The Effect of the Modality on Students' Computational Thinking, Programming Attitude, and Programming Achievement

Ibrahim Cetin[1]

Tarik Otu[1]

[1]Computer Education and Instructional Technology Department, Education Faculty, Abant Izzet Baysal University, Bolu, Turkey

[2]Orhangazi Middle School, Bolu, Turkey

DOI: https://doi.org/10.21585/ijcses.v6i2.170

Abstract

The purpose of the current study was to explore the effect of modality (constructionist mBlock, Scratch, and Python interventions) on six-grade students' computational thinking, programming attitude, and achievement. The pre-test and post-test quasi-experimental design was used to explore the research questions. The study group consisted of 105 six grade students from three different classes. A constructionist learning environment was formed for Scratch, mBlock, and Python groups. All groups were given 8 week-instruction. Instruction included two forty-minute sessions each week. The data were collected through the programming achievement test, computational thinking test, and computer programming attitude scale. The results of the study showed that mBlock group outperformed the Scratch and Python groups with respect to computer programming attitude. Students who attended mBlock and Scratch groups had higher levels of programming achievement than those of the students who attended the Python group. No significant differences with respect to computational thinking were observed between the groups. This study has implications for educators who are teaching computational thinking and programming. Further research was recommended to explore the effect of modality.

Key Words: Modality; computational thinking; programming; constructionism

1. Introduction

Computational thinking and programming have become important skills for educators around the world. Researchers are searching for the best practices to help students improve their computational thinking and programming (Tikva and Tambouris, 2021). Educational institutions are adapting computational thinking and programming concepts into their regular curriculums. It is contended that computational thinking is related to problem-solving, abstraction, critical thinking, and creativity (Korkmaz, Cakir, and Ozden, 2017; Cakiroglu, Cevik, Koseli, and Aydin, 2021; Panskyi, Rowinska,and Biedron, 2019). Nevertheless, computational thinking is a relatively new area in educational research. There is no commonly agreed-upon definition of computational thinking in the literature yet. Researchers have proposed different definitions for computational thinking.

The term was first used by Seymour Papert (1980). Papert used the term computational thinking without providing a definition. He considered computational thinking in the context of the educational theory called constructionism which is a reconstructed form of Paiget's constructivism (Papert, 1993). Papert suggested that computational thinking can be used to help students improve their mathematical knowledge. He considered programming as a medium to construct a relatively concrete product. Students can think on a relatively concrete product to improve abstract mathematical knowledge.

Ed Dubinsky (1995), like Papert, contended that formal mathematical thought should be grounded in experience. Dubinsky and his colleagues reconstructed Piaget's constructivist theory of learning in the context of collegiate mathematics education (Arnon et al., 2013). They constructed APOS theory. They did not use the term computational thinking, but they stressed the power of computing in mathematics education. The core of APOS theory is related to reflective abstraction. Dubinsky considered programming as a unique tool to help students construct necessary mathematical abstractions. Moreover, it was contended that the nature of abstraction in mathematics is the same in computational thinking (Cetin and Dubinsky, 2017). Papert and Dubinsky are important

researchers in mathematics education in that they both considered computational thinking from their systematic learning theory perspectives.

Before educational theorists, computer scientists emphasized the terms algorithm and algorithmic thinking. The term algorithmic thinking was used by the researchers before computational thinking to express the core of computer science (Denning, 2017). Knuth (1985, p.172) contended that "… Computer Science is the study of algorithms" and he stressed the importance of algorithmic thinking in the context of computer science. Newell, Perlis, and Simon (1967) took a different perspective and proposed that computers are not only tools, but there are also phenomena surrounding computers. Computer science deals with phenomena and algorithms and the hardware is the important element of the phenomena.

Computational thinking was first defined by Wing (2006) as the application of computer science concepts to solve problems design systems and understand human behavior. Aho (2012, p.832) emphasized the role of the computational model in computational thinking and defined computational thinking "…to be the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms". Cuny, Snider, and Wing (2010) modified Wing's early definition and considered computational thinking as "the thought process involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (as cited in Wing, 2011; p. 20). This definition is taken as a base for this study. Brennan and Resnick (2012), depending on their Scratch teaching experience, developed a computational thinking framework including computational concepts (programming concepts), computational practices (practices used in problem-solving), and computational perspectives (the self-reflection on computing practices).

The theoretical and historical perspectives of computational thinking are important. Moreover, an instructional perspective that provides means to help students improve their computational thinking and programming skills is important for educators. Programming and computational thinking are not easy for students to comprehend. There are studies that report students have trouble in learning programming (Chao, 2016; Moons and Backer, 2013; Sáez-López, Román-González, and Vázquez-Cano, 2016). When beginners learn programming and computational thinking there are possible pathways for them to follow. They can be introduced to computational thinking or programming with the help of block-based programming, robotics programming, text-based programming, and computer science unplugged approach. Educators can also use a blended approach by mixing some of these ways. Nevertheless, there are not enough guidelines to pick one of the modalities to help students learn computational thinking and programming. This study will explore the effect of modality (block-based, text-based, and robotics) in the context of six-grade students to start filling this gap.

1.1 Literature Review

Pioneer computer scientist Dijkstra (1982) stated that "The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities" (p. 129). However, how the tool will be used is not self-evident in the tool itself. Herrmann (2003) stated that "A technical system is not a product of its own but is made and controlled from outside… Technical systems serve purposes which do not lie within themselves but are assigned from other systems" (p. 62). Therefore, the pedagogical approach behind the tool or modality should be shortly explained before giving the details of the related literature about modalities. In the current study, constructionism will be utilized as a pedagogical approach. Constructionism was constructed on the theory of Piaget's constructivism. Constructivism is related to the origins and development of knowledge (Piaget, 1964). In the learning process, an individual acts on an internal/external object, transforms it, constructs new knowledge at a higher level of plane, and integrates the new knowledge with the existing ones at the higher level of plane. The mental mechanism is the reflective abstraction in the development of logico-mathematical knowledge. The mental structure developed through reflective abstractions is called schema. Schema is a more or less coherent collection of mental structures. Individuals actively construct their own knowledge or mental structures. Papert (1980), the constructor of the constructionist approach, agrees with Piaget's theory of knowledge construction. He furthers it with the idea of the development of a concrete entity. Abstract concepts can be represented as computer programs or algorithms. These programs are concrete in the sense that their results, e.g., moving turtle, robotic tasks, or games, can be seen immediately after running them. Therefore, individuals can deal with abstract concepts through concrete means. Individuals can share their entities/programs with others to collectively think on it. The computer becomes a cultural tool that supports individuals' learning as in the case of the support for the learning mother tongue.

Constructionism is the pedagogical approach behind all the modalities in the current study. The modalities are constiructionist Scratch, Python, and mBlock learning environments. There are many block-based programming environments available for instructional purposes. Scratch, Alice, and App Inventor are the ones that are used

frequently (Hu, Chen, and Su, 2021). When beginners start learning to program with text-based programming tools they need to handle complicated syntax issues. They need to memorize programming statements, write codes in a correct way and debug the program when needed. In contrast to text-based programming environments, block-based programming environments are intuitive (Xu, Ritzhaupt, Tian, and Umapathy, 2019). Beginners can construct games, animations, and mobile applications by using block-based environments. These are complex programming products that are hard to develop with text-based languages. Block-based programming environments can provide beginners with concrete and joyful experiences (Topalli and Cagiltay, 2018). Beginners can run the script and see the result on the screen. They can develop programs that are interesting for themselves (Mladenović, Mladenović, and Žanko, 2020). Resnick et al. (2009) summarized the expected features of block-based programming environments for beginners as low floor (easy to get start), high ceiling (allows beginners to construct complex projects), and wide walls (supports the development of meaningful products).

Programs are represented as plain text in text-based programming. There is a variety of paradigms in text-based programming and text-based programming is the norm in the industry (Kandemir, Kalelioğlu, and Gülbahar, 2021). Python, Java, and C++ can be given as examples of text-based programming languages. There is a belief that text-based programming is harder for beginners (Kölling, 2015). Nevertheless, this does not directly mean that text-based programming should not be used for the first programming experience. Some arguments support the use of text-based programming and discourage block-based programming environments for beginners. Mihci and Donmez (2017) contended that some university students are not interested in block-based programming environments. They chose text-based programming environments since they believe that text-based programming environments are the industry standard that might help them for their future career. There are several approaches for educational text-based programming that aims to introduce students in a more beginner-friendly way. These approaches are mini-language, sub-language, visualization, and frame-based programming (Brusilovsky et al., 1997; Cetin, 2020; Kandemir, Kalelioğlu, and Gülbahar, 2021; Kölling, Brown, and Altadmri, 2017). In the current study, the mini-languages approach will be focused on since it was used in one of the interventions in the study. In mini languages, there is an actor (e.g., turtle or robot) in a microworld. Students control the actor by using commands of the mini language. The mini language generally includes simple commands and basic programming structures. Students can see the immediate result of their program via the actor in the microworld. Mini languages have current implementations in programming education. For example, Python has a turtle library which is a relative of Logo turtle. Students can use this library to get the basics of Python and experience programming with the mini-language approach.

Educational robotics have a place in computational thinking and programming education. There is a variety of robotic kits that can be used for this purpose. Some of these tools are Lego robots, Bee-Bot, MBot, and Arduino kits. Beginners can use a prebuild robot (Bee-Bot); they can build a robot by a using controller unit, motors, sensors, cables, and technic elements (Lego Mindstorms EV3 and MBot); or they can build a robot by using a microcontroller, basic electronic elements, modules, sensors, motors, cables, and mechanic elements (Arduino kits). Robotic kits can be programmed by using text-based (MicroPython and Arduino IDE), block-based (Scratch, mBlock, ArduinoBlocks), hybrid (RobotC) programming environments/libraries, and by just pushing buttons on the kit. Programs for robotic kits can be written on computers just as in the case of block-based and text-based programming, then the program can be downloaded to the robot. After the program is downloaded to the physical robot, the robot can get data from the physical world, interpret it through its microcontroller, and downloaded the program; hence the robot creates a reaction through its actuators. In this way, the program in the computer gets a connection with the physical world. Moreover, data from the physical world can be transferred to the computer through robotic kits to form an interaction between the physical world and computers. The two-way physical world and computer connection can provide more meaningful activities for students (Sullivan and Bers, 2016).

Although literature reviews and meta-analysis studies (Hu, Chen, and Su, 2021; Noone and Mooney, 2018; Xu, Ritzhaupt, Tian and Umapathy, 2019) related to block-based versus text-based programming provide promising results, they are mainly inconclusive. Noone and Mooney (2018) conducted a systematic review study including 29 studies published in journals and conference proceedings. They proposed that block-based programming provides benefits over text-based programming. Xu et al. (2019) conducted a meta-analysis study to compare the effect of block-based and text-based programming environments on novice students' cognitive and affective scores. They compiled 13 studies published in journals and conference proceedings. They contended that there is a small effect size in favor of block-based programming environments with respect to cognitive scores. The overall effect size was not found to be significant. Considering the education level, the effect size in the middle school context was the smallest. Moreover, they stated that there is a trivial effect size with respect to affective scores. The effect size for affective scores in the middle school level was insignificant and the overall effect size was also insignificant. Hu et al. (2021) conducted a meta-analysis study to explore the effect of block-based programming on students' academic achievement. They examined 29 empirical studies published in journals and conference

proceedings. They reported an overall small to medium significant effect size in favor of block-based programming. Educational level was found to be a moderator variable. A large effect size was found for elementary and middle school students.

The situation is better in the robotics programming context. The literature review and meta-analysis studies mainly reported positive results in favor of robotics programming for teaching programming and computational thinking. Major, Kyriacou, and Brereton (2012) conducted a systematic literature review to explore the effect of using robots in teaching novices programming. The languages used for programming robots were mostly text-based languages (e.g., Java, C++, and Ada). They considered 23 studies for physical robot programming; (i) 16 of the 23 studies found educational robotic effective for introductory programming instruction; (ii) four of the studies had mixed results; (iii) one study was classified as ineffective; and (iv) two studies were unclassifiable. Scherer, Siddiq, and Viveros (2016) conducted a meta-analysis to consider the effectiveness of block-based programming and educational robotics. They examined 20 studies for the block-based programming condition and 7 studies for the educational robotics condition. They concluded a significantly moderate effect size in favor of block-based programming and a significantly large effect size for educational robotics. Zhang, Luo, Zhu, and Yin (2021) explored the effectiveness of educational robotics. They had considered 17 studies in the meta-analysis. They found a significant moderate effect size in favor of educational robotics with respect to computational thinking.

As seen in the literature, some studies propose that a kind of programming environment or modality has the potential to promote better learning outcomes (Weintrop and Wilensky, 2017). Some studies propose the reverse; similar tools do not result in better learning outcomes (Mihci and Donmez, 2017). Most of these studies were done in the context of programming education. The computational thinking perspective has not been given enough attention yet. Moreover, cognitive variables were the main focus in most of these studies. There is a limited number of studies related to affective variables like attitude. Therefore, there is no consensus in the literature related to effectiveness of robotics, block-based and text-based programming environments. Beside this, when the first text-based programming course should be given is another issue: is middle school context suitable for text-based programming, and if yes what is the optimum grade to start text-based programming? The aim of this study is to compare the effect of constructionist learning instruction that was given in robotics, block-based, and text-based contexts on sixth-grade students' programming achievement, computational thinking, and attitudes towards computer programming. mBlock (with MBot) was used for the robotics context; Scratch was used for the block-based context; and Python with turtle library was used for the text-based context. The followings are the research questions to be explored in the current study.

1. Is there any significant mean difference between the groups that were given mBlock (with MBot), Scratch, and Python (with turtle library) based constructionist instruction with respect to students' programming achievement?
2. Is there any significant mean difference between the groups that were given mBlock (with MBot), Scratch, and Python (with turtle library) based constructionist instruction with respect to students' post-computational thinking scores when their pre-computational thinking scores were controlled?
3. Is there any significant mean difference between the groups that were given mBlock (with MBot), Scratch, and Python (with turtle library) based constructionist instruction with respect to students' post-computer programming attitude scores when their pre-computer programming attitude scores were controlled?

2. Material and Methods

The current study utilized a quasi-experimental design with three sixth-grade introductory programming classes. There were six sixth-grade classes in the school in which the study was carried out. Three study groups were randomly chosen from six classes to construct mBlock (with MBot), Scratch, and Python (with turtle library) groups. In this study, for the simplicity mBlock (with MBot) group will be called mBlock and Python (with turtle library) group will be called the Python group. Before the intervention, all three groups were given a computational thinking test (CTT) and computer programming attitude scale for middle school students (CPAS-M) as pre-tests. After conducting pre-tests, the eight-week intervention period had started. Students were given two 40-minute sessions each week. The interventions in all three groups were designed based on a constructionist approach. The difference among the groups was the programming environment. After the intervention period, all three groups were given CTT, CPAS-M, and a programming achievement test (PAT). The design of the study is summarized in Table 1.

Table 1. The Design of The Study

Group	Pre-test	Programming Env.	Post-test
MBlock	CTT	MBlock with MBot	CTT
	CPAS-M		CPAS-M
			PAT
Scratch	CTT	Scratch	CTT
	CPAS-M		CPAS-M
			PAT
Python	CTT	Python with turtle library	CTT
	CPAS-M		CPAS-M
			PAT

2.1 Subject

105 sixth-grade students from three intact classes were included in the current study. 45 of the students were female and 60 of them were male. There were six sixth-grade classes in the school in which the study was conducted. mBlock, Scratch, and Python groups were randomly chosen from the six classes. The students were given an information technology and software course as their regular curriculum. This course generally starts at fifth grade in the country. Nevertheless, schools that have extensive English language teaching for fifth graders, provide the course at the sixth-grade level. For the study school, the information technology and software course were first given in the sixth grade since there was an English language teaching program for the fifth graders. The given course was compulsory and students' first programming course in their formal education. The mBlock group consisted of 36 students (16 females and 20 males); the Scratch group consisted of 34 students (14 females and 20 males); and the Python group consisted of 35 students (15 females and 20 males).

2.2 Intervention

The same instructor instructed in all three groups. The same approach was used in three groups. The instructions in three groups were designed based on constructionism and pair programming. The programming environments were different in the groups. mblock with Mbot was used in the mBlock group; Scratch was used in the Scratch group; turtle library of Python was used in the Python group. The students studied in pairs in the computer laboratory. Interventions lasted eight weeks, two 40-minute sessions each week. The instruction aims to improve students' computational thinking and problem-solving skills by using constructs of programming with a programming language. The main objectives of the instruction were:

i. Design algorithms,
ii. Know and use programming structures (e.g., variables, conditionals, loops, and functions),
iii. Solve problems by using programming structures,
iv. Choose and apply appropriate programming approaches to solve problems,

The instructions in three groups can be designed in such a way that they all include similar activities. The activities can be given in similar sequence. We believe that this is not a good way to compare the effectiveness of the programming environments. Each programming environment has different potential. The advantages that each modality brings to computer science education is different. Designed instructions should consider peculiarities of programming environments. Therefore, the instructions in the study were designed considering the peculiarities of programming environments. Nevertheless, this does not change main objectives of the instructions. The objectives are the same, e.g., design algorithms and use appropriate programming structures to solve problems. Our approach is different ways with their peculiarities to same ends.

The following is an activity from the mBlock group. At the beginning of the activity, students were shown the line follower robot in Figure 1. They were asked to write a mBlock code that makes the robot follow the black strip shown in Figure 1.

Figure 1. Example mBlock Activity

The following is an activity from the Scratch group. At the beginning of the activity, students were shown the screen in Figure 2. They were asked to create a game similar to the one shown in Figure 2. The bowl in the game can be controlled on the x-direction with the keyboard or the mouse. Apples spawn and fall down from random positions at the top. The player tries to take the apples. Each apple provides a constant point. There is a time limit in which the player tries to get the highest possible score.

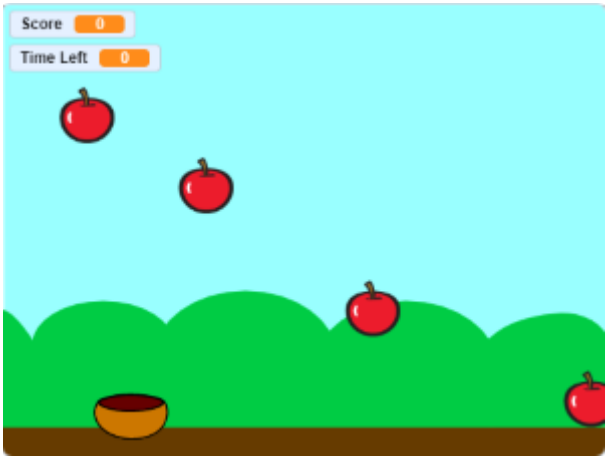


Figure 2. Example Scratch Activity

The following is an activity from the Python group. At the beginning of the activity, students were shown the shape in Figure 3. They were asked to write Python code that draws a shape similar to the one shown in Figure 3.

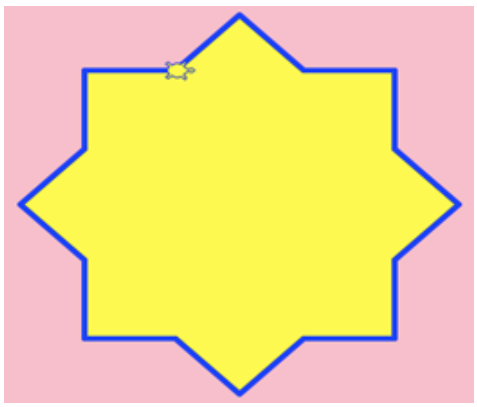


Figure 3. Example Python Activity

The instructor had the role of guide to help students explore. Students were given tasks in the laboratory. They studied to complete the given tasks. The tasks were mainly in one of the following six forms:

i. The instructor gave certain codes and asked students to find the function of these codes,
ii. The instructor gave a task and asked students to discuss as a class how to handle the given task,
iii. The instructor pointed out certain codes, and asked students to complete a given task,
iv. The instructor gave a task and asked students to complete it without any cues,
v. The instructor asked for a class discussion when a common conceptual issue appeared,
vi. The instructor asked students to develop their products.

The instructor tried to help students explore programming concepts through these tasks. The instructor tried not to give complete answers to the students. Whenever necessary, the instructor explained the code and how he handles problems at the hand. But this was kept minimum. Students were encouraged to find their own ways. The instructor was present in all laboratories. The instructor monitored group and individual work and gave group and individual feedback.

Students studied in pairs in the laboratories. Each pair had one computer in all classes. Additionally, each group had one MBot kit in the mBlock group. There were two roles in pairs. One of the students had the keyboard and mouse and was responsible for code writing. The other student reviewed the code writing process; monitored the problems; and tried to help handle the task at the hand. The pairs continuously changed their roles from task to task. Intra-group communication was allowed in the laboratories. Pairs discussed the issue whenever necessary. They were warned not to give complete answers but to negotiate their ideas.

2.3 Instruments

The study included three instruments to gather data. These instruments were a computational thinking test (CTT), a computer programming attitude scale for middle school students (CPAS-M), and a programming achievement test (PAT). CTT and CPAS-M were both used as pre and post-tests. PAT was used for the post-test.

PAT was developed by the authors of this study. PAT has three versions for mBlock, Scratch, and Python groups. All three versions included identical items. 25 items were developed considering the aims of the information technology and software course by the authors of this study. 15 of the items were selected based on content validity. The items were given to two domain experts and two language experts. Domain experts evaluated the items in terms of content validity and appropriateness for students' grade level. Language experts evaluated the items in terms of comprehensibility and grammatical aspects. Then all three versions of the PAT were sent to two domain experts. They evaluated each item in three versions and checked if the items are identical or not. They scored each item from 0 to 10. 0 means "the questions in the three versions are completely different" and 10 means "the questions in the three versions are exactly the same". Moreover, they provided feedback if the item was not given 10. One item had 8, and one item had 9 points. The scores of the remaining items were 10. Necessary changes were done depending on the feedback. Then the Scratch version of PAT was applied to 169 (73 females and 96 males) seventh graders who had already taken an information technology and software course and used Scratch in their classes. The gathered data was analyzed by using TAP software. Since one of the items had improper difficulty (0.11) and discrimination (0.05) values, it was removed from the PAT. Item difficulty and discrimination values of PAT are summarized in Table 2.

Table 2. Item statistics for PAT

Item #	Item Difficulty	Item Discrimination
1	0,86	0,31
2	0,86	0,36
3	0,26	0,49
4	0,62	0,65
5	0,44	0,68
6	0,57	0,50
7	0,38	0,73
8	0,47	0,43
9	0,84	0,36
10	0,86	0,34

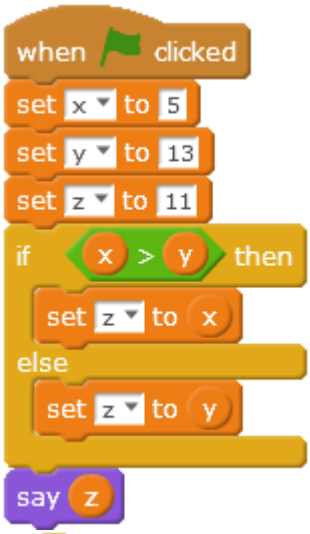
11	0,70	0,38
12	0,73	0,62
13	0,41	0,45
14	0,56	0,47

The mean item difficulty of the PAT was found to be 0.61 and the mean item discrimination of the PAT was 0.48. The internal consistency coefficient (KR20) was found to be 0.74 for the 14-item test. The following three questions can be given as an example for versions of the same question for the three groups.

Example Question for the Three Groups

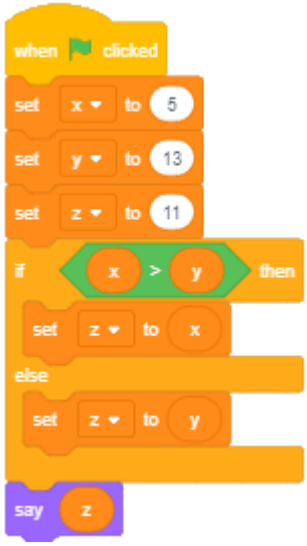
mBlock Group What is the output of the following code?

a. 5
b. 13
c. 11
d. 18



Scratch Group What is the output of the following code?

a. 5
b. 13
c. 11
d. 18



Python Group What is the output of the following code?

a. 5
b. 13
c. 11
d. 18

```
x=5
y=13
z=11
if x>y :
    z=x
else :
    z=y
print(z)
```

The CTT was originally developed by Román-González, Pérez-González and Jiménez-Fernández (2017) in Spanish. CTT was adapted to Turkish. The test aims to assess middle school students' computational thinking levels. It is a multiple-choice test and includes 24 items related to computational concepts. Each item has four choices. KR 20 value of the test was reported 0.78 in the adaptation study. KR20 value was found 0.76 in the pre-test and 0.79 in the post-test in the current study. The total score can a student get from the CTT ranges from 0 to 24.

The CPAS-M was constructed by Gul, Cetin, and Ozden (2022). It was developed to assess middle school students' attitudes towards programming. It includes 13 Likert-type items. The maximum score that a student can get from CPAS-M is 65 and the minimum score is 13. The scale is one-dimensional. Cronbach alpha coefficient of the original scale was found to be 0.93. In the current study, the Cronbach alpha coefficient was found to be 0.91 and 0.93 correspondingly for pre-test and post-test.

3. Results

For the first research question, one-way ANOVA was conducted to examine difference(s) between groups in terms of the PAT scores. Descriptive statistics related to PAT scores of groups were given in Table 3. It was observed that Scratch and mBlock groups had close means while the Python group had the lowest mean.

Table 3. Descriptive statistics of PAT

Group	N	M	SD	Skewness	Kurtosis
Scratch	35	54.57	9.58	-0.588	-0.430
mBlock	34	53.09	9.05	0.201	-0.496
Python	31	27.58	12.44	-0.746	0.403

The one-way ANOVA result showed that there was a significant effect of treatment on students' PAT scores at $p<0.05$ level for Scratch, mBlock, and Python groups [$F_{(2-97)}=68.55$, $p<0.05$]. The calculated effect size for this difference was big ($\eta^2=0.59$) (Green and Salkind, 2013). Post hoc comparisons using the Dunnett C test indicated that the difference was significant between Scratch (M=54.57, SD=9.58) and Python (M=27.58, SD=12.44) groups and mBlock (M=53.09, SD=9.05) and Python (M=27.58, SD=12.44) groups (Table 4).

Table 4. PAT ANOVA Results

Source	SS	df	MS	F	p	Sig. Dif.
Between	14788.145	2	7394.072	68.55	0.00	Scratch-Python; mBlock-Python
Within	10462.855	97	107.864			
Total	25251.000	99				

For the second research question, a one-way ANCOVA was conducted. Before the main analysis one-way ANOVA was conducted to check whether there was a significant mean difference between groups' pre-CTT scores. The ANOVA results showed that there was a significant difference in students' pre-CTT scores at $p<0.05$ level for Scratch, mBlock, and Python groups [$F_{(2-96)}=5.35$, $p<0.05$]. The effect size for this difference was found as medium ($\eta^2=0.10$). Post hoc comparisons using the Tukey test indicated that the difference was significant between Scratch (M=15.17, SD=3.82) and Python (M=11.80, SD=4.80) groups. There was no significant difference between mBlock (M=13.66, SD=3.77) and other groups (Table 5).

Table 5. pre-CTT ANOVA Results

Source	SS	df	MS	F	p	Sig. Dif.
Between	183.636	2	91.818	5.35	0.006	Scratch-Python
Within	1647.536	96	17.162			
Total	1831.172	98				

One-way ANCOVA analysis showed that there was not a significant difference between groups in terms of their post-CTT scores when their pre-CTT scores were controlled [$F_{(2-90)}=0.668$, $p>0.05$]. ANCOVA results are summarized in Table 6.

Table 6. CTT ANCOVA Results

Source	SS	df	MS	F	p

Pre-CTT	636.267	1	636.267	55.917	0.000
Group	15.204	2	7.602	0.668	0.515
Error	1024.088	90	11.379		
Total	1774.809	93			

For the last research question, one-way ANCOVA was utilized. Before the main analysis, one-way ANOVA was conducted to examine whether there was a significant mean difference between groups' pre-CPAS-M scores. The ANOVA results indicated that there was no significant difference in students' pre-CPAS-M scores at $p<0.05$ level for Scratch, mBlock, and Python groups [$F_{(2-93)}=0.783$, $p>0.05$]. The results are summarized in Table 7.

Table 7. pre-CPAS-M ANOVA Results

Source	SS	df	MS	F	P
Between	142.350	2	71.175	0.783	0.460
Within	8452.806	93	90.890		
Total	8595.156	95			

One-Way ANCOVA results showed that a significant difference between groups' adjusted mean CPAS-M scores was observed [$F_{(2-91)}=4.703$, $p<0.05$]. Results were summarized in Table 8. The effect size for this difference was small ($\eta^2=0.094$). Post hoc comparisons using the Bonferroni test indicated that the difference was significant between mBlock (M=51.95) and Python (M=45.28) groups and mBlock (M=51.95) and Scratch (M=46.31) groups. mBlock group significantly outperformed Scratch and Python groups on post-CPAS-M, $p<0.05$.

Table 8. CPAS-M ANCOVA Results

Source	SS	df	MS	F	p
pre-CPAS-M	3384.04	1	3384.04	39.755	0.000
Group	800.586	2	400.293	4.703	0.011
Error	7746.113	91	85.122		
Total	230239.0	95			

4. Discussion and Conclusion

The aim of this study was to assess the impact of modality on sixth-grade students' computational thinking, programming achievement, and programming attitude. mBlock with Mbot, Scratch, and Python with turtle library were used as programming environments. All three groups (mBlock, Scratch, and Python) were given an eight-week intervention, developed considering the principles of constructionism. CTT (computational thinking test) and CPAS-M (programming attitude scale for middle school students) were given as both pre and post-tests. PAT (programming achievement test) was given as a post-test for the groups.

There are studies in the literature contending that robotics and block-based programming provide students with better learning opportunities for programming and programming is one of the best ways to teach computational thinking (Scherer, Siddiq, and Viveros, 2020; Zhang, Luo, Zhu, and Yin, 2021). So, one might expect that programming instruction with robotics and block-based programming produces significantly better learning outcomes with respect to programming achievement and computational thinking. As expected, mBlock and Scratch groups significantly outperformed the Python group with respect to programming achievement. But there was no significant difference between groups considering students' computational thinking. Moreover, there are studies in the literature contending that robotics and block-based programming provide students with concrete and authentic learning opportunity in which students express themselves better and have joy. So, one might expect that both robotics and block-based programming are better environments related to students' attitudes. However, the Scratch group did not meet expectations. mBlock group significantly outperformed Scratch and Python groups with respect to students' CPAS-M scores.

Considering students programming achievement scores, there was (i) no significant difference between the mBlock and Scratch group, (ii) a significant difference between mBlock and Python groups in favor of the mBlock group, and (iii) a significant difference between Scratch and Python groups in favor of Scratch group. mBlock and Scratch groups were superior to the Python group. This study supports the idea that constructionist block-based and robotics programming environments can provide a better learning experience in terms of students' programming achievement (Kert, Erkoc, and Yeni, 2020). This achievement can be explained by the type of activities that students experienced in their groups. Students in mBlock and Scratch groups constructed physical robots and games/animations respectively. Students in the Python group constructed turtle-based graphics. Although students

were able to handle abstract programming concepts through concrete means (programs for the robot, game/animation, and turtle) and see results of their programs immediately in all three groups, robot and game/animation activities might be more engaging. Students can show or tell their acting robots to friends and families, or they can show their games/animations to friends and families and ask them to play their games. Nevertheless, in the case of turtle programming, the graphics on the screen might not be attractive for students themselves and their friends and families. Products of robotic and block-based programming have the potential to be a part of the wider context and to be a cultural tool (Papert, 1993) on which students, peers, friends, teachers, and families can think, talk and give feedback. Therefore, it might be said that robotics and block-based programming environments can provide a rich learning experience for students to achieve in programming since these environments have the potential to be a cultural tool to support students.

There were no significant mean differences between mBlock, Scratch and Python groups with respect to students' post-CTT scores when students' pre-CTT scores were controlled. Two issues should be considered depending on the results related to computational thinking: (i) why there was no significant difference between groups with respect to students' CTT scores and (ii) why this no significant difference phenomenon was observed while there was a significant difference between groups in terms of students' programming achievement score. One possible explanation might be related to the difference between the nature of achievement and skill. Programming achievement is related to and focuses more on the concepts that are given throughout the course. Computational thinking skill is more general, and it is hard to achieve components of computational thinking like abstraction and algorithmic thinking, problem-solving. Therefore, one can suggest that the first programming course might not be enough to help students improve their computational thinking and short-term intervention might not be representative for the general case. In addition to this, contrary to the common belief, one might contend that the complexities of text-based programming, e.g. syntax and debugging, create opportunities for students to deal with problems. These complexities might provide a learning environment in which students have to deal with problems and improve their computational thinking skills, e.g. problem solving while involved in problem-solving. The syntax of Python is not too complex. The right dosage of syntax and debugging issues might have a positive effect on students' computational thinking. The last explanation of the issues might be that there is no immediate significant relation between programming achievement and computational thinking. There is an approach called CS unplugged that aims to improve computational thinking without using programming. Bell and Vahrenhold (2018) stated that CS unplugged approach is promising for developing students' computational thinking. Therefore, improved achievement in programming might not directly mean improvement in computational thinking.

It was found that there is a significant difference between mBlock, Scratch, and Python groups with respect to students' post-CPAS-M scores when students' pre-CPAS-M scores were controlled. mBlock group significantly outperformed Scratch and Python groups. This result might be related to the interaction of constructionism, students' developmental stage, and properties ofprogramming environments. Six-grade students may not be complete abstract thinkers according to the stage theory of Piaget (Huitt and Hummel, 2003). Students might feel they are not good enough to deal with abstract concepts. Constructionism posits that by developing concrete products, students can deal with abstract concepts through concrete means. This might help students feel better in programming. Among mBlock, Scratch, and Python most concrete form of modality belongs to mBlock. mBlock brings programming into students' daily life. Exploration in the programming instruction as suggested by constructionism happens in the most concrete form in mBlock condition. Students might feel good at programming while producing a product in their physical space. This is related to the perception of students not to their actual achievement. It is possible that the use of robotics with constructionism might help students feel they are good at programming. Nevertheless, this result might simply be due to the novelty effect too. Text-based programming is the oldest way to teach programming and block-based programming is widely used in education in Turkey. Nevertheless, robotics programming is a relatively new approach that is not commonly used in state schools in Turkey yet. Therefore, more interest in the mBlock group might be due to new technology, namely robotics. Krendl and Broihier (1992) conducted a study to examine the evolution of students' perceptions about computers. They demonstrated strong evidence of novelty effect, particularly in the case of affective responses. In addition to these issues, no significant difference between Scratch and Python groups should be considered. It is contended that students can have more concrete and joyful experiences in block-based programming (Topalli and Cagiltay, 2018). So, one can expect a significant difference with respect to students' CPAS-M scores between Scratch and Python groups in favor of Scratch. We believe that the finding in the current study does not disprove the concrete and joyful experience that block-based programming can provide. It might be the case that Python is perceived as "real" programming that programmers (like game developers and hackers) use. This might affect the perception of students related to programming (Mihci and Donmez, 2017).

Considering the latest literature review and meta-analysis studies related to computational thinking and programming, there is a lack of studies related to the effect of the programming environment on cognitive and especially affective variables. It is a good idea to speculate on the results of such studies from different perspectives until focal points related to the effects of the programming environment are determined by the researchers. The current study utilized this line of reasoning to explain the findings. The results of this study provided possible answers and new questions. The most solid result is that in a constructionist learning environment mBlock is better than Scratch with respect to programming attitude and mBlock is better than Python with respect to programming achievement and attitude in the current situation. Hence practitioners and researchers can use robotics programming to increase the possibility of success of computational thinking and programming instruction for six grade students. Considering the results related to computational thinking, Python seems to be a promising tool. Nevertheless, the Python group failed in programming achievement. It might not be a good idea to use Python as the first programming environment for sixth-graders. Future studies can test the effectiveness of Python for seventh and eighth-graders. Introducing programming with block-based or robotics programming and then utilizing Python might produce effective results. Moreover, there is a newly developing game programming library called Pygame Zero for Python. As its name suggests it is a simplified version of PyGame for educational purposes. Future studies can test its effectiveness of it. In addition to these considerations, researchers and practitioners need to consider the cost. mBlock provides additional costs for students, teachers, and schools. If the cost is not affordable, then Scratch seems to be a good alternative.

There are certain limitations to this study. Firstly, the study was conducted with limited sample size. The generalizability of the findings is limited. It would be better to conduct the study with a larger sample size. Secondly, the study sample is composed of six graders. Six graders cannot be fully counted as abstract thinkers. The cognitive stage is an important factor in education. The findings of this study might not be generalized to other groups, e.g., high school students. Lastly, the study lasted for eight weeks. Variables like attitude and computational thinking might require more time to be improved. Longer studies can be done to explore these variables.

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal*, *55*(7), 832-835.

Arnon, I., Cottrill, J., Dubinsky, E., Oktac, A., Fuentes, S. R., Trigueros, M., ... Weller, K. (2013). *APOS theory: A framework for research and curriculum development in mathematics education*. New York, NY: Springer.

Bell T. & Vahrenhold J. (2018). CS unplugged—how is it used, and does it work?. In: Böckenhauer HJ., Komm D., Unger W. (Eds), A*dventures between lower bounds and higher altitudes* (pp. 497-521). Springer, Cham.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini languages: a way to learn programming principles. *Education and information technologies*, *2*(1), 65-83.

Cakiroglu, U., Cevik, I., Koseli, E., & Aydin, M. (2021). Understanding students' abstractions in block-based programming environments: A performance-based evaluation. *Thinking Skills and Creativity*, *41*, 100888.

Cetin, I. (2020). Teaching loops concept through visualization construction. *Informatics in Education*, *19*(4), 589-609.

Cetin, I., & Dubinsky, E. (2017). Reflective abstraction in computational thinking. *The Journal of Mathematical Behavior*, *47*, 70-80.

Chao, P. Y. (2016). Exploring students' computational practice, design, and performance of problem-solving through a visual programming environment. *Computers & Education*, 95, 202–215.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.

Dijkstra E.W. (1982). S*elected writings on computing: a personal perspective*. Texts and Monographs in Computer Science. New York, NY: Springer.

Dubinsky, E. (1995). ISETL: A programming language for learning mathematics. *Communications on Pure and Applied Mathematics*, *48*(9), 1027–1051

Green, S.B., & Salkind, N.J. (2013). U*sing SPSS for Windows and Macintosh: analyzing and understanding data*. New Jersey: Pearson.

Gul, D., Cetin, I., & Ozden, M. Y. (2022). A scale for measuring middle school students' attitudes toward programming. *Computer Applications in Engineering Education*, *30*(1), 251-258.

Herrmann, T. (2003). Learning and teaching in socio-technical environments. In *Informatics and the Digital Society* (pp. 59-71). Springer, Boston, MA.

Hu, Y., Chen, C. H., & Su, C. Y. (2021). Exploring the effectiveness and moderators of block-based visual programming on student learning: A meta-analysis. *Journal of Educational Computing Research*, *58*(8), 1467-1493.

Huitt, W., & Hummel, J. (2003). Piaget's theory of cognitive development. *Educational psychology interactive*, *3*(2), 1-5.

Kandemir, C. M., Kalelioğlu, F., & Gülbahar, Y. (2021). Pedagogy of teaching introductory text-based programming in terms of computational thinking concepts and practices. *Computer Applications in Engineering Education*, *29*(1), 29-45.

Kert, S. B., Erkoc, M. F., & Yeni, S. (2020). The effect of robotics on six graders' academic achievement, computational thinking skills and conceptual knowledge levels. *Thinking Skills and Creativity*, *38*, 100714.

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, *92*(3), 170-181.

Korkmaz, O., Cakir, R., & Ozden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in human behavior*, *72*, 558-569.

Kölling, M. (2015). Lessons from the design of three educational programming environments: Blue, BlueJ, and Greenfoot. *International Journal of People-Oriented Programming (IJPOP)*, *4*(1), 5-32.

Kölling, M., Brown, N. C., & Altadmri, A. (2017). Frame-based editing. *Journal of Visual Languages and Sentient Systems*, *3*(1), 40-67.

Krendl, K. A., & Broihier, M. (1992). Student responses to computers: a longitudinal study. *Journal of Educational Computing Research*, *8*(2), 215–227.

Major, L., Kyriacou, T., & Brereton, O. P. (2012). Systematic literature review: teaching novices programming using robots. *IET Software*, *6*(6), 502-513.

Mihci, C., & Ozdener Donmez, N. (2017). Teaching gui-programming concepts to prospective K12 ICT teachers: MIT App Inventor as an alternative to text-based languages. *International Journal of Research in Education and Science*, *3*(2), 543-559.

Mladenović, M., Mladenović, S., & Žanko, Ž. (2020). Impact of used programming language for K-12 students' understanding of the loop concept. *International Journal of Technology Enhanced Learning*, *12*(1), 79-98.

Moons, J., & Backer, C. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, *60*, 368–384.

Noone, M., & Mooney, A. (2018). Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education*, *5*(2), 149-174.

Panskyi, T., Rowinska, Z., & Biedron, S. (2019). Out-of-school assistance in the teaching of visual creative programming in the game-based environment–Case study: Poland. *Thinking Skills and Creativity*, *34*, 100593.

Papert, S. (1980). *Mindstorms, Children, Computers, and Powerful Ideas*. New York, Basic Books.

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. New York: Basic Books.

Piaget, J. (1964). Cognitive development in children: Development and learning. *Journal of Research in Science Teaching 2*(3), 176-186.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., ... Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Román-González, M., Perez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678-691.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129–141.

Scherer, R., Siddiq, F., & Viveros, B. S. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior*, *109*, 106349.

Sullivan, A., & Bers, M. U. (2016). Robotics in the early childhood classroom: learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education*, *26*(1), 3-20.

Tikva, C., & Tambouris, E. (2021). A systematic mapping study on teaching and learning Computational Thinking through programming in higher education. *Thinking Skills and Creativity*, *41*, 100849.

Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with scratch. *Computers & Education*, *120*, 64–74.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education*, *18*(1), 1–25.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2011, February). Research notebook: Computational thinking- what and why? *The Link Magazine*, 20–23. Retrieved from https://www.scs.cmu.edu/link.

Xu, Z., Ritzhaupt, A. D., Tian, F., & Umapathy, K. (2019). Block-based versus text-based programming environments on novice student learning outcomes: A meta-analysis study. *Computer Science Education*, *29*(2-3), 177-204.

Zhang, Y., Luo, R., Zhu, Y., & Yin, Y. (2021). Educational robots improve k-12 students' computational thinking and STEM attitudes: systematic review. *Journal of Educational Computing Research*, 1-32.