10.16920/jeet/2026/v40i1/26136

Learning Programming Language Using Color-Coding and Program Tracing

[1]Deepti Reddy Patil*, **[2]Monika Sharma, [3]Ketan Shah, [4]Alka Mahajan**
[1]Department of Computer Engineering, Mukesh Patel School of Technology Management & Engineering, NMIMS University, Mumbai Campus, India
[2]Department of Computer Engineering, Mukesh Patel School of Technology Management & Engineering, NMIMS University, Navi Mumbai Campus, India
[3]Department of Information Technology, Mukesh Patel School of Technology Management & Engineering, NMIMS University, Mumbai Campus, India
[4]Modern Education Society, Wadia Campus, Bund Garden Road, Pune, India
[1]deepti.reddy@nmims.edu, [2]Smonika15@gmail.com , [3]ketan.shah@nmims.edu , [4]alkamahajan@gmail.com

***Abstract*— Learning and teaching programming language is a challenging task as it requires learners to be engaged at both cognitive and metacognitive levels. Cognitive skills include the ability to recall and visualize various programming constructs, and metacognitive skills include the ability to monitor and reflect on the problem-solving process. The three main skills for developing programming proficiency in students are the ability to read, trace, and write the program. Research has shown that learners struggle to master the basic program comprehension skills. We propose an approach to engage learners at both cognitive and metacognitive levels. At the cognitive level, a color-coding approach is used to engage learners in writing various programming constructs using different colors, and at the metacognitive level, program tracing is used to engage learners in monitoring and reflecting on program execution using various test cases. To investigate the effectiveness of the proposed pedagogy, we conducted a two-group post-test design experiment. The comparison was conducted between current-year students who received the intervention and previous-year students who did not receive it.**

***Keywords*— Cognitive Skills; Color Coding; Metacognitive Skills; Programming Skills; Program Tracing; Trace Tables;**

***JEET Category*—Practice**

I. INTRODUCTION

Learning a programming language is a challenging task for novice programmers (Nelson, 2017), leading to low passing rates and high dropout rates in programming courses (Busjahn, 2013). It is argued that the learner should be involved in both cognitive and metacognitive levels to learn programming. The three main cognitive processes that novices use when learning to program are encoding, storage, and retrieval (Paas, 2003; Simon, 2019). Encoding involves transforming programming concepts into a mental representation. Storage refers to retaining and organizing information in memory, and retrieval involves accessing and applying that knowledge to solve programming problems. Metacognitive processes are self-regulation skills and systematic approaches that include planning, monitoring, and reflecting on problem-solving activities (Loksa, 2016; Lopez, 2008). Various instructional strategies proposed in the literature for teaching programming are visualization of program execution using visual programming tools (Mayer, 1981), hands-on exercises (Mayer, 1981), program reading, tracing, and writing exercises (Luxton-Reilly, 2016), prompts and hints (Lopez, 2008), Pair programming (Isong, 2014), etc.

Overall, the learners should be able to visualize various programming constructs as mental representations, which will help them with encoding, storage, and retrieval. Various studies (Dzulkifli, 2013; Rambally, 1986) have examined the relationship between color and memory. They highlight that color can affect memory performance by influencing attention, arousal, and emotional states, which in turn affect cognitive processes such as encoding, consolidation, and retrieval of information. The same concept is used in teaching programming language constructs (Liu, 2021).

The metacognitive skills are the ability to examine and correct errors in the program. This can be achieved by tracing the program using a trace table. Program tracing (Luxton-Reilly, 2016) is a fundamental skill for programming students, as it helps them identify errors and improve their code comprehension. The approach emphasizes the glass box approach, where the learner attempts to visualize the program's execution inside the computer, step by step.

In this article, we propose an approach to engage learners at both cognitive and meta-cognitive levels. The cognitive-level tasks involve writing programs using various color-coding schemes for different constructs. The metacognitive tasks involve tracing program execution using a trace table. The research question (RQ) that we aim to investigate is: Does the process of writing programs using various colors and tracing programs for various test cases improve the learning of the programming language?

The research study was conducted to investigate the effectiveness of the proposed intervention. The study design is a two-group post-test research design, in which current-year students' performance who were taught using the proposed intervention was compared with that of previous-year students

Deepti Reddy Patil
Department of Computer Engineering, Mukesh Patel School of Technology Management & Engineering, NMIMS University, Mumbai Campus, India
deepti.reddy@nmims.edu

who did not receive the intervention. We collected and analyzed both quantitative and qualitative data during the study. The quantitative data are the scores on the mid-test exams for both groups, and the quantitative data are the interview data from the students.

The next section, 2, discusses the literature review and contributions to learning programming languages and to various instructional methodologies. Section 3 presents the proposed methodology for teaching and learning a programming language, and Section 4 discusses the study conducted to investigate the effectiveness of the proposed intervention, followed by the discussion of results and conclusion.

II. LITERATURE REVIEW

Studies have shown that novices can effectively learn programming languages by engaging at both cognitive and metacognitive levels (Nelson, 2019). At the cognitive level, novices should be able to recall and visualize various programming constructs, engage in problem-solving activities, and actively manipulate them. Metacognitive strategies, such as planning, monitoring, and evaluating one's own learning and problem-solving processes, are crucial for developing programming expertise. Mayer (1981) suggests that novices should be encouraged to reflect on their problem-solving strategies and self-regulate their learning. He emphasizes the importance of providing clear explanations, relevant examples, and opportunities for practice and feedback.

Pair programming (Isong, 2014) is a pedagogical approach that engages students at both the cognitive and metacognitive levels. In this pedagogy, two students are assigned to one computer in the laboratory. While one student actively implements the program, the other should continuously observe the code with a view to identifying errors and provide more ideas to achieve a better solution to their task as the teacher lectures. It is important that this practice be performed interchangeably among the students to maintain a proper balance and avoid one student becoming passive.

Prior research has shown that students' problem-solving abilities and programming performance improve when they engage in self-regulation skills and systematic approaches, including planning, monitoring, and reflecting on their problem-solving activities (Loksa, 2016; Lopez, 2008). The three main skills for developing programming proficiency in students are the ability to read, trace, and write the program. The research emphasized that students who actively engaged in reading, tracing, and writing activities showed greater improvement in their programming abilities than those who passively observed or copied code examples (Luxton-Reilly, 2016). Luxton-Reilly argues that active engagement with coding tasks, along with regular practice and experimentation, is crucial for developing programming skills. Practical exercises provide opportunities for students to apply their knowledge, gain experience, and reinforce their understanding of programming concepts (Mayer, 1981). Busjahn (2013) examines the role of code reading as a pedagogical approach in teaching programming. They argue that while writing code is a fundamental aspect of programming education, reading code is equally important for developing students' comprehension, debugging, and problem-solving skills.

Another approach (Nelson, 2019) emphasizes the importance of program tracing, which involves understanding and predicting a computer program's execution. Program tracing is a fundamental skill for programming students, as it helps them identify errors and improve their code comprehension. The approach emphasizes the glass box approach, where the learner attempts to visualize the program's execution inside the computer, step by step. The author (Nelson, 2017) has proposed an online tutor, PLTutor, based on a pedagogy that allows learners to observe examples of causal relationships by stepping forward in time in the program's execution, one instruction at a time. The study (Nelson, 2017) has shown positive outcomes in learners who used program tracing of a programming language.

Tracing (Juha, 2013) is a key programming skill that expert programmers routinely use during both design and comprehension tasks; novice programmers often struggle greatly with it. It has been shown that students need instructor scaffolding to trace their programs. Another study by Hertz (2013) demonstrated that trace-based teaching led to statistically significant improvements in student grades, decreased drop and failure rates, and improved students' programming abilities.

The research study (Buffardi, 2012; Edwards, 2009) suggested using Test-Driven Development (TDD) in an introductory programming course to reinforce students' practice of writing test cases before writing solutions. The results of an empirical study showed positive outcomes in final correctness and coverage of the completed solution. This suggests that early testing yields both higher quality tests and solutions. Other studies have shown that students produced programs with 28% to 45% fewer defects per thousand lines of code than those who did not use TDD (Taufiqurrahman, 2022; Edwards, 2009; Edwards, 2003).

The findings of Rambally (1986) suggest that the choice of color scheme in programming affects program readability and comprehension. The three-color schemes used were: Color-scheme-A: Different colors indicated different blocks in a program. Color-scheme-B: Different colors were used to identify various statements or functions in the program. Black-and-White: The traditional black-and-white color scheme. The study showed that subjects who used programs with Color-scheme-B had the highest mean score for program comprehension, followed by those who used Color-scheme-A. Subjects who used black-and-white programs scored the lowest on the comprehension quiz. Another article (Mayer, 2005) also suggests that color-coding is effective in learning programming languages from video lectures. The empirical study showed that participants who studied the color-coded video lecture scored higher on the performance test than those who studied the grayscale video lecture.

According to Mayer (2005), color is classified as a form of visual signaling in the context of multimedia learning. The signaling principle suggests that incorporating signals or cues in multimedia materials can enhance learning outcomes. These signals are designed to guide learners' attention to relevant elements within the material or to highlight its organization. Cognitive load theory (Simon, 2019; Paas , 2003) posits that

there is a limit to the amount of information the human mind can process at one time. Color cues can help manage cognitive load by directing attention, reducing mental strain, and preventing cognitive overload.

The authors (Dzulkifli, 2013; Rambally, 1986) review various studies on the relationship between color and memory. They highlight that color can affect memory performance by influencing attention, arousal, and emotional states, which in turn affect cognitive processes such as encoding, consolidation, and retrieval of information. The review discusses various color-related factors, such as hue, saturation, and brightness, and their effects on memory. It presents evidence suggesting that certain colors, such as red and yellow, may enhance attention and arousal, thereby improving memory performance. However, the effects of color on memory are not uniform and can vary depending on individual differences, cultural background, and the nature of the task. The authors also discuss the use of color in educational settings and memory enhancement techniques. They suggest that incorporating appropriate colors in learning materials and instructional design can potentially enhance memory retention and recall. However, they caution that the effectiveness of color in memory enhancement should be considered alongside other factors, such as individual preferences and learning styles.

The paper (Al, 2004) examines the impact of color-coding as a visual aid in teaching English sentence structure. The study explores whether using different colors to highlight parts of speech (such as nouns, verbs, and adjectives) enhances students' comprehension, retention, and accuracy in constructing sentences. Students exposed to color-coded materials retained grammatical rules longer than those taught using traditional methods.

The author Liu (2021) investigated how color coding enhances programming education in multimedia learning environments. The study showed that participants who studied the color-coded video lecture scored higher on the performance test than those who studied the grayscale video lecture. The color coding helped learners easily identify different programming constructs (e.g., keywords, variables, functions), reducing cognitive load. Students exposed to color-coded programming syntax demonstrated better recall and faster debugging skills. The use of colors in syntax highlighting increases student interest and reduces frustration, particularly for beginners.

The paper (Liu, 2021) focuses on how color can serve as a memory cue and enhance the encoding and retrieval of visual information. The article discusses the concept of the "color code," which refers to the association between specific colors and particular information or concepts. The color code can facilitate memory by providing a visual cue that helps organize and recall information. Pruisner (1993) reviews various studies on color and memory. The findings suggest that color can significantly enhance memory for graphic information. The use of distinct and meaningful color-coding systems can improve memory performance by creating associations between colors and specific information or categories.

Prior work suggests using various instructional strategies to improve programming skills at the cognitive or metacognitive level. However, no studies have investigated the effects of instructional strategies on learner engagement at both cognitive and metacognitive levels when learning a programming language. We propose a pedagogy for learning programming languages at both levels, as discussed in the next section.

III. Methodology

Proposed pedagogy

In this section, we propose a pedagogy that focuses on learning programming at both the cognitive and metacognitive levels. At the cognitive level, learners will engage in writing programs that use different colors for different constructs. The aim of using various colors was to stimulate the learners' encoding, storage, and retrieval processes. At the metacognitive level, learners will use a trace table to simulate the program execution using various test cases. This process will enable learners to analyze and evaluate their code's execution across various scenarios, identify any logical errors, and modify their code accordingly.

The proposed steps of our method are as follows-

1. Write program with each line numbered.
2. Use various colors for different constructs in the program, as shown in Table I. We choose colors such as red, blue, green, and black that match those used in C++ editors.

Table I
Color-coding for various constructs

COLOR	PROGRAM CONSTRUCT
Red	Compiler directive
Blue	Keyword, Constant values
Green	Comments, Messages
Black	{variable/method/class} name, Operator, Punctuation.

3. Write test cases for various conditions: normal, extreme, boundary, and abnormal. The test conditions and their description are given in Table II.
4. Draw the trace table for each test condition. The trace table is used to visualize the program's execution for each case. The table (Table III) is constructed by listing the line no. of the statement being executed, the variables used as separate columns, followed by the conditions, if any, and finally the output. The trace table is illustrated using the solved example in Table V.

Table II
Test conditions

TEST CONDITION	DESCRIPTION
Normal	The input must be accepted by the system
Extreme	The input consists of the upper and lower limit that should be accepted
Boundary	The input consists of the upper and lower limit that should be accepted and rejected
Abnormal	The input must be rejected by the system

TABLE III
TRACE TABLE FOR TEST CONDITION

Line no	Variable1	Variable2	..	Condition	Output

Solved example

The process of solving a problem is illustrated with an example: Write a program to find the sum of the numbers from 1 to n. The value of n is to be taken as input from the user.

In step 1, the program is written using various color codes as shown below-

```
01.  #include <iostream>
02.  int main ()
03.  {
04.     int n, i, sum = 0;
05.     cout << "Enter a positive integer: ";
06.     cin >> n;
07.     for (i = 1; i <= n; ++i) {
08.        sum += i;
09.     }
10.     cout << "Sum = "<< sum;
11.     return 0;
12.  }
```

The next step is to write test data for the following conditions: normal, extreme, boundary, and abnormal (Table IV). The normal condition is checked for valid input. It is a typical valid input within the expected range of values. The extreme condition is a very large but still valid input, close to the upper realistic limit. Boundary condition checks for inputs at or just outside the expected limits. `-99` checks negative handling, `0` checks lower edge, and `9999999` checks beyond the usual maximum. Used to verify boundary value behavior. The abnormal condition checks for an invalid input (non-numeric character). It tests how the system responds to incorrect or unexpected data types.

The trace table (Table V) illustrates the program execution for the normal input test condition with input value 2. The first column lists the corresponding line numbers of the program. The next three columns record the values of the variables sum, i, and n, followed by the evaluation of conditions and the program output. For instance, at line 4, all variables are initialized to 0. At line 5, the message *"Enter a positive integer"* is displayed on the console. At line 6, the variable n is updated to 2. Subsequently, the loop iterations update the variables according to the specified conditions until the final output is produced.

IV. RESEARCH STUDY

This section reports the experimental design, settings, and results. The aim of the study was to investigate the effectiveness of the proposed pedagogy. The RQ formulated for the study is:

RQ- Does the process of writing programs using various colors and tracing programs for various test cases improve the learning of the programming language?

TABLE IV
TEST CONDITIONS FOR THE GIVEN EXAMPLE

TEST CONDITION	INPUT DATA
Normal	2
Extreme	999999
Boundary	-99, 0, 9999999
Abnormal	a

TABLE V
TRACE TABLE FOR NORMAL INPUT TEST CONDITION
Input: 2

Line no	sum	i	n	Condition	Output
4	0	0	0		
5					Enter a positive integer:
6			2		
7		1		1<=2? T	
8	1				
7		2		2<=2? T	
8	3				
7		3		3<=2? F	
10					Sum= 3
11,12					

Experiment design and Context

The experimental design for the study is a two-group posttest design (Cohen, 2002) aimed at comparing differences in performance scores between the two groups. The control group, which did not receive intervention, was from the previous year (2022), and the experimental group, which received intervention, belonged to the current year (2023). The students considered for our study are first-year engineering students from Computer Engineering and allied streams (Computer Engineering, Information Technology, Cybersecurity, Data Science, and Artificial Intelligence). These students are from various campuses across different cities and states in India, all under the same university. The total number of students considered for our study in the previous year (2022) was 1172, and in the current year (2023) is 1270. Admission to the first year is based on the university's common entrance exam. The admission criteria are the same for both current and previous years, and the syllabus is also the same; hence, both groups are equivalent with respect to prior knowledge and skills.

The students from the previous year were taught in a traditional way, with programming concepts taught in class,

and, during lab, students were instructed to draw a flowchart and then implement the program on a computer using the Dev-C++ compiler. The students in the current year were also taught in a manner similar to the control group; however, during lab, they were first asked to draw a flowchart, write a program using color-coding schemes, and then trace the program using a trace table in a workbook. Later, they were
asked to implement the program on the Dev-C++ compiler. To ensure uniformity in teaching, a one-week faculty orientation across campuses was held before the start of the semester. During the orientation, the teaching methodology to be adopted was discussed and practiced thoroughly by the faculty under the supervision of the experts. The faculty coordinator for the subject was selected to ensure coordination and monitoring across campuses by conducting fortnightly meetings.

As per the academic calendar, two Mid-tests were conducted: Mid-test 1 after one month of teaching and Mid-test 2 after two months of teaching. For the programming subject, the test paper consists of questions of various types: theory, output-finding, and programming. The students are supposed to write the answers on the answer sheet given to them. The performance of the students in Mid-test1 and Mid-test2 is compared for both groups.

The interview was conducted with two instructors who had taught the programming course both last year and this year. The purpose was to gather their insights into the effectiveness of the new teaching method compared with the previous year. Additionally, interviews were conducted with three students to understand their views on the usefulness of the new pedagogy.

The overall experiment design is shown in Fig. 1.

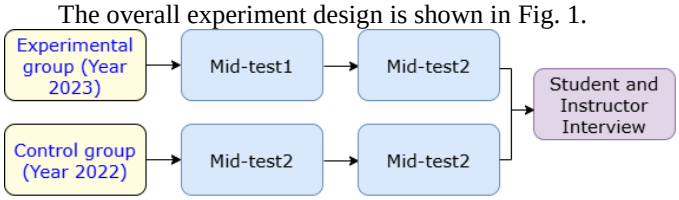

Fig. 1. Experiment design

Results

The Mid-test score analysis

The Mid-test scores of current-year students were compared with those of previous-year students. The paper pattern and the complexity level of the current-year and previous-year mid-term papers were the same. The distribution of marks for the questions for the test paper, with a maximum of 20 marks, is as shown:

i. Question on finding output of the given code snippet- 25%
ii. Theory questions- 8%
iii. Flowchart/algorithm questions- 33 %
iv. Programming questions- 33%

The mid-term marks were scaled to 10, and the comparison of mean scores of Mid-test1 and Mid-test2 of the current year with those of the previous year is shown in Table VI. The Mann–Whitney U test was used to compare the assessment scores of the independent 2022 and 2023 student cohorts. This non-parametric test was selected because the groups were independent and the score distributions did not require the assumption of normality (Mann & Whitney, 1947; Field, 2018). The green color indicated a significant difference and improvement in scores from the previous year; the red color indicated a significant improvement from the previous year; and the yellow color indicated no difference.

An analysis of 19 classes across various campuses was conducted. The results show that in Mid-test1, across 19 classes, current-year students showed a significant improvement in 5 classes compared to the previous year. However, in Mid-test2, out of 19 classes, significant improvement was observed in 10 classes among current-year students compared to the previous year. Overall, a positive outcome was observed in more than 50% of classes in the current year (2023) compared to the previous year (2022). To further analyze the performance, we selected one class from one of the campuses where no significant difference was observed and evaluated programming questions only to identify how current-year students have performed compared to previous-year students.

To identify how the students' programming skills have improved, we further investigated by evaluating the programming questions on Midtest1 for one of the classes in Campus 3 (Btech CE) in the current and the previous years of the same stream. The programming question evaluated for the previous year was:

Write a program to accept student details, including the student's name, CGPA, and the number of programming languages known. If the CGPA is above 3, and the number of programming languages known is above 2. Display Student Resume is shortlisted for an interview; otherwise, it is not.

The programming question evaluated for the current year was:

Write a program to accept student details, such as the student's name and the marks scored in five subjects. Calculate the percentage and display the grade as follows: Grade A, if the percentage is above 80, Grade B, if the percentage is between 60 and 79, Grade C, if the percentage is between 40 and 59, Grade D, if the percentage is less than 4.

The programs were evaluated for correctness on a scale of 0-4. The rubric is given in Table VII.

TABLE VI
COMPARISON OF MEAN SCORES OF MID-TEST1 AND MID-TEST2 OF CURRENT YEAR WITH PREVIOUS YEAR.

	Total students (2022)	Total students (2023)	Mid-Test1 (Mean) 2023	Mid-Test1 (Mean) 2022	Mann-Whitney U Test (p value)	Mid-Test2 (Mean) 2023	Mid-Test2 (Mean) 2022	Mann-Whitney U Test (p value)
Campus 1								
AIML*	60	66	6.12	4.68	0.01	5.67	4.35	0.03
CE*	116	126	5.97	6.34	0.51	6.04	5.15	0.00
CS*	71	77	7.55	7.92	0.89	7.18	6.80	0.08
IT*	61	59	3.82	4.23	0.63	4.18	3.28	0.03
MBATech-CE*	85	85	6.84	6.48	0.05	5.72	4.60	0.00
Campus 2								
AIDS*	62	68	4.07	2.63	0.00	3.98	3.92	0.87
CE*	73	73	5.75	4.16	0.00	5.79	4.25	0.00
MBATech-CE*	30	30	6.17	4.37	0.00	6.00	4.67	0.00
Campus 3								
CE*	85	101	6.84	6.48	0.05	5.72	4.60	0.00
IT*	56	61	4.80	6.59	0.00	4.34	6.23	0.00
MBATech-CE*	104	104	5.88	6.58	0.18	5.78	5.34	0.11
MBATech-AI*	46	52	5.61	6.48	0.10	4.98	6.35	0.00
DS*	54	70	5.93	8.17	0.00	5.67	7.57	0.00
Cybersecurity	65	65	7.70	7.15	0.02	7.03	5.27	0.00
Campus 4								
CE*	97	104	5.00	5.03	0.45	4.70	4.48	0.67
AIDS*	20	30	4.11	3.79	0.90	5.14	3.53	0.02
MBATech-CE*	20	37	3.27	3.25	0.36	5.07	2.25	0.00
Campus 5								
Btech-CSDS*	34	37	6.30	8.79	0.00	5.27	7.35	0.00
Campus 6								
CE*	17	25	5.88	5.12	0.36	5.80	4.29	0.12

*AIML-Artificial Intelligence and Machine learning, CE- Computer Engineering, CS-Computer Science, IT-Information Technology, DS-Data Science, AIDS-Artificial Intelligent and Data science, CSDS-Computer Science and Data science.

TABLE VII
RUBRIC TO EVALUATE THE PROGRAMMING QUESTIONS.

MARKS	4 MARKS	3- 1 MARK	0 MARKS
DESCRIPTION	The program produces the output correctly with no syntax error.	The program was written correctly but may not work for all test cases.	The program has syntax errors.

Two instructors assessed the responses according to the provided rubric, and the scores were subsequently compared using Cohen's Kappa coefficient to measure inter-rater reliability (Vieira, 2010). The achieved Cohen's Kappa coefficient was k=0.82, signifying a substantial level of agreement.

The descriptive statistics for the programming question scores of both the experimental and control groups are shown in Table VIII.

TABLE VIII
DESCRIPTIVE STATISTICS OF THE SCORES OF BOTH GROUPS

	EXPERIMENT GROUP	CONTROL GROUP
MEAN	3.06	2.17
SD	1.67	1.96

The Shapiro-Wilk test showed a significant departure from normality, $p < .001$, thus we did the Mann–Whitney U test to compare the mean of two sets of scores. Results of the Mann–

Whitney U test indicated a significant difference (p = .036) between the control and experimental groups.

Interview data analysis

Content analysis of interview data is a systematic process for identifying patterns, themes, and meaningful insights from qualitative data (Bengtsson, 2016). We systematically followed the steps of content analysis (Bengtsson, 2016) to analyze the interview data. First, the interview was transcribed from an audio recording into written text using online tools. The transcription was read by the authors to ensure accuracy and removal of irrelevant sections. The initial codes decided are: the effectiveness of the new pedagogy in learning a programming language, the challenges of the new pedagogy, and a comparison with the traditional pedagogy. The content analysis was conducted to manually categorize the interview sentences into predefined codes. To ensure that codes are applied consistently across all data, one author performed the coding, and another verified its accuracy.

The instructors' perceptions of each code are given:

1. Effectiveness of the new pedagogy

Instructors confirm that the color-coding and trace-table approach was a more structured and visually supportive teaching practice. This may be linked to enhancing student comprehension and reducing errors. The color coding was useful for students in identifying and understanding the keywords and variables, as it used two different colors for them. The trace table was useful, as students were able to dry-run their programs, which would otherwise be difficult for them. The students were able to write and think about test cases such as boundary cases and out-of-bound values. Compared to the traditional approach, where we ask students to use an editor to run the program, this approach lets them understand the syntax, logic, and errors on their own.

2. *Student Performance and Programming Improvement*

The instructor notes that 30% of students have improved their programming skills through the use of color coding and test cases. Because of color-coding, the students were able to follow the correct syntax and program writing in the theory exam. This suggests a positive outcome for some students but also indicates that the remaining students still struggle with programming, particularly in their first exposure to the subject.

3. *Challenges for Students*

While the new methods have shown some positive impact, there are still significant struggles for students, especially those encountering programming for the first time. This is indicative of the difficulty level in programming as a subject and the need for more effective teaching strategies.

Color-coding was difficult because students had to keep changing pens. It was effective initially for a few programs, but later it became redundant. Color coding may be used for the first few simple programs till they learn, and for complex programs, this can be avoided.

The student perceptions are given below:

1. Effectiveness of the new pedagogy

Students agreed that the color-coding was useful for remembering the exact keyword to use in the code and syntax. The Trace tables helped in understanding each step in the code, basic data structures like arrays, and the use of loops and conditions. The overall pedagogy helped to visualize the execution.

Writing a program helped them to understand the basic syntax and logic. They also mentioned that ChatGPT should not be used initially; once the basics are covered, students should be encouraged to use ChatGPT. This method, which includes writing the program using color-coding and trace tables, and then implementing it, has improved programming skills by clarifying how to use logic and where to apply it.

2. Challenges

Color coding is difficult when programs are complex, because students have to keep track of which color to use instead of focusing on the logic. Trace tables were difficult to trace for the complex programs.

3. Comparison with the traditional pedagogy

In previous years, the practice was done directly on the compiler, while the exam was based on writing code on paper, which did not help as much as it did this year.

V. Discussion

The RQ is answered based on the results presented in the previous section. Although the overall Mid-Test scores showed statistically significant improvements in only 50% of the classes, a more detailed analysis of the programming question revealed that the experimental group achieved a significantly higher mean score (Mean = 3.06, SD = 1.67) than the control group (Mean = 2.17, SD = 1.96). The Mann–Whitney U test indicated that this difference was statistically significant (p = .034), demonstrating that the proposed pedagogy positively influenced students' programming performance.

The quantitative improvement in programming scores can be explained through both cognitive and metacognitive learning processes. At the cognitive level, the color-coding approach provided visual cues that facilitated the encoding, organization, and retrieval of programming syntax. This interpretation is supported by the observed reduction in syntax-related errors among students in the experimental group. Compared with the control group, students exposed to color-coding made fewer mistakes, such as missing curly braces, incorrect variable declarations, omitted main() functions, and improperly structured if–else statements. These measurable improvements suggest that the visual representation strengthened students' recall of programming constructs.

At the metacognitive level, the use of trace tables required students to systematically predict, monitor, and evaluate program execution across multiple test cases before implementation. This reflective process encouraged learners to identify logical errors, verify program behavior, and revise their solutions prior to coding. The significantly higher programming question scores of the experimental group provide quantitative evidence that engaging students in program tracing enhanced their ability to reason about program logic rather than relying solely on trial-and-error execution in a compiler.

These findings are further supported by qualitative evidence. Students reported that the color-coding scheme made it easier to remember programming syntax, while trace tables improved their understanding of program logic. One student who experienced both the traditional and the proposed teaching approaches specifically indicated that the new pedagogy

improved both syntax recall and logical reasoning. In addition, instructors observed fewer syntax errors and more structured program development among students in the experimental group.

Overall, the findings indicate that integrating color-coding with trace tables supports programming learning through complementary cognitive and metacognitive mechanisms. The cognitive component enhances memory for syntax through visual encoding, while the metacognitive component develops self-monitoring and evaluation skills through systematic program tracing. Together, these processes resulted in significantly better programming performance compared with the traditional teaching approach. However, the results also indicate that the intervention did not benefit all classes equally, suggesting that additional scaffolding or differentiated instructional strategies may be required for students with varying levels of prior programming experience. Furthermore, instructors noted that while the approach was highly effective for introductory programming concepts, its effectiveness diminished as program complexity increased, indicating the need for adaptive strategies in advanced programming topics.

The present findings are consistent with previous research demonstrating that color serves as an effective memory cue, enhancing the encoding and retrieval of information (Pruisner, 1993). They also support studies reporting that active engagement through reading, tracing, and writing programs leads to greater improvements in programming skills than passive observation or code copying (Loksa et al., 2016; Lopez et al., 2008).

REFERENCES

Al Dulaimy, A. J. S. (2024). The effect of colour-coding on learning English sentence structure. American Journal Of Social Sciences And Humanity Research, 4(10), 133-142.

Bengtsson, M. (2016). "How to plan and perform a qualitative study using content analysis." NursingPlus Open, 2, 8-14.

Busjahn, T., & Schulte, C. (2013). The use of code reading in teaching programming. In Proceedings of the 13th Koli Calling international conference on computing education research (pp. 3-11).

Buffardi, K., & Edwards, S. H. (2012). Impacts of Teaching test-driven development to Novice Programmers. International Journal of Information and Computer Science, 1(6), 9.

Cohen, L., Manion, L., & Morrison, K. (2002). Research methods in education. Routledge.

Dzulkifli, M. A., & Mustafar, M. F. (2013). The influence of colour on memory performance: A review. The Malaysian journal of medical sciences: MJMS, 20(2), 3.

Edwards, S. H., Snyder, J., Pérez-Quiñones, M. A., Allevato, A., Kim, D., & Tretola, B. (2009). Comparing effective and ineffective behaviors of student programmers. In Proceedings of the fifth international workshop on Computing education research workshop (pp. 3-14).

Edwards, S. H. (2003). Improving student performance by evaluating how well students test their own programs. Journal on Educational Resources in Computing (JERIC), 3(3), 1-es.

Field, A. (2018). Discovering statistics using IBM SPSS statistics (Vol. 5). London: sage.

Hertz, M., & Jump, M. (2013, March). Trace-based teaching in early programming courses. In Proceeding of the 44th ACM technical symposium on Computer science education (pp. 561-566).

Isong, B. (2014). A Methodology for Teaching Computer Programming: first year students' perspective. International journal of modern education and computer science, 6(9), 15.

Juha, S. (2013). Notional machines and introductory programming education. ACM Transactions on Computing Education, 13(2), 1-31.

Liu, Y., Ma, W., & Zhu, T. (2021). Impacts of color coding on programming learning in multimedia learning: moving toward a multimodal methodology. Frontiers in Psychology, 12, 773328.

Loksa, D., & Ko, A. J. (2016). The role of self-regulation in programming problem solving process and success. In Proceedings of the 2016 ACM conference on international computing education research (pp. 83-91).

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016, May). Programming, problem solving, and self-awareness: Effects of explicit guidance. In Proceedings of the 2016 CHI conference on human factors in computing systems (pp. 1449-1461).

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the fourth international workshop on computing education research (pp. 101-112).

Luxton-Reilly, A. (2016). Learning to program is easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (pp. 284-289).

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics, 50-60.

Mayer, R. E. (Ed.). (2005). The Cambridge handbook of multimedia learning. Cambridge university press.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. ACM Computing Surveys (CSUR), 13(1), 121-141.

Nelson, G. L., Xie, B., & Ko, A. J. (2017). Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in CS1. In Proceedings of the 2017 ACM conference on international computing education research (pp. 2-11).

Nelson, G. L. (2019). Personalized, adaptive tutorials for program tracing skills. In Proceedings of the 19th koli calling international conference on computing education research (pp. 1-2).

Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. Educational psychologist, 38(1),

Pruisner, P. A. (1993). From Color Code to Color Cue: Remembering Graphic Information.

Rambally, G. K. (1986). The influence of color on program readability and comprehensibility. In Proceedings of the seventeenth SIGCSE technical symposium on Computer science education (pp. 173-181).

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive science, 12(2), 257-285.

Simon, Luxton-Reilly, A., Ajanovski, V. V., Fouh, E., Gonsalvez, C., Leinonen, J., ... & Thota, N. (2019). Pass rates in introductory programming and in other stem disciplines. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (pp. 53-71).

Taufiqurrahman, F., Widowati, S., & Alibasa, M. J. (2022). The impacts of test driven development on code coverage. In 2022 1st International Conference on Software Engineering and Information Technology (ICoSEIT) (pp. 46-50). IEEE.

Vieira, S. M., Kaymak, U., & Sousa, J. M. (2010). Cohen's kappa coefficient as a performance measure for feature selection. In International conference on fuzzy systems (pp. 1-8). IEEE.