MODESTUM
OPEN ACCESS

EURASIA Journal of Mathematics, Science and Technology Education, 2020, 16(5), em1841
ISSN:1305-8223 (online)
Research Paper https://doi.org/10.29333/ejmste/116600

Development and Validation of the Middle Grades Computer Science Concept Inventory (MG-CSCI) Assessment

Arif Rachmatullah [1], Bita Akram [1], Danielle Boulden [1], Bradford Mott [1], Kristy Boyer [2], James Lester [1], Eric Wiebe [1*]

[1] North Carolina State University, USA
[2] University of Florida, USA

Received 17 October 2019 ▪ Accepted 14 January 2020

Abstract
The increasing interest in computer science (CS) and CS-integrated STEM teaching and learning has created a need for assessment instruments that can be used to evaluate the efficacy of innovative instructional approaches to K-12 CS education. However, there is a lack of validated assessment tools aligned to core CS concepts for younger students. This paper reports on the development and validation of a CS concept assessment for middle grades (ages 11-13) students. A total of 27 multiple-choice items were developed, guided by focal knowledge, skills and abilities associated with the concepts of variables, loops, conditionals, and algorithms. These items were administered to 457 middle grades students. The items were presented in form of block-based programming code and administered in a week-long computational modeling intervention. A combination of classical test theory and item response theory approaches were used to validate the assessment. Based on results, it was found that only 24 items are considered valid and reliable items to measure CS conceptual understanding. The results also suggested that the assessment can be used as a pre and post-test to investigate students' learning gains. This work fills an important gap by providing a key resource for researchers and practitioners interested in assessing middle grades student CS conceptual understanding.

Keywords: assessment, computer science, concept inventory, middle grades, validation

INTRODUCTION

The past decade has seen an increased international policy push for more emphasis on K-12 Computer Science (CS) education (Barr & Stephenson, 2011; Cuny, 2011; Grover & Pea, 2018). In the US, a key area of focus has been on the middle grades (ages 11-13), where specific courses have been developed and research into integration strategies for core STEM curricula has been undertaken (code.org, 2018; Goode & Chapman, 2016; Manila et al., 2014; Settle et al., 2012). This policy push has resulted in bringing CS focused interventions into middle grades classrooms, but has done so without the development of effective assessment tools for use by practitioners and researchers (de Araujo, Andrade, & Guerrero, 2016; Decker & McGill, 2019).

While there are assessments associated with standalone courses, these instruments are tuned to the specific content of the corresponding course and typically reflect localized contextual factors of the curricula, including the specific programming language used or technical elements that interface with the code (e.g., robotics, gameplay elements) (Bienkowski, Snow, Rutstein, & Grover, 2015; Buffum et al., 2014). There is a significant need for a more general middle grades CS concept inventory assessment, similar to those developed for other STEM curricular areas (e.g., Hestenes, Wells, & Swackhamer, 1992) and for other age ranges in CS (Parker, Guzdial, & Engleman, 2016). Such an instrument could be used in a wide variety of ad hoc and small-scale middle grades CS interventions, and for research-based activities. We argue that such an assessment should be (1) based on the emerging student

✉ arachma@ncsu.edu ✉ bakram@ncsu.edu ✉ dmboulde@ncsu.edu ✉ bwmott@ncsu.edu ✉ kristy@learndialogue.org ✉ lester@ncsu.edu ✉ wiebe@ncsu.edu **(*Correspondence)**

Contribution to the literature

- The paper summarizes the previous literature on the CS assessments and indicates the need for a core concept-aligned assessment for use at the middle grades level (ages 11-13).
- The paper demonstrates a robust development and validation process of an instrument to measure middle grades students' understanding of computer science (CS) concepts.
- The use of a recent and well-accepted inventory of CS concepts is the focus of the assessment development.

learning standards (Computer Science Teachers Association – CSTA, 2017; k12cs.org, 2016) and thus representative of the current well-accepted CS curricular concepts of primary interest for this grade range, (2) administrable with standard testing platforms, and (3) be completed by students within a standard class period, thus the teachers can use it in their classes. Finally, assessment items should utilize block-based programming, the most popular programming representation used in middle grades (Brown, Mönig, Bau, & Weintrop, 2016). This paper provides a foundational literature review, and development and validation of the MG-CSCI, a CS concept inventory assessment for middle grades.

RELATED WORK

CS education researchers interested in analyzing the impact of prior programming experience have often resorted to asking students to self-report the level and kind of prior experience (Korkmaz, Çakir, & Özden, 2017). However, it is widely recognized that an assessment of core concepts (i.e., a concept inventory) provides more accurate, detailed information of students' prior knowledge (Smith IV, Hao, Jugodzinski, Liu, & Gupta, 2019). Such a concept inventory can also be used pre and post to measure the impact of an intervention. Taylor et al.'s (2014) review of CS concept inventories noted that the CS education community has generally lagged behind other areas in STEM education (especially physics) in the development and utilization of assessments of students' conceptual understanding. A recent large-scale effort to catalog current instruments has also concluded that research on assessment development needs to continue (Decker & McGill, 2019).

One of the challenges with the current work has been the lack of a clear construct definition for computational thinking (CT), as opposed to computer science (CS) knowledge. For the purposes of this work, CT can be thought of as a general cognitive ability linked to fluid intelligence (i.e., problem-solving ability), spatial ability, and working memory (Román-González, Pérez-González, & Jiménez-Fernández, 2017; Tukiainen & Mönkkönen, 2002; Wiebe et al., 2019). In contrast, CS conceptual understanding is linked to core concepts identified by curricular frameworks as necessary for engaging in CS practices. While there is likely high correlation between instruments that measure these two constructs, they would be used differently depending on the measurement goal.

Historically, much of the work on CS concept inventory development has focused on the undergraduate CS1 course (Gross & Powers, 2005). Caceffo, Wolfman, Booth, and Azevedo (2016) notes that typically the goal for a concept inventory for a course such as this is to develop items that are diagnostic at the single concept level, though this goal is often challenging to achieve (Luxton-Reilly et al., 2018). Because of the concept-guided nature of such an inventory, many of these types of assessments focus on a few, or even just one core concept, such as recursion (Hamouda, Edwards, Elmongui, Ernst, & Shaffer, 2017). FCS1 was developed as a concept inventory for an introductory undergraduate programming course with the goal of assessing all of the core concepts covered in the course (Tew & Guzdial, 2017). The SCS1 assessment (Parker, Guzdial & Engelman, 2016) was a revision and revalidation of the FCS1, though even through this refinement process other researchers continue to question the high difficulty level of some of the items (Xie, Davidson, Li, & Ko, 2019).

At the K-12 level, there has been work on developing assessments of student CS conceptual understanding. A prominent example is the Fairy Performance Assessment (Werner, Denner, Campe, & Kawamoto, 2012), which used coding in Alice to assess students' ability to think algorithmically and make effective use of abstraction and modeling. However, the assessment was a practicum exercise and required students to code constructions, thus limiting its scalability. Other assessments have been reported in the literature specifically associated with research-based interventions, especially at the middle grades level. This includes an assessment of a game-based learning environment (Boulden et al., 2018; Buffum et al., 2015), pair programming with Scratch (Lewis, 2011), and a teacher PD intervention with Scratch (Meerbaum-Salant, Armoni, & Ben-Ari, 2013). Due to the focused, research nature of this prior work, none of these instruments underwent further validation as a general use instrument. As formal curricula have been developed, assessments have been created matched to the curricular content and practices. These include the Exploring CS middle grades curriculum (Bienkowski et al., 2015) and the FACT curriculum (Grover, Pea, & Cooper, 2015).

Another direction has been the inclusion of nine assessment items on the Israeli nationwide exam aligned with a required middle grades CS course (Zur-Bargury, Pârv, & Lanzberg, 2013).

CURRENT WORK

The goal for the research reported here is to develop a validated concept inventory at the middle grades level that is independent of any specific curricula but is well aligned with an emerging set of student CS outcome standards for middle grades (ages 11-13). We focused on the K-12 Computer Science Framework (k12cs.org, 2016) as our main CS outcome standard. Furthermore, we utilized two sets of student outcome standards designed based on the K-12 Computer Science Framework: the CTSA framework that is designed to guide both curriculum and assessment development at the state level (CSTA, 2017) and a set of 16 FKSAs (focal knowledge, skills, and abilities) that were developed by Grover and Basu (2017) with the goal of developing CS concept assessment items. The instrument developed by Grover et al. (2017) contained items that covered multiple FKSAs and open-ended items. In contrast, we created finer-grained items focused on single concepts, which were all multiple-choice, to facilitate automated grading. In addition, this concept inventory underwent validation with a large population of students of varying prior programming experience and representative of a broad U.S. demographic.

In addition to using an evidence-centered design approach for item development, the representation of the example code in the items (e.g., text-based, pseudocode, or block-based), as both foundational cognitive affordances and prior programming experience will interact with the representational form (Werner, Denner, Campe, & Kawamoto, 2012). Research by Weintrop and colleagues (Weintrop, Killen, & Franke, 2018; Weintrop, Killen, Munzar, & Franke, 2019) has indicated that independent of individual differences, block-based representations that are independent of specialized syntax are the most accessible form for students of varying abilities.

Once items are developed, classical test theory (CTT) and item response theory (IRT) approaches, combined, have proven to be a robust approach to assessment validation. Specifically, IRT is sample-independent by virtue of a latent attribute underlying the model so that psychometric results computed through IRT are stable across different samples. CTT contributes to the dimensionality verification process and strengthens the testing performed through IRT. Moreover, IRT produces ratio-scaled scores that facilitate a more accurate score inference and comparison between item and person. Lastly, it reports a variety of fit statistics that allows for more thorough item evaluation (Bond & Fox, 2001; Boone, Staver, & Yale, 2013). A number of researchers have advocated specifically for IRT approaches to be utilized in the development of CS assessments (Werner, Denner, Campe, & Kawamoto, 2012; Winters & Payne, 2005; Zendler, 2019). In addition, utilization of both evidence-centered design for item development and Rasch modeling (a form of IRT) for validation has effectively been used for a focused CS concept inventory (only for control structures understanding) development for grades 7-10 (Mühling, Ruf, & Hubwieser, 2015), and provides an example for this research to utilize.

DATA AND METHODS

Item Development

Following an evidence-centered assessment design (ECD) approach (Bienkowski et al., 2015; Mislevy, Steinberg, & Almond, 2003), we designed a CS inventory for assessing middle grades student knowledge of essential CS constructs. ECD provides a process for identifying conceptual learning targets for students, and assessing the degree to which tasks (i.e., assessment items) provide warrantable evidence that students have mastered that concept. This methodology has been widely used in assessment development, including an assessment for the Exploring Computer Science curriculum (Goode, Chapman, & Margolis, 2012). For the MG-CSCI, we focused on the CS concepts that are commonly taught at the middle grades level and are shown to be challenging for novice programmers. Focusing on common and challenging CS concepts enables us to use the MG-CSCI assessment to evaluate students' proficiency related to the most important CS concepts and the effectiveness of an instructional intervention or curriculum in teaching them. Prior work, as noted in the previous section, was used to frame and identify the target concepts. Using an ECD process, Grover and Basu (2017) had operationalized important CS concepts for middle grades into 16 focal knowledge, skills, and abilities (FKSAs). Grover and Basu's work, in turn, was guided by the K-12 CS framework (k12cs.org, 2016). The 11 out of 16 FKSAs were used to develop the assessment and covered four primary conceptual categories: variables (3 FKSAs), loops (4 FKSAs), conditionals (1 FKSA) and algorithms (3 FKSAs). The other five FKSAs did not align with current U.S. middle grade computer science curriculum standards (CSTA, 2017) and thus were not included in this assessment development. However, in order to have comprehensive coverage of the middle grades curriculum, during early stage development, it was decided to also use definitions and examples of algorithms contained in the CSTA K-12 CS Standards (CSTA, 2017), also derived from the K-12 CS framework (k12cs.org, 2016). The list of the FKSAs and CSTA Standards used in this study is available in Appendix 1.

For item development around block-based programming code, prior work on high-school and

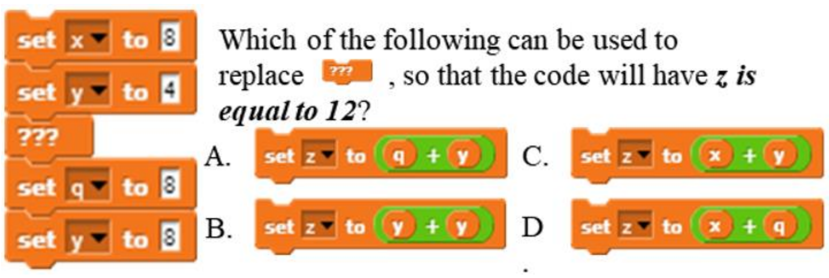

Which of the following can be used to replace ??? , so that the code will have *z is equal to 12*?

Figure 1. Item 12 Variable development type question

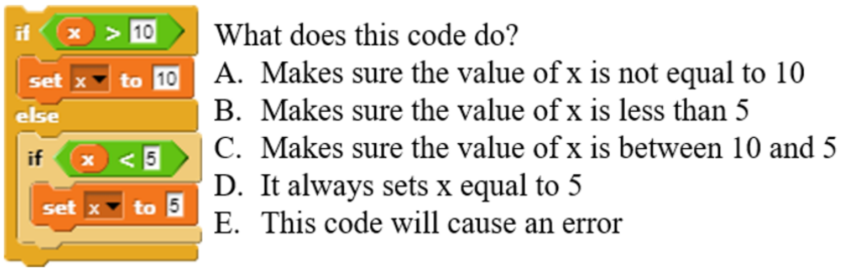

What does this code do?
A. Makes sure the value of x is not equal to 10
B. Makes sure the value of x is less than 5
C. Makes sure the value of x is between 10 and 5
D. It always sets x equal to 5
E. This code will cause an error

Figure 2. Item 16 Algorithm comprehension type question

middle-school assessments were used to guide our work (Du Boulay, 1986; Shneiderman, & Mayer, 1979, Weintrop & Wilensky, 2015; Xie et al., 2019). For each concept, we designed items with varying levels of difficulty and question types: comprehension of a code snippet, debugging a partially wrong code snippet, and developing/completing a partially built code snippet. We piloted our inventory to identify the difficulty span of the questions and refined them to maintain the appropriate balance for middle grades students. Example assessments items are a development question on variables (Figure 1) and a comprehension question on algorithms (Figure 2). After devising the first draft of the items, they were piloted with two CS graduate students and three non-CS graduate students to check for appropriate language, level of difficulty, and balance of the question types. We then revised the items based on feedback collected from this pilot study.

Sample and Administration

A total of 457 students consented to participate in the current study. The consenting students were middle grades students in sixth grade to eighth grade (ages 11-13). Sixth-grade students were 52% of the total participants and seventh and eighth grade students represented 2% and 36% of the sample, respectively. The male students were 49% of the sample. Regarding ethnicity, Caucasian, Latinx, and African-American were the most common ethnic groups in the current data set with 30%, 18%, and 17%, respectively, while the remaining 35% identified as Asian, Middle Eastern and Other. Of the total sample, 68% of students indicated that they had never experienced or were only occasionally exposed to programming activities.

Administration of the study was done in conjunction with a week-long classroom computational modeling intervention that involved block-based programming centered on science topics. Students were involved in either food web or epidemic disease modeling activities. Both activities consisted of unplugged (without computer) and plugged (with computer) sessions. The activities were developed based on the Use-Modify-Create (UMC) model of learning suggested by Lee et al. (2011). In the food web activity students were engaged in building a simulation of energy transfer in food web through coding, and in the epidemic disease activity, students were engaged in creating a simulation that shows the spread of disease and its treatment. The study collected data before and after the intervention, though not all students took the instrument both pre- and post-test. After data cleaning, the study used a total of 608 data samples from 457 students, in which 410 of them were post-test data, and 198 were pre-test data. The combination of pre- and post-test data was intended to capture a broader range of students' performance and enable additional analyses. Among the students who took both the pre- and post-test, only 148 eighth grade students had data usable for further analyses.

Data Analysis

Missing Data. Before we analyzed the collected data, we checked for missing values within the 608 data samples. While the main statistical analysis used in the current study, IRT-Rasch, uses Maximum Likelihood Estimation (MLE) which can accommodate a small percentage of missing data, both the percentage of missing data and the pattern of missingness needed to be confirmed. Thus, we explored whether the data are missing completely at random (MCAR), missing at random (MAR), or missing not at random (MNAR) (Schlomer, Bauman, & Card, 2010). We used Little's MCAR test to determine the amount and category of missingness for our data set. The null hypothesis for this test is the data is MNAR. Based on the analysis, 2.42% data were missing, and the missingness was categorized as MCAR (missing completely at random), $X^2 = 961.12$, $df = 985$, $p = .70$. Based on the results of this test, we concluded the data set was ready for analysis (Allison, 2001).

Validity and Reliability. The current study used the Classical Test Theory (CTT) and Item Response Theory (IRT) method concurrently to collect evidence of construct validity and reliability. IRT in the form of Rasch analysis was our primary guide to analysis, with CTT methods used to supplement analysis of the statistical model of the assessment instrument. In this study, using the literature reviewed, we worked on the assumption that the assessment is unidimensional meaning that it measures the FKSAs as a unitary whole. Therefore, no dimensionality test was run.

Compared to CTT methods, Rasch analysis provides more robust results of validation that tend to be more stable across different samples and administrations (Boone et al., 2013). We followed the procedure suggested by Boone et al. (2013) and Bond and Fox (2001) to identify items to remove and to retain. Boone et al. and Bond and Fox suggest looking at the values of infit and outfit mean-square (MNSQ) to evaluate the quality of the item. Items were removed when the MNSQ values were outside the acceptable cutoff range of 0.70 to 1.30 (Wright & Linacre, 1994). In addition to MNSQ values, the current study also used the Item Characteristic Curve (ICC) to evaluate the quality of the item. ICC visualizes the association between students' ability and the probability of correct and incorrect answers. Items were removed when the probability of high-ability students to answer those items correctly is lower than the low-ability students. This outcome indicates a poorly functioning item, especially for differentiating between low and high-ability students. Moreover, we sought a stable assessment instrument where item difficulty levels would be stable across different time points in an intervention and demographic factors. To do so, we used DIF contrast values to investigate stability (item-bias), where contrast values more than 0.64 indicated likely bias (Boone et al., 2013).

After all the items were evaluated with regards to retention or removal, we then performed a Rasch Dimensionality test to compare whether the model of the assessment after item removal was better than the model of the original assessment. Bond and Fox (2001) and Adams and Wu (2010) suggest using three statistics: Chi-square, Final Deviance (FD), and Akaike Information Criterion (AIC), to determine the best model. The model with lower values on these criteria would be considered superior. Additionally, we also ran a Confirmatory Factor Analysis (CFA) as a supplement to the Rasch Dimensionality test. Hu and Bentler (1999) suggest that the value of the Comparative Fit Index (CFI) >.90 and RMSEA < .06 are needed for an acceptable model. We also used a target acceptable value of X^2/df less than 2 to evaluate the models (Tabachnick, Fidell, & Ullman, 2007).

Regarding the internal consistency of the assessment, we calculated several reliability values using both CTT and Rasch methods. We report separation and person (plausible value–PV) reliabilities computed through Rasch analysis, Cronbach's alpha, composite reliability–CR (Raykov, 1997), and test-retest reliability. These calculations were evaluated using a cutoff value of greater than .70 (DeVellis, 2016). A paired-sample *t*-test was also performed for those students who took the assessment pre- and post-intervention ($n = 148$) to examine the change in student scores. The IRT-Rasch analysis was performed in ConQuest 4.14.2 (Adams, Wu, & Wilson, 2015), while other statistical analyses (CFA, paired-sample *t*-test, Cronbach's alpha, CR, and test-retest reliability) were run in Stata 15.1.

RESULTS

Rasch Modeling Analysis

Based on the Rasch modeling analysis, we found that only one out of 27 items had infit and outfit MNSQ outside the acceptable range. The item, Item_7_Con2, had values of infit and outfit MNSQ 1.39 and 1.66, respectively. We also investigated the response pattern using each item's Item Characteristic Curve (ICC) and confirmed that item Item_7_Con2 had the pattern showing low-ability students were more likely to answer the item correctly than high-ability students. Also, we identified two other items with the same pattern. Those items were Item_11_Loo4 and Item_15_Loo5 (ICCs and questions are in the Appendix 2 and Appendix 3). Item_11_Loo4 and Item_15_Loo5 did not initially have MNSQ values beyond the range; however, when we removed Item_7_Con2 from the assessment, and re-ran the analysis, the items became misfitting (MNSQ values outside the cutoff). This indicates that the three items are likely in the same dimension, but not in the same direction with the other 24 items. Thus, we removed these three items from the assessment. The remaining 24 items are in one dimension, and thus are measuring the same single latent construct. We believe this construct is represented by our FKSAs. The list of the final 24 questions and their corresponding FKSAs and CSTA standards are provided in Appendix 4.

We then compared the model of assessment with 27 items and 24 items using Rasch dimensionality test. Table 1 shows the results of this comparison. Based on the comparison, the model of 24 items had lower X^2, FD, and AIC values than the 27-items model, indicating that the 24-item assessment is superior. In line with the lower values, the chi-squared difference also showed that the difference between the two models was significant, $\Delta X^2 = 488.14$, $\Delta df = 3$, $p < .001$. Removing the three items also

Table 1. Comparing the Models

Model	X^2	*df*	FD	AIC
27 items	1221.07	26	19189.28	19245.28
24 items	732.93	23	17328.92	17378.92

Table 2. Comparing the Reliability Values of the Two Models

Model	Separation Reliability	PV Reliability	α	CR
27 items	.983	.809	.826	.859
24 items	.972	.819	.834	.862

Table 3. Alpha if Item Deleted and Rasch Item Fit Statistics for the 24-item Model

Item	α if Item Deleted	Measure	Infit MNSQ	Outfit MNSQ	DIF Contrast Pre/post
Item_1_Var1	.827	-0.003	0.98	0.97	0.15
Item_2_Con1	.829	0.836	1.03	1.10	***0.74***
Item_3_Loo1	.827	0.486	0.99	1.02	0.33
Item_4_Loo2	.825	-1.065	0.91	0.81	0.17
Item_5_Alg1	.834	0.074	1.15	1.21	0.30
Item_6_Var2	.832	-0.136	1.10	1.12	0.04
Item_8_Con3	.831	0.541	1.09	1.16	0.03
Item_9_Con4	.823	-0.493	0.92	0.86	0.12
Item_10_Loo3	.827	-0.911	0.95	0.90	0.13
Item_12_Var3	.828	-0.453	1.02	1.00	0.25
Item_13_Alg2	.829	0.268	1.00	1.03	0.00
Item_14_Var4	.826	0.152	0.98	0.99	0.29
Item_16_Alg3	.831	0.203	1.06	1.06	0.03
Item_17_Var5	.820	-0.056	0.84	0.81	0.15
Item_18_Var6	.829	0.168	1.03	1.04	0.00
Item_19_Alg4	.832	1.173	1.07	1.20	0.41
Item_20_Alg5	.833	0.211	1.11	1.17	0.46
Item_21_Var7	.826	0.139	0.96	0.95	0.14
Item_22_Var8	.827	0.507	0.97	0.99	0.23
Item_23_Con5	.830	-0.873	1.03	0.99	0.00
Item_24_Loo6	.825	0.290	0.93	0.96	0.22
Item_25_Loo7	.824	-0.214	0.92	0.89	0.00
Item_26_Alg6	.824	0.284	0.93	0.92	0.21
Item_27_Alg7	.829	-1.131	0.99	0.93	0.50

Table 4. Comparison of the results of the goodness of fit computed through CFA

Model	X^2	df	X^2/df	CFI	SRMR	RMSEA
27 items	470.44	324	1.45	0.920	0.040	0.029
24 items	372.97	252	1.48	0.931	0.040	0.030

increased the reliability values, except for the separation reliability. The models also had reliability values that were categorized as high internal consistency (> .80) as shown in Table 2.

Table 3 shows the alpha if item deleted, item fit statistics, and the DIF contrast of the items in the 24-item model. It can be seen that there is no item that has alpha if item deleted higher than the full model value of .834 and there is no item with MNSQ values beyond the cutoff. Moreover, the assessment had a good distribution of item difficulty, as shown by a Wright Map in Figure 3. These results further support the 24-item model. The DIF analysis of the 24-item assessment resulted in most of the values of the DIF contrasts being less than 0.64, indicating low bias. Item_2_Con1 had DIF pre/post contrast value of 0.74; however, we still believe that the value is acceptable given its proximity to the cutoff (cf., Cameron, Scott, Adler & Reid, 2014).

Confirmatory Factor Analysis (CFA)

Table 4 shows the comparison results between the 27 and 24-item models based on CFA. Both models can be categorized as a good model. However, the 24-item model had higher CFI and lower X^2 than the 27-item model, indicating a better fit. Based on the chi-squared difference test, this difference was significant at a .05 alpha level, ΔX^2 = 97.47, Δdf = 72, p = .025. It is worth noting the level of improvement in X^2 even as the df decreases, as indicated by a near stable X^2/df ratio lower than the target cut-off of 2.0 (Cole, 1987). The visualization of the CFA for the model with 24 items is shown in Figure 4.

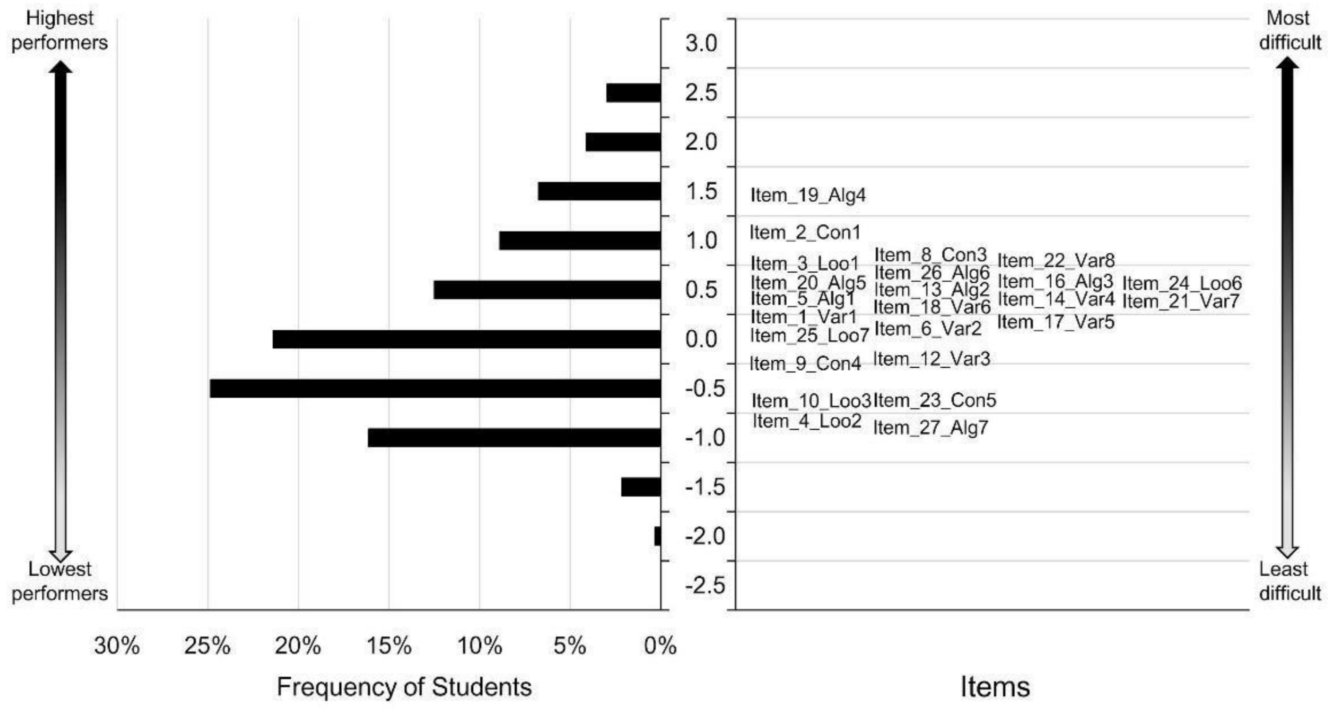


Figure 3. Wright Map of the 24-item Model. Note: the difficulty level shown here is identical to Measure in Table 3

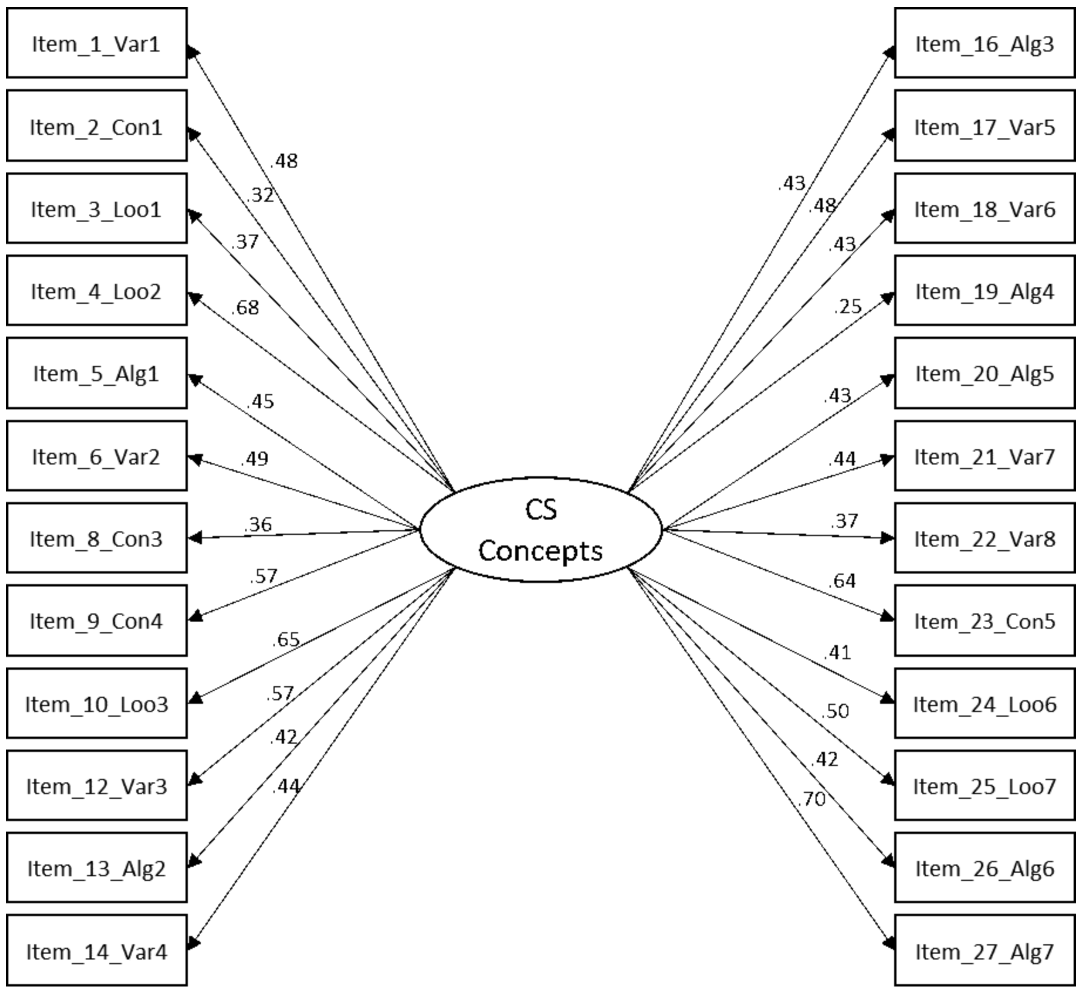


Figure 4. The CFA Model for the 24-item MG-CSCI. Note: the values are the standardized β values

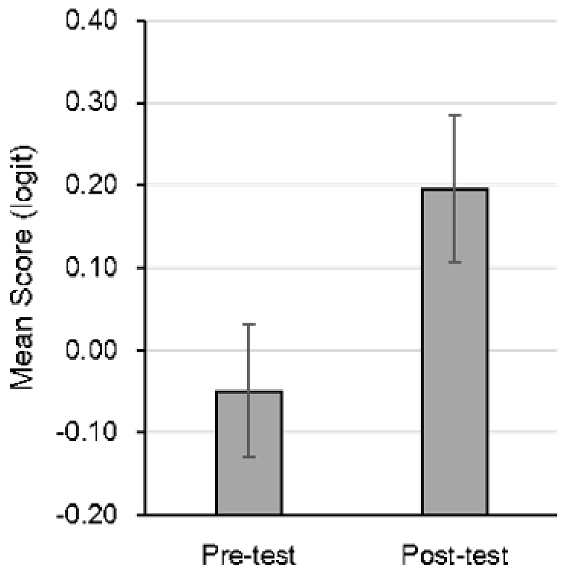


Figure 5. The mean comparison between pre- and post-test scores (Note: error bars are +/- 1 standard error of the mean)

Pre and Post-test Comparison and Test-retest Reliability

As noted above, 148 students took the instrument both pre- and post-test as part of a weeklong curricular intervention. This data was used to obtain the test-retest reliability statistics. The current study found that test-retest reliability value for the model of the 24-item instrument was acceptable (> .70), *r* = .779). Because the 24-item model had a minimal level of item bias based on time-point, this allows us to conduct further statistical analysis, such as comparing the pre- and post-test scores. Based on a paired-sample t-test, it was found that the average logit score of student post-test scores (*M* = 0.20, *SD* = 1.09) was higher than the average of the pre-test logit scores (*M* = -0.05, *SD* = 0.98). The difference was statistically significant with a small effect size, *t*(147) = -4.30, *p* < .001, *d* = 0.24 (Figure 5). These results demonstrate a functional level of sensitivity to a typical CS curricular intervention.

DISCUSSION AND CONCLUSION

We designed and developed the MG-CSCI assessment to measure middle grades (ages 11-13) students' understanding of core computer science concepts. Our objective was to create a multiple-choice assessment that could be completed by students within a standard class period. After developing the assessment, we validated it with more than 400 middle grades students. The results of the study demonstrate that the 24-item MG-CSCI can accurately measure middle grades students' understanding of core CS concepts, including variables, loops, conditionals, and algorithms. In contrast to previous efforts that have developed more focused assessments, and often in a form that uses more time-intensive scoring systems (Grover & Basu, 2017), the current study has developed a more general assessment in a scalable, multiple choice form. This type of assessment facilitates remote administration and automated grading for broader use. Moreover, this concept inventory was designed to support the new and increasingly adopted CS frameworks (CSTA, 2017; Grover & Basu, 2017; K12cs.org, 2016) and is not constrained to specific classroom curricula and activities. By focusing on the middle grades level, this work expands prior work done on concept inventories targeting undergraduate students (e.g., FCS1, SCS1). Finally, the results from the DIF testing shows that the concept inventory has low bias with regards to the timing of administration, whether as a pre- or post-test. Thus, the MG-CSCI can be used as either a pre-test (i.e., prior to a CS learning activity) or post-test (i.e., following a CS learning activity). In addition, it has the sensitivity for revealing the learning impact of a CS curricular intervention which has a relatively low effect size with a modest sample size.

Our validation study did not involve cognitive interviews and did not further investigate the dropped items qualitatively. However, the development of FKSAs underlying our instrument was based on cognitive interviews conducted with middle-grade students that also included students' difficulties in understanding certain CS concepts (Grover & Basu, 2017). As we developed our instrument based on the FKSAs, we sought statistical evidence showing the items were in alignment with the FKSAs. The items that were dropped, in turn, were a poor measurement fit with the FKSAs. While we believe that cognitive interviews on all or select items in the instrument were not a necessary part of this validation process, we also acknowledge the robustness of the cognitive interview methodology for harvesting rich data behind students' responses on the dropped items. New studies should pursue this line of inquiry. Another item of note is that our study included primarily sixth and eighth grade students due to the available sample. While these two grades bracket the middle grades and provide an appropriate sample to assess item difficulty, further studies should confirm item performance more generally with seventh grade students.

In future work, it will be important to explore instruments that expand the conceptual coverage to a greater number of finer-grained constructs underlying core CS concepts. This further work would also allow us to create more focused versions of the instrument based on subsets of the assessment centered on specific CS concepts. Part of this work would also be to create more items that are at either end of the difficulty scale to better capture a full range of abilities. To further improve the validation of the MG-CSCI, we are planning to collect a more demographically diverse data sample so that we can use DIF to further test generalizability across characteristics such as grade levels, previous block-based programming experience, underrepresented minority (URM) vs. non-URM, or English Language Learners (ELLs). Related, we would hope to develop and validate a Spanish language version of this instrument. Thus, the ultimate goal for our further work is to validate a broader set of items to better address a full range of demographics, ability levels, and developmental/age ranges. Furthermore, exploring different problem types and representational forms not limited to block-based programming are also important directions for future work.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. DRL-1640141 and CNS-1138497. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Adams R. J., & Wu, M. (2010). *Multidimensional model*. (August 2010). Retrieved on March 10, 2019 from https://www.acer.org/files/Conquest-Tutorial-7-MultidimensionalModels.pdf

Adams, R. J., Wu, M., & Wilson, M. R. (2015). *ACER ConQuest: Generalised item response modelling software* [Computer software]. Version 4. Camberwell, Victoria: Australian Council for Educational Research.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is Involved and what is the role of the computer science education community? *Inroads*, 2(1), 48-54. https://doi.org/1529-3785/20x/0700-0111

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look*. SRI technical report, 2015.

Bond, T. G., & Fox, C. M. (2001). *Applying the Rasch model: Fundamental measurement in the human sciences*. Mahwah NJ: Lawrence Erlbaum Assoc.

Boone, W. J., Staver, J. R., & Yale, M. S. (2013). *Rasch analysis in the human sciences*. Springer, Dordrecht.

Boulden, D. C., Wiebe, E., Akram, B., Aksit, O., Buffum, P. S., Mott, B., ... Lester, J. (2018). Computational thinking integration into middle grades science classrooms: Strategies for meeting the challenges. *Middle Grades Review*, 4(3), 1-16.

Brown, N. C., Mönig, J., Bau, A., & Weintrop, D. (2016, February). Panel: Future directions of block-based programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 315-316). ACM. https://doi.org/10.1145/2839509.2844661

Buffum, P. S., Lobene, E. V., Frankosky, M. H., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2015, February). A practical guide to developing and validating computer science knowledge assessments with application to middle school. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 622-627). ACM. https://doi.org/10.1145/2676723.2677295

Caceffo, R., Wolfman, S., Booth, K. S., & Azevedo, R. (2016, February). Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 364-369). ACM. https://doi.org/10.1145/2839509.2844559

Cameron, I. M., Scott, N. W., Adler, M., & Reid, I. C. (2014). A comparison of three methods of assessing differential item functioning (DIF) in the Hospital Anxiety Depression Scale: ordinal logistic regression, Rasch analysis and the Mantel chi-square procedure. *Quality of Life Research*, *23*(10), 2883-2888.

code.org. (2018). *Computer science discoveries*. Retrieved on January 6, 2019 from https://code.org/educate/csd

Cole, D. A. (1987). Utility of confirmatory factor analysis in test validation research. *Journal of Consulting and Clinical Psychology*, *55*(4), 584-594. https://doi.org/10.1037/0022-006X.55.4.584

Computer Science Teachers Association-CSTA. (2017). *CSTA K-12 Computer Science Standards 2017*.

Cuny, J. (2011). Transforming computer science education in high schools. *Computer*, *44*(6), 107-109. https://doi.org/10.1109/mc.2011.191

de Araujo, A. L. S. O., Andrade, W. L., & Guerrero, D. D. S. (2016, October). A systematic mapping study on assessing computational thinking abilities. In *2016 IEEE Frontiers in Education Conference (FIE)* (pp. 1-9). IEEE. https://doi.org/10.1109/FIE.2016.7757678

Decker, A., & McGill, M. M. (2019, February). A topical review of evaluation instruments for computing education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 558-564). ACM. https://doi.org/10.1145/3287324.3287393

DeVellis, R. F. (2016). *Scale development: Theory and applications (Vol. 26)*. Sage publications. Thousand Oaks, CA.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57-73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

Goode, J., & Chapman, G. (2016). *Exploring computer science*. University of Oregon, Eugene, OR. Retrieved on April 18, 2018 from http://www.teach21.us/uploads/1/3/0/5/13053428/samplecscurriculum.pdf

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47-53. https://doi.org/10.1145/2189835.2189851

Gross, P., & Powers, K. (2005, October). Evaluating assessments of novice programming environments. In *Proceedings of the First International Workshop on Computing Education Research* (pp. 99-110). ACM. https://doi.org/10.1145/1089786.1089796

Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267-272). ACM. https://doi.org/10.1145/3017680.3017723

Grover, S., & Pea, R. (2018). Computational Thinking: A competency whose time has come. In S. Sentance, E. Barendsen, & C. Schulte (Eds.), *Computer Science Education: Perspectives on teaching and learning in school* (pp. 19-38). London: Bloomsbury Academic, 19-37.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199-237. https://doi.org/10.1080/08993408.2015.1033142

Hamouda, S., Edwards, S. H., Elmongui, H. G., Ernst, J. V., & Shaffer, C. A. (2017). A basic recursion concept inventory. *Computer Science Education*, *27*(2), 121-148. https://doi.org/10.1080/08993408.2017.1414728

Hestenes, D., Wells, M., & Swackhamer, G. (1992). Force concept inventory. *The Physics Teacher*, *30*(3), 141-158. https://doi.org/10.1119/1.2343497

Hu, L. T., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling: A Multidisciplinary Journal*, *6*(1), 1-55. https://doi.org/10.1080/10705519909540118

k12cs.org. (2016). *K-12 computer science framework*. Retrieved on April 18, 2019 from https://k12cs.org/

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*, *72*, 558-569. https://doi.org/10.1016/j.chb.2017.01.005

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32-37.

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, *21*(2), 105-134. https://doi.org/10.1080/08993408.2011.579805

Luxton-Reilly, A., Becker, B. A., Cao, Y., McDermott, R., Mirolo, C., Mühling, A., ... & Whalley, J. (2018, January). Developing assessments to determine mastery of programming fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports* (pp. 47-69). ACM. https://doi.org/10.1145/3174781.3174784

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014, June). Computational thinking in K-9 education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference* (pp. 1-29). ACM. https://doi.org/10.1145/2713609.2713610

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264. https://doi.org/10.1080/08993408.2013.832022

Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2003). Focus article: On the structure of educational assessments. *Measurement: Interdisciplinary Research and Perspectives*, *1*(1), 3-62. https://doi.org/10.1207/S15366359MEA0101_02

Mühling, A., Ruf, A., & Hubwieser, P. (2015, November). Design and first results of a psychometric test for measuring basic programming abilities. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 2-10). ACM. https://doi.org/10.1145/2818314.2818320

Parker, M. C., Guzdial, M., & Engleman, S. (2016, August). Replication, validation, and use of a language independent CS1 knowledge assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 93-101). ACM. https://doi.org/10.1145/2960310.2960316

Raykov, T. (1997). Estimation of composite reliability for congeneric measures. *Applied Psychological Measurement*, *21*(2), 173-184. https://doi.org/10.1177/01466216970212006

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678-691. https://doi.org/10.1016/j.chb.2016.08.047

Schlomer, G. L., Bauman, S., & Card, N. A. (2010). Best practices for missing data management in counseling psychology. *Journal of Counseling Psychology*, *57*(1), 1-10. https://doi.org/10.1037/a0018082

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., & Wildeman, B. (2012, July). Infusing computational thinking into the middle- and high-school curriculum. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp. 22-27). ACM. https://doi.org/10.1145/2325296.2325306

Shneiderman, B., & Mayer, R. (1979). Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, *8*(3), 219-238. https://doi.org/10.1007/BF00977789

Smith IV, D. H., Hao, Q., Jagodzinski, F., Liu, Y., & Gupta, V. (2019, May). Quantifying the Effects of Prior Knowledge in Entry-Level Programming Courses. In *Proceedings of the ACM Conference on Global Computing Education* (pp. 30-36). ACM. https://doi.org/10.1145/3300115.3309503

Sudol, L. A., & Studer, C. (2010, March). Analyzing test items: using item response theory to validate assessments. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 436-440). ACM. https://doi.org/10.1145/1734263.1734411

Tabachnick, B. G., Fidell, L. S., & Ullman, J. B. (2007). *Using multivariate statistics (Vol. 5)*. Boston, MA: Pearson.

Taylor, C., Zingaro, D., Porter, L., Webb, K. C., Lee, C. B., & Clancy, M. (2014). Computer science concept inventories: past and future. *Computer Science Education*, *24*(4), 253-276. https://doi.org/10.1080/08993408.2014.970779

Tew, A. E., & Guzdial, M. (2011, March). The FCS1: a language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 111-116). ACM. https://doi.org/10.1145/1953163.1953200

Tukiainen, M., & Mönkkönen, E. (2002, June). Programming aptitude testing as a prediction of learning to program. In *Proceedings of 14th Workshop of the Psychology of Programming Interest Group (PPIG)*. 45-57.

Weintrop, D., & Wilensky, U. (2015, July). Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *ICER* (Vol. 15, pp. 101-110).

Weintrop, D., Killen, H., & Franke, B. E. (2018). Blocks or text? How programming language modality makes a difference in assessing underrepresented populations. In *International Society of the Learning Sciences, Inc. [ISLS]*. https://doi.org/10.22318/cscl2018.328

Weintrop, D., Killen, H., Munzar, T., & Franke, B. (2019, February). Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 1218-1224). ACM. https://doi.org/10.1145/3287324.3287348

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 215-220). ACM. https://doi.org/10.1145/2157136.2157200

Wiebe, E., London, J., Aksit, O., Mott, B. W., Boyer, K. E., & Lester, J. C. (2019, February). Development of a lean computational thinking abilities assessment for middle grades students. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 456-461). ACM. https://doi.org/10.1145/3287324.3287390

Winters, T., & Payne, T. (2005, October). What do students know?: an outcomes-based assessment system. In *Proceedings of the First International Workshop on Computing Education Research* (pp. 165-172). ACM. https://doi.org/10.1145/1089786.1089802

Wright, B. D., & Linacre, J. M. (1994). Reasonable mean-square fit values. *Rasch Measurement Transactions*, *8*(3), 370.

Xie, B., Davidson, M. J., Li, M., & Ko, A. J. (2019, February). An item response theory evaluation of a language-independent cs1 knowledge assessment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 699-705). ACM. https://doi.org/10.1145/3287324.3287370

Zendler, A. (2019). cpm.4.CSE/IRT: Compact process model for measuring competences in computer science education based on IRT models. *Education and Information Technologies*, *24*(1), 843-884. https://doi.org/10.1007/s10639-018-9794-3

Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013, July). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 267-272). ACM. https://doi.org/10.1145/2462476.2462479

APPENDIX 1

Focal Knowledge, Skills and Abilities (Grover & Basu, 2017) and CSTA Standards (CSTA, 2017) used in this study

Focal Knowledge, Skills and Abilities – FKSAs (Grover & Basu, 2017)

1. Ability to describe what a given loop is doing
2. Ability to describe the sequence that is executed in a given program when the program contains things inside a loop as well as outside of the loop.
3. Knowledge that a loop involves a repeating pattern that will terminate under a specified condition or after a certain number of repetitions
4. Ability to identify the repeating pattern within a loop
5. Ability to describe the structural components of a pattern (not in a programming context)
6. Ability to identify a pattern from a real-world phenomenon
7. Ability to describe how a conditional pathway would operate
8. Ability to create variables, assign values and update variables
9. Ability to describe how a variable changes values in a loop
10. Ability to determine what variables are required in a program to achieve the goals of the computational solution.
11. Ability to evaluate a Boolean expression
12. Ability to use Boolean operators in a programming context
13. Ability to create a Boolean expression for a given condition
14. Ability to identify sub-parts of a computational solution
15. Ability to create a Boolean test to control a loop given specifications
16. Ability to describe how the Boolean tests interacts with the loop execution

From CSTA (https://www.csteachers.org/)

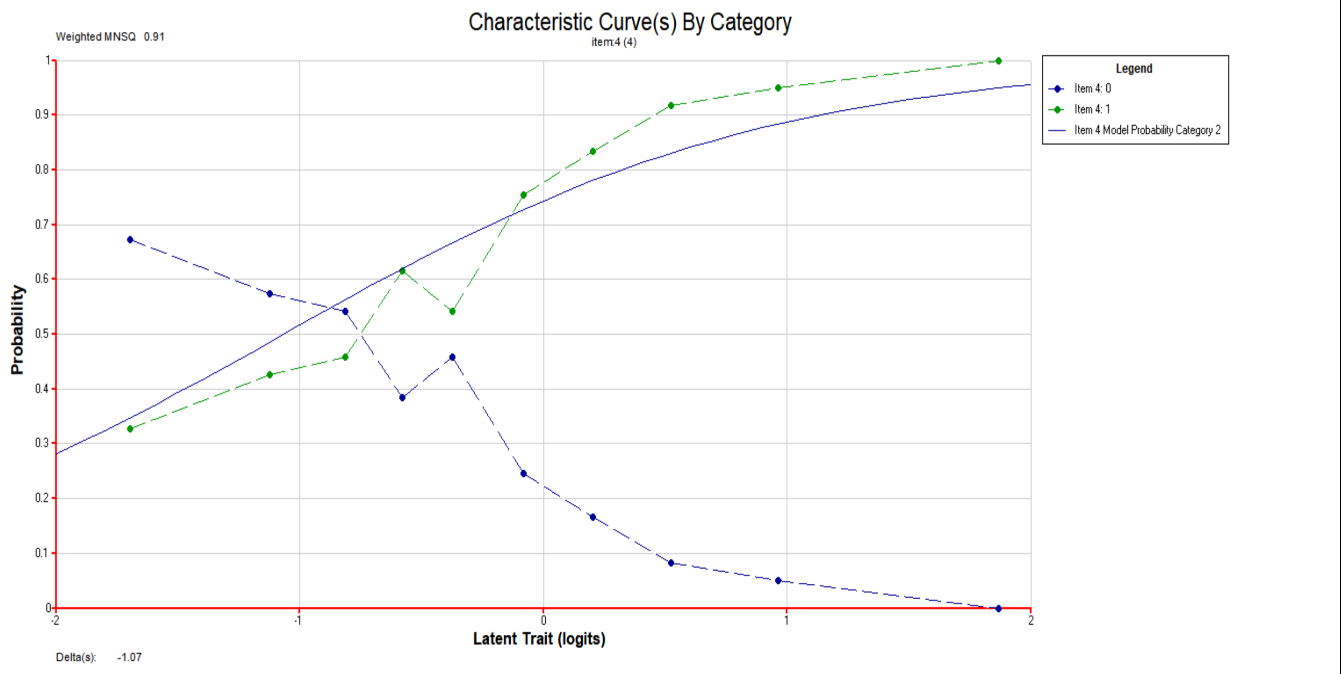
2-AP-10 Use flowcharts and/or pseudocode to address complex problems as algorithms. (P4.4, P4.1)

2-AP-11 Create clearly named variables that represent different data types and perform operations on their values. (P5.1, P5.2)

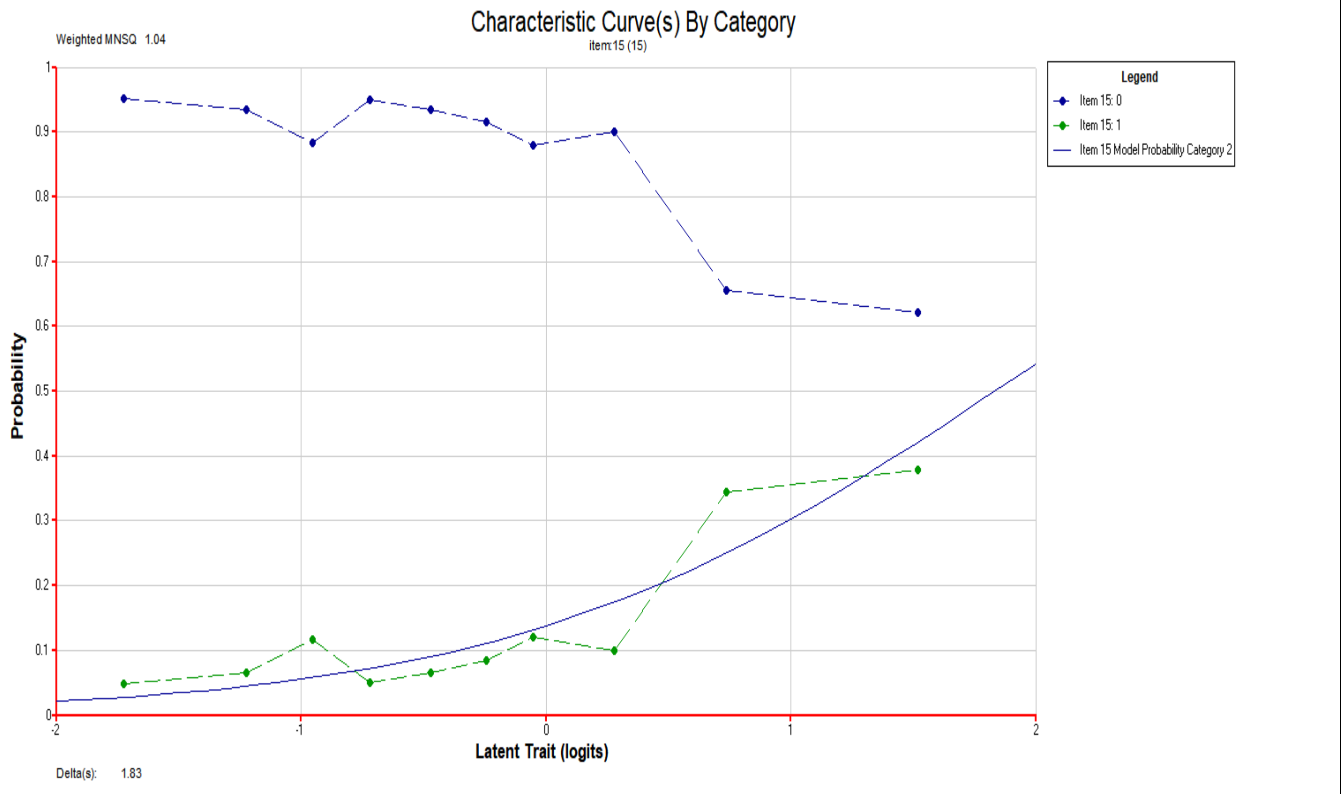
APPENDIX 2

ICCs for Item_11_Loo4 and Item_15_Loo5

Item_11_Loo4

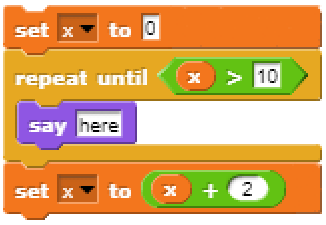
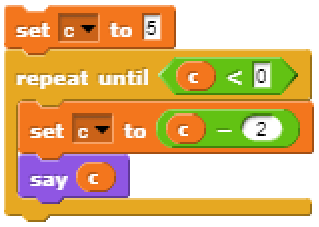


Item_15_Loo5



APPENDIX 3

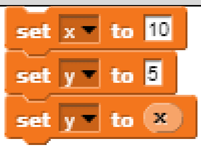
The list of problematic questions and their corresponding FKSAs

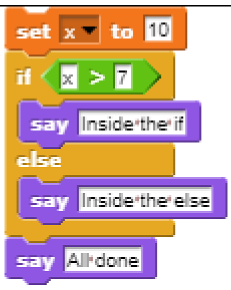
No	Item Code	Question	Category	FKSA
11	Item_11_Loo4	 `set x to 0` `repeat until x > 10` `say here` `set x to x + 2` How many times will the word "here" be said when this code is run? A. 0 B. 1 C. 5 B. 10 C. "here" will be said over and over until the code is stopped manually D. It will be different each time you run it	Loop/ Iteration	1, 2, 3, 4, 8, 9
15	Item_15_Loo5	 `set c to 5` `repeat until c < 0` `set c to c - 2` `say c` What will be said when this code is run? A. 5 4 3 2 1 B. 5 3 1 C. 5 3 -1 D. 3 1 -1 E. 3 1 F. It will be different each time you run it	Loop/ Iteration	1, 3, 4, 8, 9

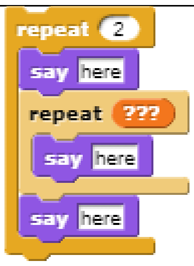
APPENDIX 4

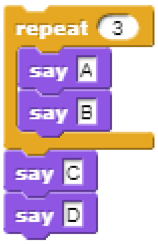
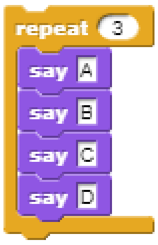
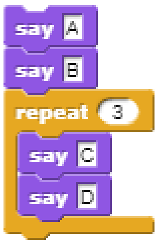
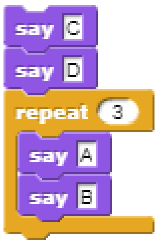
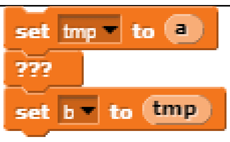
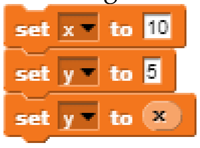
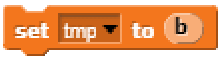
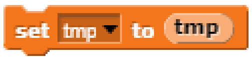
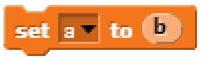
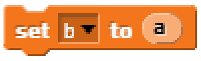
The list of the final CSCI questions and their corresponding FKSAs and CSTA Standards

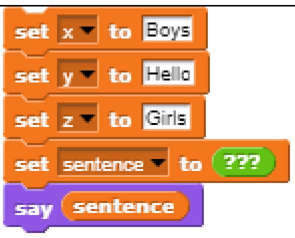
No	Item Code	Question	Category	FKSA

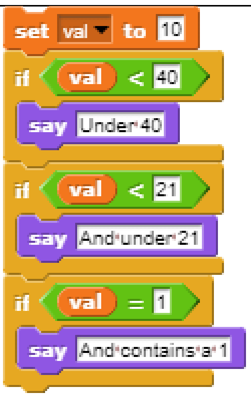
1	Item_1_Var1	
		`set x to 10`
		`set y to 5`
		`set y to x`
		What are the values of **x** and **y** after the above code runs?
		A. x is equal to 10; y is equal to 5
		B. x is equal to 5; y is equal to 5
		C. x is equal to 10; y is equal to 10
		D. x is equal to 5; y is equal to x
			Variable	8

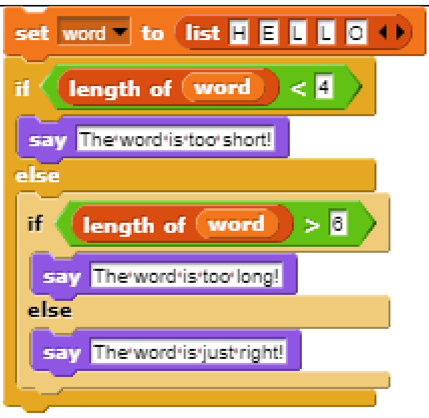
2	Item_2_Con1	
		`set x to 10`
		`if x > 7`
		`  say Inside the if`
		`else`
		`  say Inside the else`
		`say All done`
		What will be said after running this code?
		A. All done
		B. Inside the if
		C. Inside the if
		All done
		D. Inside the else
		All done
		E. Inside the if
		Inside the else
		All done
			Conditional	7, 8

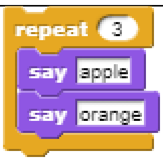
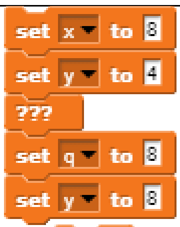
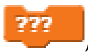
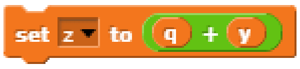
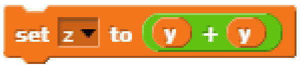
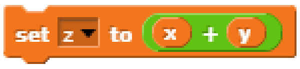
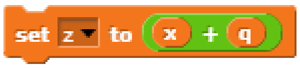
3	Item_3_Loo1	
		`repeat 2`
		`  say here`
		`  repeat ???`
		`    say here`
		`  say here`
		To ensure **"here"** is said 10 times, which number should be filled in the repeat block to replace the ??? ?
		A. 2
		B. 3
		C. 6
		D. 10
			Loop/ Iteration	1, 3, 4

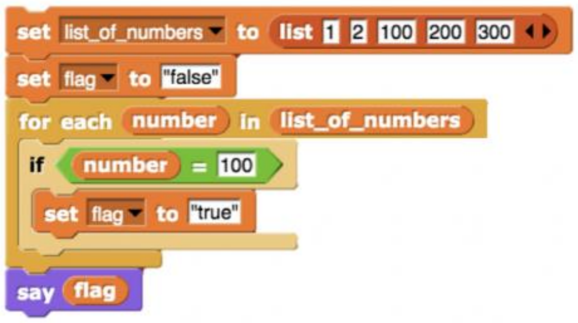
4	Item_4_Loo2	Which lines of code will result in the output saying 'ABABABCD'? A.  `repeat 3` `say A` `say B` `say C` `say D` B.  `repeat 3` `say A` `say B` `say C` `say D` C.  `say A` `say B` `repeat 3` `say C` `say D` D.  `say C` `say D` `repeat 3` `say A` `say B`	Loop/ Iteration	1, 2, 3, 4
5	Item_5_Alg1	 `set tmp to a` `???` `set b to tmp` a, b and tmp are variables with values. Which of the following can be used to replace ??? so that the code  `set x to 10` `set y to 5` `set y to x` will *swap the values of* a *and* b ? A.  `set tmp to b` B.  `set tmp to tmp` C.  `set a to b` D.  `set b to a`	Algorithm	8

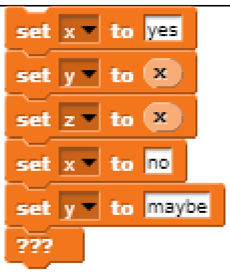
6	Item_6_Var2	
		`set x to Boys`
		`set y to Hello`
		`set z to Girls`
		`set sentence to ???`
		`say sentence`
		Which of the following should replace ??? so that the code will say *"Hello Girls and Boys"*?
		A. join z and y x
		B. join y z and x
		C. join y x and z
		D. join y and z x
			Variable	8

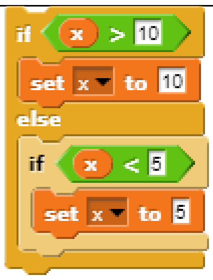
7	Item_8_Con3	
		`set val to 10`
		`if val < 40`
		`  say Under 40`
		`if val < 21`
		`  say And under 21`
		`if val = 1`
		`  say And contains a 1`
		What will be said after running this code?
		A. Under 40
		B. And under 21
		C. And contains a 1
		D. Under 40
		And under 21
		E. Under 40
		And under 21
		And contains a 1
		F. Nothing will be said
			Conditional	7, 8

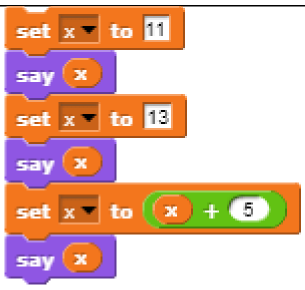
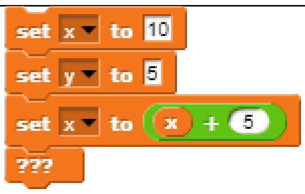
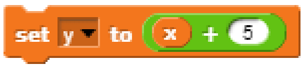
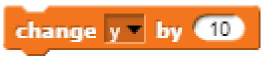
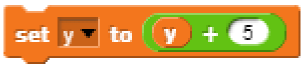
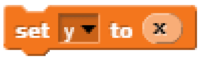
8	Item_9_Con4	
		`set word to list H E L L O`
		`if length of word < 4`
		`  say The word is too short!`
		`else`
		`  if length of word > 6`
		`    say The word is too long!`
		`  else`
		`    say The word is just right!`
		What will be said after this code has run?
		A. The word is too short!
		B. The word is too long!
		C. The word is just right!
		D. Nothing will be said.
			Conditional	7, 8

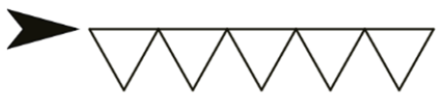
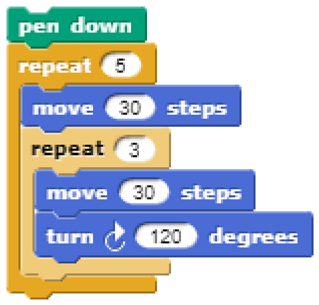
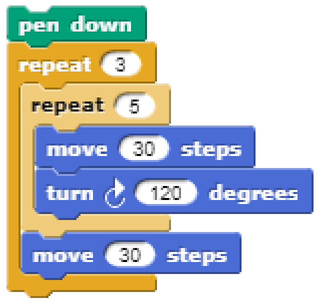
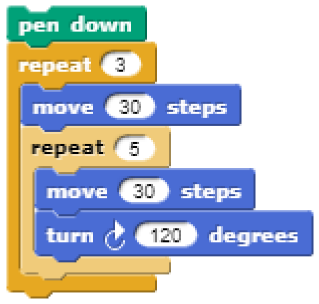
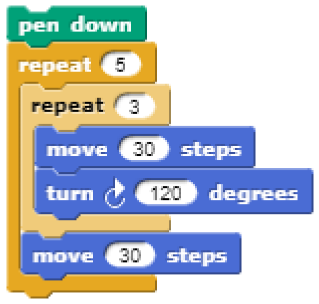
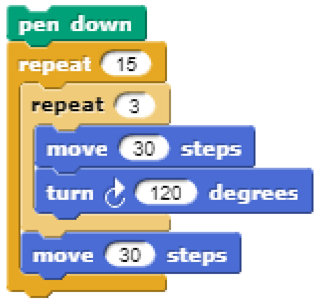
9	Item_10_Loo3	 repeat 3 say apple say orange What will be said when this code is run? A. apple apple apple orange orange orange B. apple orange C. apple orange apple orange apple orange D. Nothing will be said E. It will be different each time you run it	Loop/ Iteration	1,3,4
10	Item_12_Var3	 set x to 8 set y to 4 ??? set q to 8 set y to 8 Which of the following can be used to replace  ???, so that the code will have *z is equal to 12*? A.  set z to q + y B.  set z to y + y C.  set z to x + y D.  set z to x + q	Variable	8

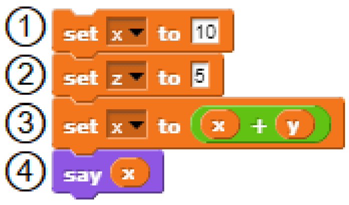
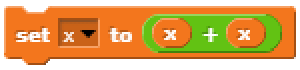
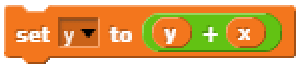
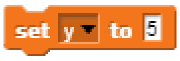
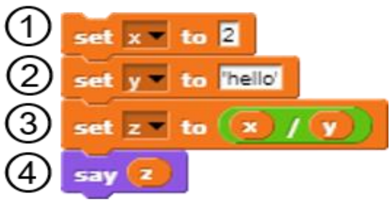
11	Item_13_Alg2	`set list_of_numbers to list 1 2 100 200 300`
		`set flag to "false"`
		`for each number in list_of_numbers`
		`  if number = 100`
		`    set flag to "true"`
		`say flag`
		For the following problem, assume that list_of_numbers is a variable that contains a list of numbers from **1 to 300**. What does this code do?
		A. says 'true' if the number 100 is in the list
		B. says 'true' if the first number in the list is 100
		C. says 'true' if the last number of the list is 100
		D. says 'true' if there are 100 in the list
		E. says 'true' or 'false' 100 times
			Algorithm	1, 2, 4, 7, 8, 9

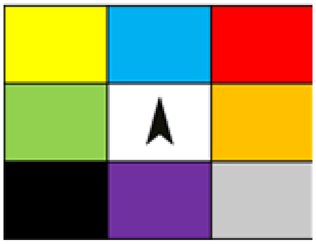
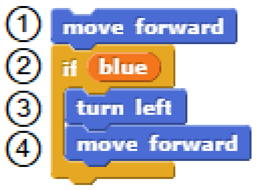
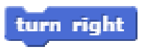
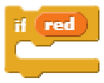
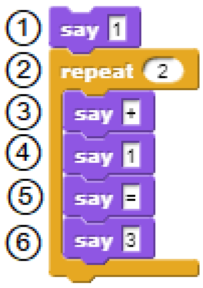
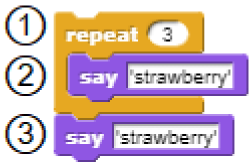
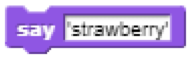
12	Item_14_Var4	`set x to yes`
		`set y to x`
		`set z to x`
		`set x to no`
		`set y to maybe`
		`???`
		Which of the following can be used to replace ???, so that the code will have **z equal to no**?
		A. `set z to x`
		B. `set z to y`
		C. `set x to y`
		D. does not need an additional block
			Variable	8

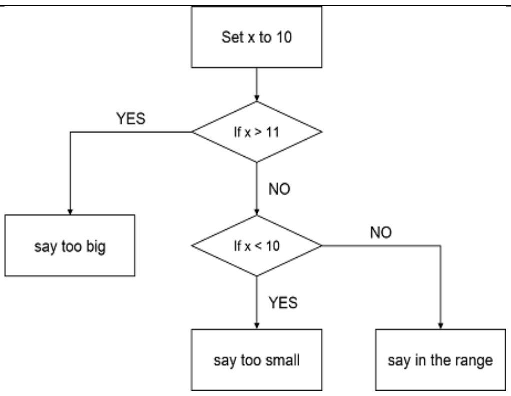
13	Item_16_Alg3	`if x > 10`
		`  set x to 10`
		`else`
		`  if x < 5`
		`    set x to 5`
		What does this code do?
		A. Makes sure the value of x is not equal to 10
		B. Makes sure the value of x is less than 5
		C. Makes sure the value of x is between 10 and 5
		D. It always sets x equal to 5
		E. This code will cause an error
			Algorithm	7

14	Item_17_Var5	 set x to 11 say x set x to 13 say x set x to (x + 5) say x What is said when this code is run? A. ‘11’ then ‘13’ then ‘18’ B. ‘11’ then ‘11’ then ‘11’ C. ‘x’ then ‘x’ then ‘x+5’ D. ‘18’ then ‘18’ then ‘18’ E. [Nothing will be said] F. This code will cause an error	Variable	8
15	Item_18_Var6	 set x to 10 set y to 5 set x to (x + 5) ??? To ensure the value of ***x is 15 and y is 10*** after running this code, which block can replace ??? ? A.  set y to (x + 5) B.  change y by 10 C.  set y to (y + 5) D.  set y to x	Variable	8

16	Item_19_Alg4	Which code snapshot will cause the arrow to draw the shape shown below? The side length of each triangle is 30 steps.
		Note: The arrow starts drawing where it is located.
		
		A. 
		`pen down`
		`repeat 5`
		`  move 30 steps`
		`  repeat 3`
		`    move 30 steps`
		`    turn ↻ 120 degrees`
		B. 
		`pen down`
		`repeat 3`
		`  repeat 5`
		`    move 30 steps`
		`    turn ↻ 120 degrees`
		`  move 30 steps`
		C. 
		`pen down`
		`repeat 3`
		`  move 30 steps`
		`  repeat 5`
		`    move 30 steps`
		`    turn ↻ 120 degrees`
		D. 
		`pen down`
		`repeat 5`
		`  repeat 3`
		`    move 30 steps`
		`    turn ↻ 120 degrees`
		`  move 30 steps`
		E. 
		`pen down`
		`repeat 15`
		`  repeat 3`
		`    move 30 steps`
		`    turn ↻ 120 degrees`
		`  move 30 steps`
			Algorithm	1, 2, 3, 4

17	Item_20_Alg5	If you want to write a code that asks a user to type in a sentence, then reports back to the user the number of times the letter 'e' appears in that sentence, which of these things would your programming language **NOT** need to be able to do: A. Compare two letters to each other to determine if they are the same B. Display text on the screen C. Convert letters into numbers and numbers into letters D. Store user entered information	Algorithm	14
18	Item_21_Var7	The following code is supposed to say "15."  1 set x to 10 2 set z to 5 3 set x to (x + y) 4 say x What needs to be changed in this code for this to happen? A. Change the block number 3 to  set x to (x + x) B. Change the block number 2 to  set y to (y + x) C. Change the block number 2 to  set y to 5 D. Nothing needs to be changed	Variable	10
19	Item_22_Var8	Look at the code below!  1 set x to 2 2 set y to 'hello' 3 set z to (x / y) 4 say z What is wrong with this code? A. Nothing is wrong with the code B. x is a numeric value C. The order of the blocks is wrong D. Dividing a numeric value with an alphabetic value	Variable	2-AP-11

20	Item_23_Con5	Look at the picture below!  The arrow is heading to the **blue** tile. If you are going to move the arrow to the red tile using the following code, which part of the code needs to be changed?  ① move forward ② if blue ③ turn left ④ move forward A. Nothing needs to be changed B. Change the block number 3 to  turn right C. Change the block number 2 to  if red D. Move the block 1 to after the block 4	Conditional	7
21	Item_24_Loo6	The following code is supposed to say “1+1+1=3”.  ① say 1 ② repeat 2 ③ say + ④ say 1 ⑤ say = ⑥ say 3 What needs to be changed in the code for this to happen? A. The block numbers 5 and 6 should come below the repeat block B. The block number 1 should go inside the repeat block C. Number of iteration should be 3 D. Nothing needs to be changed	Loop/ Iteration	2, 9
22	Item_25_Loo7	The following code needs to say ‘strawberry’ **six** times. What changes need to be made, if any, for this to happen?  ① repeat 3 ② say 'strawberry' ③ say 'strawberry' A. Nothing, the code will say ‘strawberry’ six times B. Block number 2 should come out of the repeat block C. Another  say 'strawberry' should be added to the repeat block D. The block number 3 should go inside the repeat block	Loop/ Iteration	2, 9

23	Item_26_Alg6	 Set x to 10 If x > 11 YES NO say too big If x < 10 NO YES say too small say in the range What will be said if the above code is run? A. Nothing will be said B. say too small C. say too big D. say in the range E. It is going to be different every time	Algorithm	2-AP-10
24	Item_27_Alg7	A robot is going to deliver a package to an owner. Below are the steps the robot needs to take to deliver the package. 1. Locate the owner of the package 2. Follow the fastest path from the robot location to the owner's location 3. Calculate the fastest path from the robot location to the owner's location 4. Drop the package However, there might be small mistake in the order of the steps. Can you find the mistake? E. The order of the steps is just right F. Step number 2 should be after step number 3 G. Step number 2 should be after step number 4 H. Step number one should be after step numbers 2 and 3	Algorithm	14