The CodeEazee Tool Support for Computational Thinking in Python

Francisca O. Oladipo, and Memunat A. Ibrahim

Abstract—This paper describes the development of CodeEazee, a problem solving, self-teaching tool for python programming which deploys templates and games. In this work, the authors conducted a survey to determine the factors responsible for the reduced interests of learners in programming, reviewed the various approaches used in teaching programming, and developed a python-for-python teaching system to teach programming skills, computational thinking, algorithms’ design, programming in general and Python programming specifically. The work would show how the third party environment had enabled users with limited or no programming experiences to design applications through peer supports, templates and gamification, embedded in a programming tool.

Index Terms—Algorithms; Computation; Problem-Solving Skills; Programming; Python.

I. INTRODUCTION

Over the years, there has been a paradigm shift from the use of chalkboard, biro, pencils and books, since projectors, e-books, text editors, educational websites and e-exams was introduced making computer education a necessity for every student [1]. According to research, the need for programmers in the workplace has increased, necessitating making programming a core skill for every student [2]. [3]. Some organizations and institutions consider computer education as an employment requirement [4], and programming as an additional skill for non-CS majors as it enhance their productivity and efficiency; while some non-CS students see it as a way of understanding how computers operate [5].

Learning and teaching beginners programming is considered difficult by both students and educators [6], [7], leading to a high failure rate [8] and constant reduction in the rate of enrollment of computer related courses [9], [4]. According to [7], even Computer Science students find programming to be difficult as it requires reasoning, planning and problem solving.

Most non-CS majors view programming as uninteresting and unproductive since the concept seem abstract and difficult to comprehend, and some only developed “hello world” applications in the introductory course. They also find it confusing as the lecture materials’ content does not lead to a solution when applied in finding solutions to problems; which is because most lecture materials present the non-linear and dynamic programming processes as static and linear, making them assume that the link between every problem and solution is simple and direct [10].

Basically, two factors are fundamental to teaching programming, which are: choosing a programming language and choosing an effective approach for teaching [11]. These choices are essential to the students understanding programming and acquiring the necessary programming skills, and also to prepare them for other higher programming course [12]; and a wrong choice of programming language [9] or teaching approach can make learning programming difficult [13]. Similarly, knowing the students’ learning styles will help in determining the best teaching approach to be used [14], leading to the conclusion that finding and employing an appropriate teaching technique will play an important role in increasing students’ success rate [15]. However, a study by [4] shows that students’ performance is independent of the choice of programming language.

Generally, most non-Computer Science (CS) majors have no idea about what programs or programming is, some of them assume programming is all about robots, abstract languages, hacking, nerds, computer engineering etc. Contrary to their view, programming is the art of representing solutions to the computer in a way the computer understands; every computer based system like search engines, social networks, online services, calculators, mobile phones are controlled by programs [11].

Our findings revealed that introductory courses fail to awaken students’ interest and tend to propel students from programming instead of attracting them to it as a result of these factors:

i. More focus is paid to teaching students to write programs in a particular programming language and the language syntax, than problems solving skills and programming processes [16].

ii. The course is more of lecture classes than of programming practice classes.

iii. Students are not challenged to solve real life problems with the acquired programming knowledge.

iv. Students’ desire to pass the course surpasses that of assimilating it.

v. Some interested students do not have the tools needed for their personal practice or learning.

This research is aimed at making learning and understanding of programming easier for non CS students by equipping the students with the skills required to write programs. Thereby increasing the number of students enrolled in programming based courses. We developed an easily accessed tool to teach programming to students without mathematical aptitudes using templates and
gamification. Students computational and problem-solving skills are improved through a self-teaching and individual study environment. We limited our tool support to Python programming language, as it is the introductory language for most non-CS programs, friendlier to novice programmers [17] and the skills and techniques can be ported into other more complex and less-functional programming paradigms.

This paper is organized as follows: This section on introduction provides a brief background to the concepts of teaching the syntax and semantics of a programming language to non-CS students and some of the factors responsible for decreased interests in programming among these classes of undergraduates. This is followed by an extensive literature review on relevant approaches, tools and techniques for teaching programming to both CS and non-CS undergraduates. The third section presents the materials and methods adopted in order to successfully develop the learning tool and the fourth section discusses in details the design artefacts for the CodeEazee tool. An appendix of the user interface is presented as well.

II. APPROACHES TO TEACHING PROGRAMMING

Teaching programming can be a very tedious and challenging job since there are a lot of skills to be taught, and ever since the introduction of programming education, there is still no standardized approach to teaching programming. Generally, several formal approaches to teaching programming had been identifying to include: problem-based learning [18], pair programming [19], puzzle based learning [20], game-themed [21], prerecorded lectures [22], team-based and project based learning [15]. Other approaches are programming on a board, where teachers write programs on the board, and students can follow up and ask questions; the disadvantage is programs cannot be executed and only small programs can be written or taught. Finished program on transparencies, where finished complex programs are presented to the students; here most times students are unable to follow the instructor’s progress unlike programming on a board. Live programming, where students are taught programming with the aid of projector and computer; here the class is interactive and students can run the program, this approach solves the limitation of programming on a board. Process recording, where an expert’s programming process is recorded and presented in the class, students can review the materials as much as needed [10].

Though, with these approaches in use, it has been observed that there are other factors that affect students’ performance in introductory programming courses which are the programming language being taught [23], which is opposed by [4], teaching approach used, students’ learning style, abstract thinking, logical thinking, mathematical skills, problem solving ability, analytical thinking, class size, availability of resources and how comfortable, confident, interested, motivated and persistent [24] both the students and instructors are.

III. REVIEW OF RELATED WORK

From a historical perspective, in 1947 the Department of Mathematics in the University of Sydney, introduced a new course: The Theory of Computation, Computing Practices, and Theory of Programming, marked as the first programming course in Australia. In 1956, Programming courses were offered in the University of Melbourne [25]; and by 1958, some students were learning how to code in FORTRAN with punch cards as the input device in the USA [26]. In the late 1950s, a decision was taken to use computer in the Department of Defense (DOD) and the post master general’s department by the Commonwealth Government, creating the need for professional computing personnel. At this point, universities started to wonder if computing should remain under mathematics or should be made a new discipline. Later, the Australian government’s commonwealth public service board made the Programmers in Training scheme in 1960 to provide short-term solution to the scarcity of programmers and other computer professionals in the commonwealth government departments [25]. In the 1960s, the National Science Foundation (NSF), USA supported many research projects in the area of computer technology in education, like computer assisted instruction and how to effectively use computer tools [26].

On May 1, 1964, at Dartmouth, Thomas Kurtz and John Kemeny launched their BASIC programming language [26], and in 1965, Computer Science called “discipline of information processing (IP)” was first launched as an independent discipline in Finland. Students had to use only one small computer and/or punched card as input device with paper tape [27]. Later in 1967, Logo was created at Bolt by Seymour Papert and Wally Feurzeig with the goal of creating a math land where children could play with sentences and words. By 1968-1969, Muzzey Jr High, Lexington MA, USA became the earliest school user of Logo ([26]. By 1969, in Monash University’s science degree, Computer Science was being offered [25]. In 1972, high school and middle school students in the USA started learning to use word processors in computer applications class [26]. In 1974 at Monash University, Australia, a group developed a system with mark-sense cards which could allow a class of 30 students and each can run codes; and The Monash Educational Computer System (MONECS) system was used to teach programming in FORTRAN or BASIC [25].

In the 1980s, the introduction of personal computers and user applications resulted in a shifting education focus to teaching and learning how computers can be used for office based tasks like creating presentations and word processing [28]. In 1983, a programming course in BASIC was sanctioned and included in the following academic year’s GVSU computing curriculum; later in 1986, the course was also included in the general education program for the next academic year so that students can learn problem solving skills and logical thinking while learning the syntax of BASIC.

In 1987, the Nigeria Federal Government resolved to introduce computer education into the nation’s education system after the 32nd ministerial council meeting of the National Council on Education. Thereby, launching a national policy on computer literacy at every educational level in 1988 [29]. This was aimed at equipping students with an understanding of the computing concept so as to conform to the next generation. Later in 1997, state
governments introduced computer education in secondary schools to enable both the students and teachers to use computer as a learning tool in their respective subject areas [1]. However, learning to use computers (computer education) and not to program computers, restrains the user’s ability to harness the power of computers and communicate in the digital world [28]. They proposed that learning programming should be made compulsory in every institution by including an introductory programming course in the general education program. This will result in the non-CS majors learning more skills and a broader multi-discipline degree.

Teaching programming to every non-Computer Science major is as important as teaching history and literature to every student irrespective of their majors. Programming education clarifies the world we live in to the students, a world that is connected by computers. Learning programming exposes students to the understanding, capabilities and limitations of computation and this can only be achieved by teaching students to design programs, write programs and practice programming [29].

Learning and teaching programming is considered difficult by both students and educators because of the cognitive skills involved. Teaching programming involves teaching a variety of skills which include computational thinking, problem solving, algorithmic, programming logics, the language syntax etc. However, the amount of time allocated to teaching introductory programming courses is too short to teach these skills [30], resulting in instructors compressing the course content. In solving this, programming teaching tools were introduced; some of which are Codecademy, MOOC, Alice, Online Python Tutor, Raptor, RoboProf, Wolfram Language, SICAS, etc.

Massive Open Online Course (MOOC) is a free online course designed to attract students to programming and motivate its users to write programs, by introducing its users to programming basics, providing them with learning materials and assignment, the introductory programming course is offered in python [31]. However, students also need practical tools to practice with and assess their programming skills, which MOOC fail to provide and a successfully completed assignment increase students’ programming confidence.

RAPTOR (Rapid Algorithmic Prototyping Tool for Ordered Reasoning) developed using C#, C++, Ada and also runs in the .NET framework is an environment where students can create algorithms using basic graphical symbols, and run the algorithms either step-by-step or in continuous play mode. It was developed to address the issues of both the teacher and students focusing on syntactic related difficulty, and neglecting algorithmic problem solving [32]. App Inventor is a language, developed by Google to teach introductory Computer Science. Since App Inventor is a visual language (applications are created by placing blocks together) students focus more on the events and logic of the application rather than the programming language syntax. It teaches the principles of software engineering, and requires prior knowledge of database design and SQL. This requirement prevents beginners from using the environment [33].

The EduJudge is a system for programming education aimed at managing the execution of the programming exercises based on their several contexts (assignments, tests, programming competitions, etc.). It consists of a user interface, a repository and an evaluation server. However, the system’s feedback is slow and poorly detailed; and needs improvement from the on-line judge should be improved [34]. Codecademy is an online web-based tool designed to teach students programming languages but it is focused on teaching the basics of programming languages, without teaching its users how they can apply them to real-life problems. According to [35], being a programmer requires more of thinking and having the right mindset and problem solving skill than knowing the language’s.

The Wolfram Language which was designed mainly to teach computational thinking, encourages a new approach to programming. Instead of writing codes to tell the computer what to do, the user just have to directly express computational thinking. In this approach, people provide their ideas to the system, while the Wolfram Language and the computer handle the low level details of their execution thereby allowing the programmer to focus more on computational thinking, and not mere programming [36].

Online Python Tutor is a web based tool that provides a visualization of how programs are executed line by line to its users. It was developed as a final year project aimed at making students understand how programs are executed. Since it was launched, it has been used as a teaching aid in classrooms, in digital textbooks and by students for self-learning [37]. However, python tutor can only serve as a supporting tool for teaching programming; it only shows the users how their codes are executed.

In solving the problems relating to the difficulties faced by students in learning programming languages’ syntax, Alice was developed at Carnegie Mellon University to teach young students programming. Alice teaches algorithm and other problem solving strategies in an animation environment, allowing students to build software they can relate with and use animations to visualize how their programs are executed. Alice however is only befitting for younger audience as it is event driven [38]. However, we are targeting first year students that are expected to apply these skills in solving real life problems and Alice cannot be used in developing standard applications.

Having it in mind that students’ major problem is their inability to solve problems, [14] developed SICAS (a Portuguese acronym for Interactive System for Construction of Algorithms and its Simulation) at the University of Coimbra; it’s focus is teaching students algorithms, as well as an ability to design, run and correct them. However, this tool could not provide the necessary support considering that others skills are required to program and develop a problem-solving mindset.

RoboProf which is a web based program used to automatically assess students’ programs. It was developed help students acquire problem solving skills and programming confidence; it provides an environment for students to practice programming, randomly generates questions, marks students’ solutions and provide feedback to the students [39]. According to the researchers, with RoboProf provides an environment to correct poor syntax and programming skills by showing students the expected
output; and giving chance for resubmission can also help them with writing a correct solution. This tool is used in Dublin City University, it only evaluates students programming skills and cannot be assessed outside the class. However, some students do not appreciate the system, as they are scored zero for even almost correct programs.

In order to keep non-CS majors motivated, attracted to, and interested in programming, and also quickly equip them with the programming experience and skills needed, [39] developed a tool at West Chester University (WCU) that teaches students analysis and thinking processes using examples, it designate tasks that are not too difficult to solve to them, and also automatically generate outcome (grades) of immediate tasks done.

To provide immediate assistance to students remotely and also to help improve students’ weak points in programming, [40] created a tool where students can develop programs increasingly while learning programming. This tool provides students with feedbacks and hints while coding, and solutions to programming exercises. However, the techniques used in these solutions differ from the students’ techniques, and the students’ approaches are often too complex for the system to handle. Also, sometimes, the response time of the system is very slow, and the tutor only recognizes solutions that tally with the model solutions but cannot recognize other correct solutions submitted by the students.

To solve the problem of the inability of students with good to write programs, [41] developed an online training system (Kaleidia system) at Tsinghua University aimed at improving students’ computational and problem solving skills; and teaching basic programming using the C programming language. The Kaleidia system edits, compiles, and automatically marks students’ programs; it also teaches students programming with examples and pattern. The users of the system are the students, teachers, and administrators. The teachers post assignment on the system weekly, the students can write, compile, and debug programs online.

Reference [12] developed a teaching tool at the University of Sharjah, UAE, to develop students’ creativity in solving problems and program solving skills; and teaching basic programming using the C programming language. The Kaleidia system edits, compiles, and automatically marks students’ programs; it also teaches students programming with examples and pattern. The users of the system are the students, teachers, and administrators. The teachers post assignment on the system weekly, the students can write, compile, and debug programs online.

Over the period of four weeks, an observation of the perception of programming by non-CS students were carried out. Several in person surveys were conducted within the target population to determine the factors responsible for the lack of interest of these students in courses that teach programming. The Extreme Programming (XP) methodology was deployed in the development of the teaching tool named CodeEazee as it provides support for changing requirements which was one of thing that characterised this human-centred development. The research successfully passed through all the stages of XP. The project management plan was draw and continuously refined as a result of discovery of new facts. This is a common thing in human-centred research as students sometimes find it difficult to describe their expectations in concise terms. Next the requirements were elucidated where technical, economic and organizational feasibilities were determined. In addition, the system’s requirements and constraints were modeled into logical design that described the relationship between the components, and the low level models were designed (Fig. 1). The system was implemented using the following development tools: HTML 5; Sublime text due to its support for cross-platform code editing; CSS 3; JavaScript 1.8.5; PHP 5.5.12. MySQL 5.6.17 DBMS was deployed for the database specification paradigm of the system and the web application was tested in the Mozilla Firefox 43.0 environment.

IV. MATERIALS AND METHODS

A. High Level Model

The High Level Model for the CodeEazee system consists of two major parts: the information management system and the e-learning management system (Fig. 2). The student management system takes care of users’ registration, login, and other information related process; while the e-learning management system takes care of the learning and tutoring session of the system; this is where the contents of the system is managed.
B. Data Flow Diagram

In the Data Flow Diagram for the CodeEazee system (Fig. 3), users’ data is accepted at the user interface, where it is transferred to the interpreter if the data was entered at the interpreter’s interface; or to the database if the data was accepted at the user’s registration or login form. Before a user can use CodeEazee, the user must first register by creating an account, a registered user can then gain access to the system with their login details. If a user is granted access to the system, the user can learn programming in general, problem solving, and python programming skills from the system, update profile, download reading materials and write Python codes and execute them.

V. SYSTEM ARCHITECTURE AND DESIGN ARTEFACTS

Fig. 4 illustrates the client-server architecture as applied to the development of the CodeEazee system, consisting of both hardware and software components. The software components consist of the data access logic programs, the presentation logic programs and the database while the hardware components are a server and clients which could be smart phones, PDAs, PCs, and every other resource-seeking system. When a client sends an HTTP request to the server through the CodeEazee interface; the Apache Server processes the request and check if it is a request for an HTML file or a request for an application. If it is a request for an HTML file, it sends the file to the interface; else it forwards the request to the app server. The app server, if the request needs to access the database, the request is sent to the database server. The request is processed and executed with the results sent to the app server, then to the Apache server, and back to the CodeEazee interface where the client receive responds from the server.

The design artefacts for CodeEazee were produced in UML descriptions. While Fig. 5 illustrates the use case description for the system, showing two categories of use cases and each use case represents the operation a user can perform using the system; the control flow from one activity to another is depicted in (Fig. 6) using a UML activity diagram. The upper filled circle is used to depict the start of a process while the lower a filled circle inside another circle is used to depict the end of a process. Rounded rectangles represent user activities (some of which are concurrent) and arrows between activities. When there is a flow from an activity to a number of activities, these activities may be executed in parallel.

Fig. 2. High Level Model of the CodeEazee System

Fig. 3. Data Flow Diagram

Fig. 4. The System Architecture

Fig. 5. Use Case diagram of CodeEazee

Interactions between the actors and the system’s components are shown in a dynamic model as explicit sequence of messages in a sequence diagram (Fig. 7). Messages are illustrated by solid lines (links) while arrows
on the link depicts the direction of the message. The order of messages goes from the top of the diagram to the bottom; higher messages are messages that occurred earlier in the sequence, and the lower messages occur later. Fig. 8 shows the class hierarchies for the system.

VI. USER INTERFACE DESIGN AND SPECIFICATION

Since the system is majorly designed for undergraduate students, simple graphics, images and colors were used since these more of these components are mostly preferred for younger users. Also, the input data are restricted to texts, numbers, and button clicks, while the outputs are images and texts for consistency; messages or feedbacks gotten from the system are as explicit, that is, the use of computer science jargons is avoided since the main users are not computer science majors.

Fig. 6. CodeEazee Activity diagram

The index screen (Fig. 9) is first presented to the user at the start of the CodeEazee application and it is the same for every user. A new user selects the "Register" button and is directed to the registration page (Fig. 10) to create an account and an existing user selects the "Login" option to be directed to the login page (Fig. 11) to continue into the system.

Fig. 7. CodeEazee Sequence Diagram
After Login, a new user is directed to the Home Page (Fig. 12.), while an existing user is given an option to continue from the previous page (Fig. 13).

The CodeEazee code window (Fig. 14a, b) mimics the Skulpt Python interpreter to provide a portable programming environment similar to most python editors thereby allowing for the writing and interpreting of python codes.

**Fig. 8.** CodeEazee Class Hierarchy

**Fig. 9.** CodeEazee Index Screen

**Fig. 10.** Sample UI to create an account in CodeEazee

**Fig. 11.** Sample UI to Login in to CodeEazee

**Fig. 12.** The Home Page for a new user

**Fig. 13.** CodeEazee Resume Page for existing user
VII. CODEEAZEE PERFORMANCE EVALUATION

An initial survey consisting on 73 users across four user groups (non-CS departments) of ten metrics each with five-point scale (1-Strongly Disagree, 2-Disagree, 3-Neutral, 4-Agree, 5-Strongly Agree) gave an average score as shown in Table I.

| Metrics                      | Non-CS User Group Average Score |
|------------------------------|----------------------------------|
| Ease of Use                  | BIO: 4.3, PHY: 4.5, CHM: 4.2, GEO: 4.5 |
| User-friendliness            | BIO: 4.4, PHY: 4.6, CHM: 4.3, GEO: 4.3 |
| Attractiveness               | BIO: 4.7, PHY: 4.4, CHM: 4.6, GEO: 4.6 |
| Operational stability        | BIO: 4.0, PHY: 4.1, CHM: 4.0, GEO: 4.3 |
| Ease of accessing shared materials | BIO: 4.6, PHY: 4.5, CHM: 4.5, GEO: 4.3 |
| Feedback                     | BIO: 4.6, PHY: 4.4, CHM: 4.4, GEO: 4.4 |
| Responsiveness               | BIO: 4.1, PHY: 4.2, CHM: 4.2, GEO: 4.2 |
| Sufficient content           | BIO: 4.5, PHY: 5.6, CHM: 4.6, GEO: 4.4 |
| Useful content               | BIO: 5.0, PHY: 5.0, CHM: 4.6, GEO: 4.4 |

Fig. 15 gives the user experience static journey map from 152 non-CS students who used the system for a period of one semester (13 weeks).

REFERENCES

[1] Oluwatayo, J. A. (2012). Assessment Of Computer Literacy Of Secondary School Teachers In Ekiti State, Nigeria. Journal of International Education Research 8(2), 97-104.
[2] Qian, Y., & Lehman, J. D. (2016). Correlates of Success in Introductory Programming: A Study with Middle School Students. Journal of Education and Learning; Vol. 5, No. 2, 73-83.
[3] Daly, C., & Waldron, J. (2002). Introductory Programming, Problem Solving and Computer Assisted Assessment. 6th CAA Conference, Loughborough, Loughborough University.
[4] Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. ITiCSE '14 (pp. 39-44). New York: ACM.
[5] Chilana, P. K., Alcock, C., Dembla, S., Ho, A., Hurst, A., Armstrong, B., & Guo, P. J. (2015). Perceptions of Non-CS Majors in Intro Programming: The Rise of the Conversational Programmer. VLHCC, Atlanta, Georgia, USA.
[6] Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. SIGCSE Bulletin, 39(2), 32-36.
[7] Baldwin, L., & Kuljis, J. (2001). Learning Programming Using Program Visualization Techniques. Proceedings of the 34th Hawaii International Conference on System Sciences – 2001. Hawaii: IEEE Computer Society Digital.
[8] Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. the 17th Annual Workshop og the Psychology of Programming Interest Group (pp. 293-304). Brighton, UK: University of Sussex.
[9] Ali, A., & Mensch, S. (2009). Issues and Challenges for Selecting a Programming Language in a Technology Update Course. ISEJ 7 (85), 1-10.
[10] Bennedsen, J., & Caspersen, M. E. (2005). Revealing the Programming Process. SIGCSE '05 (pp. 186-190). St. Louis, Missouri, USA: ACM.
[11] Mohorovics, S., & Stric, V. (2011). An Overview of Computer Programming Teaching Methods. CECSI 2011. Croatia: CECSI.
[12] Al-Imamy, S., Alizadeh, J., & Nour, M. A. (2006). On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. Journal of Information Technology Education vol. 5, 271-283.
[13] Burther M., & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. ICT: Providing choices for learners and learning Proceedings ascilite Singapore 2007 (pp. 99-107). Singapore: ascilite.
[14] Gomes, A., & Mendes, A. J. (2007). An environment to improve programming education. CompSysTech '07 (pp. 1-6). ResearchGate.
[15] Whittinghill, D. M. (2014). Improving the Affective Element in Introductory Programming Coursework for the "Non-Programmer" Student. 121st ASEE Annual Conference and Exposition. Indianapolis: American Society for Engineering Education,2014.
[16] Carlisl, M. C., T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving. SIGCSE ’05. St. Louis Missouri, USA: ACM.
[17] Grandell, L., PeltoMäki, M., Back, R.-J., & Salakoski, T. (2006). Why Complicate Things? Introducing Programming in High School Using Python. ACE2006. Tasmania, Australia: Australian Computer Society, Inc.
[18] Nuutila, E., Torma, S., & Malm, i. L. (2005). PBL and Computer Programming - The Seven Steps Method with Adaptations. Computer Science Education, Vol. 15 (2), 123-142.
[19] Zacharis, N. Z. (2011). Measuring the Effects of Virtual Pair Programming in an Introductory Programming Java Course. IEEE Transactions on Education, Vol. 54 (1), 168-170.
[20] Merrick, K. E. (2010). An Empirical Evaluation of Puzzle-Based Learning as an Interest Approach for Teaching Introductory Computer Science. IEEE Transactions on Education, Vol. 53 (4), 677-680.
[21] Sung, K., Hillyard, C., Angotti, R. L., & Panitz, M. W. (2010). Game-Themed Programming Assignment Modules: A Pathway for Gradual Integration of Gaming Context Into Existing Introductory Programming Courses. IEEETransactions on Education, Vol PP (99), 1-12.
[22] Smith, G., & Fidge, C. (2008). On the efficacy of prerecorded lectures for teaching introductory programming. Proceedings of the tenth conference on Australasian computing education, Vol. 78, 129-136.
[23] Russell, J., Russell, B., Pollacia, L. F., & Tastle, W. J. (2010). A Study of the Programming Languages Used in Information Systems and in Computer Science Curricula. ISEDI 8 (56), 1-15.

DOI: http://dx.doi.org/10.24018/ejers.2018.3.3.637
[24] Rogerson, C., & Scott, E. (2010). The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment. Journal of Information Technology Education 9, 147-171.

[25] Tatnall, A., & Davey, B. (2004). Streams in the History of Computer Education in Australia: An Overview of School and University Computer Education. In: Impagliazzo J., Lee J.A.N. (eds) History of Computing in Education (pp. 1-8). Springer, Boston, MA: IFIP vol 145.

[26] Moursund, D. (2017, February 13). History of Computers in Education. Retrieved April 15, 2017, from iae-pedia: http://iae-pedia.org/History_of_Computers_in_Education

[27] Kurki-Suonio, R. (2003). Birth of Computer Science Education and Research in Finland. In: Bubenko J., Impagliazzo J., Sovberg A. (eds) History of Nordic Computing. HiNC (pp. 111-121). Springer, Boston, MA: IFIP vol 174.

[28] Ferguson, R. C., Leidig, P. M., & Reynolds, J. H. (2015). Including a Programming Course in General Education: Are We Doing Enough? ISEDJ 13 (3), 34-42.

[29] Jegede, P. O., & Owolabi, J. A. (2003). Computer Education in Nigerian Secondary Schools: Gaps Between Policy and Practice. Meridian: A Middle School Computer Technologies Journal 6(2).

[30] Babb, J., Herbert E. Longenecker, J., Baugh, J., & Feinstein, D. (2014). Confronting the Issues of Programming In Information Systems Curricula: The Goal is Success. ISEDJ 12 (1), 42-72.

[31] Staubitz, T., Klement, H., Renz, J., Teusner, R., & Meinel, C. (2015). Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses. 4th Annual IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 10-12 December 2015. Zhuhai: IEEE.

[32] Cortina, T. J. (2007). An Introduction to Computer Science for Non-Majors Using Principles of Computation. SIGCSE’07 (pp. 218-222). Covington, Kentucky, USA: ACM.

[33] Soares, A. (2014). Reflections on Teaching App Inventor for Non-Beginner Programmers: Issues, Challenges and Opportunities. ISEDJ 12 (4), 56-65.

[34] Shivers, O. (2008). Why Teach Programming Languages. 2008 SIGPLAN Workshop on Programming Language Curriculum (pp. 1-3). MA, USA: ACM.

[35] Hughes, M. (2015, December 2). 4 Reasons Why You Shouldn’t Learn to Code from Codecademy. Retrieved April 21, 2017, from MakeUseOf: http://www.makeuseof.com/tag/4-reasons-shouldnt-learn-code-codeacademy/

[36] Wolfram, S. (2016, September 7). How to Teach Computational Thinking. Retrieved 3 21, 2017, from Stephen Wolfram Blog: http://blog.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/

[37] Guo, P. J. (2013). Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. SIGCSE’13. Denver, Colorado, USA: ACM.

[38] Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D Tool for Introductory Programming Concepts. Journal of Computing Science in Colleges 15(5), 107-116.

[39] Jiang, Z., Fernandez, E. B., & Cheng, L. (2011). P2N: A Pedagogical Pattern for Teaching Computer Programming to Non-CS Majors. 38th Conference on Pattern Languages of Programs (2011). Portland: ACM.

[40] Gerdes, A., Jeuring, J., & Heeren, B. (2012). An Interactive Functional Programming Tutor. ITICSE’12. Haifa, Israel: ACM.

[41] Chen, W., Li, X., & Liu, W. (2011). Teaching Computer Programming to Non-computer Science Students. The Third Asian Conference on Education 2011 Official Proceedings (pp. 784-795). Osaka, Japan: iafor.

[42] Grunewald, F., Meinel, C., Totsch, M. & Willems, C. (2013). Designing MOOCs for the Support of Multiple Learning Styles: Scaling up Learning for Sustained Impact (pp. 371-382). Springer.