Received April 30, 2021, accepted June 4, 2021, date of publication June 15, 2021, date of current version June 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3089560

A Systematic Mapping of Introductory Programming Languages for Novice Learners

PIUMI PERERA, GEETHYA TENNAKOON, SUPUNMALI AHANGAMA, (Associate Member, IEEE), RANGANA PANDITHARATHNA, AND BUDDHIKA CHATHURANGA

Faculty of Information Technology, University of Moratuwa, Moratuwa 10400, Sri Lanka

Corresponding author: Supunmali Ahangama (supunmali@uom.lk)

This work was supported in part by the Faculty of Information Technology, University of Moratuwa, and in part by the Senate Research Committee, University of Moratuwa.

ABSTRACT Delivery of core programming principles to novices is a challenging task and many introductory programming languages and platforms have been designed to support this process. Educational programming languages generally focus on alleviating the syntax overhead enforced on novice learners by designing languages with simple and concise keywords. Furthermore, only the most basic programming concepts and principles are incorporated and many languages follow unique methods to provide more simplified learning environments. However, considering the way programs are authored using these platforms, two common contrasting approaches to program representation are identified as text-based and block-based representations. Additionally, a hybrid approach of dual-modality interfaces, which combines the best of both techniques has gained traction as a current trend in the development of educational programming platforms. However, despite these extensive features, not all introductory programming languages can cater to the exact requirements of novice learners and a dearth of comprehensive studies and literature reviews have been conducted to investigate this context. This paper explores and presents a comprehensive review of how different elements of educational programming languages and platforms contribute towards learning by novices under the Technology Acceptance Model (TAM). The review is conducted under two main constructs of TAM as (1) Perceived Usefulness (PU) and (2) Perceived Ease of Use (PEOU) and external factors regarding the programming environment, language design, included programming concepts and supporting features such as the target audience group, language extensibility, and availability of learning materials are thoroughly investigated considering the typical behavioral patterns of novices concerning computer programming education.

INDEX TERMS Computer languages, computer science education, introductory programming, novice programmers.

I. INTRODUCTION

Computer programming is considered an important skill in today's society. However, it is widely accepted that learning programming in itself is a difficult task, especially for novices who come from all manner of backgrounds and experiences [1]–[6]. Beginners to programming often struggle with the most basic tasks including predicting outputs, identifying the correct order of commands, and writing simple programs that address real-world problems [6], [7]. Moreover, programming requires an adept knowledge of both the syntax and semantics of the chosen programming language which in most cases, does not relate well with normal spoken English [8], [9]. Furthermore, the use of the English language in coding and related learning materials itself can act as a barrier to novices, as non-native English speakers must learn both the programming language as well as English to properly understand basic programming principles [1], [2], [10]. Hence, even though there is a high interest in computer programming, these barriers often deter novice programmers from mustering up enough motivation to even begin learning.

Due to this situation, there are many educational programming platforms today which target novice learners of different age groups specifically. These platforms and

The associate editor coordinating the review of this manuscript and approving it for publication was Davide Patti.

languages have been designed to build interest and lower the barriers for novices to enter into programming by reducing the frustrations caused by excessive focus on syntax and semantics [11]. Many of these languages including Scratch, Alice, and Blockly make use of rich multi-media content, interactive environments, and predefined code blocks to eliminate syntax errors and engage users more actively [12]–[14]. Furthermore, the simplified syntax of these languages shows greater support for the translation of the programming languages into different natural languages [12], [13], [15]. This language extensibility has allowed computer programming knowledge to become more accessible in countries where native language education is more prominent as opposed to education delivered in English (English is a secondary language).

In comparison, the other approach of educational programming platforms is to introduce a simplified text-based programming language. These languages allow users to write the code themselves and whilst they do not eliminate the possibility of syntactical errors, they provide the user with a more realistic semblance as to what actual computer programming entails [7], [16], [17]. However, it must be acknowledged that novices may still find that this approach is too difficult initially, as they must be aware of the simplified syntax and semantics of the language. Furthermore, text-based languages are markedly more difficult to translate due to the more complex syntax and semantical constructs. Hence, there are fewer opportunities to localize these languages to cater to non-English speaking students.

Past literature has studied the various features exhibited by these educational programming platforms as well as the influence they have on supporting the learning of basic programming concepts by novice programmers. Comparisons between visual-based programming environments and text-based languages have identified that students are inclined to show greater interest and motivation towards learning to program when using visual and block-based programming languages [7], [8]. However, it has also been noted that students cannot grasp the concept of ‘real’ programming with visual environments and that they feel it is more discouraging when they begin to switch into text-based programming languages after working in such illusionary environments [5], [14], [15], [17], [18]. Furthermore, studies have determined that visual-based programming environments are responsible for cultivating certain undesirable habits among novice programmers that distort their understanding of basic programming concepts [19].

Many languages and platforms with numerous features exhibiting different language representations have been introduced to the existing field of programming education. However, the factors directly affecting the user’s attitude towards using a certain language, and thus the behavioral intention to use the said language changes over time. Therefore, the identification of factors that influences the user acceptance of such languages and platforms is complex and equally vital. Therefore, questions regarding the user acceptance of these languages and platforms are an ever-present problem domain [20], [21]. Hence, this study seeks to identify the existing literature related to introductory programming languages/platforms and systematically review the factors affecting the novice user’s acceptance of them.

Drawing from Technology Acceptance Model (TAM) proposed by Davis [21], the accumulated literature would be reviewed to understand the learner behavior in acceptance of educational programming platforms. TAM is one of the most influential and used models in the field of research of Information Technology (IT) [20]–[23]. Although many variations and modifications of the TAM exist, the original model introduced by Davis [21] is used in this review. For a better understanding of the motivations and requirements behind the development of educational programming platforms, this paper will provide an overview of the common distinguishing features exhibited by such platforms and languages and the effects they may have upon novice programmers.

Furthermore, both past literature and the authors’ own experiences with various introductory programming languages and platforms will be considered when conducting this systematic mapping. As programming languages and platforms are constantly updating, it is possible that past literature will not reflect the present situation clearly. Hence, the authors’ personal experiences will also be considered in identifying relevant and significantly permissible data to be included in the review. Additionally, this paper will also attempt to identify which language features should be ideally included in similar platforms and languages in development, to best support the learning of basic programming concepts by novice programmers.

II. THEORETICAL FRAMEWORK

A. THE TECHNOLOGY ACCEPTANCE MODEL (TAM)

TAM is one of the most prominent models among academics in the field of IT. Numerous literature reviews have been conducted by using TAM as the basis of methodology in several application domains. Therefore, a brief introduction into TAM’s origin and the modifications done to the model over time and several literature reviews following the TAM model are discussed in this section.

In 1986, Davis [21] presented the original TAM which included two variables ’Perceived Ease of Use’ (PEOU) and ‘Perceived Usefulness’ (PU). The term ’Perceived Usefulness’ can be defined as the extent to which a person believes that using a particular system would improve one’s job performance and ’Perceived Ease of Use’ can be defined as the extent to which a person believes that using the said system would be effortless. Davis hypothesized that both these constructs directly influenced a person’s attitude to use a certain system which in turn influenced the said person’s intention to finally use it [21], [23]–[25].

Even though TAM is proposed to investigate how users accept and use technology, some studies had extended its applicability to construct systematic mappings related to the introduction of technologies. For instance, Wirtz and

Göttel [20] used TAM as the basis of the methodology to do the literature review regarding social media acceptance. Similarly, Al-Aulamie [26] used TAM as the reference model to establish an enhanced TAM to investigate the learners' behavioral intentions in using Learning Management Systems (LMS). Furthermore, TAM was selected as the theoretical basis as it can be used to evaluate the features of a system based on the usefulness and ease of use, which are the most critical features of a system that would drive users to use it. In that background, this study too uses TAM as the overarching model to do a systematic mapping to understand the user preferences of educational programming platforms based on features related to PEOU and PU.

B. RELATED STUDIES

Considering past literature, multiple studies have been carried out to evaluate introductory programming languages, educational platforms, and associated coding tutorials available to novice learners. In any systematic mapping, identification and classification of the research material relevant for the corresponding topic of research hold significant importance. Hence, specifying inclusion and exclusion criteria for the paper selection process and declaring a set of classification criteria to classify the papers into subsections should be completed before the evaluation process. In the context of the introductory programming languages, researchers often classified the selected papers using criteria such as pedagogy [11], [27]–[30], curriculum, and the general-purpose programming language it caters to [11], [27], [28], [31], language representation [7], [8], method of assessment [28], [29] and the students [28].

Since inclusion and exclusion criteria are defined based on the context of the study, several frequently used conditions can be noticed in the introductory programming literature. Exclusive inclusion of the published papers (conferences or journal papers) written in English and programming tools/platforms catered specifically towards a certain level of education were frequently observed in the literature of introductory programming [28], [31], [32]. In the comprehensive review on introductory programming published by Luxton-Reilly *et al.* [28], publications during the years from 2003-2007 were included, while Salleh *et al.* [29] used publications from 2005-2011 inclusive. Additionally, partially prototyped platforms, studies dedicated exclusively to non-programming learners, and languages catered towards other branches of programming such as data structures, flowcharts, and database query languages were also observed to be common exclusion criteria [28], [31]. Although there are multiple systematic mappings conducted focusing on the introductory languages and platforms catering to tertiary education, a significant lack of literature concerning comprehensive reviews on primary and secondary education can be noticed [11], [28], [31], [32]. Hence, the key focus of the literature evaluated in the review presented in this paper is the primary and secondary education levels. Additionally, factors regarding the popularity of a language/platform such as the number of users, modes of use (online or desktop solutions), and open-source contributions are not considered in most literature. In the analysis done by Kim and Ko [33] regarding the pedagogical approach in different types of platforms that provide online coding tutorials, the platforms were chosen based on their popularity and website traffic generated. Similarly, the current review considers the popularity and research significance of the languages/platforms to ensure the inclusion of the introductory programming languages with a significant impact on the industry and the field of research.

The process of evaluation and interpretation of the nominated publications is considered as the core of a systematic mapping. Kim and Ko [33] used four core principles to evaluate thirty platforms providing programming tutorials, namely; (1) connecting to learners' prior knowledge, (2) organizing declarative knowledge, (3) practice and feedback, and (4) encouraging meta-cognitive learning. Although most of the tutorials provided sets of exercises that gave immediate feedback to the user, these methods lacked goal-oriented learning and maturity. Therefore, Kim and Ko [33] recommended that educators should use educational games and interactive tutorials that provide individualized support and context-sensitive feedback to aid the learners effectively. Considering literature regarding introductory programming languages and educational platforms, it can be noticed that factors such as user interface [12], [26], liveness, and tinkerbility [12], making the execution visible [12], [27], [31], [32], error messages and adaptive feedback [12], [31], [33] are consistently used by researchers to review and evaluate these systems.

In addition to the evaluation criteria, evaluation and interpretation of an academic paper can be done by defining a set of research questions in the study. Some of such formulated research questions that are frequently used in introductory programming literature are listed below [28], [29], [31].

1. What introductory programming aspect is the focus?
2. What developments have been reported in introductory programming during a given period?
3. Which programming language is taught?
4. What types of supplementary features are incorporated into the introductory programming language/platform?
5. What are the current issues related to the introductory programming language/platform?

Although many studies have been conducted to review introductory programming languages, only a few have considered including factors relating to the audience demographics such as gender, age, native language, and educational background of a user [33]. Therefore, Luxton-Reilly *et al.* [28] acknowledged that there is a dilemma of choosing the most suited programming language and paradigm to teach programming to learners and that it is still a widely debated and open topic of research. Hence, how the audience demographics such as age and native language influences the novice learners' learning experience with the introductory programming language is also discussed in this review. Furthermore, in the systematic review of Intelligent tutoring systems presented by Crow *et al.* [31], the primary focus is on the evaluation of the system as a whole and thus the effectiveness of individual features was not studied. However, the focus of the review presented in this paper is on the evaluation of the individual features of an introductory programming language/platform and determining how they affect the attitude of a novice learner.

III. METHODOLOGY

A. ARTICLE SEARCH METHOD

This review concerning the applications of introductory programming language for novice learners is multidisciplinary research that encompasses both the fields of education and computer science. Therefore, selecting search phrases that would encompass a broad and inclusive range of introductory programming literature was found to be challenging. Phrases that are too general resulted in a higher percentage of papers that are irrelevant and out of scope. However, phrases that are too specific tended to miss relevant and important papers. Therefore, after a recursive trial and error process using several databases, a combined search phrase that yielded the highest percentage of relevant literature was selected.

> ''Introductory programming language'' OR ''novice programming'' OR ''Introduction to programming'' OR ''novice learners'' OR ''localized programming languages'' OR ''coding for beginners'' OR ''learn programming''

To verify the pertinence of the selected search string, a trial set of papers was tested against the search phrase, and the outcome was then compared with a manual screening process by the authors. 94 papers retrieved from the proceedings of ITiCSE 2019 and ICER 2019 were included in this verification process. As the first step, 4 members individually classified the papers into 3 categories; namely, relevant, irrelevant, and undecided which depicted the level of relevancy of the paper for the systematic mapping. Thereafter, two groups consisting of two members were formed and the differences of their categorizations were discussed and resolved. During this step, papers that were classified into the 'undecided' group were also re-classified as relevant or irrelevant. The authors used the title, abstract, and keyword fields of the papers for the mentioned manual screening process.

The manual screening process included a fixed number of raters (4 members) classifying multiple items (94 papers) into a fixed number of categories (3 and 2 categories for the individual and group screenings respectively). Therefore, to measure the inter-rater reliability of the individual and the paired screening process, Fleiss-Davies kappa [34] was used, which measures the level of agreement between a set of raters. The individual classification process resulted in a Fleiss-Davies kappa of 58%, while the paired classification process gave a 71% of Fleiss-Davies kappa. The increase in the Fleiss-Davies kappa from the individual screening to the paired screening can be attributed to the increased reliability of the raters and the reclassification of the 'undecided' category. According to Simon *et al.* [35], kappa values are interpreted and categorized into three value ranges which depict the degree of agreement of the raters. Kappa values greater than 0.75 represent excellent agreement beyond chance while kappa values below 0.40 depict weak agreement beyond chance. Similarly, kappa values between 0.40 and 0.75 represent a fair to a good level of agreement. Therefore, it can be noticed that the paired screening agreement was substantially greater than that of the individual screening. Hence, the results of the paired screening were considered for the verification process which resulted in a selection of 32 papers.

After the manual screening process, the search phrase was then automatically applied to the title, abstract, and keyword fields of the 94 papers, which resulted in a selection of 38 papers. Out of the 38 papers selected, 29 were included in the set of papers selected via the manual screening process, resulting in a false positive of 9 papers. Since all the papers selected by the automatic search phrase will be examined by one of the members during the application of inclusive and exclusive criteria, the papers that would be irrelevant for the systematic mapping can be eliminated. Hence, the percentage of false positives was dismissed as a minor concern. However, the impact of the false negatives on the integrity of the review is substantial, since they represent the papers that would be relevant for the review but were not chosen during the selection process via the search phrases. However, only three papers were given out as false negatives for the above-mentioned search phrase. Therefore, the best combination of search terms was selected which gave the least percentage of false negatives (10%).

The search terms were then applied to the title, abstract, and the keyword fields of the ACM Digital Library, IEEE Explorer, SpringerLink, and ScienceDirect. A total of 1534 papers were acquired during the article search and the number of papers selected in each database is available in the following Table 1. To ensure that all articles relevant to the research, including those that are not available in the above-mentioned databases, are considered, a thorough search has been conducted in google scholar as well.

TABLE 1. Databases and the selected paper set.

Source	Number of Papers	Distribution (as a % of the total number selected)
ACM Digital Library	804	52.41
IEEEXplore	321	20.93
SpringerLink	241	15.71
ScienceDirect	168	10.95

B. INCLUSION AND EXCLUSION CRITERIA

All the papers that have been considered for this review are in between the years 2010 and 2019. The starting year is chosen as 2010 considering the availability of the papers.

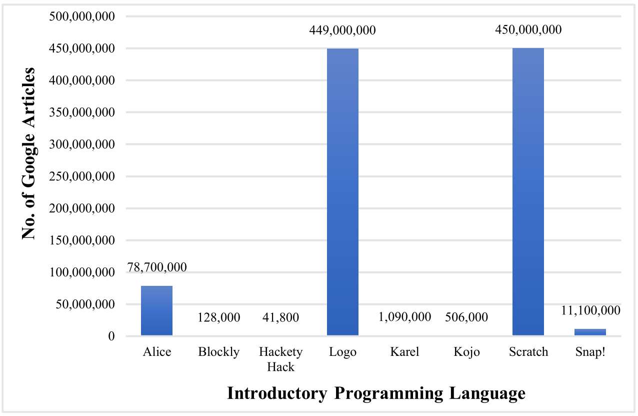


FIGURE 1. The popularity of introductory programming languages based on the number of Google articles.

To expand the article search, literature reviews in the relevant domain were also considered. As for the conferences that are considered when searching for articles, Innovation and Technology in Computer Science Education (ITiCSE) and the ACM Technical Symposium on Computer Science Education (SIGCSE) and the ACM Conference on International Computing Education Research (ICER) was the conference proceedings where the most number of research papers were published related to the domain of the study. The research papers and articles collected from each of the digital libraries and the searches in google scholar were filtered based on four main inclusion criteria,

1. Year of publication.
2. The popularity of the programming language or educational programming platform.
3. The relevance of the programming language to teaching novice learners.
4. Target audience group of the programming language

The popularity of the chosen programming languages and platforms were determined by taking into account the number of google articles and research articles available for each programming language at the time of writing. The numeric results of these metrics are illustrated in Fig. 1 and 2.

However, when considering educational programming platforms, the entire scope of all educational programming platforms would be too large to be addressed in this paper alone. Hence, for this review, mainly two categories of educational programming platforms have been selected depending on the program language representation. These are namely, block-based and text-based educational programming languages. While there are many different types of educational programming languages and platforms, the two categories noted above are the most predominant when used in delivering key programming concepts to novice learners [15], [36], [37]. A comparison between the various platform and language features exhibited by both block-based and text-based introductory programming languages, including the programming environment, programming language, and other supporting features such as the target audience

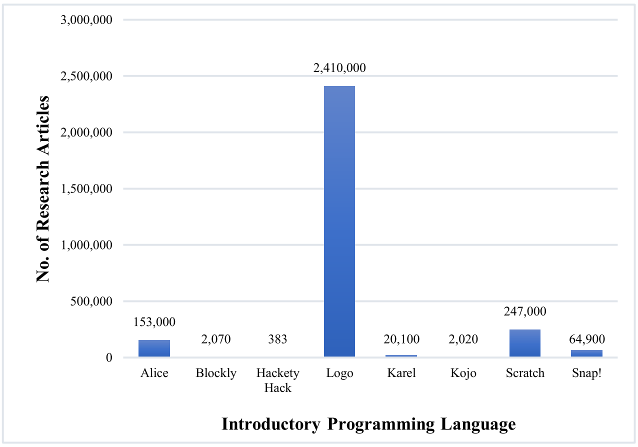


FIGURE 2. The popularity of introductory programming languages based on the number of research articles.

TABLE 2. Comparison of platform and language features.

Available Features	Language representation	
	Visual/ Block-based	Text-based
Programming Environment	Drag and drop interfaces	No specific interfaces.
	Statement execution visibility available.	Execution visibility is generally unavailable.
	Some languages also have a text-based component to write code.	Some languages targeting younger audiences have a visual component.
Programming Language	Basic programming concepts, constructs are included.	Basic programming concepts, constructs are included.
	Visual grammar is represented by block properties like shape and color.	The grammar is more complex and syntax is based on symbols and keywords.
	Short and concise keywords to enable translatability.	Short and concise keywords to avoid syntactic overhead.
	Eliminate syntax and type errors.	Syntax and compile errors are displayed.
Target Audience	Children 8 years and above.	Children 14 years and above.
Language Extensibility	The most extensive translations are for these types of languages.	Almost no extensibility for translating into other languages.

and language extensibility is described in Table 2 below [2], [13], [16]–[18], [36], [37]. Furthermore, due to the high availability of these educational programming platforms, especially in the block-based category, it was decided to exclude any block-based programming language which did not have its own stable programming environment from this review.

However, when considering educational programming platforms, the entire scope of all educational programming platforms would be too large to be addressed in this paper alone. Hence, for this review, mainly two categories of educational programming platforms have been selected depending on the program language representation. These are namely, block-based and text-based educational programming languages. While there are many different types of educational programming languages and platforms, the two categories noted above are the most predominant when used in delivering key programming concepts to novice learners [15], [36], [37]. A comparison between the various platform and language features exhibited by both block-based and text-based introductory programming languages, including the programming environment, programming language, and other supporting features such as the target audience and language extensibility is described in Table 2 below [2], [13], [16]–[18], [36], [37]. Furthermore, due to the high availability of these educational programming platforms, especially in the block-based category, it was decided to exclude any block-based programming language which did not have its own stable programming environment from this review.

Hence, the languages which adhered to the above-mentioned criteria and displayed a relevant number of Google and research articles were analyzed and it was decided to select Scratch and Alice to represent block-based programming languages and Logo, Karel, and Kojo to represent text-based programming languages. Hence, the research articles collected were further filtered to give greater inclusion priority to those discussing these languages. The relevance of the stated programming languages to teaching novice learners including factors such as the programming platform and availability of learning materials and resources was also considered in further filtrations.

Additionally, it was noticed that many educational programming languages and platforms targeted users of specific age groups, though they are accessible and used by others as well. Following such observations, target audience groups could be broadly categorized into two groups as those from primary to secondary education level and those of college education level or higher. Highly visual programming languages such as Scratch, Alice, Snap, and Blockly fall into the former category where their audience often ranged from those between 8 – 16 years of age [12]–[14], [38], [39]. In contrast, text-based learning environments such as Kojo, Hackety Hack, Oz, and Curry were seen to target older audiences while typically covering more advanced concepts as well as basics [40], [41]. For this review, the main focus was given to languages targeting audiences of the primary education level as most novices are introduced to programming at this time as per many school curriculums [42]–[44]. Hence, languages including Scratch and Alice which fall into this category were given priority among the block-based languages, and Logo, Karel, and Kojo were given priority among the text-based languages as they best fit the criteria considered.

C. RESEARCH METHODOLOGY

Following the initial selection of relevant papers and articles based on the filtered query string, the research was then carried out under the following steps on the 1534 papers thus acquired.

1) CATEGORIZATION OF STUDIES

In this stage of the systematic mapping, the focus was given towards identifying the studies that would be most relevant to forming the basis of the current review. As such, studies relating to different introductory level programming languages were divided based on three broad categories as (1) The programming environment, (2) The Programming Language, and (3) Supporting Features for programming ecosystems which were decided upon after considering both past literature and the authors' opinions following several brainstorming sessions. Therefore, for this classification, the entire result set from the initial search query was divided among the authors for an initial screening where the titles, abstracts, and keywords of each study were referred to, and used to deliver the correct categorization. Papers that were found to refer to or compare between multiple categories were added to all relevant groups accordingly while studies that could not be classified were grouped separately to be considered in more detail during the latter stages. Additionally, compliance of the studies to criteria concerning the year of publication, relevance, and popularity of the subject programming language was also evaluated during this initial screening, and those papers showing significant deviation were eliminated from the outset. Additionally, it was identified that a majority of these eliminations were attributed to papers gathered from ScienceDirect and Springerlink databases, as the more generalized nature of content allowed studies from different fields to also be captured by the search string (For example in ScienceDirect a majority of papers captured using the keywords "novice learners" referred to studies supporting novices in multiple fields such as medicine and healthcare rather than programming). Hence, the overall contribution to the review by the papers from generalized databases was observed to be significantly less than those of the more specific IEEE Xplore and ACM digital libraries.

2) ELIMINATIONS BASED ON INCLUSION CRITERIA

Based on this categorization, the two groups on the programming environment and language were divided among two groups of authors with two members each for a second screening. Hence, each paper was further evaluated in depth by considering the remaining inclusion criteria and the relevance of the content to the subject of this review. For this purpose, separate summaries illustrating the various research areas and gaps indicated by each paper were considered to support the filtration process. The remaining two categories cited as 'Supporting features' and 'Unspecified' were given special consideration by all the authors partly due to the low number of papers available as well as the ambiguity of some of the paper content in terms of relevance to the review conducted. Subsequently, several field experts including two lecturers for undergraduate level programming students and two teachers handling the same subject at the school level were also consulted regarding the validity of each classification. Hence, this methodology was used to limit the scope of papers to

TABLE 3. Categorization of selected papers by group.

Category	Number of Papers	Distribution (as a % of the total number selected)
Programming Environment related	357	75.48
Programming Language related	323	68.28
Supporting Features related	198	41.86
Unspecified	49	10.35

be considered and several more papers were eliminated for failing to meet the set inclusion criteria.

As stated previously, particular priority was set to papers subjecting the languages Alice, Logo, Karel, Kojo, and Scratch as they were selected based on their popularity among users. However, the main papers subjecting other introductory programming languages such as Blockly, Hackety Hack, and Snap! were also considered to better capture common features of introductory programming languages and platforms. Hence, a final total of 473 studies passed the second screening and were selected to act as the basis of this systematic mapping paper. The distribution of these papers among the previously specified categories is indicated in Table 3 below.

3) EXTRACTING DATA AND SYNTHESIZING RESULTS

Consequently, the data thus gathered during the previous stages was synthesized to present the findings regarding introductory programming languages and educational programming platforms as per PU and PEOU in TAM [20]–[22], [26] as described in Fig. 3. Hence, the categorically classified papers were now reviewed in full detail while referencing the dedicated summaries created during the previous screening. These summaries were further updated at this time to be used in all future referrals required during this review. Additionally, several more papers were eliminated from the review process after considering the full content and also due to the unavailability of access to the full paper.

Furthermore, the inclusion of several sub-features with regards to the different platform components including the programming environment, the programming language, and other supporting features of the platforms were decided as the main topics to be explored during this review. The selection of these sub-features was carried out by referring to past literature to identify the components likely to have the greatest impact on novices' acceptance towards an introductory programming language [12] and authors were able to indicate which sub-categories were discussed under each paper in their assigned group.

The sub-features selected were classified as PU and PEOU according to the intended goal of identifying the language features having the greatest impact on novice learners. Hence, when considering PU and PEOU in this context, it was identified that 'usefulness' referred to the ability of the introductory programming platform to deliver key concepts and principles

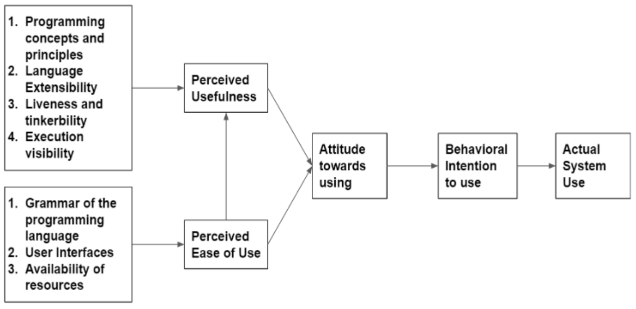

1. Programming concepts and principles
2. Language Extensibility
3. Liveness and tinkerbility
4. Execution visibility
1. Grammar of the programming language
2. User Interfaces
3. Availability of resources
Perceived Usefulness
Perceived Ease of Use
Attitude towards using
Behavioral Intention to use
Actual System Use

FIGURE 3. Technology acceptance model (TAM).

TABLE 4. Categorization of language features.

Main Language Features	Included sub-features	Categorization under TAM	Number of papers
Programming Environment	User Interfaces	Perceived Ease of Use	362
	Execution Visibility	Perceived Usefulness	314
	Liveness and Tinkerbility	Perceived Usefulness	215
Programming Language	Programming concepts and principles	Perceived Usefulness	306
	Grammar and Syntax	Perceived Ease of Use	298
Supporting Features	Language extensibility	Perceived Usefulness	47
	Availability of resources	Perceived Ease of Use	165

effectively to novices [21]. Similarly, it was noted that 'ease of use' signified the effort required by novices to understand and use the programming platform at the initial stages [21]. Accordingly, the sub-features, programming principles and concepts, language extensibility, liveness and tinkerbility, and execution visibility were included under PU while the grammar of the programming language, user interfaces, and the availability of resources was allocated under PEOU as stated in Table 4.

The following observations were considered to support the basis of this categorization [2], [7], [13], [16], [18], [45].

1. The inclusion of basic theories plays a direct role in delivering required concepts to novice learners. Hence, the programming principles and concepts show a greater inclination towards PU.
2. Language extensibility allows novices to bypass the language barrier and focus solely on grasping key concepts and thus tends towards PU.
3. 'Liveness' and 'Tinkerbility' allow users to learn by constantly changing the computer programs and observing their results. As this feature supports greater flexibility in delivering concepts to novices, it is more relevant to PU.

4. Visualization of the program flow execution provides a more enhanced learning experience to novices and thus tends towards PU.
5. A novice programmer should prioritize the learning of basic principles and concepts over memorizing syntax and grammar rules of a programming language. As the keywords, syntax patterns and representation depicts the ease of understanding the language, they are considered under PEOU.
6. User interfaces of an introductory programming platform should aid the learning process and not cause added complexity. Hence, it is more relevant to PEOU.
7. Availability of tutorials and user guides for programming platforms supports users in familiarizing themselves with the system for more efficient learning and thus tends towards PEOU.

The identification and classification of these sub-features were carried out predominantly by reference to past literature and the authors' own experiences with the subject programming languages. While the authors' experience was sufficient to make the necessary allocations, it was identified that this same experience may cause unwarranted biases in the observations as the paper focuses on the effect of such features on novice programmers. Hence, greater significance was given to the reasoning cited in past literature to support the classification process. However, it was further noted that complete reliance on past literature alone could not be accepted as there was a high possibility of systems changing and updating between the time of publication and the time of review. Hence, as reliance on such obsolete data would severely undermine the quality of the review results, initial findings obtained through referring to past literature were screened a third time by the authors to remove and update irrelevant data. Thus, a few papers whose content was identified to be completely obsolete were further eliminated as there was no useful information that could be used to support the review conducted.

Subsequently, based on the identification of these sub-features and their classification under the three main language features considered in this review, it was identified that certain papers needed to be reclassified based on the more detailed inspection of sub-feature content discussed. Furthermore, few more studies which had previously passed the screenings were eliminated due to a lack of appropriate content following this more detailed analysis. Subsequently, upon the conclusion of all elimination and categorization processes, a total of 442 papers were considered for this review with some falling under multiple categories. distribution based on the sub-features selected is also presented in Table 4 below.

IV. RESULTS

Considering the categorization of language features observed in Section 4, the following section describes the significance of PU and PEOU towards behavioral intention in different introductory educational programming languages as per past literature [7], [12], [13], [17], [18], [46].

TABLE 5. Languages and their programming paradigms.

Language	Related Paradigms
Scratch	Object-based (Not Object-oriented), Event-driven programming
Alice	Object-oriented, Imperative, Functional, Distributed, and Concurrent programming
Logo	Functional, Procedural programming
Kojo	Modular programming, Object-oriented programming, Functional programming, Concurrent programming
Karel/Karel++	Procedural and Object-oriented programming

A. PERCEIVED USEFULNESS (PU)

Perceived usefulness is one of the two key factors in TAM which directly influences attitude and the behavioral intention towards the system use. Programming principles incorporated in the language, Language extensibility, Execution visibility, and Liveness and tinkerbility are considered as the key factors in determining the significance of 'Perceived Usefulness'. Considering the past literature, the selected introductory programming languages are reviewed under the above-mentioned factors to determine the behavioral intention.

1) PROGRAMMING CONCEPTS AND PRINCIPLES

As stated in Table 5, most of the educational programming languages are catered towards one or more programming paradigms. Although the underlying paradigms and teaching objectives are different, almost all of the educational languages have incorporated the most basic programming concepts such as those of variables, data types (e.g.: Boolean, Number, String), control structures, decision-making constructs, concepts of recursion, functions/procedures and operators. The main aim of introductory programming languages and educational platforms is to enable novice learners to learn and practice programming concepts. Due to the inclusion of the most basic programming concepts and principles, it can be deduced that these platforms have fulfilled their perceived usefulness of delivering core programming concepts to novice learners. Thus, this creates a positive attitude among the novices and tutors towards using such languages/platforms since they deliver the intended use.

However, due to the design nature of the languages and features of its inherent type systems, some of these programming concepts are not properly understood by the students [15]–[17]. In some languages including Scratch, Logo, Kojo, variables are implicitly typed [12], [16], [36]. Additionally, the drag and drop system in block-based languages would not allow function blocks with incompatible return types to be inserted into parameter slots. Furthermore, automatic data type conversions between number and string types are common in most visual-based languages. Although these types of syntactic sugar have been incorporated to unburden novices of type errors, they might not be able to grasp the concepts of data types and data typing constraints successfully [47]. Therefore, some of these languages occasionally fail to convey their main intended use of delivering the most basic programming concepts and principles successfully. Although the use of syntactic sugar may create a short-lived positive attitude to use among novice programmers, the behavioral intention would be impacted negatively in the long run. However, visualizing certain programming concepts and their structures have proven to be more effective for novices [13], [48]. In Scratch and Alice, variables are visualized as concrete objects by using variable monitors and this enables the novices to see and manipulate them through tinkering. Additionally, animations are used to display the effects of list operations. These visualizing concepts will assist the novices to generate mental maps on how variables, arrays, and lists work instead of regarding them as abstract concepts [12], [15], [49]. Therefore, it can be noticed that the usage of visualization to depict and convey the programming concepts/principles in the forms of different shapes, colors, objects, and animations can create greater ease of use for the novices and hence deliver the programming concepts more successfully.

In visual-based languages, shapes of blocks and embedded slots are designed to characterize different programming constructs [15]. Therefore, to add a new data type, a new parameter slot shape and function block shape must be introduced. Hence grammar extensions to add more data types or functionalities in visual languages are complex compared to text-based languages. Additionally, more categories and shapes on the block palette may lead to visual clutter on the platform and thus confusion to novices [12]. Although an extensive block palette and advanced user modifications/features increase the overall perceived usefulness of the system, they may decrease the ease of use and thus create a negative attitude towards using these languages/platforms.

Students have often shown enthusiasm in learning programming concepts using visual-based languages and platforms [17], [47]. While using only text-based languages, novices have shown signs of inability to apply the programming concepts and ideas used in coding to other domains that are outside of the programming context [16]. For example, the novices may think the variables learned in programming are different from the variables used in algebra. Since the context in which they have been used is different, they find it difficult to realize the concept and purpose of the use is the same and the lack of a visualization aspect in text-only languages to map the concepts into real-world objects may be the reason for this [13]. Hence, to address this issue, languages such as Alice and Logo tend to incorporate both text-based and visual-based aspects into their language designs to give the 'best of both worlds'. Hence, the inclusion of both the text-based and visual-based components increases both the perceived usefulness and the perceived ease of use of these languages, thus creating a positive attitude and behavioral intention towards them. However, for languages that are only-text-based, simply the exposure to programming for a significant period will not be adequate to deliver the basic programming concepts to novices successfully [13], [16]. Hence, a meticulously structured set of exercises that would gradually introduce the use of syntax and complexity of programming concepts, would help novice learners to grasp the principles more effectively. The existence of a set of exercises to guide novice learners will increase the perceived ease of use of the text-based languages, thus influencing the perceived usefulness. Furthermore, since these exercise guides will enable novice learners to learn the concepts without the supervision of a tutor, it will increase the usefulness of the text-based languages and create a positive attitude and behavioral intention towards using them. '*Introduction to programming*' documentation [36], [50] prepared by the developers of Kojo is a successful attempt in using a structured set of exercises that gradually introduces the language and its concepts to novices.

Additionally, the tasks can be presented as a set of scenario-based exercises that will follow a common storyline [15], [49]. This will encourage the novices to follow the programming tasks and learn basic concepts and principles more effectively. Therefore, using scenario-based exercises combined with a storyline will increase the perceived usefulness of the introductory language. For example, Scratch and Alice both have multiple environments and actors with some recurring themes from popular movies and stories. Many visual-based programming languages and platforms focus on maintaining their respective micro-worlds and actors that function within that world. Hence, presenting the programming concepts according to a storyline and using the concepts of micro-worlds and actors may create a positive attitude among the novice learners towards using these introductory programming languages. However, they should also focus on delivering the core programming concepts at the same time. Encouraging the activities such as world and character building as per different storylines is more in line with computer game designing than programming which is the focus of this paper. Thus, it can be considered that influencing the novices to create different environments for their projects may indirectly distract them from their learning goals [15]. Therefore, although the usage of micro-worlds and actors may create a positive attitude towards using, overusing these concepts may result in deterring these languages from fulfilling their intended overall use. This may impact the behavioral intention to use among both novice learners and programming tutors negatively.

2) LANGUAGE EXTENSIBILITY

For the context of this paper, language extensibility is considered as the availability of localizations of a given programming language, thus providing users with opportunities to code in their native language. However, it was observed that this is a very limited feature with only a few large-scale visual-based languages such as Scratch and Alice maintaining complete translations to different natural languages [2], [12], [13]. In text-based languages, this feature is even less accessible, and there are no successful completed translations available [2], [10]. Unfortunately, localization is an important feature that would be extremely useful for novice students since it would eliminate the language barrier in understanding and applying programming concepts. Past studies have shown that younger audiences, especially those completely new to programming, respond better when being taught concepts in their native language [2], [3]. However, other studies have proven that this observation does not apply to older students or those who already have some basic understanding of computer programming. Nevertheless, the overall outcome remains that localization is a valuable addition to educational programming platforms [1]. Furthermore, language extensibility would also make computer programming more accessible to students who have completed their education thus far in their mother tongue. However, the high cost involved in facilitating such translations, as well as the technical difficulties that arise in attempting to translate high-level languages with already well-established ecosystems has halted these efforts for many platforms, thus depriving students of a critical opportunity [1], [2].

Furthermore, language extensibility adds immense worth to the existing programming languages since the availability of introductory programming languages/platforms catering specifically towards non-English languages is scarce. Considering these features of language extensibility, it can be noticed that the availability of one's native language to introduce the programming concepts to novices contributes towards increasing the perceived usefulness of the introductory language/platform. Hence, this creates an approving attitude towards using these languages which consequently influences the behavioral intention of usage positively. The immense popularity of the programming languages such as Scratch and Alice can be attested to this.

3) LIVENESS AND TINKERBILITY

The term 'Liveness' refers to whether a distinction exists between the editing mode and running mode of a computer program, i.e., if a compilation stage exists. In contrast, 'Tinkerbility' refers to the ability of the users to experiment with commands and code snippets to gain hands-on experience [12]. Block-based languages such as Scratch and Blockly have high tinkerbility as users can arrange blocks in numerous ways and still observe the associated visual output. Scratch also demonstrates high liveness as the program can be edited while the program is being executed and running. However, languages like Alice, Karel, and even Logo require code to be syntactically correct and compiled before execution, which leads to limited tinkerbility as the knowledge of the syntax and semantics of the language is required to get a correct output [17]. To teach programming, both liveness and tinkerbility are key aspects as they allow users to gain knowledge of writing programs. It is further identified that students indicate greater interest and activeness when they can experiment with code without being concerned about syntax and semantics [17].

Therefore, features such as Liveness and Tinkerbility make it easier for novice learners to adapt to the platform and hence create a positive attitude towards learning programming. Since 'Tinkerbility' allows the novices to focus on learning by making constant changes to the program and see how those changes affect the outcome, the perceived usefulness of the introductory programming language is also increased. These in turn will act as catalysts in increasing the behavioral intention of the novices to use that particular introductory programming language/platform. However, it is also noted that this carefree environment is ill-equipped to shift beginners to more advanced languages like Python, C, and Java where basic knowledge regarding syntax and semantics is required to try out even basic simple concepts.

4) EXECUTION VISIBILITY

Most of the learning platforms provide visual feedback for the programs in execution by presenting the output of each line of code. Languages such as Scratch, Alice, and Karel provide visual feedback by allowing the relevant actors to perform the actions depicted in the program. Languages such as Logo and Kojo that employ turtle graphics, visualize the result as a graphical illustration embodying each command. However, the extent of this visualization varies from platform to platform. For example, Scratch can illustrate which line of code is causing the current actions of the actor, while Karel has no direct indications as to whether the program being executed is even the one that was last written [17].

The approach of visualizing the program flow step by step has been adopted by many modern learning platforms due to its effectiveness in educating novice learners on simple program structures and principles. Being able to see each command executed allows users to gain a better understanding of the meaning behind each step and construct logical relationships between instructions. Furthermore, it allows better insight into mistakes in program logic as even a wrong output is carefully illustrated [12], [45]. Therefore, execution visibility of an introductory programming language/platform directly affects novice learners' intention in using them since it is a useful concept in aiding the beginners to learn the concepts more effectively. Additionally, since novice learners often prefer the learning methodologies to be in visual mode, visualization of the program flow execution will contribute towards generating a positive attitude towards these platforms.

B. PERCEIVED EASE OF USE (PEOU)

To identify the perceived ease of use of a programming language, three main factors as the Grammar of the Programming Language, User Interfaces, and Availability of Resources have been considered. Hence, the programming languages that have been selected according to the inclusion criteria previously noted are reviewed under these factors to perceive the behavioral intention to use.

1) THE GRAMMAR OF THE PROGRAMMING LANGUAGE

The grammar of a language consists of two main features, the syntax and the inherent programming conventions of the language that should be followed during the coding process. To master these attributes, a novice learner may require constant practice and effort spanning over a considerable time. During the early stages of programming, novice learners struggle to understand, use and memorize the keywords and other elements of the syntax. Subsequently, these complications will demotivate the programmers in applying the concepts to solve problems. This will inevitably act as an obstacle when learning basic programming principles of the language [3], [18]. Hence, syntax-based coding issues act as an immense barrier to programming for most novices. Therefore, to minimize the overhead of the language syntax for the novices, most educational programming languages use simple and short keywords in their language design [5], [36].

There are two types of programming languages depending on the way the programs are written, i.e., text-based languages and visual-based languages. Although the majority of the general-purpose languages are text-based, most of the educational programming languages tend to lean more towards visual-based programming [7], [8].

Visual-based languages like Scratch, Blockly, and Alice use drag and drop functionality on pre-defined building blocks to create programs. Different attributes of the blocks like shapes and colors build up the visual grammar of the language which would play the role of syntax. Blocks are snapped together to make statements, expressions of the programs [15]. Due to the additional context regarding the commands and the parameters, most of the block-based programming language commands can be read as sentences [18]. It would help novices to better understand what the commands dictate and identify the program flow.

In text-based languages, programs are generated by typing the syntax into a code editor. Logo, Karel, and Kojo are examples of these languages and their syntax is lightly similar to existing general-purpose text-based languages[13], [36]. However, the keywords used in those languages are concise and often explicitly state their use in the programming as well as in general context. However, each line of code typed by the students must adhere to syntactic constraints of the corresponding language [18]. Hence, during the initial stages of coding in text-based languages, constant syntax errors may become frustrating to the novices. Therefore, to alleviate the above-mentioned issue faced by students, most text-based languages introduce a handful of primitive commands containing limited and concise instructions to be used at the initial stages [49]. Additionally, some of these text-based languages have introduced two modes of syntax practices to help novices gradually transit from simple one-line commands to complex syntactical programs. In Logo, ‘Immediate mode’ enables the novices to use the primitive set of commands like *Forward, Back, Right, Left,* and *Clear screen* as a single line and execute them. Once a student masters this level of complexity, ‘Program mode’ is introduced which enables the novices to program using a series of commands and execute the corresponding program [38].

Furthermore, most of the educational language designs are based on a ‘real’ general-purpose programming language, hence similarities in the syntax can be noticed. Although Alice is language independent, as it is built on top of Python, the grammar is interleaved with several Python syntax. Alice’s ‘if’ statements are directly drawn from Python. Similarly, Karel’s syntax is highly identical to that of Pascal, and Kojo’s grammar is directly derived from Scala [13], [17], [36].

Syntax of educational text-based languages is composed of keywords and symbols such as parentheses, curly braces, and semicolons. However, most of the block-based languages often forgo the use of these symbols and provide the students with pre-defined blocks to drag-and-drop [12]. Additionally, meaningless block connections will not be allowed by the drag-and-drop system itself, eliminating the possibility of syntactical errors [15], [49]. Therefore, the novices can solely focus on resolving semantic errors, logical errors, and grasping the basic programming concepts. Unfortunately, while this does seem appealing initially, having no error messages at all presents its problems as users find it much more difficult to shift into other high-level languages afterward [16], [17], [36]. Even though error messages often intimidate novices, their importance is also recognized as debugging and error handling are important skills to learn for any programmer [3], [47]. In text-based languages like Karel and Kojo, syntax errors and compile error messages are indicated. This enables novice programmers to gain a clearer insight into advanced learning opportunities and concepts in computer science. Therefore, although visual-based languages strive to unburden the novices from the syntax errors, educational text-based languages may assist beginners better when transferring to ‘real’ programming languages in later stages of their learning [17], [36].

Considering these language features of the grammar and syntax of each of the selected programming languages, it can be noticed that block-based languages show greater support to delivering key programming concepts and thus, creates a positive attitude towards the usage of these languages by beginners. However, block-based languages also omit some major concepts related to computer programming to increase the perceived ease of use which creates a negative attitude towards advanced learning of computer programming and contradicts the intended behavior of the user to learn basic programming concepts and principles. In contrast, text-based languages have lower perceived ease of use due to the possibility of syntax errors and complicated grammar but create a positive attitude towards learning over time as it fully supports the long-term intended behavior of delivering key programming concepts.

2) USER INTERFACES

The UI was observed to deliver different approaches to teach programming in each platform and language. However, many of these interfaces followed a common structure of supporting two main windows for code editing and displaying the output. Some languages including Scratch, Logo, and Karel presented this output in the form of graphics and animations while others like Kojo and Alice presented both a visual component and an output terminal for displaying either results or error messages. Many languages, especially those that were reliant on visual and block-based coding were noted to maintain their environment as a micro-world with actors (entities such as the Scratch Cat, Karel the Robot, and Logo's Turtle) that would carry out the commands defined within each program [17]. The use of actors to visualize the flow of a computer program has been extremely effective in delivering concepts, especially to younger audiences [13]. Furthermore, novices to programming find their fear of coding eliminated by the attractive interfaces and are often encouraged to explore the micro-world for themselves. However, when compared with text-based coding environments, it can also be argued that the visual and block-based components distract users from the task of 'real' programming [18], [19].

With consideration to these factors, it can be noted that the highly visual nature of the user interfaces and programming environment creates an overall positive attitude towards the learning of basic programming principles and concepts. Furthermore, the use of actors and micro-worlds also supports this and promotes the expected behavior of encouraging novice learners to participate in computer programming.

3) AVAILABILITY OF RESOURCES

Another important feature to be considered is the availability of language resources such as active communities and proper documentation that could support new users to correctly adopt the language. It is noted that the presence of detailed beginner-friendly documentation and learning materials as is available with languages like Scratch, Alice, and Kojo greatly influence the performance and interest of novice programmers attempting to self-learn concepts. Many languages maintain learning materials in the form of wikis, lessons, textbooks, curriculums, and online videos. Particularly, Scratch, Alice, Kojo, and Logo have a very high repository of learning materials where some are even available in multiple languages. Discussion forums and active communities are also important supporting features towards learning by novices and these are particularly well-maintained by popular languages like Scratch and Kojo. Others may have relatively inactive communities, but the lack of such communication groups is often compensated by good documentation in most cases.

Additionally, schools play a major role in programming education, with many countries introducing basic programming concepts to students at early levels. For example, in Japan and Singapore, computer programming is a fixed component of the national school curriculum [44]. The subject material involved typically covers basic programming concepts including variables, constants, data types and sequence, selection, and iteration control structures. In others, programming is often not integrated at the national level, but it is optionally taught using the same subject matter. In many countries including Japan, China, and Singapore, there is no compulsory programming language included in the curriculum to deliver concepts [42]–[44]. Hence, the programming environment used can greatly differ among student groups. However, many schools make use of various introductory programming environments such as Scratch, Logo, and Karel to deliver basic concepts to novice learners [15], [17], [18]. Support for such educational programming platforms from school curriculums has also contributed to the popularity of these platforms among novice programmers.

Hence, it can be determined that the availability of resource materials to support the learning of a particular programming language directly supports the goal of easing the delivery of key programming concepts. Furthermore, it can be noted that the high availability of such support materials creates a positive attitude among users with regards to the usage of the language and promotes the intended behavior of using the programming language for learning basic concepts and principles.

V. DISCUSSION

This paper aims to review existing educational programming languages and platforms targeting novice learners and to identify the main features of those languages that may affect the learning of key programming principles. Hence, for the context of this paper, such introductory languages are broadly categorized into two groups as visual/ block-based and text-based according to the approach presented to users for writing computer programs. Furthermore, it was decided to consider the common language features of programming languages/platforms belonging to these two categories. As such, several notable features were identified as the programming concepts and principles included in the language, execution visibility, language extensibility, liveness and tinkerbility, the grammar and syntax of the language, user interfaces, and the availability of resources.

Each of these noted language features was reviewed under TAM to identify the effect on the attitude of users and the behavioral intention to use the programming language in question. Hence, it was observed that each language feature differently influenced novice programmers towards the goal of learning basic programming principles and concepts to various degrees. A summary of these observations is discussed in Table 6.

While each language followed its unique approaches to deliver key concepts to users, each aspect was seen to have been incorporated to support learning by novices in some way or another. Hence, the following behaviors of novice programmers were seen to be considered in the design of these languages to support ease of learning [1], [4], [45].

TABLE 6. Summary of language features according to TAM.

TAM Dimension	Language features	Effect on Attitude towards Using
Perceived Usefulness	Programming concepts and principles	Both the block-based and text-based languages contain basic programming concepts. Hence both create a positive attitude towards learning. However, at times block-based languages present the concepts more effectively to the novices.
	Execution visibility	Execution visibility creates a positive attitude towards the use of the programming language.
	Language extensibility	Language extensibility has a major contribution towards a positive attitude to use. This feature is mostly available in Block-based languages.
	Liveness and Tinkerbility	Block-based languages create a positive attitude towards learning as they offer better liveness and tinkerbility features.
Perceived Ease of Use	Grammar and Syntax	Block-based languages create a positive attitude towards learning. However, the long-term effect is negative due to difficulties in switching to more advanced programming languages.
		Text-based languages generate a less positive attitude initially due to syntax and grammar errors.
	User Interfaces	Highly visual interfaces create a positive attitude towards the use of the programming language/ platform.
	Availability of Resources	Greater availability of learning resources and support materials create a positive attitude in novices towards using the programming language/platform.

1. Preference for a visually appealing coding environment for coding over regular text editors.
2. The desire for visual and understandable outputs regarding familiar scenarios or objects.
3. Preference for simplified syntaxes as opposed to the complex structures of mainstream programming languages.
4. Favor minimum interaction with cryptic error messages, especially syntax errors.
5. Preference for new or complex theories to be taught in their native language, particularly for non-native English speakers.
6. Preference to use languages or platforms that have many learning resources or active communities.

In consideration of these behaviors commonly exhibited by novice programmers, it was observed that many languages incorporated highly visual environments, with elements such as graphics, animations, and storylines to support the teaching process. Even those languages that expected the user to write the code themselves were seen to include some form of visual output to support this behavior. Furthermore, almost all introductory languages provided extremely simplified syntaxes and semantics with limited functionalities [18]. As these languages and platforms are targeting novice programmers, it was noted that extensive documentation and reference materials were also often freely available. Among the features considered localization of the languages was noted to have positive impacts on learning by novices. However, this concept was not seen to be implemented effectively by many of the languages considered by this paper.

Additionally, it was noted that some of these tendencies shown by novices, were further encouraged by these learning platforms and may have some long-term negative effects on the users' learning abilities. For example, users who grow used to visual environments are often seen to struggle when attempting to shift to high-level mainstream programming languages. Their reduced understanding of syntax, semantics, and error handling was seen to contribute to this result [17]–[19].

Therefore, considering the observations of this review as well as past literature, the inclusion of the following language features in the design of introductory programming languages and platforms is suggested to capture the advantages and mitigate the risks thus identified [7], [8], [15], [17]–[19].

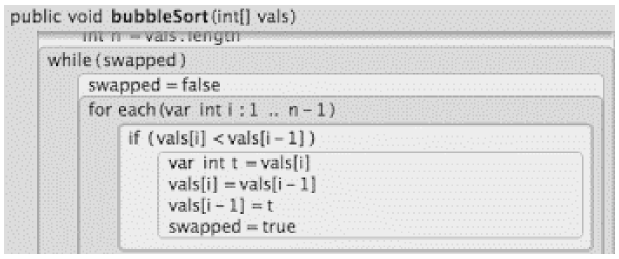
1. Text-based language representation as a compulsory feature (with a block or visual representation depending on the target audience)
2. Visualize outputs and maintain high execution visibility.
3. Interaction with errors and debugging with descriptive error messages suited for the target audience.
4. Simplified syntax and limited functionalities suitable for the main goal of the language.
5. Language localization support.
6. Beginner-friendly documentations tutorials

A. TRENDS AND RECOMMENDATIONS FOR FUTURE RESEARCH

The previously discussed results are observed to have direct implications towards current trends and potential areas of research in the field of introductory programming languages and platforms. The existing trends displayed during the development of the introductory programming languages and the possible future areas of research thus identified are further discussed below.

1) MODERN TRENDS IN INTRODUCTORY PROGRAMMING LANGUAGES

Considering previous literature, various pros and cons of both block-based and text-based educational programming languages have paved the way for the emergence of several trends with regards to the future directions of these learning platforms [37], [51], [52]. Two of the more prominent trends in this area will be further discussed in this section as follows.



```
public void bubbleSort(int[] vals)
    int n = vals.length
    while(swapped)
        swapped = false
        for each(var int i : 1 .. n - 1)
            if (vals[i] < vals[i - 1])
                var int t = vals[i]
                vals[i] = vals[i - 1]
                vals[i - 1] = t
                swapped = true
```

FIGURE 4. A simple frame-based program in Greenfoot 3.

2) FRAME-BASED PROGRAMMING

Multiple studies have revealed that the use of block-based visual languages enhances the delivery of programming concepts to novice programmers [17], [47]. As a result of the immense popularity and the positive research results of block-based programming tools, 'frame-based programming', which incorporates the basics of block-based programming has emerged [51], [52]. As shown in Fig. 4, although frames are conceptually similar to blocks, they provide a better structure to the program than block-based languages. Furthermore, features such as the 'frame cursor' and the single-keypress frame insertion mitigate the overhead related to the block palette and the drag-and-drop functionality of the block-based languages. Hence, scrolling and selecting a block category, selecting the desired block out of the category, and dragging it to the accurate position of the code can be done using a single keypress. Frame-based programming is currently incorporated in Greenfoot 3 [52]. With future improvements on keyboard support, frame manipulation, frame navigation to enhance program readability, and structuring, frame-based programming can transcend text-based programming [52].

3) DUAL MODALITY ENVIRONMENTS

Despite high preference and widespread use, novice programmers often display reluctance to perceive block-based languages as 'real' programming languages [16], [17]. Hence, novices may struggle to transit from block-based to conventional text-based languages [36]. To address these issues and deliver programming concepts more fluently, dual-modality environments are introduced. They encompass the interface design features of both block-based and text-based languages. The novice programmers can choose to program in either block-based or text-based interfaces depending on their preference [8], [51]. Languages such as Alice and Logo have incorporated the dual-modality concept into their language designs to give the 'best of both worlds' [13], [16]. In programming environments such as Pencil Code, novice learners are allowed to convert their block-based program into textual formats and vice-versa [8].

4) RECOMMENDATIONS FOR FUTURE RESEARCH

The field of programming languages is a heavily researched area. However, the areas concerning educational programming languages and introductory programming platforms to assist novice learners are emerging topics of research under this domain. Some of these potential areas and topics of research are listed below.

1. The effectiveness of using text-based programming languages with simple and short syntax patterns to teach core programming concepts to novice learners.
2. The effectiveness of integrating competitive-based learning techniques and game theory principles to teach core programming concepts.
3. The effectiveness of using a localized programming language to deliver core programming concepts.
4. An approach to design a simplified and translatable introductory programming language.
5. The effectiveness of incorporating a storyline-based exercise set to enhance the learning experience of novice programmers.
6. The effectiveness of using interactive programming platforms to promote self-learning in novice learners.
7. The use of automated feedback of exercises and error messages in introductory programming languages and platforms.

VI. LIMITATIONS

One of the major limitations of this review is most of the reviewed and analyzed literature revolve around the pre-selected 5 introductory programming languages and platforms, namely Scratch, Alice, Kojo, Karel, and Logo. Since there are many other introductory programming languages and platforms available, there may be other educational programming languages that were not included in this analysis which may have had important information. However, to address this issue, the five languages and platforms were selected based on the popularity among the users and the number of google articles and research articles available for these languages. Even though many languages and platforms were excluded in the review, the two metrics used ensured the languages with the most impact were considered. Additionally, to counter this limitation, some of the literature on languages such as Pencil Code, Blockly, Hackety Hack, and Flowgorithm which displayed interesting similarities to the main five languages were also included in the review. Therefore, the writers are satisfied with the inclusion criteria for this article since the most effective languages and their features which encompass a diverse range of introductory programming languages were empirically analyzed.

Additionally, due to the exclusive inclusion of five introductory programming languages and platforms and the existence of multiple different keywords associated with the topic of review, there may be articles and publications that were excluded. To counter this limitation, only the most recent published literature was chosen for the process of review. Therefore, both the past and the most recent findings regarding the topic of introductory programming languages were analyzed and reviewed.

VII. CONCLUSION

At present, many students from various backgrounds and knowledge levels attempt to learn computer programming and many introductory level programming languages that have been designed to support their learning process. This paper reviews the different language features incorporated within such languages and how they affect the learning rates of novices and their acceptance towards the usage of an introductory programming language according to TAM. Hence, the following conclusions were arrived upon, following the observations of the review conducted.

1. Visual-based coding environments are more suitable for teaching concepts to beginners of younger age categories, but text-based representation is necessary to support long-term learning and shifting to mainstream high-level programming languages.
2. Simplified syntax and limited functionality are optimal for better understanding. However, simplification should maintain some semblance of logical constructs, syntax, semantics, and error management.
3. Writing computer programs in one's native language is favorable for delivering concepts to novices effectively, but knowledge of English keywords is also necessary for long-term sustainability.
4. The presence of beginner-friendly documentation and user guides as well as an environment that supports self-learning by novices is highly influential to effective learning interest building.

Further work of this study will entail more in-depth analysis into the approaches to design a simplified and translatable introductory programming language aimed at novices, the integration of competitive learning theories in delivering basic programming concepts, and the effectiveness of simplified and localized programming languages to learning by novices.

REFERENCES

[1] A. G. S. Raj, K. Ketsuriyonk, J. M. Patel, and R. Halverson, "Does native language play a role in learning a programming language?" in *Proc. 49th ACM Tech. Symp. Comput. Sci. Educ.*, Feb. 2018, pp. 417–422.

[2] S. Dasgupta and B. M. Hill, "Learning to code in localized programming languages," in *Proc. 4th ACM Conf. Learn. Scale*, Apr. 2017, pp. 33–39.

[3] P. K. Sevella and Y. Lee, "Determining the barriers faced by novice programmers," *Int. J. Softw. Eng.*, vol. 4, no. 1, pp. 10–22, 2013.

[4] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "Understanding the syntax barrier for novices," in *Proc. 16th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Jul. 2011, pp. 208–212.

[5] C. Malliarakis, M. Satratzemi, and S. Xinogalos, "CMX: The effects of an educational MMORPG on learning and teaching computer programming," *IEEE Trans. Learn. Technol.*, vol. 10, no. 2, pp. 219–235, Apr. 2017.

[6] W. S. Burleson, D. B. Harlow, K. J. Nilsen, K. Perlin, N. Freed, C. N. Jensen, B. Lahey, P. Lu, and K. Muldner, "Active learning environments with robotic tangibles: Children's physical and virtual spatial programming experiences," *IEEE Trans. Learn. Technol.*, vol. 11, no. 1, pp. 96–106, Jan. 2018.

[7] L. Moors, A. Luxton-Reilly, and P. Denny, "Transitioning from block-based to text-based programming languages," in *Proc. 6th Int. Conf. Learn. Teach. Comput. Eng. (LaTICE)*, Apr. 2018, pp. 57–64.

[8] D. Weintrop and N. Holbert, "From blocks to text and back: Programming patterns in a dual-modality environment," in *Proc. Conf. Integr. Technol. Into Comput. Sci. Educ. (ITiCSE)*, Jan. 2018, pp. 633–638.

[9] A. Black, K. B. Bruce, and J. Noble, "Panel: Designing the next educational programming language," in *Proc. ACM Int. Conf. Companion Object Oriented Program. Syst. Lang. Appl. Companion (SPLASH)*, 2010, pp. 201–203.

[10] P. J. Guo, "Non-native english speakers learning computer programming: Barriers, desires, and design opportunities," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, pp. 1–14.

[11] E. Mehmood, A. Abid, M. S. Farooq, and N. A. Nawaz, "Curriculum, teaching and learning, and assessments for introductory programming course," *IEEE Access*, vol. 8, pp. 125961–125981, 2020.

[12] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–15, Nov. 2010.

[13] R. Pausch and W. Dann, "ALICE: A 3-D tool for introductory programming concepts," *J. Comput. Sci. Colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[14] J. Cardenas-Cobo, A. Puris, P. Novoa-Hernandez, J. A. Galindo, and D. Benavides, "Recommender systems and scratch: An integrated approach for enhancing computer programming learning," *IEEE Trans. Learn. Technol.*, vol. 13, no. 2, pp. 387–403, Apr. 2020.

[15] A. Ebrahimi, S. Geranzeli, and T. Shokouhi, "Programming for children: Alice and scratch analysis," in *Proc. 3rd Int. Conf. Emerg. Trends Comput. Inf. Technol. (ICETCIT)*, Nov. 2013, pp. 106–115.

[16] D. H. Clements and J. S. Meredith, "Research on logo: Effects and efficacy," *J. Comput. Child. Educ.*, vol. 4, no. 4, pp. 263–290, 1992.

[17] A. Ruf, A. Mühling, and P. Hubwieser, "Scratch vs. karel: Impact on learning outcomes and motivation," in *Proc. 9th Workshop Primary Secondary Comput. Educ. (WiPSCE)*, 2014, pp. 50–59.

[18] C. M. Lewis, "How programming environment shapes perception, learning and goals: Logo vs. scratch," in *Proc. 41st ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2010, pp. 346–350.

[19] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," in *Proc. 16th Annu. Joint Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2011, pp. 168–172.

[20] B. W. Wirtz and V. Göttel, "Technology acceptance in social media: Review, synthesis and directions for future empirical research," *J. Electron. Commer. Res.*, vol. 17, no. 2, pp. 97–115, 2016.

[21] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quart. Manage. Inf. Syst.*, vol. 13, no. 3, pp. 319–339, 1989.

[22] W. M. Al-Rahmi, N. Yahaya, A. A. Aldraiweesh, M. M. Alamri, N. A. Aljarboa, U. Alturki, and A. A. Aljeraiwi, "Integrating technology acceptance model with innovation diffusion theory: An empirical investigation on students' intention to use E-learning systems," *IEEE Access*, vol. 7, pp. 26797–26809, 2019.

[23] S. A. Salloum, A. Q. M. Alhamad, M. Al-Emran, A. A. Monem, and K. Shaalan, "Exploring students' acceptance of E-learning through the development of a comprehensive technology acceptance model," *IEEE Access*, vol. 7, pp. 128445–128462, 2019.

[24] V. Venkatesh and H. Bala, "Venkatesh_et_al-2008-decision_sciences," *J. Decis. Sci. Inst.*, vol. 39, no. 2, pp. 273–315, 2008.

[25] V. Venkatesh and F. D. Davis, "A theoretical extension of the technology acceptance model: Four longitudinal field studies," *Manage. Sci.*, vol. 46, no. 2, pp. 186–204, Feb. 2000.

[26] A. Al-Aulamie, "Enhanced technology acceptance model to explain and predict learners' behavioural intentions in learning management systems," Univ. Bedfordshire, London, U.K., Tech. Rep., 2013, pp. 1–124.

[27] A. N. Pears *et al.*, "A survey of literature on the teaching of introductory programming Arnold," in *Proc. 12th Annu. Conf. Innov. Technol. Comput. Sci. Educ.*, Dundee, Scotland, vol. 1846, 2018, pp. 204–223.

[28] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "A review of introductory programming research 2003–2017," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Jul. 2018, pp. 342–343.

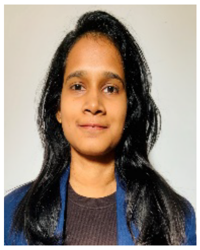
[29] S. M. Salleh, Z. Shukur, and H. M. Judi, "Analysis of research in programming teaching tools: An initial review," *Procedia-Social Behav. Sci.*, vol. 103, pp. 127–135, Nov. 2013.

[30] A. Gomes and A. Mendes, "A teacher's view about introductory programming teaching and learning: Difficulties, strategies and motivations," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2014, pp. 1–8.

[31] T. Crow, A. Luxton-Reilly, and B. Wuensche, "Intelligent tutoring systems for programming education: A systematic review," in *Proc. 20th Australas. Comput. Educ. Conf. (ACE)*, 2018, pp. 53–62.

[32] J. Sorva, V. Karavirta, and L. Malmi, "A review of generic program visualization systems for introductory programming education," *ACM Trans. Comput. Educ.*, vol. 13, no. 4, pp. 1–64, Nov. 2013.
[33] A. S. Kim and A. J. Ko, "A pedagogical analysis of online coding tutorials," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ. (ITiCSE)*, Mar. 2017, pp. 321–326.
[34] M. Banerjee, M. Capozzoli, L. McSweeney, and D. Sinha, "Beyond kappa: A review of interrater agreement measures," *Can. J. Statist.*, vol. 27, no. 1, pp. 3–23, Mar. 1999.
[35] S. Simon, A. Carbone, M. de Raadt, R. Lister, M. Hamilton, and J. Sheard, "Classifying computing education papers: Process and results," in *Proc. 4th Int. Workshop Comput. Educ. Res. (ICER)*, 2008, pp. 161–171.
[36] B. Regnell and L. Pant, "Teaching programming to young learners using Scala and Kojo," *LTHs Pedagog. Inspirationskonferens*, vol. 8, p. 17, Dec. 2014.
[37] D. Weintrop and U. Wilensky, "How block-based, text-based, and hybrid block/text modalities shape novice programming practices," *Int. J. Child-Comput. Interact.*, vol. 17, pp. 83–92, Sep. 2018.
[38] B. Pardamean, E. Evelin, and H. Honni, "The effect of logo programming language for creativity and problem solving," in *Proc. 10th WSEAS Int. Conf. E-Activities*, Jun. 2014, pp. 151–156.
[39] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.
[40] M. Hanus, H. Kuchen, J. J. Moreno-Navarro, J. Votano, M. Parham, and L. Hall, "Curry: A truly functional logic language," in *Proc. Workshop Vis. Future Log. Program. (ILPS)*, Mar. 2013, pp. 95–107.
[41] P. Van Roy, S. Haridi, C. Schulte, and G. Smolka, "A history of the OZ multiparadigm language," *Proc. ACM Program. Lang.*, vol. 4, pp. 1–56, Jun. 2020.
[42] S. Çapuk, "ICT integration models into middle and high school curriculum in the USA," *Procedia-Social Behav. Sci.*, vol. 191, pp. 1218–1224, Jun. 2015.
[43] W. Guo and Z. Yang, "A study of integration ICT into curriculum in China's developed areas—A case of Foshan city," in *Proc. 4th Int. Conf. Manage., Educ., Inf. Control (MEICI)*, 2016, pp. 594–598.
[44] L. Sturman, "International comparison of computing in schools," Nat. Found. Educ. Res., Slough, U.K., Tech. Rep., 2011.
[45] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein, "Modeling how students learn to program," in *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2012, pp. 153–158.
[46] M. S. Farooq, A. Abid, S. A. Khan, M. A. Naeem, A. Farooq, K. Abid, and M. Shafiq, "A qualitative framework for introducing programming language at high school," *J. Qual. Technol. Manage.*, vol. 8, no. 2, pp. 135–151, 2012.
[47] M. Hjorth, "Strengths and weaknesses of a visual programming language in a learning context with children," School Comput. Sci. Commun., Sweden, Tech. Rep., 2017.
[48] M. Jancheski, "Improving teaching and learning computer programming in schools through educational software," *OLYMPIADS Informat.*, vol. 11, no. 1, pp. 55–75, Jul. 2017.
[49] S. M. Biju, "Taking advantage of Alice to teach programming concepts," *E-Learn. Digit. Media*, vol. 10, no. 1, pp. 22–29, Feb. 2013.
[50] L. Pant, "Introduction to programming with Kojo," Tech. Rep., Aug. 2018.
[51] N. C. C. Brown, J. Mönig, A. Bau, and D. Weintrop, "Panel: Future directions of block-based programming," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, Feb. 2016, pp. 315–316.
[52] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-based editing: Easing the transition from blocks to text-based programming," in *Proc. Workshop Primary Secondary Comput. Educ.*, vols. 9–11, 2015, pp. 29–38.

PIUMI PERERA is currently a Research Scholar with the Faculty of Information Technology, University of Moratuwa, Sri Lanka. She is a published author at a reputed international conference and represented the faculty in several international programs, including the Huawei Seeds for the Future program in her student years. Her current research interests include visual-based educational programing environments and native language programing education.



GEETHYA TENNAKOON is currently a Research Scholar with the Faculty of Information Technology, University of Moratuwa, Sri Lanka. She is also acting as a Software Engineer with the Graduate Program offered by the London Stock Exchange Group. Her research interests include compiler theory and programing languages.

SUPUNMALI AHANGAMA (Associate Member, IEEE) received the Ph.D. degree in information systems from the National University of Singapore. She is currently a Senior Lecturer with the Department of Information Technology, University of Moratuwa, Sri Lanka. She also holds the position as the Director of undergraduate studies, Faculty of Information Technology, University of Moratuwa. Her research interests include data science, design science, information systems, and e-education. She has presented her work and served as a Reviewer in numerous top-tier forums, including *e-Service Journal*, *Information Systems Frontiers*, International Conference on Information Systems (ICIS), Pacific-Asia Conference on Information Systems (PACIS), the Workshop on Information Technologies and Systems (WITS), and HCI International.

RANGANA PANDITHARATHNA is currently a Researcher with the Faculty of Information Technology, University of Moratuwa, Sri Lanka, focused on game theory, competitive learning, and education.

BUDDHIKA CHATHURANGA is currently a Research Scholar with the Faculty of Information Technology, University of Moratuwa, Sri Lanka. He participated in GSoC 2020 contributing to Jenkins and presented at the DevOps World 2020 conference. His current research interests include compiler theory and theory of programing.