere: The Turtle Finally Escapes the Plane (*Doctoral dissertation, University of Colorado at Boulder*).

Nanavati, A., Owens, A., & Stehlik, M. (2020, February). Pythons and Martians and Finches, Oh My! Lessons Learned from a Mandatory 8th Grade Python Class. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 811-817).

Papert, S. (1972). Teaching children thinking. *Programmed Learning and Educational Technology*, 9(5), 245-255.

Papert, S.A., *Mindstorms: Children, Computers, and Powerful ideas*. 1980, New York: Basic books, Inc., Publishers.

Papert, S. (1993). The children's machine: Rethinking school in the age of the computer. New York.[8] Kafai, Y.B. and Q. Burke, *Computer programming goes back to school.* Phi Delta Kappan, 2013. **95**(1): p. 61-65.

Rahim M. H., (1997) Turtle Geometry at a Secondary School Level, *Computers in the Schools*, 14:1-2, 129-142, DOI: 10.1300/J025v14n01_10.

Selby, C. and J. Woollard, *Computational thinking: the developing definition.* University of Southampton, 2013.

Sinclair, N. and M. Patterson, *The Dynamic Geometrisation of Computer Programming.* Mathematical thinking and learning, 2018. **20**(1): p. 54-74.

Svistkov, A.I., SutchenkovA. A., and Tikhonov A. I. , (2021) "STEM and STEAM Technologies in Problem Solving with Python", 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE), Moscow, Russia, pp. 1-6, doi: 10.1109/REEPE51337.2021.9388001.

Williams, L., & Mead, B. (2022). “Literacy from Python” Using Python for a Proposed Cross-curricular Teaching and Learning Model. In *Digital Transformation of Education and Learning-Past, Present and Future: IFIP TC 3 Open Conference on Computers in Education, OCCE* 2021, Tampere, Finland, August 17–20, 2021, Proceedings (pp. 41-53). Springer International Publishing.