European Journal of Engineering and Technology Research
Vol 1 | Issue CIE | December 2023
ISSN 2736-576X

RESEARCH ARTICLE

Assessment of Data Representation in Scratch Via the SOLO Taxonomy

Anastasios Ladias[1,*], Theodoros Karvounidis[2], Dimitrios Ladias[3], and Christos Douligeris[4]

ABSTRACT

The work addresses the significance of robotics in education, emphasizing its role in enhancing STEM skills through programming and sensory feedback. Scratch, a multimedia programming environment, is highlighted as a tool for robotic projects. Within Scratch, this work discusses data representation, distinguishing between visible and transparent data. The current work focuses on the visible data. Variables in Scratch are made tangible, helping users understand their function. Developers define their own data, such as values, variables, lists, and call parameters, while Scratch also provides system data. This system data can be numeric, alphanumeric, or logical, and its representation in code varies. To evaluate how data are used in Scratch by novice programmers, this work also proposes an evaluation framework using the Structure of the Observed Learning Outcome (SOLO) taxonomy. This evaluation framework can be used by the teacher as a tool either to evaluate with measurable criteria the students' code (on issues related to the way the data indicates their presence in Scratch) or to develop their personal teaching paths, thus creating mental scaffolds that assist students to master new knowledge.

Keywords: Assessment, Data representation, Scratch, SOLO Taxonomy.

Submitted: December 04, 2023
Published: December 29, 2023

10.24018/ejeng.2023.1.CIE.3134

[1] *Former Bureau of School Directors, Ministry of Education, Greece.*
[2] *Department of Informatics, University of Piraeus, Piraeus, Greece.*
[3] *Department of Informatics, National and Kapodistrian University of Athens, Greece.*
[4] *Department of Informatics, University of Piraeus, Greece.*

* ***Corresponding Author:***
e-mail: ladiastas@gmail.com

1. Introduction

The use of robotics in education enables pupils to enhance their skills in a variety of disciplines, including STEM [1]. These skills include math physics, engineering, programming, and beyond and they are essential in a world where technology is progressing rapidly. By programming a robot, the students are able to receive additional sensory feedback from the robot's enactment of digital codes, thus elevating the learning process from an abstract concept into something real [2]. One may not find a scientific framework with measurable evaluation criteria to assess Educational Robotics works that use Scratch. Even though there are various evaluation frameworks for the qualitative and quantitative assessment of other type of programming languages [3], [4]. In the context of the Panhellenic Educational Robotics Competition for Primary School of WRO-Hellas [5], in which Scratch is used, it is found that there is a "lack of measurable evaluation criteria. These observations led to the development of a long-term research to find measurable criteria evaluation first in the code and then in the construction of the projects". This research attempts to cover three axes of the robotic structures (see Fig. 1):

(a) the movement mechanism
(b) the energy the robot uses
(c) the control of the robot's mechanism [6].

The research focuses on the field of control via computer. In this area, a framework for evaluating visual programming using tile codes is about to be completed. This framework considers both the anatomy of the code and its functionality. In the field of code anatomy, a program consists of algorithm plus data. The study of the algorithms as functionality in Scratch has been developed in a series of scientific papers [7].

The research on the data attempts to formulate a proposal for assessing how these are being used by students in the Scratch programming environment. The factors that are been taken into consideration when evaluating the use of data are: their form, the types and symbolism of their content, the ways of their use, their carriers, their representation, the organization of simple and multimedia data, their processing, and their naming. In this paper, we examine the ways that data are being represented in the Scratch programming environment. Data in Scratch can exist (a) inside the program either within commands or as mediators in the program flow, (b) in the "code" folder,

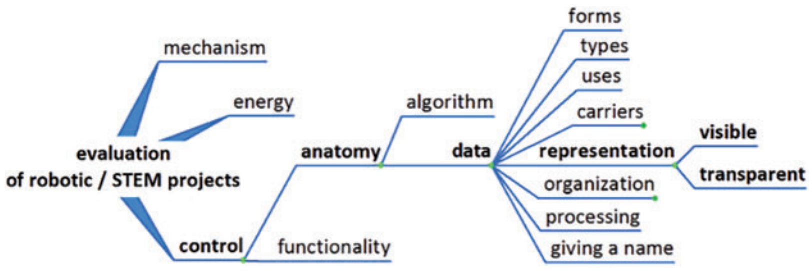

Fig. 1. The research framework in data representation in Scratch.

and (c) on the stage as "display variables". Data in Scratch based on the way that are being represented are distinguished into "classic data" (visible) and "transparent" data (Fig. 1). In the current work we deal with the visible data.

To study, how novice programmers understand programming data, we adopted the Structure of the Observed Learning Outcome (SOLO) taxonomy proposed by Biggs and Collis [8] as a promising educational taxonomy [9]–[11]. Research findings have demonstrated that SOLO taxonomy is an efficient framework for studying students' representations of programming concepts and studying their development in programming [12].

Based on all aforementioned the rest of the document is organized as follows: The next section is a thorough investigation on the data in the Scratch environment. It starts with a brief description of the Scratch programming Environment itself and the data in it, followed by a description of data visibility, the developer defined data, and system data and how they are represented. It continues with the properties of objects as data, the coordinates of the clones as undefined data, and the scenarios threads with "broadcast message and wait" messages. Hierarchical and network navigation structures using backdrops and clothing embedded in multimedia recordings as data are then provided. The first part concludes with the use of data as program flow control, a transparent data structure during a recursive procedure call, and the prioritization in the execution of pseudo-parallel scenarios. The second part starts with a brief outline of the SOLO taxonomy description. Based on the SOLO it follows the assessment of the data, categorized as visible and transparent. The work concludes in Section IX.

2. The Scratch Programming Environment

Scratch is an educational multimedia programming environment in which the novice developer can engage in digital projects by creating animations, simulations, and games [13]. One key feature of Scratch that enhances its educational value is its ability to make variables tactile and understandable. As learners handle these variables, they gain a clearer understanding of their functional role within programming. Additionally, while Scratch allows developers to define their unique sets of data–encompassing values, variables, lists, and call parameters–the platform also has inbuilt system data. In the Scratch environment, the code consists of fragments-parts called scenarios, which are distributed in "objects with properties to which they can assign behaviors by specifying events and actions" [14]. In Scratch, a program is understood as a set of scenarios that run in all objects and backgrounds. Making use of the metaphor that parallels the development of a project in Scratch with the staging of a shadow theatre play, the scenarios are those that control the behavior of objects corresponding to the Greek traditional cartoon "karagiozis" (Fig. 2).

The shadows/projections of the figures/objects on the screen correspond to the roles/sprites in the Scratch stage. Each projected onto the stage role occupies a layer, thus creating overlaid layers.

3. The Data in the Scratch Environment

Maloney *et al*. [13] state that in most text-based programming languages, variables are invisible, abstract, and unintelligible but Scratch turns variables into concrete objects that the user can see and manipulate, making them more understandable through editing and observation. In Scratch, a variable, which exists in the "Code" folder in the "Variables" field (left in Fig. 3), can be displayed as a "variable display" in the stage (right in Fig. 3).

These "variable displays" in the stage allow the users to see the result of using certain commands during program execution such as "change x by 1", helping them to build a mental picture of how the values of variables evolve and ultimately how the variables work.

In addition, the "variables display" in the stage are useful either as indicators (e.g., for displaying the score in a game) or as adjustments of a variable via a slider display during a program execution. An example of the benefits of using the "variable display" in Scratch is the execution of the "network classification" program [15], as shown in Fig. 4.

4. Data Visibility by the Developer and the User

The existence of a piece of data in Scratch does not imply its clear visibility to the developer and the user of the program. The representation of this piece of data can vary from being entirely visible and manageable ("classical" data) to being undefined and "transparent" (also called "control" data).

This "visibility" is determined on the one hand by how a piece of data justifies its presence in Scratch and on the other hand by how the developer perceives it. Hence, there are cases where a less experienced developer may look at visible data but not see (perceive) their existence and there exist cases where an experienced programmer (developer)

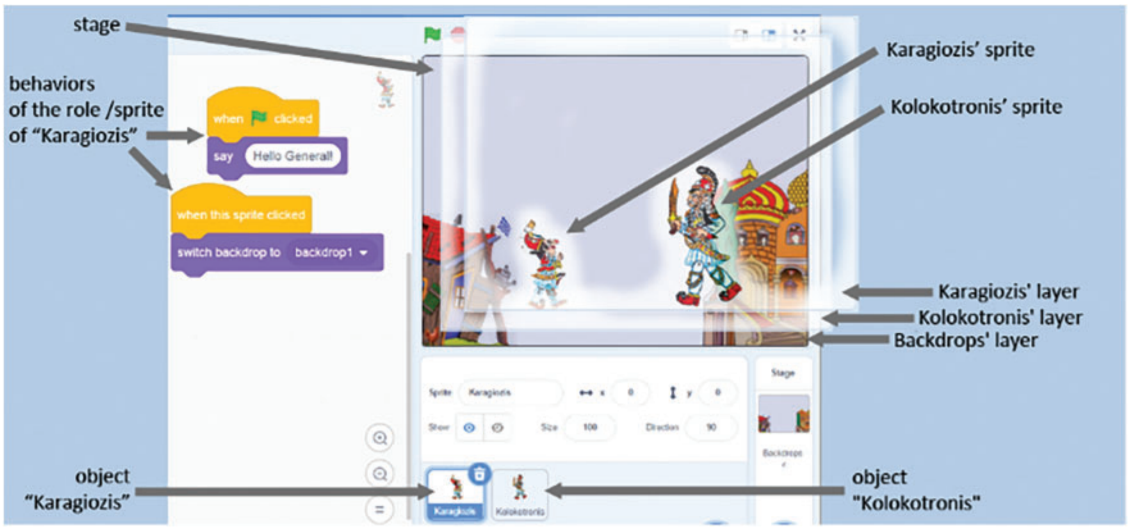

Fig. 2. The Scratch environment with the objects, their projection on the stage layers and the programming scripts that define their behaviors.

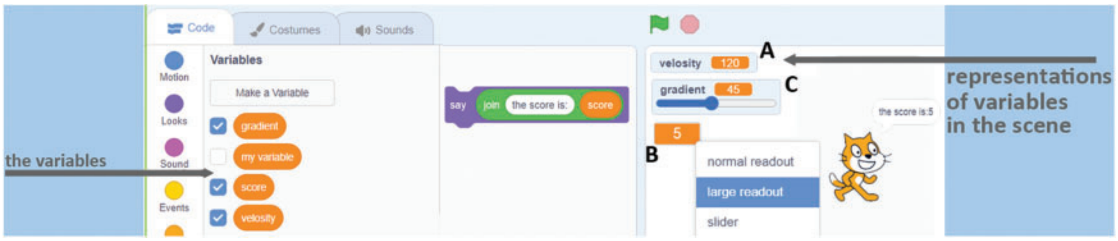

Fig. 3. In Scratch, the display of a variable in the stage can appear as (A) a "normal text view" e.g., the variable "velocity", (B) a "large text view" e.g., the variable "score", and (C) a "slider" e.g., the variable "gradient".

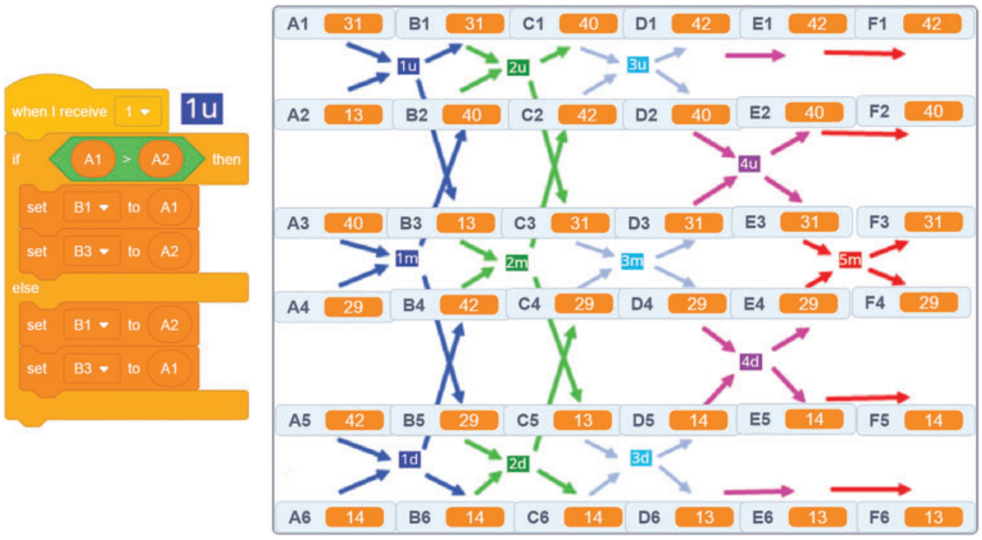
```
when I receive 1        1u
if A1 > A2 then
    set B1 to A1
    set B3 to A2
else
    set B1 to A2
    set B3 to A1
```
A1 31 B1 31 C1 40 D1 42 E1 42 F1 42
1u 2u 3u
A2 13 B2 40 C2 42 D2 40 E2 40 F2 40
4u
A3 40 B3 13 C3 31 D3 31 E3 31 F3 31
1m 2m 3m 5m
A4 29 B4 42 C4 29 D4 29 E4 29 F4 29
4d
A5 42 B5 29 C5 13 D5 14 E5 14 F5 14
1d 2d 3d
A6 14 B6 14 C6 14 D6 13 E6 13 F6 13

Fig. 4. Presentation of variable values in the stage, via "variable display", which shows the changes in variable values when performing a "network classification" are shown.

can perceive the existence of a non-visible and implicit data structure.

The "visible" or "classic" data include the input, the output, and the intermediate values resulting from calculations during the program execution. This data is visible to the programmer and is displayed in Scratch in three areas: (a) in the "Code" folder (Fig. 5 to all area A's), (b) within the program instructions (Fig. 5, area B), and (c) in the stage during program execution (Fig. 5, area C). In the case of displaying the data in the stage, the data is also visible to the user/viewer during program execution. In the case where the data concerns the communication between the scenarios of the program (Fig. 5, area D) these are not displayed; they are invisible and they are classified as "transparent". Transparent data are typically used as program flow control data.

5. The Developer-Defined Data (DDD)

The Developer Defined Data (DDD) are the most visible/discernible data and this is aimed by the fact that it is created after a conscious choice by the developer to meet a

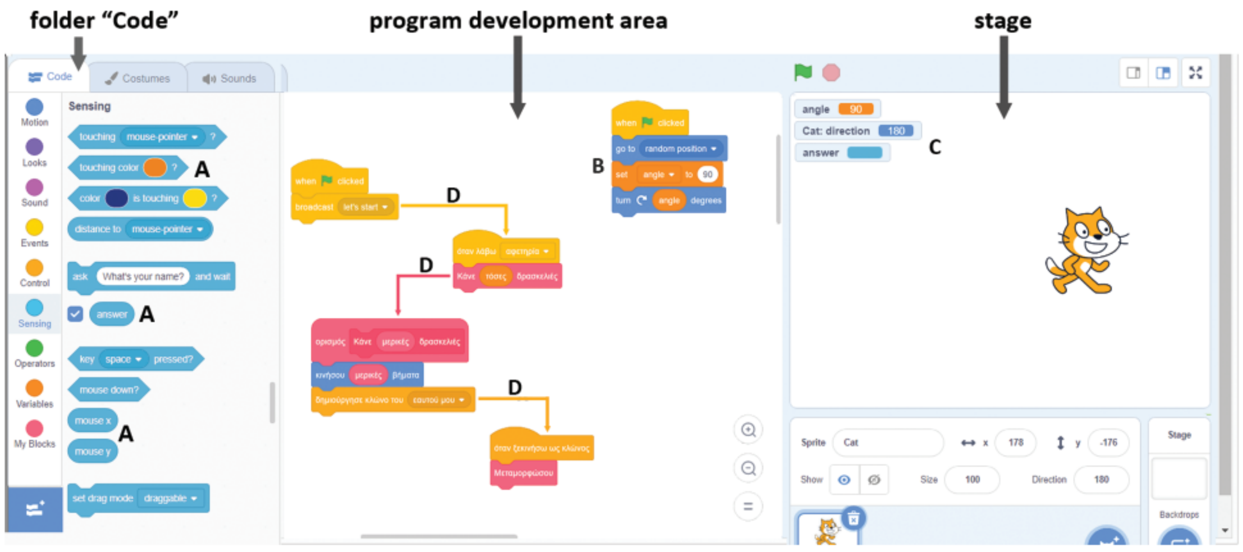

Fig. 5. The places where "classical" data can be found in the Scratch environment.

programming need. The DDDs are (a) values, (b) variables, and lists, and (c) call parameters to procedures.

The values set by the developer in the program commands are considered as data embedded in the code.

The variables and the lists appear in the "Code" folder in the "Variables" field (Fig. 6A), within the program commands and in the stage. It should be noted that only for these variables and lists within this category there is a possibility to be or not to be visible in the stage; these variables can then be visible when they are within commands as shown in Fig. 6B with the variable "this", apart from checking them in the "Code" folder. Variables and lists in Scratch when they use the option of "variables display" in Scratch, they can act as indicators (e.g., for the display of the score in a game or the time evolution of a size). In addition, the user can adjust the variable value using the slider during program execution (Fig. 3C).

The call parameters to procedures are declared by the developer when the procedures are defined (Fig. 6, area C) and they are activated when the procedure is called by a scenario during the program execution. They, therefore, only appear in the program and they act as variables only during the procedure call and execution (Fig. 6, area D).

6. System Data (SD)

The "System Data" (SD) is a second category of ready-to-use data provided by Scratch to the developer. SD are located in the "Codes" folder in the objects and background. The SD can be either numeric, alphanumeric, or logical type. Moreover, for the SD there is the option of either ticking to be displayed as a "variable display" (D+VD) or they are without the option of "variable display" (D−VD). In particular, a numeric or alphanumeric type SD, which has an oval shape in Fig. 7 may be D+VD or D−VD while all of the logical type SD (which have a hexagonal shape in Fig. 7) are D−VD. There is no option using commands to make SD appear as "variable representations" in the stage.

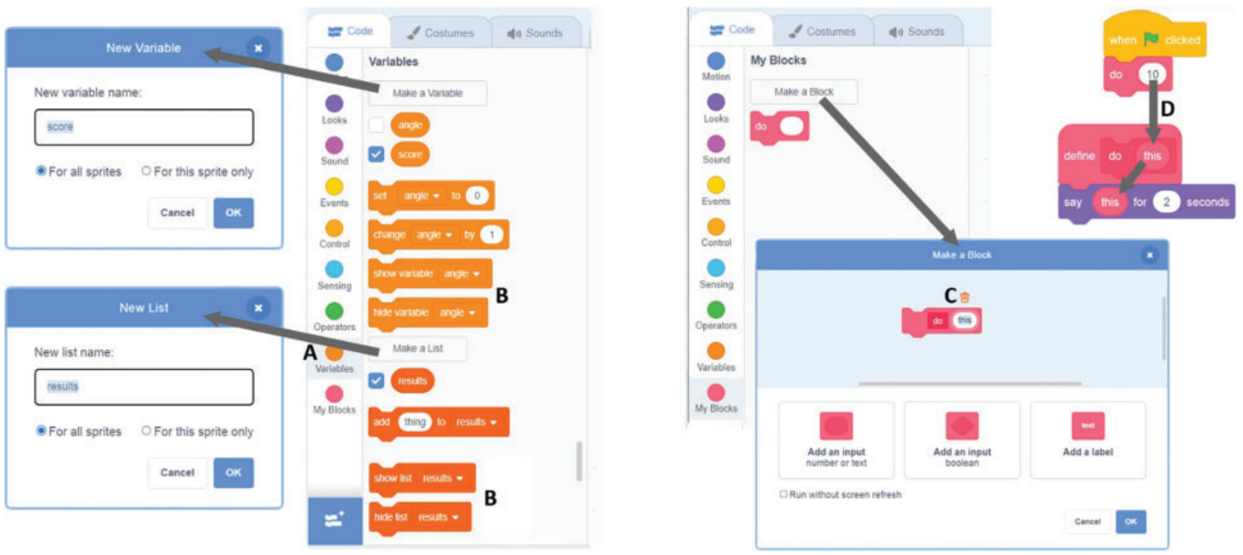

Fig. 6. Areas where the data defined by the developer are displayed.

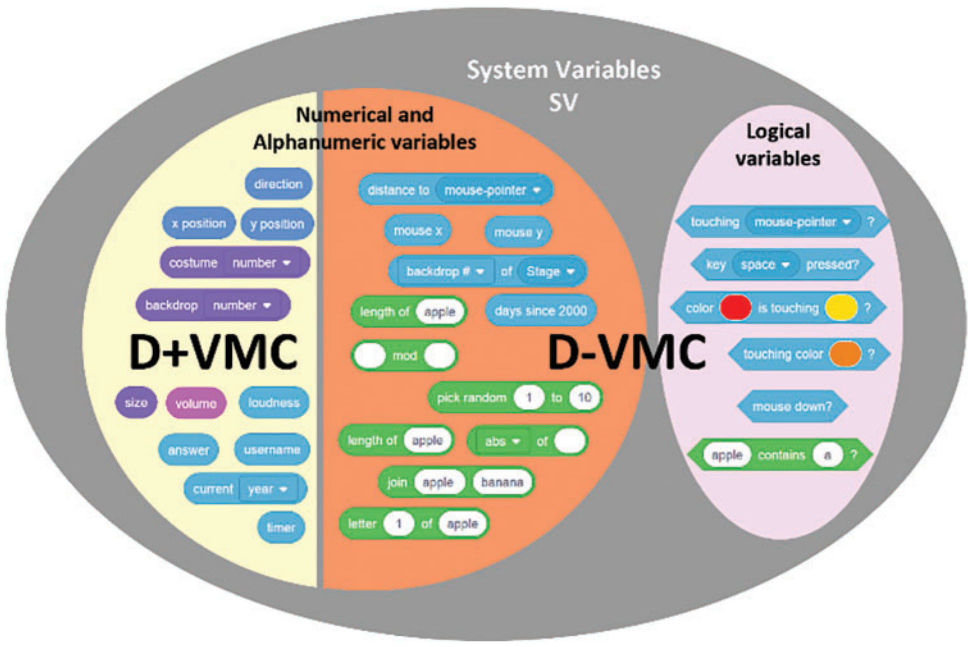

Fig. 7. Indicative presentation of the system data. Their categorization depends on whether they can be displayed (D+VD) or not on the stage (D−VD).

7. Representation of the System Data in the Code

In the commands “set effect (parameter = color) to (value)”, “when (parameter = intensity) > (value)”, “set (parameter = variable) to (value)”, the data is the “value” and the “parameter”. Based on the above, two additional ways of distinguishing system data can be derived: (a) their value domain which can be closed or open, and (b) their ability to change their value during the program execution.

Combining the two previous ways of distinguishing system data, we obtain the table of Fig. 8. We observe that there are data in the commands that remain constant during the program execution; these data are displayed in a rectangular frame with rounded corners (Figs. 8A and 8C). There are also data that can be changed as a result of the program execution and are depicted inside an oval box (Figs. 8B and 8D). Moreover, the data whose value domain is closed (and predetermined by the system) are shown in A and B areas of Fig. 8, while those data whose value domain can (in various ways) be extended are shown in the areas of C and D areas of Fig. 8.

In Fig. 8 grey areas indicate correlations between “neighboring” codes. Thus, the grey area A1-B1 shows the two ways that an event is detected-on the left by an interrupt technique and on the right by a polling technique [16]. The grey area B1 shows a comparison between codes where the left one receives a value from a finite value list (the left command within the B1 cell), while the right one receives a value through a variable whose value is derived from the previous finite value list of Greek characters on the keyboard. However, as shown in the codes within the grey area B1-D5, there is a way to bypass the finite value list of values from which the variable takes values and to extend the values of this list, e.g., to the Greek characters that are available on the keyboard. Similarly, in the grey areas C1−D1 and C3−D3 a comparison is made between codes where the left code fetches a value from a finite value list via an interrupt technique, while the right code fetches a value via a polling technique either through the previous finite value list or through a variable.

8. Assessment of Data

The assessment of programming code in education ensures that students are not just coding, but are thinking, understanding, and growing as budding software developers and computer scientists. The assessment of data, in particular, is also important. Data are more than just placeholders for values in code. They represent a coder’s ability to think logically, efficiently, and clearly. By assessing their use in education, instructors can ensure students are building a strong foundation for all future coding endeavors. For this reason, as mentioned in the Introduction, we adopted the SOLO.

8.1. The SOLO Taxonomy

The SOLO taxonomy is the tool that will be used to evaluate how data is used in Scratch code. The Structure of Observed Learning Outcomes (SOLO) taxonomy of [8] proposes the assessment of knowledge based on the structure of the observed learning outcome, classifying these outcomes into five hierarchical levels:

1) *The pre-structural level* or use of unrelated and disorganized information that does not make sense.

2) The *mono-structural level*, where a limited perspective is observed–mainly one element or aspect is used or emphasized–while the other components are omitted, and no significant connections are made between the parts.

3) The *multi-structural level*, in which there is a multipoint perspective–several relevant elements or aspects are used or recognized–but there are no significant connections and a complete picture has not yet been formed.

4) The *relational level*, in which there is a holistic perspective where the meta–connections between the parts are

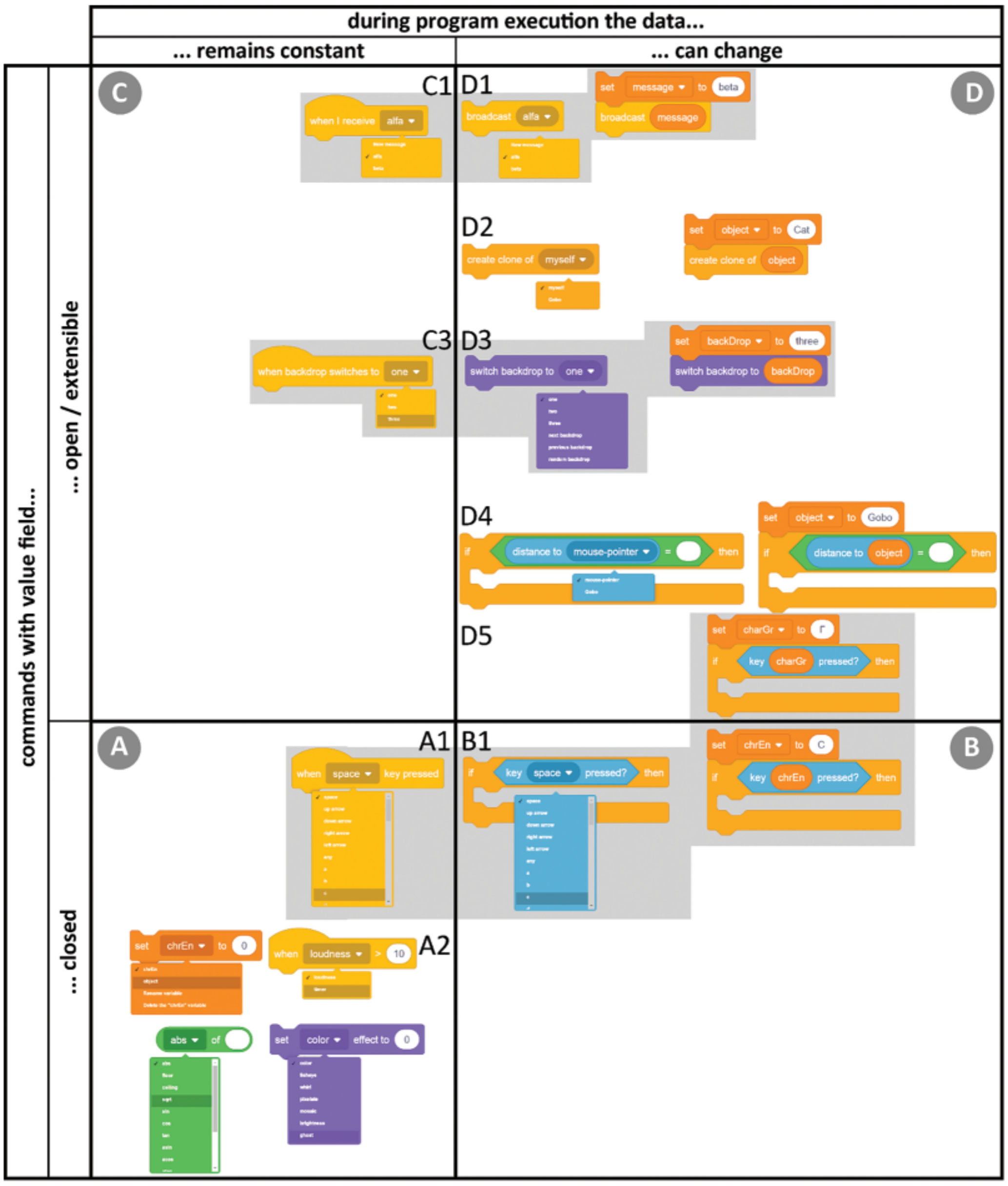

Fig. 8. System data with predefined value domain (A, B) or with expandable value domain (C, D) and variables that during the program remain either constant (A, C) or can be changed (B, D).

realized and the importance of the parts in relation to the whole is demonstrated and appreciated and

5) The *level of extended abstract*, in which in addition to the features of the previous relational level, the content is treated as an instance of a more general case.

8.2. Assessment of "Visible" Data With the SOLO Taxonomy

Taking into account what has been mentioned in the paragraphs "Developer Defined Data" (DDD) and "System Data" (SD) we create Table I. The criteria by which the data are classified in terms of their representation are, on the one hand, the way the developer and the user perceive them and, on the other hand, the way they are represented in the Scratch environment.

An assessment of the representation of data in the Scratch environment with the SOLO taxonomy can be based on the table summary of Table I. The following observations emerge from this table:

- The values appear only inside the commands in the program and may not be recognized as data by non-experienced programmers.
- As the DDD creation is a conscious choice of the developer, they result in more intense mental representations compared to the existing SD. We believe this justifies why DDD is being considered more observable by the developer compared to SD.
- Simple variables are the only ones that provide the user/viewer of the program with the ability to interact through the "slider display" of variables during the program execution.
- The DDD variables (simple and lists) are the only ones that can appear/disappear in the stage via commands during program execution.
- The procedures' parameters are the only visible data that "live" temporarily and for as long as the execution of the process lasts.

TABLE I: Possibilities of Data Representation Depending on Where They Appear in Scratch and as a Function of the Mode (Visible or Transparent)

Representation of the data					In terms of space					SOLO taxonomy level
					On stage as "display variables"			In the "code" folder	In the program (inside the commands)	
					Without slider	With slider	Using commands			
In display terms	Visible	Developer-defined data (DDD)	Value						X	Mono-structural
			Variable	Simple	X	X	X	X	X	Extended abstract
				List	X		X	X	X	Relational
			Procedure parameters						X	Relational
		System data	D+VD		X			X	X	Multi-structural
			D−VD					X	X	Mono-structural

- As the System Data are not declared by the programmer as variables, inexperienced programmers do not realize that they are data; they often consider them as parts of the code. Such a view indicates that some programming instructors do not take advantage of teaching selection statements using logical type system variables (e.g., if mouse down) without requiring them to use DDD for the statements' logical conditions (if my variable > 0).

Based on the foregoing and with the criterion of the degree to which the data becomes visible, the following classification of the representation of the visible data in the levels of the SOLO taxonomy is proposed:

- At the *level of the extended abstract*, the simple variables of the DDD (Developer Defined Data) are ranked as those data with the most representations.
- At the *relational level*, the lists and parameters of the DDD procedures are included as conscious creations of the programmer.
- At the *multi-structural level* are classified as the Boards that can appear on stage.
- At the *mono-structural level*, the values that cannot be displayed on the stage (D−VD) are included as the least visible from the DDD and SD (System Data).

A more in-depth assessment of the System Data using the SOLO taxonomy can be based on the table in Fig. 8 of the "Display of the System Data in the Code" paragraph. Based on the criteria of the value fields (opened–closed) and the possibility of changing the parameters of the commands during the execution of the program, it following classification is proposed for the representation of the System Data at the levels of the SOLO taxonomy:

- The *mono-structural level* includes the parameters-data of the commands with a closed value domain, which remains constant during program execution (Fig. 8A).
- The *multi-structural level* includes the parameters-data of the commands with an open and extensible values domain which remain constant during the execution of the program (Fig. 8C).
- The *relational level* includes the parameters-data of the commands with a closed value field which can be changed during the program execution (Fig. 8B).
- The *level of extended abstract* includes the data parameters of the commands with an open and extensible value field that can be changed during the execution of the program (Fig. 8D).

9. Conclusion

Within the Scratch programming environment, data representation emerges as a significant aspect of the learning curve. The text draws a distinction between data that is visible to the user and that which remains "transparent" or hidden in the backdrop. One key feature of Scratch that enhances its educational value is its ability to make variables tactile and understandable. As learners manipulate these variables, they gain a clearer understanding of their functional role within programming. Additionally, while Scratch allows developers to define their unique sets of data encompassing values, variables, lists, and call parameters–the platform also has an inbuilt system data.

This work is part of a larger research on data assessment in Scratch. Other parts, beyond the present research on data representation, are being developed in parallel and relate to the formats, types, and symbolization of their content, ways of using them, their operators, the organization of simple and multimedia data, and their processing and naming. The data research belongs to the subcategory dealing with the anatomy of code, which in turn is part of a broader framework that attempts to define the specifications for the evaluation of STEM educational projects in the context of the WRO-Hellas National Educational Robotics Competition.

The assessment of expected learning outcomes with the SOLO taxonomy related to the representations of visibilities considered in this paper can be tools for educators. These tools can be used both to assess the code developed by students in Scratch but also to formulate guidelines regarding the order of instruction by designing their own personal teaching paths. Code evaluation can be done by identifying the features of a student's code data and ranking them into specific levels of the SOLO taxonomy, thus providing measurable data on these features.

Acknowledgment

This work has been partially supported by UPRC (University of Piraeus Research Center) and by COSMOTE through a PEDION24 grant.

Conflict of Interest

Authors declare that they do not have any conflict of interest.

References

[1] Ching Y-H, Yang D, Wang S, Baek Y, Swanson S, Chittoori B. Elementary school student development of STEM attitudes and perceived learning in a STEM integrated robotics curriculum. *TechTrends*. April 2019;63:590–601. doi: 10.1007/s11528-019-00388-0.

[2] Ching Y-H, Hsu Y-C, Baldwin S. Developing computational thinking with educational technologies for young learners. *TechTrends*. April 2018;62(6):563–73. doi: 10.1007/s11528-018-0292-7.

[3] Friedman MA, Voas JM. *Software Assessment: reliability, Safety, Testability*. John Wiley & Sons, Inc; 1995.

[4] Jackson M, Crouch S, Baxter R. Software evaluation: criteria-based assessment. Software sustainability institute, The university of Edinburgh. 2011. (last accessed 1 October 2021). Available from: http://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf.

[5] Somalakidis I, Ladias A. The what, the how and the why of the panhellenic educational robotics competition for the primary school of WRO Hellas. *Erkyna, Rev Educ-Sci Issues*. 2021;Issue 23:84–108, Last accessed: 31January 2023. Available from: https://erkyna.gr/e_docs/periodiko/dimosieyseis/pliroforiki/t23-06.pdf.

[6] Ladias A, Georgopoulos K, Souvatzoglou G, Ladias D. Proposal for an educational robotics & STEM evaluation framework with the SOLO taxonomy. *Proceedings of the 3rd Panhellenic Conference: open Educational Resources and E-Learning*. Florina-GR, October 3–4, 2020.

[7] Ladias A, Karvounidis T, Bellou I. Curriculum proposal for computer programming within STEM. *Proceedings of the Panhellenic Conference: curricula in an Ever-Changing World, PAPEDE*. Athens-GR, 1–3 November 2020.

[8] Biggs JB, Collis KF. *Evaluating the Quality of Learning. The SOLO Taxonomy*. NY: Academic Press; 1982.

[9] Lister R, Fidge C, Teague D. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bull*. 2009;41(3):161–5.

[10] Thomson E. Holistic assessment criteria: applying SOLO to programming projects. *Proceedings of the 9th Australian Computer Society*, pp. 155–62, Ballarat, Victoria: Australian Computer Society; 2007.

[11] Jimoyiannis A. Using SOLO taxonomy to explore students' mental models of the programming variable and the assignment statement. *Themes Sci Technol Educ*. 2011;4(2):53–74.

[12] Resnick M, Maloney J, Monroy-Hernández A, Rusk N, Eastmond E, Brennan K, *et al.* Scratch: programming for all. *Commun Acm*. 2009;52(11):60–7.

[13] Maloney J, Resnick M, Rusk N, Silverman B, Eastmond E. The scratch program-ming language and environment. *ACM Trans Comput Educ*. November 2010;10(4). doi: 10.1145/1868358.1868363. Article 16, pp. 1–15.

[14] Mavrochalivides G, Makris G, Bekos N. Didactic approach to object oriented programming with scratch. *Proceedings of the 6th Panhellenic Informatics Teaching Conference*. Florina-GR, 20–22 April 2012. Last accessed March 20 2023. Available from: https://docplayer.gr/9680425-Didaktiki-proseggisi-toy-antikeimenostrafoys-programmatismoy-me-to-scratch.html.

[15] Bell T, Witten I-H, Fellows T. Computer science unplugged... offline activities and games for all ages. 2010. Last accessed: March 20–2023. Available from: https://jmvidal.cse.sc.edu/library/bell98a.pdf.

[16] Ladias A, Ladias D, Karvounidis T. Categorization of requests detecting in scratch using the SOLO taxonomy. *Proceedings of the 4th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. Piraeus-GR, 20–22 September 2019. doi: 10.1109/SEEDA-CECNSM.2019.8908438.