International Journal of Information and Communication Technology Education
Volume 18 • Issue 1

A Blended Learning Approach:
Motivation and Difficulties in Learning Programming

Su Ting Yong, University of Nottingham, Malaysia*
https://orcid.org/0000-0001-5910-6721

Kung Ming Tiong, University of Nottingham, Malaysia

ABSTRACT

This study explored students' motivation and difficulties in learning programming in a blended learning environment. The face-to-face classroom instructions were blended with digital learning instructions. The study adopted a convergent parallel design mixed methods research and involved 209 pre-university students. The findings were as follows: (1) Looping was hard. Students faced difficulties in program design, problem-solving, and debugging of repetition structures. (2) Students with prior programming experience were more motivated to learn by innate psychological needs: autonomy and competence. (3) Competence-motivated students performed better in programming, but no significant findings for autonomy and relatedness motivated students. A blended learning environment must be structured around the nature of the subject to satisfy students' innate psychological needs. Digital learning materials can support understanding of certain programming concepts, but teaching instructors play an important role in providing academic, mental, and emotional support.

KEYWORDS

Autonomy, Challenge, Flipped Classroom, Loop, MATLAB, Nested Loop, Peer Learning, Pre-University Students, Relatedness, Self-Determination Theory

INTRODUCTION

The Covid-19 pandemic has changed the education landscape entirely. When it comes to a new normal in education, *information and communication technologies* (ICT) integration is often part of the teaching and learning strategies. Since the Covid-19 outbreak, many educational institutions could not operate fully, and blended learning is one of the alternative instructional models. Blended learning is an educational approach that combines face-to-face (offline) and online learning instructions, and the effectiveness of blended system has been reported in many past studies (e.g., Alhazbi, 2016; Chiu, 2021; Tang et al., 2020; Yigit et al., 2014). Despite the positive outlook, younger students (e.g., primary, secondary and pre-university education) are more likely to face difficulties in adapting to the online learning instructions, and even well performed students may lose motivation when learning remotely (Di Pietro et al., 2020). Students who are isolated from their peers and instructors may

DOI: 10.4018/IJICTE.301276 *Corresponding Author face learning difficulties and psychological problems that demotivate them to learn independently (Di Pietro et al., 2020). Very few studies have investigated the design of technological environments that satisfy students' inner psychological needs (Chiu, 2021). Hence, more studies are required to understand how educational technologies can support learning motivation, resulting in better learning experience and positive learning outcomes (Ryan & Deci, 2020).

Blended learning is the best educational model to learn programming (Alhazbi, 2016). Unfortunately, students with no self-directed learning skills could find it hard and discouraging to learn programming independently, e.g., watching video lectures followed by in-class programming practices (Baldwin, 2015; Di Pietro et al., 2020). Responding to these findings, this study investigated motivation and difficulties of pre-university students in learning programming in a blended learning environment. This study addressed the following research questions:

1. How students rated their proficiency in programming?
2. What were the difficulties faced in learning programming?
3. How were students motivated to learn programming?

To answer these research questions, quantitative and qualitative data were collected to triangulate the findings of the study.

RELATED WORK

Blended Learning

Blended learning is an educational model that enriches student learning experience by integrating technology-based instructions (e.g., online learning and digital learning materials) and physical face-to-face classroom instructions (Di Pietro et al., 2020). Many studies have stated that blended learning is somewhat more effective than traditional classroom learning (Spanjers et al., 2015). In a blended learning environment, students are more positive and motivated to learn programming (Alhazbi, 2016; Yigit et al., 2014). Blended learning is claimed to be the best educational model to learn programming because it facilitates learning by giving students the flexibility to learn at their own time, pace, and location as well as supporting their social-emotional needs (Alhazbi, 2016). The sole use of blended learning model does not improve teaching quality. Instead, blended learning is often combined with a flipped classroom model to deliver lectures online (i.e., whether asynchronous, or synchronous) prior to the face-to-face classroom interactions (e.g., exercises and practices) (Akçayır & Akçayır, 2018; Giannakos et al., 2014; Lacher & Lewis, 2015). A flipped classroom is a pedagogical model that flips the learning practices. Typically, the in-class activities (e.g., lectures) and homework outside of the classroom are reversed (Akçayır & Akçayır, 2018; Lacher & Lewis, 2015). Many studies related to flipped learning have reported increase in learning motivation (Akçayır & Akçayır, 2018). In a flipped programming course, students can use in-class time working on programming exercises and at the same time, interacting with their peers and instructors. To learn programming effectively, students need the in-person guidance and help to overcome programming difficulties and doubts. A flipped programming course promotes computational thinking, deep learning, higher-order-thinking skills, and active learning (Giannakos et al., 2014; Lacher & Lewis, 2015). Conventional lecturing is no longer effective in programming education because many instructors tend to focus on syntactic structures rather than coaching students to improve problem-solving skills (Alhazbi, 2016). Ultimately, blended learning and flipped classroom have potentially become the dominant instructional models in this pandemic.

Programming Learning Difficulties

Novice learners encounter different types of programming difficulties. They may have basic programming knowledge, but they do not have problem-solving skills to write logical and correct program algorithms (Alhazbi, 2016). Students with poor problem-solving skills tend to believe that programming is difficult because they do not have any strategy (Bain & Barnes, 2014). Computer programming is not only about understanding the language and syntactic details, but also involves the use of other important skills such as problem analysis, algorithm design, coding and debugging (Alhazbi, 2016). Past studies have reported the difficulties of novice learners in program development, algorithm construction, modular design, syntax usage, and bugs detection (Lahtinen et al., 2005; Piteira & Costa, 2013). Novice learners also failed to comprehend some programming concepts such as repetition, recursion, parameters, pointers, data types, error handling and libraries (Lahtinen et al., 2005; Piteira & Costa, 2013; Settle et al., 2014). Some of these abstract constructs are cognitively hard to understand because they do not correspond to physical things in real life (Alhazbi, 2016; Lahtinen et al., 2005). In a blended programming course, students struggle to comprehend *for* loop and *while* loop (Baldwin, 2015). Another challenge is understanding the larger entities of a program instead of focusing on many minor ones (Bain & Barnes, 2014; Lahtinen et al., 2005). In computer programming, learners must utilize abstract design to understand a program at various degrees of abstraction (Alhazbi, 2016).

Learning Motivation

Motivation is the most important element in effective learning. Students who are intrinsically motivated will devote more time and effort to learn for their own sake rather than being driven by external rewards or threats. Intrinsic motivation is the inner drive that pushes someone to act freely (Deci & Ryan, 2000; Van den Broeck et al., 2010). Self-determination theory (SDT) suggests that humans are motivated to act by three innate psychological needs: *competence* (the desire to feel effective when doing something), *autonomy* (the desire to feel in control of their actions and experience) and *relatedness* (the desire to feel attached to other people) (Deci & Ryan, 2000; Van den Broeck et al., 2010). Instructional models that satisfy human innate psychological needs can prolong the passion of learning, and it is more fulfilling for both learners and instructors. According to Ryan and Deci (2020), SDT research should be designed around educational technologies. In a blended learning environment, student engagement can be fostered through digital and teacher support (Chiu, 2021). Chiu (2021) has proposed digital support strategies based on the three innate psychological needs: (1) *autonomy* – use various forms of electronic learning materials for the same subject matter, (2) *competence* – develop exercises and interactive resources with different level of difficulties and challenges, and (3) *relatedness* – use constructive interactional support (Chiu, 2021).

In a blended learning environment, programming theories and concepts can be presented in lecture slides, diagrammatic representations (e.g., flowcharts, animations) and video lectures to enable students to direct their own learning independently, which might result in better understanding and cognitive engagement. Students have the *autonomy* to choose their preferred modalities. The learning motivation is linked to the perceived relevance; a tool is useful if it can help them to achieve the programming learning objectives (Settle et al., 2014). Visualization tools (e.g., visual programming languages) also promote the sense of *autonomy* (Settle et al., 2014; Tsai, 2018). Students can visualize and explain the graphical examples to themselves, and this results in a sense of ownership over the learning process and encourage positive learning engagement (Settle et al., 2014). Some visual programming languages (e.g., Scratch) are helpful for beginners to understand the basic programming concepts, whereby they can manipulate the program structures graphically (Tsai, 2018). Learning would be more engaging and enjoyable if students have a sense of control over the learning processes (Csikszentmihalyi, 1975).

A *competency*-based programming course strikes a balance between support and challenges. Students are challenged, but they must feel *competent* and self-efficacy to overcome the challenges (Chiu, 2021; Csikszentmihalyi, 1975; Settle et al., 2014). Setting high learning expectations could be detrimental, especially for beginners with only basic programming knowledge and routine learning experience (Settle et al., 2014). According to Csikszentmihalyi's (1975) model of the flow state, one will get anxious and worried if challenges are overwhelming and beyond their capabilities; however, one will experience boredom if challenges are too easily achievable (Csikszentmihalyi, 1975, 1988). In both scenarios, engagement efforts have failed to enhance the sense of *competence*. Cognitive competency can be enhanced by the state of flow, i.e., when the challenge is increased gradually, the learning experience will be more enjoyable (Chiu, 2021; Csikszentmihalyi, 1988). Learning is more motivating and engaging if students can use their programming knowledge to solve challenging problems.

A *relatedness* supportive instructor offers students with academic, emotional, and motivational assistance, so that they feel connected (*relatedness*) and experience the sense of belonging in the learning community (Chiu, 2021). Although digital *relatedness* was proposed in Chiu (2021), the face-to-face component of a blended learning model could better satisfy students' need for *relatedness*. Studies conducted during the pandemic have suggested that students need the in-person classroom interactions (whether with peers or instructors) to maintain a good mental health and positive emotions (Aguilera-hermida, 2020; Di Pietro et al., 2020; Shim & Lee, 2020). No matter how much efforts have been done in a virtual learning environment, the digital *relatedness* can never replace the in-person classroom interactions. Social interactions (whether formal or informal) happened spontaneously in a physical classroom. Education is not only about academic; it also includes socializing, communicating, and emotional development.

METHODOLOGY

MATLAB Programming Course

A MATLAB programming course was taught over a period of 10 weeks (a weekly three hours in-class lecture/tutorial) at a pre-university program in Malaysia. The course was delivered through blended learning and flipped classroom models. Face-to-face classroom instructions were blended with digital learning materials such as lecture slides, animations, diagrams, video lectures, interactive practices, and self-guided tutorials. Students were encouraged to use the electronic resources to learn programming before attending the in-class lecture/tutorial. The self-directed learning (*autonomy*) served as a preparation for the upcoming lessons. Most of the students did not have prior programming experience so they might not have the ability to learn programming independently. Hence, short lectures, additional practices, and constructive discussions were conducted during the in-class sessions to support their need for *relatedness*. The topics covered were: using MATLAB as a calculator, array and matrix operations, array manipulation, built-in functions, graphs, functions, calculate the sum of series using a *for* loop, calculate the area under a curve using a *for* loop, write input/output program, *if* statements, nested *for* loop, *while* loop, and polynomial functions. Each programming lesson was designed with multiple levels of difficulty (*competency*).

Data Collection

The study employed the most well-known mixed methods research design, i.e., the *convergent parallel design* (Creswell & Plano Clark, 2011). The approach was used to triangulate the methods by comparing quantitative and qualitative findings for corroboration (Creswell & Plano Clark, 2011). Both quantitative and qualitative data were collected during the same phase of the study. Upon completion of the course, students were given 50 multiple-choice questions test to assess their programming knowledge and understanding. Then, survey and interview were administered to explore student learning experience, e.g., motivation and learning difficulties. The response rate was 91% (209 students) for both survey and programming test. Subsequently, a sub-sample of seven students were drawn for the interview on a voluntary basis: six students without prior programming experience and one student with Java programming knowledge. The survey was structured in two sections:

1. Programming proficiency for each lesson was measured using a 5-point Likert scale ranging from very easy (1) to very hard (5).
2. SDT motivation level was evaluated using a 5-point Likert scale ranging from strongly disagree (1) to strongly agree (5):
   a. I am motivated to solve challenging programming problems [*competence*].
   b. I am motivated to learn programming at my own pace and time [*autonomy*].
   c. I am motivated to learn programming together with my friends [*relatedness*].

The survey and interview were used to explore the same issues, but the conversational nature of the interview allowed more flexibility in the formulation and adaptation of queries than survey. During the interview, students' point of views, learning difficulties, motivations, experiences, and feelings were explored in depth.

Data Analysis

Drawing on the *convergent parallel design*, the quantitative and qualitative data were analyzed independently, and the findings were merged into an overall interpretation. Qualitative methods offered in-depth and rich information, but the results were not generalized beyond the sample of study (Creswell & Plano Clark, 2011; Johnson & Christensen, 2008). To complement the findings, quantitative methods were used to provide statistical generalizations of a population from the sample of study (Creswell & Plano Clark, 2011; Johnson & Christensen, 2008). The convergent design offered complementary strengths and non-overlapping weaknesses of quantitative and qualitative methods (Creswell & Plano Clark, 2011; Johnson & Christensen, 2008). During the final interpretation, quantitative and qualitative results were merged to provide a consolidated understanding of the study.

Three types of analyses were performed: (1) item analysis of the programming test, (2) statistical analysis of the survey questionnaire and (3) content analysis of the interview data:

1. Item analysis was performed using *SmartScan*. Two types of item analysis were performed as suggested by Oosterhof (1990): (i) the *item discrimination index* was calculated to measure how well an item was able to distinguish between examinees who were knowledgeable (mastered) and those who were not (non-mastered) and (ii) the *difficulty index* was calculated to measure the proportion of examinees who answered the item correctly.
2. Data collected from the survey was analyzed using *SPSS* software. Statistical analyses performed included descriptive statistics, correlation, and Mann-Whitney test. Descriptive statistics was used to summarize the percentage of the findings in a tabular form. Correlation analysis was used to measure the strength of the relationship between two variables. And finally, Mann-Whitney U test was used to compare two independent sample means (non-parametric).
3. Interview data was analyzed using the content analysis approach. *A priori codes* and *inductive codes* were used during segmenting and coding. The initial *a priori codes* were derived from the research questions such as learning difficulties and motivation. Then, *inductive codes* were drawn from the interview data such as difficulties in learning *for* loop, nested loop and *while* loop. Finally, the common themes were derived to provide a holistic and consolidated finding.

RESULTS AND DISCUSSION

Item Analysis

The programming test used in the study demonstrated strong validity (see Table 1). The discrimination index showed no question fall in the range of [-1.0, 0.0), and 68% of the questions were either good or very good at discriminating high scorers and low scorers. Although 32% of the questions were considered fair, they were acceptable. In the test, high performing students were more likely to answer the questions correctly and low performing students were more likely to answer the questions wrongly.

As shown in Table 2, the difficulty indices for 48% of the questions were easy [+0.8, +1.0], while 4% were difficult [0.0, +0.3], and the remaining 48% of the questions were within an optimal range (+0.3, +0.8). Moderate questions had great discriminative power while easy and difficult questions demonstrated poor discrimination index. About half (48%) of the questions were acceptable as far as difficulty and discriminative indices were concerned.

A pass index was calculated for each lesson assessed in the test (see Table 3). The larger the pass index, the more students had mastered the lesson. All lessons tested were considered moderate level of difficulty, except lesson 4 and lesson 11. A high past index of 0.8 or more indicated easy test questions for lesson 4 (built-in functions) and lesson 11 (*while* loops). To master the built-in functions in lesson 4 (e.g., *sqrt, factorial, exp, log10, isprime, acos, cosd, fliplr, etc.*), students only needed to understand the purpose and syntax of the functions. However, it was surprising to find that students could master *while* loops in lesson 11, which was considered one of the hardest programming structures. The students might be familiar with the *while* loop structures (e.g., calculate the sum of a series) that had been learned in the tutorial classes.

Programming Proficiency

The majority of the students (61% to 87%) reported proficient in lesson 1 to 4: using MATLAB as a calculator, array and matrix operation, array manipulation and built-in functions (see Table 4). There was no programming involved in the four lessons; students worked with commands, calculations, and built-in functions in the *command window*. In the subsequent lessons, students started to learn programming. The majority of the students (82% to 93%) reported lessons 5 to 7 and lesson 9 as very easy, easy, or moderate. In these lessons, students learned some programming structures such as plotting graphs, developing functions, using *for* loop to calculate the sum of a series, and using *if* statement to

Table 1. Validity of the test

Discrimination Index	Percentage of Question	Measure of Discrimination
[-1.0, 0.0)	0%	Bad
[0.0, +0.2)	32%	Fair
[+0.2, +0.6)	64%	Good
[+0.6, +1.0]	4%	Very Good

Table 2. Difficulty of the test

Difficulty Index (p-value)	Percentage of Question	Level of Difficulty
[0.0, +0.3]	4%	High (Difficult)
(+0.3, +0.8)	48%	Medium (Moderate)
[+0.8, +1.0]	48%	Low (Easy)

Table 3. Pass index of MATLAB lessons

Lesson	Pass Index
(1) Using MATLAB as a calculator	0.79
(2) Array and matrix operation	0.79
(3) Array manipulation	0.66
(4) Built-in functions	0.85
(5) Plot graphs	0.67
(6) Create functions	0.68
(7) Calculate sum of series using a *for* loop	0.64
(8) Calculate area under a curve using a *for* loop	0.60
(9) Write input/output program/ *if* statements	0.67
(10) Nested *for* loop	0.71
(11) *while* loop	0.86
(12) Polynomials	0.78

Table 4. Student programming proficiency

Lesson	Easy/ Very Easy	Moderate	Hard/ Very Hard
(1) Using MATLAB as a calculator	87%	11%	2%
(2) Array and matrix operation	83%	15%	2%
(3) Array manipulation	74%	23%	3%
(4) Built-in functions	61%	34%	5%
(5) Plot graphs	53%	40%	7%
(6) Create functions	41%	43%	16%
(7) Calculate sum of series using a *for* loop	39%	43%	18%
(8) Calculate area under a curve using a *for* loop	26%	40%	34%
(9) Write input/output program/ *if* statements	44%	38%	18%
(10) Nested *for* loop	25%	41%	34%
(11) *while* loop	34%	43%	24%
(12) Polynomials	38%	38%	24%

write a program that included input, process, and output. Ultimately, students reported lesson 8, and lesson 10 to 12 as hardest: calculate area under a curve using a *for* loop, nested *for* loop, *while* loop and polynomials. A significant proportion of the students (24% to 34%) rated the four lessons as hard or very hard. In these lessons, students must demonstrate strong logical thinking, problem solving and visualization skills. This result coincided with the findings of past studies that looping was hard to learn (e.g., Baldwin, 2015; Lahtinen et al., 2005; Piteira & Costa, 2013; Settle et al., 2014).

Programming Learning Difficulties

During the interview, students also expressed their difficulties in learning *for* loop, nested *for* loop, *while* loop, and polynomial functions. The data corresponded to the quantitative findings.

for Loop

In lesson 8, students were required to calculate the area under a curve using the right-hand Riemann approximation approach. According to Reimann Sum, an interval was split into several rectangles. To make the approximation better, more and more rectangles should be packed into the interval. A rough approximation of the area under the curve was calculated by adding up the areas of the rectangles, which was the sum of the height of the rectangles multiplied by the width respectively. Students must understand the principles behind Riemann Sum and transform it into the equivalent programming algorithm. The sample code is shown as follows:

%ALGORITHM: Sample Program of Reimann Sum

```
function[Area]=zDistribution(a,b)
    Area=0;
    for z=a:0.0001:b-0.0001
        Area=Area+0.0001*(1/sqrt(2*pi)*exp(-1/2*z^2));
    end
end
```

Students revealed their difficulty in visualizing the algorithm of Reimann Sum. They could not understand how the variables in a *for* loop accumulated the results:

This is the hardest to understand because for me to understand the question, I need to imagine. Like this question consists of many for loops and it's quite hard to imagine that.

for loop is in a range of something like you plus it up, and then just add up and add up. But sometimes when you put the condition that you thought it will be like that. The result you are expecting is not given out from the computer. The structure is weird.

Diagrams and animations were used to describe the principle of Riemann Sum. However, the difficulties faced by the students were beyond the underlying principle of Riemann Sum. The fact is, they could not relate and transform the principle into a programming structure.

Nested for Loop

In Lesson 10, students learned nested *for* loop in which there was an outer *for* loop and an inner *for* loop. For example, the students were required to write a function to change all the negative numbers in a matrix to zeros. The sample code is shown as follows:

%ALGORITHM: Sample Program of a Nested Loop

```
function[y]=matrixchange(A)
[row, column]=size(A);
for r=1:row
    for c=1:column
        if A(r,c)<0
            A(r,c)=0;
        end
    end
end
y=A;
end
```

A student said this lesson was hardest:

Because you need to know which value to assign and how to use the for loop. Must know how to jump, that is the problem…the increment.

The student seems to be confused how the inner and outer loops connected to each other. For instance, how the program flowed between the inner and outer loops, and what the sequence of increment between the counters was. The confusion might have arisen because of the complex dynamics between the counter *r* and counter *c*. In this case, the student might face difficulty in understanding the logic of nested loop structures. Although a video lecture was recorded to visualize how the elements were checked and updated one by one, it had no added value in explaining the abstract concept. Apparently, most of the students cleared their doubts and confusions during the in-class tutorials.

while Loop

In lesson 11, students learned how to calculate the sum of a series using a *while* loop. For example, they were required to develop a function to calculate the total of an alternating harmonic series:

$$\sum_{k=1}^{n}\frac{\left(-1\right)^{k+1}}{k}=1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\ldots+\frac{1}{n}=\ln\left(2\right)=0.6931471806$$

The sample program is shown as follows:

% ALGORITHM: Sample Program of Alternating Harmonic Series

```
function[total]=alternating_harmonic_series(n)
    total=0;
    k=1;
    while k<=n
        total=total+(-1)^(k+1)/k;
        k=k+1;
    end
end
```

A student said calculating the sum of a series using a *while l*oop is the hardest:

The while is the hardest for me because you have to make sure that the while loop is always true… If you put calculation outside the while loop and you put calculation inside the while loop would be totally different thing… You have to break it, break the loop or to continue the loop. That's the most challenging part.

The student seems to be confused with the basic concept of a *while* loop. For instance, how the loop would start or stop, and how the loop would process the calculations inside the loop. *while* loop was hard especially for those who could not understand the fundamental concept of iterations and how the controlled variables started or stopped the iterations.

Also, in lesson 11, students learnt how to find the maximum or minimum point using a *while* loop. For example, students were required to find the minimum point of a quadratic function

$y = x^2 - 6x + 3$. The basic principle behind the program algorithm was that, as the points (x1, y1) and (x2, y2) moved from the left to right, the gradient of the graph changed from negative to positive at the minimum. When the gradient changed from negative to positive, the minimum point was found at (x1, y1). Based on this principle, the sample solution was developed with a few assumptions (e.g., the initial value of x=0, the initial value of gradient d=-1, and the shape of the graph was known). The sample program is shown as follows:

% ALGORITHM: Sample Program of Sentinel-Controlled Loop

```
d=-1; x=0;
while d<0
        x1=x;
        y1=x1^2-6*x1+3;
        x2=x+0.01;
        y2=x2^2-6*x2+3;
        d=(y2-y1)/(x2-x1);
        x=x+0.01;
end
fprintf('Minimum point at (%f, %f)\n',x1,y1);
```

Students mentioned that this lesson was hard:

Because the method used to determine maximum and minimum is different from maths… Last time when we learn maths, we use differentiation to find the maximum and minimum point.

The concept of finding those things, the principle behind it. The while loop is fine. The application part is a bit hard.

The students found it hard to comprehend the strategy used to determine the maximum or minimum point. They seemed to understand the functionality of a *sentinel-controlled loop*. However, the principle used to solve the problem contradicted with what they had learned in mathematics lessons. There were perhaps two reasons for this confusion: (1) the students had to accommodate a new problem-solving strategy (assimilation), (2) the students had to convert the newly learned approach to programming codes. An animation was used to demonstrate how a maximum or minimum point was found by shifting the points (x1, y1) and (x2, y2) along the graph. Unfortunately, the new problem-solving strategy was hardly accepted by most students, and it stirred up more confusion when they tried to construct the program algorithm.

Polynomials

In lesson 12, students learnt polynomial functions. For example, they were required to find a quadratic curve that best fitted a set of data and draw the best quadratic equation approximation to the graph. The polynomial functions *polyfit()* and *polyval()* were used:

% ALGORITHM: Sample Program of Curve Fitting

```
x=[.5 1 1.5 2 2.5 3 3.5 4];
y=[2.2 2.1 2.3 2.9 4.1 6.2 8.3 10.7];
a=polyfit(x,y,2)
```

```
x2=0:.1:5;
y2=polyval(a,x2);
plot(x,y,'+',x2,y2)
```

According to a student, the hardest part of this program was understanding the functionality of *polyval()*:

The polyval is just very confusing like don't know what is to write inside the polyval function. Because the sequence is not logical, is a function.

When the student said "the sequence is not logical", he/she could not understand the syntax of *polyval(),* and how to enter the correct *input arguments* to produce the required output. A video lecture was recorded to explain the functionality of *polyfit()* and *polyval()*. Yet, most students would still prefer to ask the instructor to explain the functionality of *polyval()* during the in-class sessions. One of the limitations of video lectures was the lack of interactivity in providing feedback to students.

Debugging

During the interview, students also expressed their difficulties in debugging a program. Syntax errors could be detected and fixed easily because error messages were shown in the MATLAB IDE (Integrated Development Environment). However, students expressed their difficulties in detecting and fixing logical errors.

Syntax Errors

Easy is like the typo errors and the bracket errors, spacing errors.

The easiest to detect is like you didn't put the parentheses.

The easiest one is normally about syntax, like missing of comma, missing of colon or semi colon or bracket.

Logical Errors

Like if I use a for loop and it supposed to be like this, but is not giving the result that I want… Why it doesn't work?

The harder one you have constructed the if statement, and the returning result is not your wanted result.

Maybe you look at the thing, you think you are correct. That's how you understand the concept and is like nothing seems wrong.

There was no error message shown in the IDE to indicate logic errors. The programs usually ran without crashing (no syntax errors) but incorrect results were produced. In this case, the instructor played a significant role in facilitating the debugging process.

Self-Determination Theory of Motivation

Most of the students (60% to 73%) agreed or strongly agreed that they were motivated to learn programming (see Table 5).

Table 5. Motivation factors in learning programming

	Disagree/Strongly Disagree	Not Sure	Agree/ Strongly Agree
Challenge	15%	25%	60%
Own pace and time	7%	21%	72%
Peer learning	6%	22%	73%

Up to 60% of the students agreed that they were motivated to solve challenging problems (*competence*). They claimed that programming problems without any challenges were boring and not engaging:

I like challenging questions more because I can use my brain more. I find it more interesting. Easy questions are quite boring.

It helps you to think more… Complicated one because it stimulates you to think more.

I want to think outside of the box. I don't want to lock inside the box, with the same frame and same idea. It is boring.

I prefer challenging question because it normally consists of many easy questions. So, you get one question, you revise all the other questions already.

About 72% of the students claimed that they liked to have control over their learning pace and time (*autonomy*). The students were positive with the idea of self-studying and independent learning. Digital learning materials fitted well with their learning motivation:

I normally learn faster than other people. So I won't wait other people's pace… I will learn with my pace… Do you think if I learn in the time that I feel is not convenient or is not the time I should learn, do you think it will be better? No, I am not happy at all.

Yes is because if I feel that the pace is too fast or too slow, I can adjust it so that it is suitable for me.

Approximately 73% of the students liked to learn together with their friends (*relatedness*). They wanted to learn distinct problem-solving strategies and common mistakes from their peers. The students valued the benefits of peer learning:

Normally I learn among my friends… Like I got my way to solve it and he also got his special way to solve it. I can also learn from his way and get more information.

I need other people's ideas to think about a concept or something… I will think why they want to do this and why they want to do that, and I will see is good or not.

We won't be the only one doing the mistakes. They will also doing mistakes. So sometimes when they did the mistakes, we might look at it and we can see… learning from their mistakes.

They might already know where is the mistake and may be they did the same mistakes and they found out and they can help you with it.

The correlations of SDT of motivational factors and test performance are presented in Table 6. The motivation to solve challenging problems (*competence*) was weakly positive correlated with test performance (challenge: $r_s = .204$, $p = .003$). Students who liked to solve challenging problems were more likely to perform better in the test. There was no significant correlation between other motivational factors and test performance. The findings suggested that students who were motivated to learn by *competence* were more likely to perform better in the test. However, students who were motivated to learn by *autonomy* and *relatedness* might not perform better or worse in the test. All motivational factors were weakly positive correlated with each other. The findings suggested that innate psychological needs for *competence, autonomy,* and *relatedness* were linearly related.

Subsequently, a Mann-Whitney test was used to compare the mean differences between students with prior programming experience (20% of the students) and without prior programming experience (80% of the students) in the three motivational factors. Students with prior programming experience were significantly more motivated to learn at their *own pace and time* (U =2705, $p < .05$, r = .159), and solve *challenging* problems (U =2269, $p < .01$, r = .245). There was no significant difference between the groups in *peer learning*.

CONCLUSION

In this study, the pre-university students faced various programming difficulties in a blended learning environment. They reported lack of proficiency in *for* and *while* loops. The findings coincided with past studies (e.g., Baldwin, 2015; Lahtinen et al., 2005; Piteira & Costa, 2013; Settle et al., 2014). There are a few justifications to these findings. Firstly, students do not understand the looping structures. They could not visualize the flow and process of the loops. For instance, how the control variables are initialized, updated, tested, and terminated to start or end the loops. Secondly, students could not apply the looping structures to the problems given, in such a way that each problem entity corresponds to certain segments of a program algorithm. In other words, they could not transform their problem-solving strategies into logical program algorithms. Thirdly, the problem-solving strategies introduced in the programming class could be different from what they have learned in the mathematics class. Students must adapt to the new problem-solving strategies, and they may struggle to understand the logic and flow of the program algorithms. Finally, logical errors are hard to detect and fix, especially when dealing with multiple looping structures. For instance, the programs are compiled successfully without any syntax errors; unfortunately, the results are wrong, or the programs are terminated without executing any of the loops. Students must identify the appropriate input values to test the programs thoroughly.

To solve programming questions, students must grasp various facets of problem-solving skills. Online instructions alone may not fully support students needing help with their difficulties in certain programming concepts. Teaching instructors play an important role in providing appropriate academic

Table 6. Correlations of SDT of motivational factors and test performance

		Own pace and time	Challenge	Peer learning
Test Performance	Correlation Coefficient	0.065	.204**	-0.008
	Sig. (2-tailed)	*0.349*	*0.003*	*0.906*
Own pace and time	Correlation Coefficient		0.281**	0.179**
	Sig. (2-tailed)		*0.000*	*0.010*
Challenge	Correlation Coefficient			0.371**
	Sig. (2-tailed)			*0.000*

guidance and motivational help (Chiu, 2021). Digital learning materials can support understanding of certain programming concepts, but they are not fully interactive. Blended learning would offer a better option, in which physical face-to-face classroom strategies are used to complement the weaknesses of online instructions. Past studies also revealed that students need the in-person interactions to maintain a good psychological and emotional health (Aguilera-hermida, 2020; Di Pietro et al., 2020; Shim & Lee, 2020).

Most of the pre-university students were motivated to learn by *competence, autonomy*, and *relatedness*. In a blended learning environment, students with prior programming experience are better prepared with self-directed learning skills to learn at their own pace and time using the digital learning materials (*autonomy*). They are also equipped with some problem-solving skills that motivate them to undertake challenging problems (*competence*). *Competence* motivated students seek programming mastery and take extra time to explore various creative programming strategies and problem-solving skills. As expected, *competence* motivated students are more likely to perform better in a programming course. Programming is a complex knowledge that includes understanding of language, syntactic details, problem analysis, algorithm design, coding and debugging (Alhazbi, 2016; Lahtinen et al., 2005; Piteira & Costa, 2013). During in-class sessions, students can interact with their peers to learn alternative programming solutions and common mistakes. Although peer interaction may not enhance programming performance, it could satisfy students' innate psychological need for *relatedness*; indirectly influencing the other motivational factors.

There are a few limitations of this study. Firstly, the programming proficiency surveyed was confined to the 12 lessons learned. Future studies could expand the selection of attributes in programming difficulties, e.g., syntax, logical design, problem-solving, modular design, and debugging. Secondly, the *convergent parallel design* was adopted as the methodology of the study. Future studies could employ the *exploratory sequential design* in which the qualitative findings generalized from the interview are used to develop and inform the design of quantitative survey. Thirdly, there were very easy and very difficult questions found in the programming test with poor discrimination. The questions should be reviewed and reconstructed in the future.

The use of blended learning model does not guarantee the success of a programming course. A blended learning environment must be structured around the nature of the subject to satisfy students' innate psychological needs. *Autonomy*: Digital learning materials should be designed to encourage self-directed learning. The difficulty of the learning content should be increased gradually, and various types of multimedia should be used to present the programming concepts, e.g., videos, animations, texts, and sounds. For instance, animations can be used to visualize looping structures. *Competence*: Challenging programming problems related to mathematics, science and engineering applications may pose more interest to the students. When a programming problem is connected to the students, they are more engaged to solve the problem, e.g., find the probability by calculating the area under a z distribution. Learning should be fun and stimulating. Learning without a challenge is not fun. *Relatedness*: Student mental and emotional health should not be overlooked. Teaching instructors play a significant role in teaching as well as providing mental and emotional support. The instructor-student bonding should not be underestimated. Ultimately, education is a comprehensive package of formal and informal interactions in academic and non-academic settings.

ACKNOWLEDGMENT

We would like to thank the publisher and all research participants.

FUNDING AGENCY

This research received funding from the University Teaching and Learning Grant (2018). The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Aguilera-hermida, A. P. (2020). College students' use and acceptance of emergency online learning due to COVID-19. *International Journal of Educational Research Open*, *1*, 100011. doi:10.1016/j.ijedro.2020.100011 PMID:35059662

Akçayır, G., & Akçayır, M. (2018). The flipped classroom: A review of its advantages and challenges. *Computers & Education*, *126*, 334–345. doi:10.1016/j.compedu.2018.07.021

Alhazbi, S. (2016). Active blended learning to improve students' motivation in computer programming courses: A case study. In Advances in Engineering Education in the Middle East and North Africa (pp. 187–204). Springer. doi:10.1007/978-3-319-15323-0_8

Bain, G., & Barnes, I. (2014). Why is programming so hard to learn? In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. Uppsala, Sweden: ACM. doi:10.1145/2591708.2602675

Baldwin, D. (2015). Can we "flip" non-major programming courses yet? In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Kansas City, MO: ACM. doi:10.1145/2676723.2677271

Chiu, T. K. F. (2021). Digital support for student engagement in blended learning based on self-determination theory. *Computers in Human Behavior*, *124*, 106909. doi:10.1016/j.chb.2021.106909

Creswell, J. W., & Plano Clark, V. L. (2011). *Designing and conducting mixed methods research*. Sage Publications, Inc.

Csikszentmihalyi, M. (1975). Play and intrinsic rewards. *Journal of Humanistic Psychology*, *15*(3), 41–63. doi:10.1177/002216787501500306

Csikszentmihalyi, M. (1988). Motivation and creativity: Towards a synthesis of structural and energistic approaches to cognition. *New Ideas in Psychology*, *6*(2), 159–176. doi:10.1016/0732-118X(88)90001-3

Deci, E. L., & Ryan, R. M. (2000). The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological Inquiry*, *11*(4), 227–268. doi:10.1207/S15327965PLI1104_01

Di Pietro, G., Biagi, F., Dinis Mota Da Costa, P., Karpinski, Z., & Mazza, J. (2020). *The likely impact of COVID-19 on education: Reflections based on the existing literature and recent international datasets*. EUR 30275 EN. Publications Office of the European Union. 10.2760/126686

Giannakos, M. N., Krogstie, J., & Chrisochoides, N. (2014). Reviewing the flipped classroom research: Reflections for computer science education. In *Proceedings of the Computer Science Education Research Conference (CSERC '14)*. ACM. doi:10.1145/2691352.2691354

Johnson, B., & Christensen, L. (2008). *Educational research quantitative, qualitative and mixed approaches*. Sage Publications.

Lacher, L. L., & Lewis, M. C. (2015). The effectiveness of video quizzes in a flipped class. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Kansas City, MO: ACM. doi:10.1145/2676723.2677302

Lahtinen, K., & Jarvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*(3), 14-18. 10.1145/1151954.1067453

Oosterhof, A. (1990). *Classroom Applications of Educational Measurements*. Merrill Publishing Company.

Piteira, M., & Costa, C. (2013). Learning computer programming: Study of difficulties in learning programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication (ISDOC '13)*. Lisboa, Portugal: ACM. doi:10.1145/2503859.2503871

Ryan, R. M., & Deci, E. L. (2020). Intrinsic and extrinsic motivation from a self-determination theory perspective: Definitions, theory, practices, and future directions. *Contemporary Educational Psychology*, *61*, 101860. doi:10.1016/j.cedpsych.2020.101860

Settle, A., Vihavainen, A., & Sorva, J. (2014). Three views on motivation and programming. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. Uppsala, Sweden: ACM. doi:10.1145/2591708.2591709

Shim, T. E., & Lee, S. Y. (2020). College students' experience of emergency remote teaching due to COVID-19. *Children and Youth Services Review*, *119*, 105578. doi:10.1016/j.childyouth.2020.105578 PMID:33071405

Spanjers, I. A. E., Könings, K. D., Leppink, J., Verstegen, D. M. L., de Jong, N., Czabanowska, K., & van Merriënboer, J. J. G. (2015). The promised land of blended learning: Quizzes as a moderator. *Educational Research Review*, *15*, 59–74. doi:10.1016/j.edurev.2015.05.001

Tang, T., Abuhmaid, A. M., Olaimat, M., Oudat, D. M., Aldhaeebi, M., & Bamanger, E. (2020). Efficiency of flipped classroom with online-based teaching under COVID-19. *Interactive Learning Environments*, 1–12. doi:10.1080/10494820.2020.1817761

Tsai, C. Y. (2018). Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior*, *95*, 224–232. doi:10.1016/j.chb.2018.11.038

Van den Broeck, A., Vansteenkiste, M., De Witte, H., Soenens, B., & Lens, W. (2010). Capturing autonomy, competence, and relatedness at work: Construction and initial validation of the Work-related Basic Need Satisfaction scale. *Journal of Occupational and Organizational Psychology*, *83*(4), 981–1002. doi:10.1348/096317909X481382

Yigit, T., Koyun, A., Yuksel, A. S., & Cankaya, I. A. (2014). Evaluation of blended learning approach in computer engineering education. *Procedia: Social and Behavioral Sciences*, *141*, 807–812. doi:10.1016/j.sbspro.2014.05.140

Su-Ting Yong is an Associate Professor in the Department of Foundation in Engineering, University of Nottingham Malaysia. She joined the university in 2008, having taught in a few universities for a number of years. Dr Yong obtained her Bachelor's Degree in Science and Computer with Education (Mathematics) and Master's Degree in Information Technology from University of Technology Malaysia. She completed her PhD in Engineering Education at University of Nottingham. She is a Senior Fellow of the Higher Education Academy and has a wide variety of research interests, largely focused on technology in mathematics education, educational games, gamification, and programming. Her latest project is funded by the University Teaching and Learning Fund.

Tiong Kung Ming, currently an Assistant Professor, has more than 18 years of experience in higher education, and has taught at foundation and undergraduate levels. He holds a MSc (Mathematics) from UTM and a BScEd (Hons) (Mathematics and Physics Education) from UM. He is a trained data scientist and a Certified Analytics Professional (CAP®), a HRDF certified trainer, and a Professional Scrum Master I. His research interests are in mathematics education, higher education, games, and analytics. He has served as a reviewer and exam item writer for the CAP® certification and was part of a UNDP-University of Nottingham Malaysia-Bangladesh project as the curriculum developer for 4-year BSc programs in mathematics. He is the author of a higher education guidebook for post-secondary students, "Decide Wisely", and has published in various journals and proceedings as well as presented papers in various conferences and seminars.