The Development of a Visual Output Approach for Programming via the Application of Cognitive Load Theory and Constructivism

Marini Abu Bakar[1], Muriati Mukhtar[2]
Faculty of Information Science and Technology
Universiti Kebangsaan Malaysia
Bangi, Selangor, Malaysia

Fariza Khalid[3]
Faculty of Education
Universiti Kebangsaan Malaysia
Bangi, Selangor, Malaysia

***Abstract*—Programming is a skill of the future. However, decades of experience and research had indicated that the teaching and learning of programming are full of problems and challenges. As such educators and researchers are always on the look-out for suitable approaches and paradigms that can be adopted for the teaching and learning of programming. In this article, it is proposed that a visual output approach is suitable based on the current millennials affinities for graphics and visuals. The proposed VJava Module is developed via the application of two main learning theories, which are, the cognitive load theory and constructivism. There are two submodules which consist of eight chapters that cover the topics Introduction to Programming and Java, Object Using Turtle Graphics, Input and Output, Repetition Structure, Selection Structure, More Repetition Structures, Nested Loops and Arrays. To enable Java programs to produce graphical and animated outputs, the MJava library was developed and integrated into this module. The module is validated by three Java programming experts and an instructional design expert on the module content, design and usability aspects.**

***Keywords*—Introductory programming; CS1; novices; Java programming; learning; objects-first**

I. Introduction

Programming education research has been going on for over five decades. Teaching and learning of programming have continuously drawn the attention of researchers among academics. Many studies are conducted at respective institutions including studies on the programming languages used, curriculum aspects, teaching and learning approaches as well as supporting materials and software tools. Most researchers around the world agree that teaching and learning of programming are difficult for novice students as well as for teachers [1]–[5]. Hence, many institutions have taken measures to address this problem to motivate, enhance students' interest, skills and competitiveness in programming.

The main challenges that novice students face in learning programming are related to problem solving [6] , understanding programming concept [7], programming language syntax [8] and motivation [9], [10]. While the challenges that teachers face are the need for appropriate teaching methods and tools [10].

Starting with a review of existing approaches to teaching and learning of programming, this paper proposes a new approach that addresses three key issues. First is the current generation of students who prefer visual approaches; second is the nature of programming courses that cause students, especially novice students, to experience high cognitive load in writing programs [11]–[13]; and third is the nature of programming courses that require active participation of students in building their own knowledge based on existing knowledge [14]–[17]. Thus, this new approach, presented in the form of teaching modules, was built using visual elements and based on two main theories, cognitive load theory and constructivism.

This paper is organized as follows: Section II presents previous studies on teaching of programming approaches. Section III describes the learning theory applied in this study. Section IV details the result and discussion of the study. Section V describes the conclusion and further work.

II. Teaching of Programming Approaches

Many previous studies have discussed the approaches used in the teaching of programming. In addition to the traditional approaches, the commonly used approaches are visual programming, graphical and animation library and object-first.

A. Traditional Approach

Generally, the traditional approach to computer programming courses follows closely the order of the topics in most textbooks. The first section covers the topics of introduction, data types, assignment statements, arithmetic expressions, input/output followed by three basic programming structure namely sequence, selection and repetition structure. The second part of the course covers advanced topics such as arrays, strings, methods and classes. Students need to apply the basic concepts in the first section to solve the problems presented in the second section.

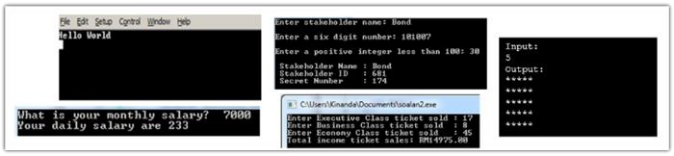


Fig. 1. Program Output in Text Mode.

In the traditional approach input and output are textual in which the students' program gets input from the keyboard and displays the output on the screen in text mode. Fig. 1 shows some examples of the output in text mode. Input and output in text mode were common in the early 1990s as most of the computers available in the programming lab were in text mode and the use of personal computers and laptops was quite limited. This traditional approach to output in text mode is less appealing to today's digital native students who are familiar with the latest computers and gadgets with graphical interfaces and touch screens.

Almost all institutions of higher learning in Malaysia adopt a traditional approach in teaching programming courses.

B. Visual Programming Approach

Among the popular visual programming language environment are Scratch and Alice which is widely used in primary and secondary schools [3], [18], [19]. In the visual programming language environments, programs are created by manipulating graphical components rather than writing textually. Creating programs is easier as there is no compilation errors and students are not required to know the syntax of a specific programming language.

Scratch was developed by Lifelong Kindergarten, MIT Media Lab in 2007 and is designed for students ages 8 and older. Author in [20] reports that students learning Scratch during primary school will easily learn advanced topics in high school. They don't have much trouble learning new topics and can reach a higher level of understanding for most basic concepts. As a result, some students choose to take programming courses in higher education. Students are also seen to have a high level of motivation and self-confidence.

Alice is a programming environment developed by the researchers in Carnegie Mellon. Alice provides an environment where students are able to drag-and-drop objects to create animations in three-dimensional. C++/Java programs are generated automatically. In higher institution Alice is usually used as a course in parallel with Computer Science 1 (CS1) or Computer Science 0 (CS0) courses [21]. CS1 generally refers to the first computer programming course in the computer science programme while CS0 is the programming related course at the pre-university level. From this study it was found that students taking Alice courses are better compared to those who do not take Alice courses in CS1. In this case, Alice is an additional course rather than being used extensively in the CS1 programming course.

C. Graphics and Animation Library Approach

Most graphics and animation library approaches are derived from LOGO. The LOGO programming language was introduced in 1967 by Seymour Papert with several researchers at the Massachusetts Institute of Technology [22]. LOGO is a language that teaches kids the basics of computer programming. In LOGO, the turtle is a cursor that can be controlled and operated according to the simple instructions given. Lines are drawn according to the movement of the turtle cursor. When it was first introduced in the 1960s, the LOGO language commands controlled turtle-shaped physical cursors. Later, with the technological advancements of the computer display screen, the turtle is represented as a cursor on the screen as in today's computer.

Graphics libraries have long been used for teaching of programming in universities worldwide. In [23] from Standford University initially developed the Turtles graphics library in ANSI C and subsequently translated it into Java. He encountered some problems in the implementation of the library because the earlier version of Java introduced in 1995 was relatively unstable.

Author in [22] developed Turtles package for use in teaching CS1 courses using Java at the University of Aarhus, Denmark. He uses the inverted curriculum approach proposed by [24]. With this approach important topics and concepts are introduced first and the details are explained later. The Turtles package has been used in all topics in the course. They reported positive effects, but no analysis was performed to show its effectiveness. Author in [9] also use turtle graphics in programming courses for prospective teachers. They report that this approach increases student motivation.

Another very similar approach is Karel the Robot in which the character is a robot named Karel in a simple world represented as a grid indicating streets and avenues. The Karel the Robot character can move one step forward, turn left, place and collect an object known as a beeper [25]. Author in [25] has used the Karel the Robot in programming courses at the University of Waterloo. Karel the Robot was first developed in 1981 by Richard Pattis to introduce Structured Programming courses using Pascal Language. Another popular variations of Karel are Robots is Karel J Robot [26] which are widely used in teaching basic programming courses using Java.

D. Objects-First Approach

Object-oriented programming is a widely used programming paradigm in both industry and education [27]. Almost all universities have object-oriented programming courses in the curriculum. Object-oriented programming is initially considered as an advanced course and is included in the middle or at the end of the curriculum. This situation has changed and today many universities have introduced object-oriented programming as their first course of programming. Among the object-oriented programming languages are Java, C++, C#, Eiffel, Python and Ruby.

The objects-first approach which was introduced in the ACM Computing Curricula 2001 document emphasized that the principles of object-oriented programming are introduced from the very beginning. The strategy was to begin introducing the concept of objects and classes and then followed by the structure of control, repetition and subsequent topics as in the traditional method.

Many studies highlight that teaching the basic concepts of object-oriented programming is difficult [27]–[30]. This is because many different concepts need to be understood as well as the skills that must be learned before students can write the program. Author in [27] emphasizes that object-oriented teaching is best used as an object-first approach compared to the object-later approach, by starting with a procedural programming approach and then switching to

objects. This is because the transition from procedural to object is more difficult compared to the difficulty of learning from the beginning. However, many textbooks use the object-later approaches.

The traditional approach to programming courses that produce text output is less appealing to today's generation Z students who are more inclined to visual learning styles. However, popular visual programming approaches such as Scratch and Alice are not suitable for higher education particularly for Computer Science students. Therefore, a new approach to the generation Z student learning style is needed. The proposed module named VJava Module uses the graphical and animation library and objects-first approaches. This module allows students to write Java programs textually that uses objects from the developed MJava graphics library to produce visual outputs in the form of graphics and animations. This VJava Module aims to increase students' interest and reduce anxiety at the beginning of the programming course which is perceived as difficult. To develop the VJava Module, two learning theories are applied; the cognitive load theory and constructivism theory. Both theories are discussed in the following sections.

III. Learning Theory

Learning theory describes how knowledge is absorbed, processed, and stored during learning. Learning theories need to be taken into account in designing the module to make them more effective and to achieve the objectives. This section discusses the learning theory applied to develop the VJava Module and the methodology of the research.

A. Cognitive Load Theory

Cognitive load theory was introduced by Sweller [31], [32] in the 1980s as a study of problem solving. It emphasised that all information is processed in a working memory and then stored in long-term memory for later use. Cognitive load theory basic premise is that the capacity of working memory is limited and can only process some information in a short duration of time.

Computer programming is a skill based course that is difficult and challenging which places a heavy cognitive load on the learners. Learning will be restrained by limited information processing capacity. If a learning task or activity requires cognitive capacity beyond its limits, that learning will be hindered [33].

There are three types of cognitive load which are intrinsic, extraneous and germane [32]. Internal cognitive load is related to the complexity of learning material and existing knowledge. Someone expert in programming and have extensive knowledge will learn easily compare to a student who has no direct knowledge of programming. This means that internal cognitive loads cannot be modified through instructional design [33].

Extraneous cognitive load is related to the design of teaching materials which can be modified by organizing the content of the materials. Novice students frequently use the means-ends analysis strategy in solving problems in the problem-solving approach [32]. Using this strategy will result in high usage of working memory resources resulting in a lack of existing cognitive resources. This causes cognitive activity to fail in working memory and thus impedes learning.

Germane cognitive load is an important cognitive load to explore in this study. This load refers to the construction of subsequent schemes as the primary goal of learning [32]. For example, giving students an example to solve a problem will help them understand the important steps in solving the problem and subsequently develop the problem solving scheme. The instructional design should guide the students to develop a scheme to increase the germane cognitive load.

The relationship between intrinsic, extraneous and germane cognitive load can be seen in the following three situations: (1) For situations where intrinsic cognitive load is low (easy learning content), and sufficient memory resources, students will be able to perform the learning process despite the high extraneous cognitive load (poor presentation of teaching material); (2) In situations where high intrinsic cognitive load (difficult teaching content) and high extraneous cognitive load, the cognitive load overcome mental resources and learning processes may fail; (3) Situations in which the external cognitive load in (2) is reduced, and the germane cognitive load is enhanced to facilitate the learning process [34].

Intrinsic cognitive load cannot be changed with the design of teaching materials. To produce meaningful learning, the design of instructional materials should reduce extraneous cognitive load and nurture germane cognitive load. This is because the extraneous cognitive load does not have a positive effect on the learning process, in contrast to the germane cognitive load that can help to improve the learning process.

B. Constructivism Learning Theory

The constructivism learning theory pioneered by Jean Piaget is based on the premise that knowledge is built by a person as a result of his mental activity rather than being conveyed by an instructor. Learning happened by interpreting the meaning of a concept based on existing knowledge and experience [14], [15]. Teachers encourage students to explore how an activity helps them to understand a concept. Constructivist teachers provide learning environments based on problems that need to be solved individually or collaboratively, while students produce their meaningful artifacts [15]. Learning occurs actively in solving problems with teachers acting as facilitators in nurturing meaningful learning.

Constructivism does not deny the role of lecturers or knowledge expert. Constructivism has changed that role, so lecturers helps students to build knowledge instead of just presenting facts. Constructivist lecturers provide tools such as problem solving and inquiry-based learning activities, sharing experiences, discussions, creating concept maps and building a broader picture of concept [35]. Students formulate and test their ideas, draw inferences and conclusions, and integrate their knowledge in a collaborative learning environment. Constructivism transforms students from passive recipients to active participants in the learning process. Guided by lecturers, students actively build their knowledge rather than

receiving knowledge directly from a lecturer or textbook. With a well-designed classroom environment, students will learn how to learn.

One face of Constructivism learning theory is the Constructionism introduced by Seymour Papert [36], [37] that asserted learning occurs "especially felicitously" when learners engaged in constructing artifacts. Papert introduced constructionism in association with LOGO, a programming language designed to enable the study of abstract concepts in mathematics, geometry, physics and others by manipulating computational objects [38]. The most common artifacts in constructionism today are in digital form.

C. Methodology

The VJava Module is developed using ADDIE, the five-phase development methodology [39]. The first phase is the analysis phase which comprises of literature review and preliminary study to determine the problems faced in the learning of programming, determine the appropriate approach and relevant learning theories to apply in the proposed method. The second phase is the design phase, which involves the design of the graphics library and the learning module. This is followed by the third phase, which is the development phase to develop the module. At the end of this phase, the process of verification of the module by the experts is conducted. The next phase is the implementation phase where the updated modules based on expert reviews are tested in the pilot study before being implemented in the actual learning environment. The final phase is the evaluation phase on the students' response after learning using this module. This paper discusses the results of a study that covers phases one to three.

IV. Results and Discussions

This study applies two learning theories, namely cognitive load theory and constructivism in designing and developing VJava Modules for basic programming course. The VJava Module consists of two submodules; the VJava Module I consists of five chapters and the VJava Module II that consists of three chapters. This module uses the MJava graphics library to produce graphics and animated outputs.

A. MJava Library

The MJava library consists of two Java packages, the MTurtle package and the MGraphics package. The MTurtle library implements the turtle graphics concept introduced by Seymour Papert. The output generated by programs using MTurtle library are display in animated form as the turtle moves. The MGraphics library can produce basic graphical shapes output such as line, rectangle, oval, polygon and text. Combination of these basic forms can produce complex drawing.

B. Application of Cognitive Load Theory in VJava Module

The application of cognitive load theory in the VJava Module are (1) graphical and animated output; (2) learn programming by examples; (3) program tracing; (4) step by step guide. These feature are aim to reduce the students' cognitive load in learning programing.

1) Graphical and animated output: Using the MJava library, programs written by students can produce graphical and animated output. Students can better understand the basic concepts of Java such as sequence, repetition and selection structure by associating the graphical and animation output with the written program. The output helps students to understand the program flow especially when they need to correct errors.

Fig. 2 shows an example of program code and the output generated written in the traditional approach as compared to the graphical and animated output approach using MTurtle library. This method will increase students' interest and facilitate the understanding of basic concepts. Hence, this method will reduce students' internal cognitive load in understanding basic programming concepts.

2) Learn programming by example: All concepts in the VJava Module are presented using appropriate examples which include programs to solve specific problems and the output produced. For example, a program to draw a rectangle is shown in Fig. 3 while the output is shown in Fig. 4.

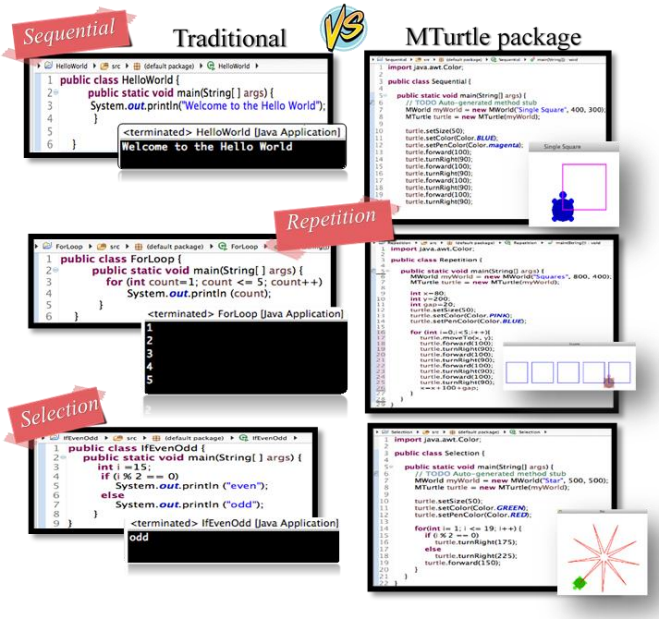


Fig. 2. Traditional Approach vs Graphical Output Approach.

```
public class MyTurtleApp {
    public static void main(String[] args) {
        MWorld myWorld = new MWorld("My First Turtle World");
        MTurtle turtle = new MTurtle(myWorld);          // (a)
        // draw a square
        turtle.forward(100);                            // (b)
        turtle.turnRight(90);                           // (c)
        turtle.forward(100);                            // (d)
        turtle.turnRight(90);                           // (e)
        turtle.forward(100);                            // (g)
        turtle.turnRight(90);                           // (h)
        turtle.forward(100);                            // (i)
    }
```

Fig. 3. Program Example.

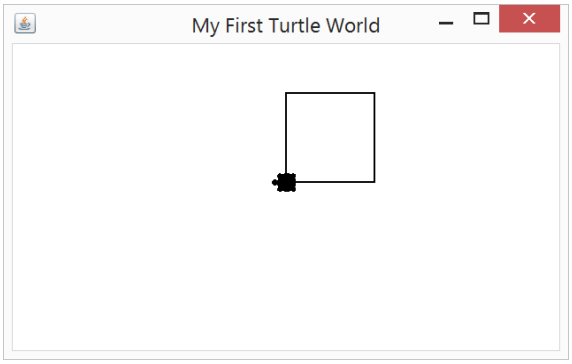


Fig. 4. The Output.

3) Program tracing: Programs in the VJava Module are explained by tracing the program line by line. Program tracing is a method to simulate how the program is executed on paper, by step through it line by line. With this explanations students can understand the program discussed before they can write their code. Fig. 5 shows an example of tracing a program to draw a square.

4) Step by step guide: The step by step guide is to explain important tasks. For example, to describe the process to install JDK and Eclipse on a computer. The step by step guide enables students to follow it to perform the tasks on their own.

C. Application of Constructivism Theory in VJava Module

Features of the theory of Constructivism applied in the module are (1) scaffolding; (2) forming new ideas; (3) exploration of new ideas; (4) construction of new knowledge.

1) Scaffolding: The MJava Module implements scaffolding learning that consisting of 5 stages namely requirements, concepts and syntax, reinforcement, program segment exercise and programming exercise (Fig. 6).

The five stages are explained with an example as follows:

a) Requirements: At this stage the requirements for a topic are described. For example, in Topic 4 Repetition Structure, it is explained how a repetition structure can simplify the writing of the program without duplicating the program statements. This is demonstrated by showing an example of a program to draw two squares side by side by copying the program segment to draw a square and repeating it twice with some changes to set the location and direction of the turtle of the second square. This is discussed in detail with program examples until the purpose is achieved.

The next query put forward is "how to output 5 or maybe 10 squares in a row. Do we need to repeat the program segment 10 times?" (Fig. 7) This situation justifies the need for a repetition structure.

a) Concepts and syntax - In this stage the concepts of a topic are discussed with simple examples and the syntax of the statement is also explained. For example, the program to draw 10 squares in a row as discussed in (a) is shown in Fig. 8. In this example, a repetition statement, that is for loop is introduced and the syntax is explained in Fig. 9.

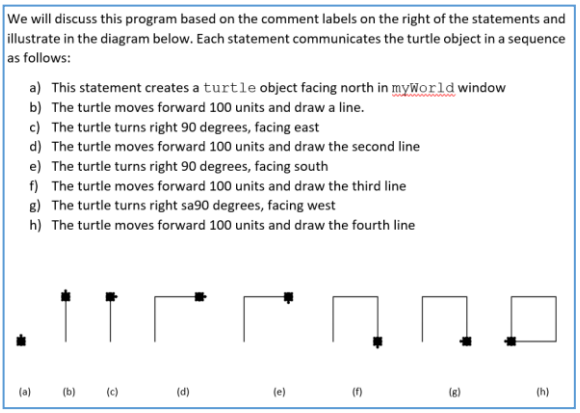


Fig. 5. Tracing a Program that Draws a Square.

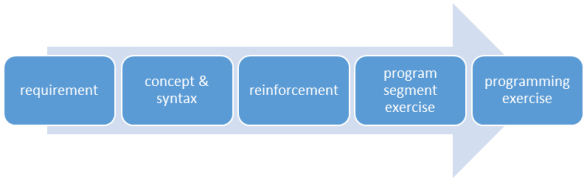


Fig. 6. Scaffolding.

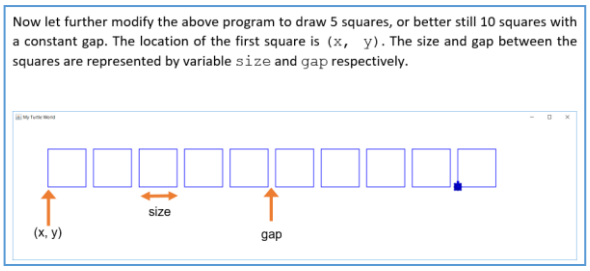


Fig. 7. Question Put Forward to Draw 10 Squares Side by Side.

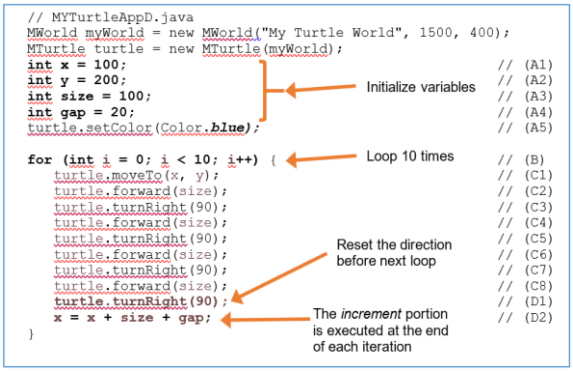


Fig. 8. Program to Draw 10 Squares in a Row using for Statement.

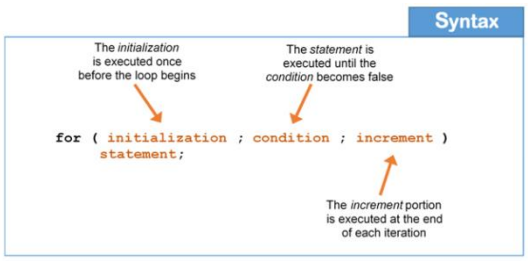


Fig. 9. Syntax for "for Loop".

b) Reinforcement- This stage further explains the program example by tracing the program. Using the same example to draw 10 squares, the program is traced for each loop to show the drawing displayed and the value and calculation of variables representing the coordinates and direction of the turtle.

c) Program segment exercise - Simple questions such as to trace some program segments were given to familiarize students with the concepts being discussed. Further exercises related to the discussed problems are also provided, for example to draw 5 horizontal lines. Using the program to draw 10 squares, students can write a solution to this problem.

d) Programming exercise - The question given in this stage requires students to apply the concepts discussed to solve the problem on their own. An example of the question is to draw 5 cascading squares.

These five scaffolding stages of learning are emphasized in the learning outcomes of the related topics. Examples of learning outcomes for Topic 4 Repetition Structure are shown in Fig. 10. Stages 1 to 5 refer to learning outcomes 1 to 5, respectively.

2) Forming new ideas: Forming new ideas is a key feature of the theory of constructivism. Based on the examples given, students are anticipated to form new ideas in solving problems. For example, based on the given example to draw a square, students should be able to form new ideas to draw a triangle (Fig. 11). The idea needed is to determine the degree of angle for the turtle to turn to draw the lines.

This chapter demonstrates the concept of repetition control structure using MTurtle class library. The learning outcome of this chapter is:

- understand the need of loop control structure
- understand the concept of loop control structure
- understand the syntax of `for` loop using `MTurtle` objects
- understand the flow of control of `for` loop using `MTurtle` objects
- able to apply `for` loop to draw multiple `MTurtle` objects

Fig. 10. Learning Outcome for Topic 4 Repetition Structure.

Task 3: Draw equilateral triangle

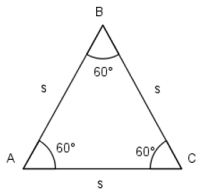
1. Create a new project and name it **AxxxxxxTask3** (Axxxxxx is your student number). In that project, add a new class called **AxxxxxxTask3**.

   Start with a new `main` method. Create a `turtle` object in `MWorld`.

   a. Set pen width to 3.

   b. Draw an equilateral triangle with sides of length = 150.

   An equilateral triangle has three sides of equal length, connected by three angles of equal width.



Question: What is the statement to set pen width to 3?

Fig. 11. Example of Question to form New Ideas.

3) Exploration of new ideas: In VJava Module, students were given examples based on the concepts discussed. In the programming exercises, students are asked to explore the MTurtle and MGraphics software library by referring to the library documentation. The documentation listed all the methods that can be used to create a drawing using the respective library. For example, in an exercise, students are asked to change the color and line thickness of the turtle. Students are guided to explore by referring to the MTurtle documentation.

4) Construction of new knowledge: One of the exercises in Topic 4 Repetition Structure is to trace a program segment that produces a pattern called polyspiral. Students were given a note that describes polyspiral as a hint. This is then followed by a programming exercise where students were asked to write a program that produces the polyspiral pattern. New knowledge constructed in this exercise is that pseudo-code for the polyspiral that can be converted into program code. Students can refer to the program tracing exercise and modify the program code to solve this problem.

Other ensuing exercise questions require students to change the values of some variables to produce interesting polyspiral shapes. A new knowledge constructed that polyspiral patterns can be produced with different variable values that applies the geometric coordinate concept in mathematics.

D. Validation of Module by Experts

The expert validation process is conducted on the developed VJava Module, to determine the validity, and usability of the module. The evaluation was performed by three programming experts and one instruction design expert as shown in Table I. PK1, PK2 and PK3 are the programming experts, while the PRB is the instruction design expert.

The questionnaire for programming experts consists of three sections, Part A for the demographics of respondents, Part B for module content and Part C for module usability. The survey for instruction design experts also comprises three sections, Part A for the demographics of the respondent, Part B for module design and Part C for module usability.

1) Module Content Validation: Content validity is based on the mean score of learning outcomes of the topics in the VJava Module. Findings for expert evaluation of learning outcomes are summarized based on topics as shown in Table II.

The measure used is a 5-point Likert scale (1 = strongly disagree, 2 = disagree, 3 = moderate agree, 4 = agree, 5 = strongly agree). The descriptive range for the mean score is divided into three levels: low (1.00 - 2.33), medium (2.34 - 3.66) and high (3.67 - 5.00). Based on the expert evaluation, all chapters showed high scores, in the range of 3.67 to 5.00. The highest score was for Topic 4 Repetition Structure with a mean score of 4.93 while the topic with the lowest score was Topic 8 Arrays with a score of 3.89. Topic 4 is the topic that introduces the concept of repetition that is discussed in detail. Topic 8 is on arrays which is considered as a difficult topic for novices.

TABLE. I. EXPERT PROFILE

No.	Designation	Expertise	Experience
PK1	Senior lecturer	Programming education, software testing	20 years
PK2	Professor	Artificial intelligence	24 years
PK3	Associate professor	Programming language design, software testing	17 years
PRB	Lecturer	Education technology, computational thinking	11 years

TABLE. II. EXPERT EVALUATION ON LEARNING OUTCOME

Topic	Title	Mean score
1	Introduction to Programming and Java	4.40
2	Objects : Using Turtle Graphics	4.47
3	Input and Output	4.58
4	Repetition Structure	**4.93**
5	Selection Structure	4.67
6	More Repetition Structure	4.87
7	Repetition Structure : nested loop	4.67
8	Arrays	**3.89**

Expert reviews of the overall content of the module are shown in Table III. The reviews are positive with some suggestions for improvement.

2) Modul design validation: The design expert evaluates and validates that module design includes features of cognitive load theory, constructivism theory. The overall review by the design expert are as follows:

"The content structure of this module is clear and provides some examples of incremental program development: 1) examples to illustrate concepts; 2) guided examples for reinforcement (work-example); and 3) activities/exercises that students need to complete without guidance. Need to include more exercises - a broader context that resembles the program used in the industry/field (suggestion)."

3) Usability of the module: All experts have evaluated and validated the usability of the module in terms of learnability, efficiency, memorability and satisfaction. As for the error aspects, experts recommend the need for minor changes in terms of spelling and grammar. An overall review of the usability of the module is shown in Table IV. All of the experts confirm that the VJava module is suitable to be used in basic programming course.

E. Limitation of the Study

This study focuses on the early part of a basic programming course in which most novice learners have difficulties in understanding the basic concept of programming and the syntax of a programming language. Using a simple approach to write programs that produce graphical and animated output will increase students' interest and reduce anxiety, hence will reduce the cognitive load in learning programming. The next part of the programming course which focuses on problem solving and programming skills is part of a different study.

TABLE. III. EXPERT REVIEW ON MODULE CONTENT

Expert	Remarks
PK1	1. This module is very interesting. It's generally very interesting and easy to understand. 2. Only the topic on array seem difficult.
PK2	1. An interesting learning method using visuals and graphics. 2. Simple examples are used to introduce a concept.
PK3	1. This is a great effort to improve the teaching quality of programming. The proposed concept can be further refined to increase students' understanding of Java. 2. I strongly agree with the concept of 'tracing' used and recommend to emphasis on this. 3. Need to include additional references especially from youtube to enable students to learn more about each topic. 4. Overall, I agree with this module and hope to improve it in the future.

TABLE. IV. EXPERT REVIEW ON USABILITY OF THE MODULE

Expert	Remarks
PK1	1. Practical. The use of visuals makes it easier for students to remember/understand what to do (visual/visual algorithms)
PK2	1. Support materials for the laboratory are very suitable 2. It should be used with textbooks and reference books for an in-depth explanation. 3. Overall, it is simple, interesting and appropriate approach for 21st-century learning
PK3	1. This module is simple to understand.
PRB	1. Generally, this module is easy to use. The content is presented in a structured way. 2. The use of icons to mark 'sections' in module content is good.

V. CONCLUSIONS

This article describes the development of the VJava Module that uses visual output approaches for a basic programming course. This module uses the developed MJava library that can be integrated into student programs to produce graphical and animated outputs. Cognitive load theory and constructivism are applied in the design of this module. The VJava module has been validated by programming and instruction design experts in terms of content, design and usability of the module. All experts have responded positively and agreed that this module is suitable for use in the teaching and learning of programming in higher education institutions.

In the next phase of this project, the VJava module will go through a pilot test process before being implemented in a real learning environment and evaluated for its effectiveness. The results of this evaluation will be reported in the subsequent paper.

ACKNOWLEDGMENT

Appreciation to the Ministry of Education of Malaysia for supporting this work through its research grant FRGS/1/2016/ICT01/UKM/02/3.

REFERENCES

[1] M. M. Bashir and A. S. M. L. Hoque, "An effective learning and teaching mod el for programming languages," J. Comput. Educ., vol. 3, no. 4, pp. 413–437, 2016.

[2] A. Luxton-Reilly et al., "Introductory programming: A systematic literature review," in Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, 2018, pp. 55–106.

[3] M. Noone and A. Mooney, "Visual and Textual Programming Languages: A Systematic Review of the Literature," Oct. 2017.

[4] P. C. Tavares, P. R. Henriques, and E. F. Gomes, "A computer platform to increase motivation in programming students-PEP," in CSEDU 2017 - Proceedings of the 9th International Conference on Computer Supported Education, 2017, vol. 1, pp. 284–291.

[5] R. Latih, M. Abu Bakar, N. Jailani, N. M. Ali, S. M. Salleh, and A. M. Zin, "A Design for Challenge-based Learning of Programming," Int. J. Adv. Sci. Eng. Inf. Technol., vol. 8, no. 5, pp. 1912–1918, 2018.

[6] R. Smetsers-Weeda and S. Smetsers, "Problem solving and algorithmic development with flowcharts," in ACM International Conference Proceeding Series, 2017.

[7] A. Vihavainen, J. Airaksinen, and C. Watson, "A Systematic Review of Approaches for Teaching Introductory Programming and Their Influence on Success," in Proceedings of the Tenth Annual Conference on International Computing Education Research, 2014, pp. 19–26.

[8] R. M. Kaplan, "Using Problem-Based Learning in a CS1 Course -Tales from the Trenches," in Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS), 2015, pp. 86–90.

[9] R. Horváth and S. Javorský, "New Teaching Model for Java Programming Subjects," Procedia - Soc. Behav. Sci., vol. 116, pp. 5188–5193, 2014.

[10] R. P. Medeiros, G. L. Ramalho, and T. P. Falcao, "A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education," IEEE Trans. Educ., 2019.

[11] M. Yousoof and M. Sapiyan, "Optimizing instruction for learning computer programming- A novel approach," Commun. Comput. Inf. Sci., vol. 516, pp. 128–139, 2015.

[12] S. Mohd Salleh, Z. Shukur, and H. Mohamad Judi, "Scaffolding Model for Efficient Programming Learning Based on Cognitive Load Theory," Int. J. Pure Appl. Math., 2018.

[13] X. Li, "Application of Cognitive Load Theory in Programming Teaching," J. High. Educ. Theory Pract., vol. 16, no. 6, pp. 57–65, 2016.

[14] M. Guzdial, "Constructivism vs. Constructivism vs. Constructionism," Computing Education Research Blog, 2018. [Online]. Available: https://computinged.wordpress.com/2018/03/19/constructivism-vs-constructivism-vs-constructionism.

[15] M. Rob and F. Rob, "Dilemma between constructivism and constructionism: Leading to the development of a teaching-learning framework for student engagement and learning," Journal of International Education in Business, vol. 11, no. 2. Emerald Group Publishing Ltd., pp. 273–290, 05-Nov-2018.

[16] A.-M. Gasaymeh, I. A. AlJa'afreh, A. Al-Dmour, and M. A. Alrub, "Higher Education Students' Preferences for Applying the Principles of Constructivism in Learning Programming Languages with the Use of ICTs," J. Stud. Educ., vol. 6, no. 3, pp. 168–187, 2016.

[17] M. A. Bakar et al., "Kerangka Bagi Persekitaran Pembelajaran Berpusatkan Pelajar untuk Latihan Pengaturcaraan Kendiri," ASEAN J. Teach. Learn. High. Educ., vol. 10, no. 1, pp. 24–37, 2018.

[18] J. M. Sáez-López, M. Román-González, and E. Vázquez-Cano, "Visual programming languages integrated across the curriculum in elementary school: A two year case study using 'scratch' in five schools," Comput. Educ., vol. 97, pp. 129–141, Jun. 2016.

[19] H. Husain, N. Kamal, M. F. Ibrahim, A. B. Huddin, and A. A. Alim, "Engendering problem solving skills and mathematical knowledge via programming," J. Eng. Sci. Technol., 2017.

[20] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From Scratch to 'Real' Programming," ACM Trans. Comput. Educ., vol. 14, no. 4, pp. 1–15, 2015.

[21] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," ACM SIGCSE Bull., vol. 35, no. 1, p. 191, 2003.

[22] M. E. Caspersen and H. B. Christensen, "Here, there and everywhere - on the recurring use of turtle graphics in CS1," in Proceedings of the Australasian conference on Computing education - ACSE '00, 2000, pp. 34–40.

[23] E. Roberts and A. Picard, "Designing a Java graphics library for CS 1," ACM SIGCSE Bull., vol. 30, no. 3, pp. 213–218, 1998.

[24] B. Meyer, "Towards an object-oriented curriculum," J. Object-Oriented Program., vol. 6, no. 2, pp. 76–81, 1993.

[25] B. W. Becker, "Teaching CS1 with Karel the Robot in Java," Proc. thirty-second SIGCSE Tech. Symp. Comput. Sci. Educ. (SIGCSE '01)., pp. 50–54, 2001.

[26] J. Bergin, M. Stehlik, J. Roberts, and R. Pattis, Karel J Robot : a gentle introduction to the art of object-oriented programming in Java. Dream Songs Press, 2013.

[27] M. Kölling, "The problem of teaching object-oriented programming. Part 1," J. Object Oriented Program., vol. 11, no. 8, pp. 8–15, 1999.

[28] N. Thota and R. Whitfield, "Holistic approach to learning and teaching introductory object-oriented programming," Comput. Sci. Educ., vol. 20, no. 2, pp. 103–127, 2010.

[29] M. Berry and M. Kölling, "Novis: A notional machine implementation for teaching introductory programming," in Proceedings - 2016 International Conference on Learning and Teaching in Computing and Engineering, LaTiCE 2016, 2016.

[30] E. J. Johan, S. Idris, M. A. Bakar, and M. Mukhtar, "Persuasive Object Oriented Programming Lab Assignment Framework," Int. J. Technol. Incl. Educ., vol. 4, no. 1, pp. 557–565, 2015.

[31] J. Sweller, "Cognitive load theory, learning difficulty, and instructional design," Learn. Instr., vol. 4, no. 4, pp. 295–312, 1994.

[32] J. Sweller, J. J. G. van Merriënboer, and F. Paas, "Cognitive Architecture and Instructional Design: 20 Years Later," Educational Psychology Review. 2019.

[33] T. de Jong and T. Jong, "Cognitive load theory, educational research, and instructional design: some food for thought," Instr. Sci., vol. 38, no. 2, pp. 105–134, 2010.

[34] N. H. Jalani and L. C. Sern, "The Example-Problem-Based Learning Model: Applying Cognitive Load Theory," Procedia - Soc. Behav. Sci., 2015.

[35] T. Pittayapiboolpong and P. Yasri, "Development of an Integrative Learning Unit to Enhance Students' Conceptual Understanding of Dissolution and Their Reasoning Sophistication," J. Res. Sci. Math. Technol. Educ., vol. 1, no. 3, pp. 283–310, 2018.

[36] S. Papert, Mindstorms: Childern computer and powerful ideas. 1980.

[37] G. Walton, M. Childs, and G. Jugo, "The creation of digital artefacts as a mechanism to engage students in studying literature," Br. J. Educ. Technol., vol. 50, no. 3, pp. 1060–1086, May 2019.

[38] M. Lodi, D. Malchiodi, M. Monga, A. Morpurgo, and B. Spieler, "Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey," Olympiads in Informatics, 2019.

[39] R. M. Branch, Instructional design: The ADDIE approach. 2010.