Original Research Article

Journal of Educational Computing Research
2023, Vol. 0(0) 1–28
© The Author(s) 2023
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/07356331231204653
journals.sagepub.com/home/jec
Sage

Transitioning From Introductory to Professional Programming in Secondary Education: Comparing Learners' Computational Thinking Skills, Behaviors, and Attitudes

Dan Sun[1], Chengcong Zhu[2], Fan Xu[3], Yan Li[4], Fan Ouyang[4], and Miaoting Chen[5]

Abstract
Although previous research has provided some insights into the effects of block-based and text-based programming modalities, there is a dearth of a detailed, multi-dimensional analysis of the transition process from different introductory programming modalities to professional programming learning. This study employed a quasi-experimental design to address this gap, involving 64 secondary school students in two groups. For the beginning five weeks, the first group used an introductory block-based programming environment, while the second group used an introductory text-based programming environment. Then, both groups transitioned to professional text-based programming for the subsequent eight weeks. The results showed that participants who transitioned from introductory text-based programming to professional text-

[1]Jing Hengyi School of Education, Hangzhou Normal University, Hangzhou, China
[2]Xiaoshan High School, Hangzhou, China
[3] Department of Educational Studies, The Ohio State University, Columbus, OH, USA
[4]College of Education, Zhejiang University, Hangzhou, China
[5]Faculty of Education, Shenzhen University, Shenzhen, China

Corresponding Author:
Yan Li, College of Education, Zhejiang University, Yu Hang Tang Rd. #866, 310058, Hangzhou, Zhejiang, China.
Email: yanli@zju.edu.cn

based programming (1) significantly outperformed in computational thinking skills; (2) had more code-writing and debugging behaviors and fewer irrelevant behaviors, and (3) had more interactions with the instructor. No significant differences were observed between the two groups regarding enjoyment, confidence, and interest in programming. Drawing on these findings, this study proposes pedagogical implications that could facilitate the adoption of programming modalities within the broader context of STEM education.

Keywords
STEM education, block-based and text-based programming modality, computational thinking skills, behavioral patterns, learning attitudes, secondary education

Introduction

Computer programming is an integral component of STEM education, encompassing science, technology, engineering, and mathematics disciplines. Its significance lies in developing students' computational thinking (CT) skills (Sun et al., 2021b), enhancing their motivation and engagement (Schnittka et al., 2015), and encouraging them to pursue careers in computer science (Chittum et al., 2017). In primary and secondary education, two predominant modalities have emerged: block-based programming (also known as graphical programming) and text-based programming (Weintrop & Wilensky, 2019). The availability of tools such as Scratch, Blockly, and App Inventor has facilitated the engagement of increasing young learners or beginners worldwide in learning block-based programming. Meanwhile, recognizing the necessity of professional programming knowledge and skills as a prerequisite for entering fields like Artificial Intelligence, educators have increasingly focused on integrating text-based programming languages (e.g., Python, Java, C#) into secondary school curricula and beyond (Grover, 2021; Ministry of Education, 2017).

However, regarding the transition from introductory (text-based or block-based) programming to professional (mainly text-based) programming in K-12 school settings, previous empirical research has yielded conflicting results on students' summative performance, such as programming knowledge, computational thinking skills, and attitudes. Some studies indicate that introductory block-based programming leads to positive learning processes and outcomes in professional text-based programming (Armoni et al., 2015; Grover et al., 2015; Mladenović et al., 2018), while others suggest that introductory text-based programming is more effective than introductory block-based programming at improving students' programming ability (Weintrop & Wilensky, 2017, 2019; Xu et al., 2019). In addition, few studies actually examine how learners engage in programming practices from a process-oriented perspective. Many open questions have not been addressed regarding the transition between programming modalities (Lin & Weintrop, 2021), therefore, further research is necessary to conduct a detailed, multi-dimensional analysis of the transition process.

Given the above-mentioned needs, this study aimed to investigate two transitions: (1) the transition from the learning of introductory text-based programming to professional text-based programming (e.g., Python), namely TPP; (2) the transition from the learning of introductory block-based programming to professional text-based programming (e.g., Python), namely BPP. A quasi-experimental research design was employed to examine these two transitions and their impacts on learners. Specifically, the following research questions were answered.

RQ1. What is the impact of the introductory programming modality (block-based vs. text-based) on secondary school students' *computational thinking skills* as they transition to professional text-based programming?

RQ2. What is the impact of the introductory programming modality (block-based vs. text-based) on secondary school students' *programming behaviors* as they transition to professional text-based programming?

RQ3. What is the impact of the introductory programming modality (block-based vs. text-based) on secondary school students' *attitudes toward programming* as they transition to professional text-based programming?

Literature Review

Block-Based versus Text-Based Programming Modalities

Block-based programming has emerged as an accessible and intuitive to visual programming, employing the "programming-primitive-as-puzzle-piece" technique to simplify the programming process (Good, 2018). The rising popularity of tools such as Scratch, Snap!, and Blockly has led to the widespread adoption of block-based programming in primary education. In a block-based environment, program blocks can only be connected if they form a valid statement, preventing syntax errors while maintaining a step-by-step instruction process (Lin & Weintrop, 2021). These tools also utilize color-coding to signify different programming concepts and nested structures to indicate block scope (Maloney et al., 2010). Block-based programming's user-friendly nature has made it appealing to novice learners globally. For instance, Code.org's "Hour of Code" features numerous block-based programming activities, with over 100 million participants from more than 180 countries (HourofCode, 2023). Furthermore, block-based programming is increasingly integrated into primary school curricula, specifically in information technology courses (Ministry of Education, 2017).

In contrast, text-based programming necessitates learners to input command lines and is a prevalent method in secondary programming education and beyond, especially for secondary students with a firm grasp of syntax and text-based code logic (Yücel &

Rızvanoğlu, 2019). Common professional text-based programming languages include Python, C++, and Java. Text-based programming allows learners to attain advanced programming expertise, engage in professional projects (Weintrop & Wilensky, 2019), and access more robust and realistic features (Weintrop & Wilensky, 2015).

With the development of different programming modalities, researchers have gradually focused on the differences between block-based programming and text-based programming. Kölling et al. (2015) identified 13 key distinctions between block-based and text-based programming, such as code readability, code memory, syntax memory, typing/spelling, command line quantity, prototypes and definitions, identifiers (e.g., variables), scopes, written expressions, data types, error reporting, code layout, and programming paradigms. Researchers have acknowledged that text-based programming cannot be supplanted by block-based programming. Instead, text-based programming languages are often designed as learning objectives for students who have mastered block-based programming languages (Xu et al., 2019).

In block-based programming, students do not need to learn the syntax of a specific programming language in advance, which reduces the cognitive load of elementary and secondary school students and allows them to focus more on the structure of the code (Hu et al., 2021; Sayginer & Tuzun, 2023). However, according to Weintrop and Wilensky (2019) research, block-based programming was seen by students as lacking authenticity and being less powerful, which was more likely to impact older students keen to develop transferable skills for potential employment or future computer science studies.

Research on the Transition across Different Programming Modalities

The increasing popularity of block-based programming in formal educational settings has driven research efforts to examine its effectiveness in preparing learners for professional text-based programming in the future. Early studies identified that, despite success in block-based programming, learners often faced challenges when transitioning to text-based programming (Scholtz & Wiedenbeck, 1990). Recent scholars continue to investigate the obstacles encountered by learners during this transition, revealing that experiences in block-based programming do not automatically transfer to text-based programming (Grover, 2021; Lin & Weintrop, 2021). Hsu and Gainsburg (2021) revealed that the block-based environment used in introductory programming courses did not offer an advantage in preparing students for learning a text-based programming language (Java). According to Saygıner and Tüzün's (2023) research, the implementation of a block-based environment in programming training had a positive impact on the development of students' logical thinking skills and motivation to learn programming. However, the study found no significant differences in programming success between students who received this training and those who did not. Espinal et al. (2022) also found that most students were capable of transitioning between block-based and text-based programming languages, yet many struggled with interpreting programs and addressing new challenges in text-based environments.

These findings suggest that block-based programming tools can enhance learners' attitudes toward programming education, but their influence on overall learning performance remains inconclusive (Lin & Weintrop, 2021; Xu et al., 2019). Furthermore, the majority of prior research has examined the comparative effects of block-based and text-based programming at different stages of learning, with few studies exploring the impact of these experiences on secondary school students' professional programming (such as Python) skills through quasi-experimental methods (Weintrop & Wilensky, 2019). Consequently, the potential benefits of block-based programming experiences for professional text-based programming remain uncertain.

Influential Factors on Learning Across Varied Programming Modalities

Empirical research suggests that the transition from introductory to professional programming could have an impact on several aspects of learners' experience, including their CT skills, level of engagement, and attitudes towards programming. Firstly, computational thinking entails utilizing basic computational concepts and methods to scrutinize and solve problems (Wing, 2006). The transitions from block-based to text-based programming can have a considerable impact on the development of CT skills. According to Tabet et al. (2016), students who transitioned from block-based to text-based programming witnessed enhancement in their CT skills, in contrast to those who solely received text-based programming instruction. Similarly, Espinal et al. (2022) discovered that exposure to block-based programming aided students in developing CT skills such as decomposition, while text-based programming in their algorithm design skills. Hence, when assessing the efficacy of transition between different programming modalities, the advancement of CT skills is a crucial consideration.

Secondly, as programming necessitates cognizant problem-solving, meaning-making, and construction of knowledge (Sun et al., 2021a), empirical research has undertaken the analysis and demonstration of various facets of block-based and text-based programming processes. For instance, Schnittka et al. (2015) demonstrated that middle-school students manifested increased engagement and motivation toward programming through the introduction of block-based programming languages. Weintrop and Wilensky (2015) discovered that block-based programming strengthened novice programmers' debugging abilities. Similarly, according to Xu et al. (2019), block-based programming aided learners in improving their coding habits, including the ability to write concise and readable code.

Finally, the modalities of programming instruction can have an impact on attitudes toward programming. Chittum et al. (2017) found that introducing Python programming to high school students improved their attitudes toward programming and careers in computer science. Furthermore, Wang (2021) found that female students' attitudes toward programming were more positive when they learned through a block-based programming language, as opposed to a text-based programming language. Taken together, an essential aspect of investigating the efficacy of programming modality transition is the examination of learners' CT skills, programming behaviors, and attitudes toward programming.

Methodology

Context and Participants

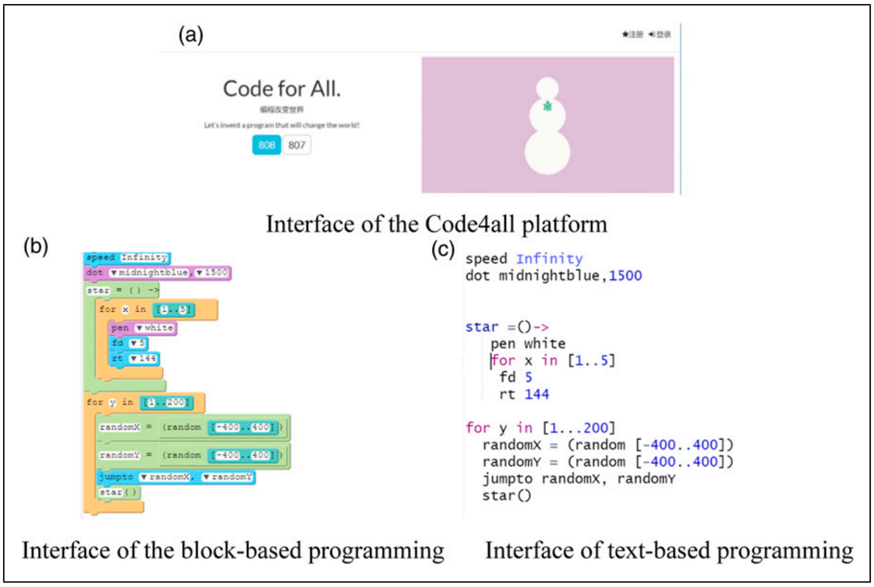
The research context was a formal course titled "Creative Programming Algorithms", which took place during Autumn 2021 in a secondary school located in the Eastern area of China. A total of 64 students from two first-grade classes at the secondary school were recruited in this study, after obtaining consent from both the school and the students. The Review Committee's written agreement was obtained to interact with and collect data for research purposes without ethical issues. The students, with no formal text-based or block-based programming learning experience, were instructed by the same teacher (the first author of this paper).

The 64 participants came from two natural classes were divided into two groups. Over a period of five initial weeks, the first group interacted with an introductory block-based programming environment (BPP group: 32 students, 20 boys, and 12 girls). This group was designated as the control group, as it was consistent with the default environment in the secondary school setting. While the second group engaged with an introductory text-based programming environment (TPP group: 32 students, 17 boys, and 15 girls), establishing the experimental group for this study. Then, both groups transitioned to professional text-based programming instruction with Python.

Course Design

The instructional sessions for introductory text-based and block-based programming were based on Bau's (2013) book, where they provided 26 projects covering basic to deep knowledge of coding concepts such as lines, points, loops, events, sorting, and searching. Additionally, the book showcased 17 elaborated examples including running pencil code, strings/numbers, using "if" to choose, etc. The instructor adjusted the instructional content and procedures to suit the programming abilities of local students. The instructional content included starting the art of drawing, using for and while loops to draw, implementing if statements to guess numbers, and utilizing functions to create both basic and advanced starry night drawings. For the professional text-based programming instruction, the sessions were based on the Information Technology textbook for secondary students (Zhejiang Provincial Department of Education and Research Office, 2019). The instructional content comprised getting started with Python, data and operations, sequential structures in Python, if structures in Python, loop structures in Python, and Python Programming with functions.

This study was conducted on two programming platforms (Code4all, and PyCharm). In the introductory text-based and block-based programming phases, learners were taught and practiced on the Code4all platform. The platform's backend server was


(a)
Code for All.
编程改变世界
Let's invent a program that will change the world!
808
807
Interface of the Code4all platform
(b)
(c)
Interface of the block-based programming
Interface of text-based programming

```
speed Infinity
dot ▾midnightblue, ▾1500
star = ( ) ->
  for x in [1..5]
    pen ▾white
    fd ▾5
    rt ▾144
for y in [1..200]
  randomX = (random [-400..400])
  randomY = (random [-400..400])
  jumpto ▾randomX, ▾randomY
  star( )
```

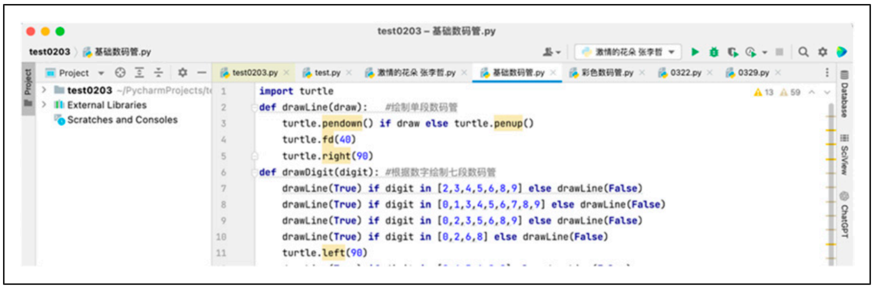
```
speed Infinity
dot midnightblue,1500


star =()->
   pen white
   for x in [1..5]
    fd 5
    rt 144

for y in [1...200]
  randomX = (random [-400..400])
  randomY = (random [-400..400])
  jumpto randomX, randomY
  star()
```

Figure 1. Code4all Platform Interfaces. (a) Interface of the Code4all platform. (b) Interface of the block-based programming (c) Interface of text-based programming.

built using Node.js web architecture and express, while the front end was developed in JavaScript with custom messaging middleware. The modules collaborated and passed information entirely through events. Additionally, the platform uses jQuery Turtle to draw graphics and can support the block-based or text-based presentation of the programming language CoffeeScript (see Figure 1(a)), which has a 'light syntax' property that makes it more suitable for beginners to learn introductory programming (CoffeeScript, 2023).

In the block-based programming interface (see Figure 1(b)), learners can drag and drop programming blocks and auto-link them, while in the text-based programming interface (see Figure 1(c)), learners need to write programs by entering code character by character. Both interfaces have the same functionality, except for the different presentation of the programming language, including the keywords and syntax of the programming language, the visual execution environment, and other platform features such as login, new, save, delete, and other operations. Code4all does not allow learners to switch between the two modalities. Instead, learners are required to use either the block-based interface or the text-based interface exclusively.

During the professional text-based programming instruction phase, this study used PyCharm as the Integrated Development Environment (IDE) in the course. The main reason for adopting this platform is that it supports the writing of Python language (see Figure 2). By developing relevant tools, the researcher could use this platform to



```
import turtle
def drawLine(draw):     #绘制单段数码管
    turtle.pendown() if draw else turtle.penup()
    turtle.fd(40)
    turtle.right(90)
def drawDigit(digit): #根据数字绘制七段数码管
    drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)
    drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)
    drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)
    drawLine(True) if digit in [0,2,6,8] else drawLine(False)
    turtle.left(90)
```

Figure 2. PyCharm platform interface.

automatically capture learners' programming learning behavior, which laid the necessary foundation for the study.

Analytical Framework

As demonstrated in Figure 3, the research proposed an analytical framework to investigate the differences between TPP and BPP from the process and summative perspective, which was adapted from Sun et al. (2021b). From the perspective of process-oriented assessment, research can gather behavioral data by recording in-class behaviors and programming activities through classroom video recordings and learners' programming operations through log data or computer screen recordings. Classroom video analysis, clustering analysis, and temporal analysis can be applied to analyze behavioral data. Furthermore, recordings of classroom audio have the potential to capture discussions that take place between students and teachers during class. These recordings can be analyzed using quantitative content analysis, lag-sequential analysis, and ethnographic interpretations to explore discourse patterns and features. With regards to summative assessment, data on programming knowledge and skills (e.g., pre-, mid-and post-tests) and final products (e.g., programming projects) can be gathered as performance data, and statistics can be used to explore the significance of performance changes. Moreover, learners' attitudes toward various programming modalities can be better understood by collecting data from surveys conducted from pre-, mid-, and post-surveys. Taken together, this analytical framework provides a comprehensive approach to both the process and final assessments of different programming modalities.

Data Collection

This study collected data from four sources. First, we conducted pre- and post-test of learners' computational thinking skills. The questionnaire used a validated computational thinking scale (5-point Likert scale) for Chinese K-12 students adapted from Bai and Gu (2019) based on Korkmaz et al. (2017).

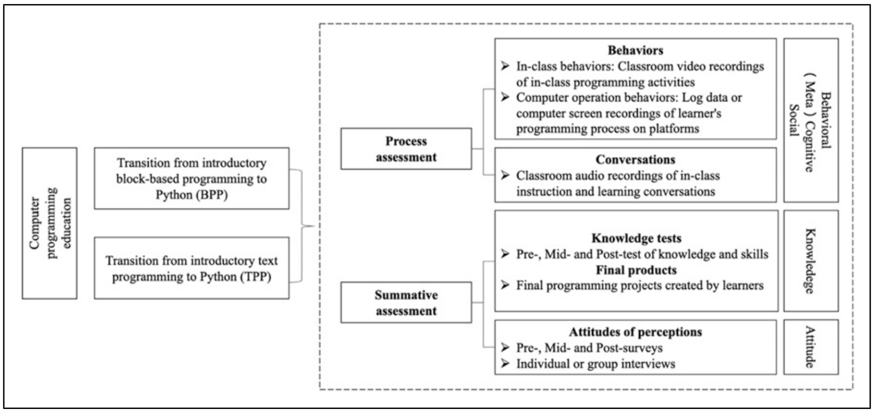

Computer programming education
Transition from introductory block-based programming to Python (BPP)
Transition from introductory text programming to Python (TPP)
Process assessment
Summative assessment
Behaviors
➤ In-class behaviors: Classroom video recordings of in-class programming activities
➤ Computer operation behaviors: Log data or computer screen recordings of learner's programming process on platforms
Conversations
➤ Classroom audio recordings of in-class instruction and learning conversations
Knowledge tests
➤ Pre-, Mid- and Post-test of knowledge and skills
Final products
➤ Final programming projects created by learners
Attitudes of perceptions
➤ Pre-, Mid- and Post-surveys
➤ Individual or group interviews
Behavioral
(Meta) Cognitive
Social
Knowledge
Attitude

Figure 3. Analytical framework.

Second, we collected data on learners' programming behaviors from two aspects. On the one hand, we recorded students' online programming behaviors through programming platform log data. By modifying the helper's file of the PyCharm platform, the debugging information generated by each student's click on the "debug" button is transferred to the cloud server for storage in the form of time stamps. The stored data includes "operation time, written code, operation result, error message, etc." On the other hand, we recorded students' offline programming behaviors (90 minutes in total) through a whole class video recording.

Finally, this study conducted pre-, mid-, and post-tests in three phases of the experiment to investigate changes in students' attitudes, which include self-confidence, enjoyment, and interest. The survey was adapted from the Georgia Computes project (Bruckman et al., 2009) and the Computing Attitudes Survey, which were validated from previous research (Sun et al., 2021b). The survey includes five items on a 10-point Likert scale ranging from 1 (strongly disagree) to 10 (strongly agree). Please refer to Appendix A.

As illustrated in Figure 4, the study lasted for 16 weeks, with one class period of 40 minutes per week. In the first week, students were acclimated to the programming environment and were administered a pre-test of CT and learning attitudes. In the seventh week, students took a mid-test of CT and learning attitudes along with a programming project. During the eighth to 15th week, both sets of students were taught Python programming. In the 16th week, students underwent a post-test of CT and attitudes, and complete a programming project.

Data Analysis

First, analysis of variance (ANOVA) was used to compare the differences in CT skill levels among pre-, mid-, and post-test, and one-way analysis of covariance

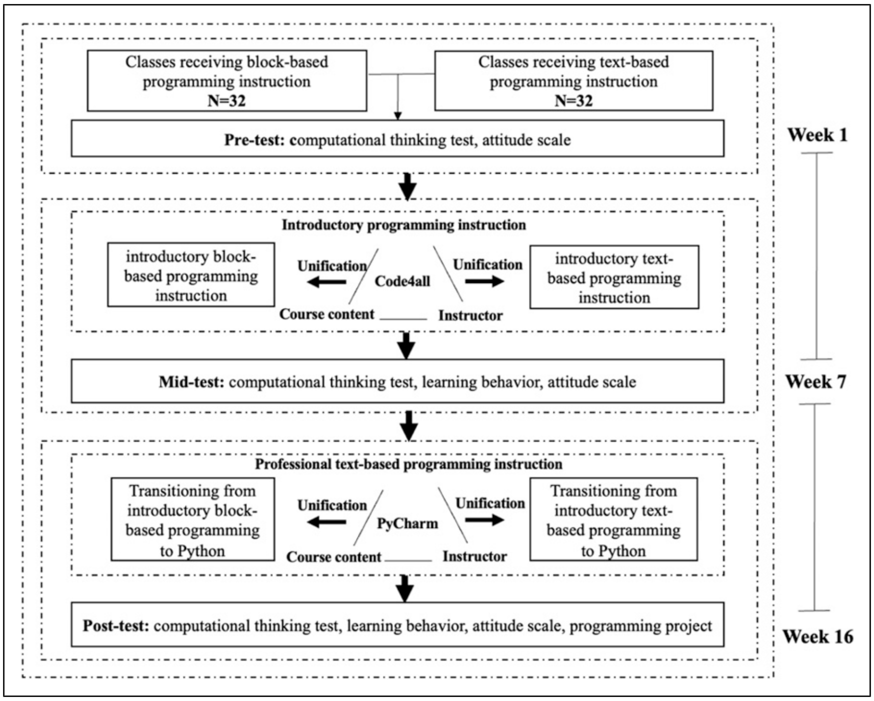

Classes receiving block-based programming instruction N=32
Classes receiving text-based programming instruction N=32
Pre-test: computational thinking test, attitude scale
Week 1
Introductory programming instruction
introductory block-based programming instruction
Unification
Code4all
Unification
introductory text-based programming instruction
Course content
Instructor
Mid-test: computational thinking test, learning behavior, attitude scale
Week 7
Professional text-based programming instruction
Transitioning from introductory block-based programming to Python
Unification
PyCharm
Unification
Transitioning from introductory text-based programming to Python
Course content
Instructor
Post-test: computational thinking test, learning behavior, attitude scale, programming project
Week 16

Figure 4. Experimental design.

(ANCOVA) methods was applied to compare the post-test computational thinking skills between the two groups. These data analyses were conducted using SPSS 25.0.

Secondly, to explore programming behavior differences between the two groups, this study collected and analyzed learners' online (2313 lines of log data) and offline (class video lasting 40 minutes) programming behaviors. Based on previous research (Pereira et al., 2020) and the data types of programming platform log files, this study identifies the coding types based on programming operation data (see Table 1), which are *average line of code written* (ACo), *average lines of code changed* (ACh), *number of platform operations* (NPo), *number of debugging* (ND), *number of syntax errors* (NSe), *average time between two* debugging (AtD), and *average irrelevant on irrelevant behavior* (AIb). The two coders then coded the data again independently in chronological order based on the coding framework, with a rater reliability of .890.

For the log data recorded by the programming platform, the study analyzed the descriptive statistics and then conducted a clustering analysis (Dutt, 2015). Clustering algorithms can discover hidden patterns in complex datasets, and new relationships in educational data can be mined through unsupervised learning methods (Dutt et al.,

Table 1. The Coding Framework of Learner's Online Programming Behaviors.

Code	Behavior	Description
ACo	Average number of codes written	The average number of lines of code written by the learner between two clicks of the debug button.
ACh	Average number of changes	The average number of lines of code the learner modified between two clicks of the debug button.
NPo	Number of platform operations	The number of actions the learner took on the platform (e.g., login, save, new, etc.).
ND	Number of debugging	The total number of times the learner clicked the debug button.
NSe	Number of syntactical errors	The number of syntax errors made by the learner during programming.
AtD	Average time between two debugging	The average time (in seconds) that the learner spends between clicks on the debug button.
AIb	Average time on irrelevant behavior	The average amount of time (in seconds) the learner spent on unrelated actions during coding, such as playing games, idle actions, etc.

2017). Based on the fact that learners' behaviors are heterogeneous, this study clustered learners based on the data recorded in the platform logs to explore the programming learning behavior patterns of different learner clusters. In addition, the study used the *K*-means algorithm, which derives *N* observations among a predetermined number of *K* clusters, where each observation belongs to the nearest group mean (MacQueen, 1967), and then uses the average profile coefficient of the observations to select the most appropriate number of clusters (Rousseeuw, 1987). Log data analysis was conducted using RStudio 2022.

In addition, we used classroom video analysis (Kersting, 2008) to analyze the video data to identify behavioral differences between the two classes during the programming learning process. Two coders first watched the whole-class video recordings separately to determine the initial coding of classroom behaviors and then engaged in a discussion to determine the final coding framework (see Table 2). The following six programming learning behaviors were included: *operating on the computer* (OoC), *discussion with peers* (DwP), *asking questions to the instructor* (AsQ), *listening to the instructor* (LtI), *irrelevant behavior*s (IB), and *walking around* (WR). The two coders then coded the data again independently in chronological order based on the coding framework, with a rater reliability of .870. Students' behaviors were reported in a summative way and further demonstrated in a temporal graph. Temporal analyses were conducted using RStudio.

Finally, regarding the difference in attitudes, the Wilcoxon Rank Sum test (reported as a U statistic) was performed to reveal the differences in learners' attitudes in terms of confidence, enjoyment, and future interest between the two groups. This test was used because the two samples are independent, and the underlying data is ordinal and non-parametric (Fay & Proschan, 2010).

Table 2. Coding Framework of Learners' Offline Programming Behaviors.

Code	Behavior	Behavior description
OoC	Operating on the computer	Learners are doing it on the computer by themselves.
DwP	Discussing with peer	Learners are having a discussion with their peers.
AsQ	Asking questions	Learners asking questions to the teacher.
Ltl	Listening to the instructor	The learner is listening to the instructor's explanation.
IB	Irrelevant behaviors	The learner is exhibiting unrelated behaviors (fiddling with the computer, etc.).
WR	Walk around	The learner is walking around the classroom.

Results

Results Regarding Computational Thinking Skills

We present the results of learners' computational thinking skills at the pre-, mid-, and post-tests under two modalities (see Table 3). The learners in the BPP group had mid-test scores significantly higher than their pre-test and post-test scores on the CT skill level ($F = 5.55, p < .01$). In contrast, TPP learners had significantly higher post-test CT scores than mid-test ($F = 2.89$, $p < .05$) scores.

This study conducted an ANCOVA to compare the CT kills of learners under the BPP and TPP modalities. The study accepted the modality used for introductory programming instruction (block-based vs. text-based) as a fixed factor, with CT pretest scores as covariates and CT posttest scores as dependent variables. As shown in Table 4, after excluding the effect of the pretest, learners in the BPP and TPP groups showed significant differences in computational thinking skills ($F = 5.47$, $p < .05$, $eta^2 = .08$).

Results Regarding Programming Behaviors

Log Data: Online Programming Behaviors. Table 5 shows the distribution of learners' online programming behaviors recorded by the log data in PyCharm. Compared to the TPP group, learners in BPP had more behavior of code-changing (Ach, $M_{\mathrm{BPP}} = 3.83$, $SD_{\mathrm{BPP}} = 2.80$; $M_{\mathrm{TPP}} = 2.64$, $SD_{\mathrm{TPP}} = 2.13$), longer debugging intervals (AtD, $M_{\mathrm{BPP}} = 356.75$, $SD_{\mathrm{BPP}} = 416.14$; $M_{\mathrm{TPP}} = 266.70$, $SD_{\mathrm{TPP}} = 276.74$) and more than three times the number of irrelevant behaviors (AIb, $M_{\mathrm{BPP}} = 21.25$, $SD_{\mathrm{BPP}} = 17.90$; $M_{\mathrm{TPP}} = 6.50$, $SD_{\mathrm{TPP}} = 6.34$). In addition, learners in TPP had longer lines of code (ACo, $M_{\mathrm{BPP}} = 19.88$, $SD_{\mathrm{BPP}} = 6.97$; $M_{\mathrm{TPP}} = 22.67$, $SD_{\mathrm{TPP}} = 2.50$), more platform operations (NPo, $M_{\mathrm{BPP}} = 6.75$, $SD_{\mathrm{BPP}} = 4.22$; $M_{\mathrm{TPP}} = 9.00$, $SD_{\mathrm{TPP}} = 5.02$), more errors (NSem, $M_{\mathrm{BPP}} = 4.31$, $SD_{\mathrm{BPP}} = 2.65$; $M_{\mathrm{TPP}} = 6.59$, $SD_{\mathrm{TPP}} = 7.68$), and a greater number of debugging (ND, $M_{\mathrm{BPP}} = 5.72$, $SD_{\mathrm{BPP}} = 3.27$; $M_{\mathrm{TPP}} = 7.40$, $SD_{\mathrm{TPP}} = 7.91$).

Table 3. Distribution and ANOVA Results of Learners' Computational Thinking Skills.

		(1) Pre-test		(2) Mid-test		(3) Post-test			
Group	N	*M*	*SD*	*M*	*SD*	*M*	*SD*	*F*	Significant comparison
BPP	32	3.91	.37	4.23	.53	3.91	.36	5.55**	(2) > (1); (2) > (3)
TPP	32	3.90	.52	3.79	.75	4.15	.47	2.89*	(3) > (2)

Note. *p < .05; ** p < .01.

Table 4. ANOVA Results of Learners' Computational Thinking Skills.

		Pre-test		Post-test		One-way ANCOVA			
Group	N	*M*	*SD*	*M*	*SD*	*M(adjusted)*	*SE*	*F*	*eta*^{2}
BPP	32	3.91	.37	3.91	.36	3.91	.07	5.47*	.08
TPP	32	3.90	.52	4.15	.47	4.15	.07		

Note. p*< .05.

The results of clustering showed that the *K*-means converged in the 10th iteration, with a *K* value equal to 5 being the best value and the corresponding highest mean profile coefficient (.59). As shown in Table 6, 17.18% of the participants were assigned to Cluster 1 (C1), 6.25% to Cluster 2 (C2), 28.13% to Cluster 3 (C3), 46.88% to Cluster 4 (C), and 1.56% to Cluster 5 (C5).

To examine the differences in the level of CT skills across five clusters, an ANOVA was conducted (see Table 7). Overall, there were statistically significant differences in CT levels among the five clusters (F = 2.641, p < .05), with cluster 2 performing significantly better than Cluster 3 and Cluster 5; Cluster 4 performed significantly better than Cluster 5. Besides, Cluster 4 performed significantly better than Cluster 3.

Looking into the programming behaviors of the high-performing clusters, the 4 members in Cluster 2 (2 learners for BPP and 2 learners for TPP) had an average level of programming behaviors. The 29 members in Cluster 4 (10 learners for BPP and 19 learners for TPP) used the "trial and error" or "tinkering" strategy, as they showed the greatest number of debugging behaviors (ND: M = 7.80, SD = 2.61) and the least number of code-modifying behaviors (ACh: M = 2.40, SD = 1.09).

Similar to learners in Cluster 4, the 11 members in Cluster 1 (4 learners for BPP and 7 learners for TPP) had intensive debugging behaviors, as they showed the shortest debugging interval (AtD: M = 42.08, SD = 42.46). However, they also showed the least number of code-writing (ACo: M = 20.20, SD = 6.73) and the greatest number of errors (Nse: M = 6.91, SD = 10.54), which lead to slightly lower performance in CT.

Table 5. Distribution of Learners' Online Programming Behaviors.

	N	ACo	ACh	NPo	ND	NSe	AtD	AIb
BPP	32	19.88 ± 6.97	3.83 ± 2.80	6.75 ± 4.22	5.72 ± 3.27	4.31 ± 2.65	356.75 ± 416.14	21.25 ± 17.90
TPP	32	22.67 ± 2.50	2.64 ± 2.13	9.00 ± 5.02	7.40± 7.91	6.59 ± 7.68	266.70 ± 276.74	6.50 ± 6.34

Table 6. Clustering of Learners' Online Programming Behaviors.

Cluster	N	ACo	ACh	NPo	ND	NSe	AtD	AIb
C1	11 (17.18%)	20.20 ± 6.73	4.07 ± 2.90	8.39 ± 2.51	7.64 ± 10.58	6.91 ± 10.54	42.08 ± 42.46	4.04 ± 2.71
C2	4 (6.25%)	21.63 ± .74	3.38 ± 1.47	7.94 ± 1.07	2.25 ± .43	2.25 ± .43	925.38 ± 151.99	3.92 ± 2.53
C3	18 (28.13%)	20.77 ± 1.36	3.50 ± .84	6.75 ± 2.35	5.07 ± 1.31	4.28 ± 1.09	395.01 ± 71.90	9.01 ± 10.94
C4	29 (46.88%)	21.82 ± 4.70	2.40 ± 1.09	8.32 ± 3.61	7.80 ± 2.61	6.16 ± 2.42	225.77± 45.24	3.88 ± 3.56
C5	2 (1.56%)	24.50 ± .07	14.00 ± .12	9.00 ± .23	2.00 ± .00	2.00 ± .00	1903 ± 50.13	.02 ± .07

Table 7. ANOVA Results of Five Clusters' Computational Thinking Skills.

C1	C2	C3	C4	C5	*F*	Significant comparison
4.05 ± .55	4.44 ± .43	3.84 ± .45	4.09 ± .33	3.60 ± .02	2.641*	C2>C3 C2>C5 C4>C3

Note. *$p < .05$.

For the relatively lower-performing groups, Cluster 3 had 18 members (15 learners for BPP and 3 learners for TPP) who had the least number of operational (NPo: $M = 6.75$, $SD = 2.35$) behaviors, but had the greatest number of irrelevant (AIb: $M = 9.01$, $SD = 10.94$) behaviors. Lastly, Cluster 5 had 2 members (both members are BPP learners), and they had the highest number of code writing (ACo: $M = 24.50$, $SD = .07$), code modifying (ACh: $M = 14.00$, $SD = .12$), platform operation (NPo: $M = 9.00$, $SD = .23$), and had the longest debugging interval (AtD: $M = 1903$, $SD = 50.13$).

Class Video: Offline Programming Behaviors. In terms of offline programming behaviors during the course (see Figure 5), in the introductory programming period, BPP learners were more focused on operating computers (OoC; frequency = 347) and peer discussions (DwP; frequency = 274) but showed limited interactions with the instructor (AsQ; frequency = 35, LtI; frequency = 160). More casual walking behavior (WR; frequency = 136) occurred in the following professional text-based programming learning period. In comparison, learners in the TPP group also focused on their computer operations (OoC; frequency = 380) during the introductory instruction period and had more interactions with the instructor (AsQ; frequency = 117, LtI; frequency = 318) and discussions with peers (DwP; frequency = 221).

Results Regarding Attitudes

As shown in Figure 6, this study analyzed three aspects of students' attitudes on the three administrations (pre-, mid-, and post-tests): confidence, enjoyment, and interest. In terms of confidence, both BPP and TPP showed a trend of decreasing, with the mean level of BPP learners slightly higher than that of TPP learners from beginning to end. However, there was no statistically significant difference between the scores of the two groups in the post-test ($U = 424.50, p > .05$). Similarly, learners' enjoyment in both BPP and TPP did not show a significant difference between the two groups in the post-test ($U = 459.00, p > .05$). Both groups experienced a decrease in engagement first and then a slight increase. Regarding interest, there was no statistically significant difference in learners' scores on the post-test interest score either ($U = 415.50, p > .05$). The two groups appeared to have different trends: Learners in the BPP experienced a trend of increasing and then decreasing, while learners in the TPP maintained a more stable level of interest.

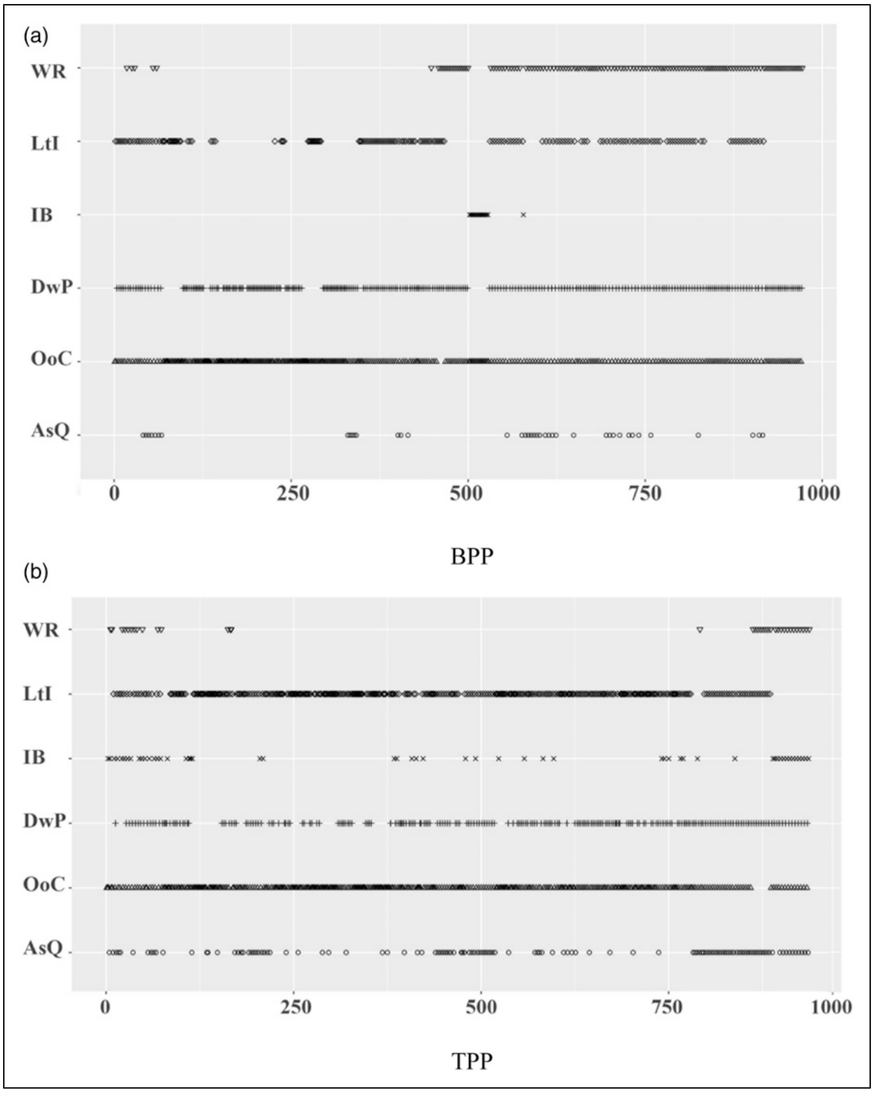


Figure 5. The Temporal Graph of Learners' Classroom Behaviors. (a) BPP. (b) TPP. *Note*. The *x*-axis represents the time period; the *y*-axis represents classroom behaviors.

Discussions

As one area of STEM education, computer programming focuses on fostering learners' computational thinking skills, learning motivations and interests in programming, as well as programming engagement (Grover, 2021; Sun et al., 2021b; Wakhata et al., 2022). This study explored the transitions from two introductory programming

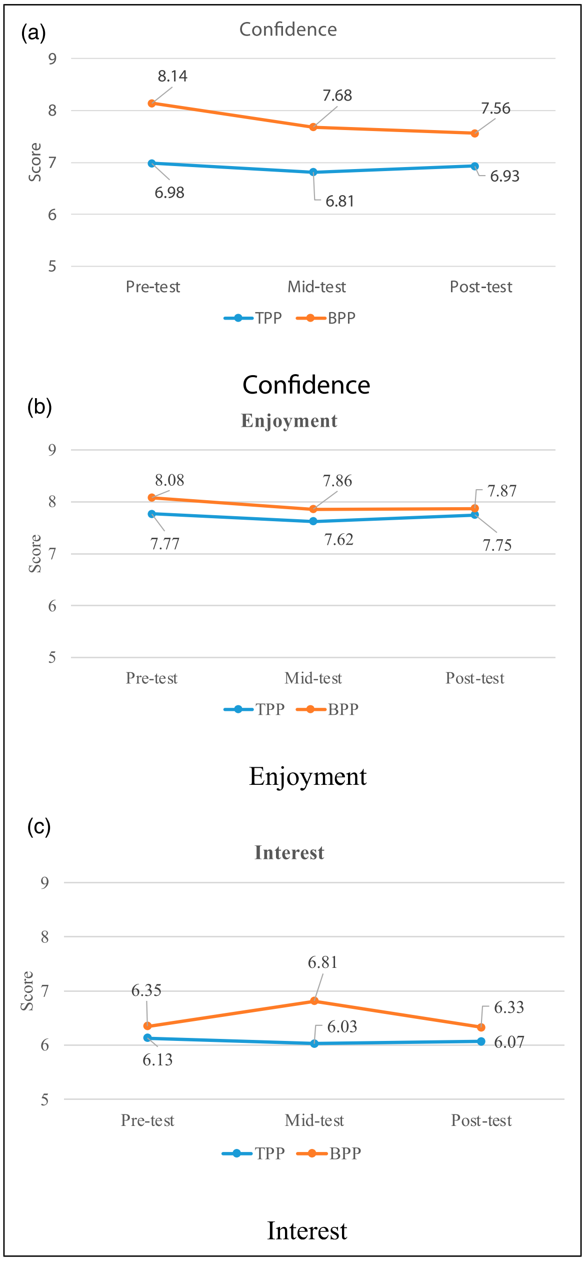


Figure 6. Changes of learners' Confidence (a), Enjoyment (b), and Interest (c).

modalities, namely text-based and block-based programming, to professional text-based (Python) programming in secondary education in China. The computational thinking skills, programming behaviors, and programming attitudes of 64 participants were collected and analyzed.

The results found that, compared with learners who began with introductory block-based programming, those who transitioned from introductory text-based programming to professional text-based programming outperformed in CT skills. This study found that although block-based programming improved learners' levels of CT skills during the introductory programming period, the improvement was less likely to persist in the learning of professional programming (Python). The findings are aligned with previous research (Mladenović et al., 2018), suggesting that block-based programming tools may offer some features to simplify programming complexity and impart certain logical and structural concepts, whereas they have a minor impact on learners' ability to learn professional programming learning. In contrast, learners who were taught with introductory text-based programming are better equipped to learn Python and demonstrate stronger CT skills due to the smooth transition between introductory and professional text-based programming languages (Weintrop & Wilensky, 2019), and Espinal et al. (2022) also found, as the students advanced through the lesson plans, their progress in transition tasks improved due to the enhancement of their initial CT abilities.

Regarding programming behaviors, learners in the BPP group showed a greater number of code-modifications, which could be attributed to the various assistive features provided by the block-based programming environment (Grover & Basu, 2017). Learners in the BPP group also had longer debugging intervals and displayed over three times the number of irrelevant behaviors. This suggests that although block-based languages could reduce the difficulty in coding for beginners (Mladenović et al., 2018), it run the risk of leading to more behaviors unrelated to programming. In contrast, students in TPP showed more code-writing and debugging behaviors. We infer that after transitioning to Python, learners in the TPP group became more comfortable with a text-based programming language and engaged in more debugging behaviors when they encountered syntax errors. While learners in the BPP group took longer to adapt to the Python language and made more changes to their codes, resulting in fewer debugging behaviors.

Based on the results of clustering analysis, learners in Cluster 3 who had many irrelevant behaviors and fewer operations of the programming platform underperformed in the CT test, from which we can conclude that irrelevant behaviors or cognitive absence are one of the barriers to developing CT skills through programming. Consistent with prior research (Fields et al., 2021; Hwang et al., 2012), Cluster 5, which performed lower in CT had the least number of debugging behaviors, which suggested that students could benefit from continuous debugging and modification of codes. Without testing their codes and detecting bugs in the codes, learners can hardly develop CT skills through the problem-solving process. Additionally, most learners in Cluster 3 and Cluster 5 were from the BPP group. We can infer that instructors should monitor the progress of learners with block-based languages and provide timely support (e.g., process-oriented scaffoldings) in the introductory instruction stage (Sun et al., 2021b).

Meanwhile, the analysis of offline behaviors through class videos indicated that students in BPP displayed fewer interactions with the instructor. One possible explanation is that students who have worked with block-based programming may have developed the habit of exploring code blocks independently through drag and drop. Therefore, when students transition from introductory block-based languages to more complicated professional text-based languages, instructors need to increase their communication with them (Sun et al., 2021a). This includes the provision of comprehensive code explanations, monitoring students' advancement, addressing their queries, and undertaking similar supportive measures.

Moreover, the obtained results revealed no statistically significant difference between learners in the BPP and TPP groups concerning their confidence, enjoyment, and interest. This finding contradicts previous arguments that emphasized the positive impacts of block-based programming on motivation, attitudes, and engagement for K-12 learners (Grover, 2021; Hu et al., 2021; Saygıner & Tüzün, 2023). However, this finding echoes studies investigating transition effect of different programming modalities. For instance, Hsu and (2022) found that, compare with Java-only group, students with introductory block-based programming experience felt less confident in later Java course. Likely,Weintrop and Wilensky (2019) conducted a study comparing transitions between different modalities and found no significant differences in confidence and enjoyment across the conditions. The absence of a significant difference suggests that the increased positive attitudes observed in previous studies employing block-based tools may be more influenced by the curriculum employed or the contextual factors surrounding learners' programming experiences, rather than the programming modality itself (Lin & Weintrop, 2021). Another potential factor contributing to these results could be the age range of the research participants. Given the developmental aspects of metacognition among high school students in Weintrop and Wilensky's (2019) study, they may exhibit different patterns in attitudes compared to secondary students in this study.

Overall, despite prior claims by researchers regarding the potential benefits of block-based programming for fostering a favorable initial learning encounter, this study reveals that individuals who were exposed to introductory text-based programming exhibited a higher degree of consistency and fluidity during their progression towards professional text-based programming in Python. Learners in the TPP group demonstrated superior computational thinking abilities, displayed a greater propensity for desirable programming-related behaviors, and exhibited comparable levels of attitudes toward programming when compared to their block-based programming counterparts.

Pedagogical Implications

Based on the results, this study proposes three pedagogical implications for future instructional design of programming education in secondary school. Firstly, to continue learning professional programming languages, text-based programming is a more beneficial modality in introductory programming instruction. In other words, teachers could select appropriate programming modalities based on specific goals of programming education. To do this, teachers should specify the learning goals and consider whether students will transition to professional text-based programming languages such as Python or pursue a programming-relevant career in the future. According to Weintrop and Wilensky (2019), for educator who has doubts about the authenticity of block-based tools or recognizes the pedagogical benefits of having learners write programs from the beginning, it could be more suitable to introduce learners to a text-based language instead of a block-based one.

Secondly, teachers could provide different scaffoldings for learners with different programming learning experiences. On the one hand, this study shows that block-based programming could help learners reduce the likelihood of encountering syntax errors compared to text-based programming, but learners may exhibit more irrelevant behaviors during the learning process. Grover and Basu's research (2017) also found that students in block-based environments may face challenges in fully understanding fundamental programming concepts and the flexibility of manipulating data due to the absence of variable and data type usage. Therefore, when dealing with learners who have experience with block-based programming, teachers should provide more instructional interventions and support on learning strategies, such as providing explicit task lists, incorporating gamified learning strategies (Mladenović et al., 2018), engaging students in functions-describing activities (e.g., loop iterations), promoting the practice of choosing appropriate variable names, and consistently assessing students' comprehension of constructs and concepts with ongoing evaluations (Grover & Basu, 2017).

On the other hand, text-based programming learners wrote longer lines of code, spent more time debugging, and encountered more syntax errors. They showed limited creativity in solving problems because they relied mainly on the teacher's examples (Pereira et al., 2020). Therefore, educators can foster learners' engagement in code-writing and debugging by leveraging their experience with text-based programming. They can offer additional support through scaffolding techniques for error correction strategies. This might involve equipping students with a compilation of typical debugging errors (Žanko et al., 2023), compiling syntax errors commonly encountered during early programming practices, and distributing them to learners through feedback charts or tables. Furthermore, the adoption of innovative approaches in text-based programming can enhance learning experiences. For example, Kölling et al. (2015) suggest frame-based editing as a method that combines the error resistance and user-friendliness of block-based programming with the flexibility and conventional programming semantics inherent in text-based programming languages.

Thirdly, teachers should also consider learners' emotional feedback during the learning process. This study found that learners in both programming modalities showed decreased enjoyment and confidence levels when they faced programming challenges. Therefore, teachers could be attentive to learners' emotional fluctuations and offer timely feedback to help adjust their attitudes and learning strategies accordingly (Jeon & Song, 2019). Throughout the instructional process, teachers have the opportunity to tailor the difficulty level of materials according to learners' programming experience, use diverse learning strategies like peer communication support (Sun et al., 2021a), and provide verbal or behavioral encouragement to foster positive attitudes towards programming learning (Sherin, 2001).

Conclusions, Limitations, and Future Directions

This research aimed to investigate how transitioning from introductory text-based or block-based programming to professional text-based programming (i.e., Python) instruction affects learners' computational thinking skills, programming learning behaviors, and learning attitudes. The study involved 64 secondary school students as experimental subjects. The findings indicated that learners who transitioned from introductory block-based programming instruction to Python showed higher levels of computational thinking skills than those who transitioned from introductory block-based programming instruction. On the other hand, learners who transitioned from introductory text-based programming to Python showed more code-writing and code-debugging behaviors and fewer irrelevant behaviors. The study suggests that introducing learners to introductory text-based programming can be more beneficial for formal education scenarios in secondary schools, where the primary learning goal is to teach professional text-based programming in the future. However, introductory block-based programming can be more suitable for learners who use programming skills to develop comprehensive skills like logical thinking. These findings offer evidence to support the use of either block-based or text-based programming instruction in secondary school settings.

Nonetheless, this study has a few limitations. Firstly, the assessment of computational thinking relied primarily on self-reported questionnaires completed by students, introducing the possibility of bias due to inaccurate comprehension and interpretation of the questionnaire items. Future research should assess learners' computational thinking through knowledge tests and fine-grained examination of their programming learning behaviors. Secondly, though the sample size of 64 participants was appropriate, all participants in this study were recruited solely within the context of secondary education in China. A larger and more diverse sample would provide more statistical power and enhance the robustness of the results.

Thirdly, the study's duration was relatively brief, encompassing only 16 weeks and 16 instructional sessions, leading to limited insight into the lasting effects of block-based or text-based programming on learners. Due to technical constraints, the analysis of data relied heavily on manual coding, preventing the utilization of alternative methods. Future research should focus on gathering multimodal data over an extended timeframe to gain a more comprehensive understanding of learners' long-term changes resulting from transitions between different modalities. This could involve incorporating various data types such as audio/video recordings, clickstream data, facial expressions, motion and gesture analysis, and eye-tracking data. A mixed-method approach has the potential to provide more meaningful and robust results, which could be incorporated in future research. Moreover, the development or implementation of automated data analysis tools and the execution of longer-term experimental studies would significantly enhance a more comprehensive evaluation.

Overall, as the essence of programming lies in its process, future research and practices must integrate both block-based and text-based programming approaches and adopt a process-oriented perspective in investigating, advancing, and evaluating learners' programming quality. This research represents a significant step forward in conducting a comprehensive analysis of learners' performances, processes, and attitudes in computer programming education within the context of compulsory secondary education in China. Furthermore, this study has the potential to contribute to the global adoption of programming modalities, particularly in the broader realm of STEM education.

Appendix A

Attitudinal Survey

1. I will be/am good at programming.
2. I will be doing well/did well in this course.
3. I like programming.
4. I am excited about this course/I was excited about this course.
5. I might take more programming courses in the future/I plan to take more programming courses in the next semester.

Acknowledgements

The authors would like to thank all the participants for their active participation and collaboration in this study.

Authors' Contributions

Sun Dan: Conceptualization; Data analysis; Writing – original draft, review & editing. Chengcong Zhu: Data analysis; Writing – original draft. Yan Li: Writing – original draft, review & editing; funding acquisition. Fan Xu: Writing –review & editing. Fan Ouyang: Writing – original draft, review & editing. Miaoting Cheng: Writing – review & editing, funding acquisition.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by funding from the National Natural Science Foundation of China [Grant No. 62307011], National Natural Science Foundation of China [Grant No. 61977057], Guangdong Basic and Applied Basic Research Foundation, China [Grant No. 2021A1515110081], Guangdong Planning Office of Philosophy and Social Science, China [Grant No. GD22XJY12], Shenzhen Science, Technology and Innovation Commission, China [Grant No. 20220810115236001].

Ethical Statement

Ethical Approval

Active consent forms have been signed by the participants, and the research project has acquired ethical approval from the institution.

ORCID iDs

Dan Sun https://orcid.org/0000-0003-2467-7406
Fan Xu https://orcid.org/0000-0002-2518-4075
Fan Ouyang https://orcid.org/0000-0002-4382-1381
Miaoting Chen https://orcid.org/0000-0001-5329-8857

Data Availability Statement

Requests for data details may be made to the corresponding author.

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to “real” programming. *ACM Transactions on Computing Education*, *14*(4), 1–15. https://doi.org/10.1145/2677087

Bai, X. M., & Gu, X. Q. (2019). Research on the construction and application of the computational thinking instrument in K12. *China Educational Technology*, *393*(10), 83–90.

Bau, D. (2013). *Pencil code: A programming primer* (2nd ed.). David Bau. https://www.google.com/books/edition/Pencil_Code/kJRhAgAAQBAJ?hl=en&gbpv=1&dq=Pencil+Code:+A+Programming+Prime&pg=PP9&printsec=frontcover

Bruckman, A., Biggers, M., Ericson, B., McKlin, T., Dimond, J., DiSalvo, B., Hewner, M., Ni, L., & Yardi, S. (2009). “Georgia computes!”: Improving the computing education pipeline. *ACM SIGCSE Bulletin*, *41*(1), 86–90. https://doi.org/10.1145/1539024.1508899S

Chittum, J. R., Jones, B. D., Akalin, S., & Schram, Á. B. (2017). The effects of an afterschool STEM program on students’ motivation and engagement. *International Journal of STEM Education*, *4*(1), 1–16. https://doi.org/10.1186/s40594-017-0065-4

CoffeeScript. (2023). Retrieved 7 September, 2023, from https://coffee-script.org/

Dutt, A. (2015). Clustering algorithms applied in educational data mining. *International Journal of Information and Electronics Engineering*, *5*(2), 112–116. https://doi.org/10.7763/ijiee.2015.v5.513

Dutt, A., Ismail, M. A., & Herawan, T. (2017). A systematic review on educational data mining. *IEEE Access*, *5*, 15991–16005. https://doi.org/10.1109/access.2017.2654247

Espinal, A., Vieira, C., & Guerrero-Bequis, V. (2022). Student ability and difficulties with transfer from a block-based programming language into other programming languages: A case study in Colombia. *Computer Science Education*, 1–33. https://doi.org/10.1080/08993408.2022.2079867

Fay, M. P., & Proschan, M. A. (2010). Wilcoxon-mann-whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, *4*, 1–39. https://doi.org/10.1214/09-SS051

Fields, D., Lui, D., Kafai, Y., Jayathirtha, G., Walker, J., & Shaw, M. (2021). Communicating about computational thinking: Understanding affordances of portfolios for assessing high school students' computational thinking and participation practices. *Computer Science Education*, *31*(2), 224–258. https://doi.org/10.1080/08993408.2020.1866933

Good, J. (2018). Novice programming environments: Lowering the barriers, supporting the progression. Retrieved from https://sro.sussex.ac.uk/id/eprint/77651/1/Novice-Programming-Environments_-Lowering-the-Barriers-Supporting-the-Progression.pdf

Grover, S. (2021). Teaching and assessing for transfer from block-to-text programming in middle school computer science. In C. Hohensee & J. Lobato (Eds.), *Transfer of Learning* (pp. 251–276). Springer International Publishing.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. In M. Caspersen, S. Edwards, T. Barnes, & D. Garcia (Eds.), Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education (pp. 267–272). March 8–11, 2017, Seattle, WA, USA.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

HourofCode. (2023). Retrieved 20 June, 2023, from https://hourofcode.com/us

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. https://doi.org/10.1177/0735633120945935

Hwang, W.-Y., Shadiev, R., Wang, C.-Y., & Huang, Z.-H. (2012). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. *Computers & Education*, *58*(4), 1267–1281. https://doi.org/10.1016/j.compedu.2011.12.009

Hsu, W., & Gainsburg, J. (2021). Hybrid and non-hybrid block-based programming languages in an introductory college computer-science course. *Journal of Educational Computing Research*, *59*(5), 817–843. https://doi.org/10.1177/0735633120985108

Jeon, I., & Song, K. S. (2019). *The effect of learning analytics system towards learner's computational thinking capabilities*, In Proceedings of the 11th international conference on computer and automation engineering (pp.12–16), New York, NY, USA, ACM. https://doi.org/10.1145/3313991.3314017

Kersting, N. (2008). Using video clips of mathematics classroom instruction as item prompts to measure teachers' knowledge of teaching mathematics. *Educational and Psychological Measurement*, *68*(5), 845–861. https://doi.org/10.1177/0013164407313369

Kölling, M., Brown, N. C., & Altadmri, A. (2015). Frame-based editing: Easing the transition from blocks to text-based programming. In Proceedings of the workshop in primary and secondary computing education (pp. 29–38), ACM. https://doi.org/10.1145/2818314.2818331

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Lin, Y.H., & Weintrop, D. (2021). The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, *67*(4), 101075. https://doi.org/10.1016/j.cola.2021.101075

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth berkeley symposium on mathematical statistics and probability (pp. 281–297). University of California Press.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Ministry of Education. (2017). General high school information technology curriculum standard (2017 Edition). Retrieved from https://www.moe.gov.cn/jyb_xxgk/xxgk_jyta/jyta_kjs/202002/.html

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, *23*(4), 1483–1500. https://doi.org/10.1007/s10639-017-9673-3

Pereira, F. D., Oliveira, E. H. T., Oliveira, D. B. F., Cristea, A. I., Carvalho, L. S. G., Fonseca, S. C., Toda, A., & Isotani, S. (2020). Using learning analytics in the Amazonas: Understanding students' behaviour in introductory programming. *British Journal of Educational Technology*, *51*(4), 955–972. https://doi.org/10.1111/bjet.12953

Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*(1), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

Sayginer, S., & Tuzun, H. (2023). The effects of block-based visual and text-based programming training on students' achievement, logical thinking skills, and motivation. *Journal of Computer Assisted Learning*, *39*(2), 644–658. https://doi.org/10.1111/jcal.12771

Schnittka, C. G., Evans, M. A., Won, S., & Drape, T. D. (2015). Looking for learning in afterschool spaces: Studio STEM. *Research in Science Education*, *46*(3), 389–412. https://doi.org/10.1007/s11165-015-9463-0

Scholtz, J., & Wiedenbeck, S. (1990). Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human–Computer Interaction*, *2*(1), 51–72. https://doi.org/10.1080/10447319009525970

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning*, *6*(1), 1–61. https://doi.org/10.1023/A:1011434026437

Sun, D., Ouyang, F., Li, Y., & Chen, H. (2021a). Three Contrasting Pairs' Collaborative Programming Processes in China's Secondary Education. *Journal of Educational Computing Research*, *59*(20), 073563312097343. https://doi.org/10.1177/0735633120973430

Sun, D., Ouyang, F., Li, Y., & Zhu, C. (2021b). Comparing learners' knowledge, behaviors, and attitudes between two instructional modes of computer programming in secondary education. *International Journal of STEM Education*, *8*(1), 54–54. https://doi.org/10.1186/s40594-021-00311-1

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016). From Alice to Python: Introducing text-based programming in middle schools. In Proceedings of the ACM conference on innovation and technology in computer science education (pp. 124–129). ACM.

Wang, J. (2021). Use hopscotch to develop positive attitudes toward programming for elementary school students. *International Journal of Computer Science Education in Schools*, *5*(1), 48–58. https://doi.org/10.21585/ijcses.v5i1.122

Wakhata, R., Mutarutinya, V., & Balimuttajjo, S. (2022). Secondary school students' attitude towards mathematics word problems. *Humanities and Social Sciences Communications*, *9*(1), 444. https://doi.org/10.1057/s41599-022-01449-1

Weintrop, D., & Wilensky, U. (2015). *Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs* (pp. 101–110). Association for Computing Machinery.

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. https://doi.org/10.1145/3089799

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, *142*, 103646. https://doi.org/10.1016/j.compedu.2019.103646

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

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. https://doi.org/10.1080/08993408.2019.1565233

Yücel, Y., & Rızvanoğlu, K. (2019). Battling gender stereotypes: A user study of a code-learning game, "Code Combat," with middle school children. *Computers in Human Behavior*, *99*(4), 352–365. https://doi.org/10.1016/j.chb.2019.05.029

Žanko, Ž., Mladenović, M., & Krpan, D. (2023). Mediated transfer: Impact on programming misconceptions. *Journal of Computers in Education*, *10*(1), 1–26. https://doi.org/10.1007/s40692-022-00225-z

Zhejiang Provincial Department of Education and Research Office. (2019). *Middle school information technology grade*, (Vol. *7*). Zhejiang Education Publishing House.

Author Biographies

Dan Sun (Ph.D.) is an assistant professor in Jing Hengyi School of Education at Hangzhou Normal University. Her research interests include artificial intelligence in education, programming education, learning analytics and computational thinking, etc. Email: dansun@hznu.edu.cn, ORCID: https://orcid.org/0000-0003-2467-7406

Chengcong Zhu is a high school teacher in Xiaoshan High School. His research interests include information technology, programming education, etc. Email: m18969036832@163.com

Fan Xu (Ph.D.) is a Senior Learning Designer in the Center on Education and Training for Employment (CETE) of the College of Education and Human Ecology at The Ohio State University. Her research interests include learning design, learning analytics, computational thinking, collaborative learning, etc. Email: xu.3849@osu.edu, ORCID: https://orcid.org/0000-0002-2518-4075

Yan Li (Ph.D.) is a professor in College of Education at Zhejiang University. Her research interests include e-learning, distance education, ICT education, etc. Email: yanli@zju.edu.cn, ORCID: https:// orcid.org/0000-0002-0640-1783

Fan Ouyang (Ph.D.) is a ZJU 100 Young Professor in College of Education at Zhejiang University. Her research interests include learning analytics and educational data mining, computer-supported collaborative learning, etc. Email: fanouyang@zju.e-du.cn, ORCID: https://orcid.org/0000-0002-4382-1381

Miaoting Chen (Ph.D.) is an assistant professor in Department of Educational Technology at Shenzhen University. Her research interests include Artificial intelligent education, digital education, technology acceptance. Email: chengmt1@szu.edu.cn, ORCID: https://orcid.org/0000-0001-5329-8857