

Effective
Fall 2017

AP[®] Computer Science Principles

**ASSESSMENT OVERVIEW AND PERFORMANCE TASK
DIRECTIONS FOR STUDENTS**

INCLUDES:

- ✓ Assessment overview
- ✓ Explore performance task directions
- ✓ Create performance task directions
- ✓ Exam reference sheet

Contents

AP Computer Science Principles Assessment Overview for Students

- 1 Investigation and Citation
- 2 Programming Language Requirements
- 2 Peer-to-Peer Collaboration
- 3 Preparing for the Through-Course Performance Tasks

Performance Task: Explore – Impact of Computing Innovations

- 7 Preparing for the Explore Performance Task
- 8 Guidelines for Completing the Explore Performance Task

Performance Task: Create – Applications from Ideas

- 12 Preparing for the Create Performance Task
- 12 Guidelines for Completing the Create Performance Task

AP Computer Science Principles Exam Reference Sheet

AP Computer Science Principles Assessment Overview for Students

The AP Computer Science Principles course has three assessments, consisting of two performance tasks and an end-of-course AP Exam. All three assessments are summative and will be used to calculate a final AP score (using the 1–5 scale) for AP Computer Science Principles.

Assessment	Timing	Percentage of Total AP Score
Explore Performance Task	8 hours	16%
Create Performance Task	12 hours	24%
End-of-Course Exam	2 hours	60%

Students who are completing the AP Computer Science Principles course in a nontraditional classroom situation (e.g., online, homeschool, independent study) should consult a school-based AP Coordinator for instructions on taking the AP Exam and submitting work for the performance tasks.

Investigation and Citation

The through-course performance tasks require you to create computational artifacts. A computational artifact is a visualization, a graphic, a video, a program, or an audio recording that you create using a computer. For the Create performance task, you will develop a computer program and for the Explore performance task, you will create a computational artifact of your choosing to represent or illustrate the intended purpose, function, or effect of a computing innovation using any computational tool(s) you wish.

In creating your computational artifact, you can create your own original work, including video, music, text, images, graphs, and program code. If you use external work to integrate into your computational artifact, you must acknowledge, attribute, and/or cite sources and include a bibliography with your submission. External work that should be acknowledged include video, music, text, images, graphs, and program code that are used in the creation of your computational artifacts.

AP Computer Science Principles Policy on Plagiarism

A student who fails to acknowledge (i.e., through citation, through attribution, by reference, and/or through acknowledgment in a bibliographic entry) the source or author of any and all information or evidence taken from the work of someone else will receive a score of 0 on that performance task.

To the best of their ability, teachers will ensure that students understand ethical use and acknowledgment of the ideas and work of others as well as the consequences of plagiarism. The student's individual voice should be clearly evident, and the ideas of others must be acknowledged, attributed, and/or cited. A computational artifact without acknowledgment of the media used in the creation of the computational artifact, and program code segment(s) written by someone else used in a program without appropriate acknowledgment, are all considered plagiarized work.

Programming Language Requirements

AP Computer Science Principles is language agnostic. This means that there is no specific language requirement. When completing the Create – Applications from Ideas performance task for this course, you are allowed to select a language you feel is most appropriate to meet the requirements of the task. When selecting a language or program, you should review the requirements section of the performance task to ensure that your program will be sophisticated enough to implement mathematical and logical concepts, create abstractions, and implement algorithms.

Peer-to-Peer Collaboration

Collaboration is only allowed on designated sub-components of the Create performance task.

For the Explore – Impact of Computing Innovations performance task, collaboration of any kind is not allowed.

For the Create – Applications from Ideas performance task, you are encouraged to collaborate on the development of their program with another student in your class. Collaboration is not allowed during the creation of the video or when answering the written responses.

Students completing AP Computer Science Principles in a nontraditional classroom situation (e.g., online, homeschool, independent study) are encouraged to collaborate with another student peer when completing the Create performance task.

Preparing for the Through-Course Performance Tasks

The following guidelines are meant to help you be successful on the performance tasks as well as to clarify or address any questions you may have regarding the process of completing these tasks.

Prior to your teacher administering the performance tasks, you should:

- ▶ obtain content knowledge and skills that will help you succeed on the performance tasks;
- ▶ practice either an entire performance task or individual prompts of the tasks;
- ▶ review the scoring guidelines, found on AP Central, to understand how your work will be assessed;
- ▶ examine examples of performance task found on AP Central of submissions at high, medium, and low levels. If you select a computing innovation that is represented in one of the examples, or that was discussed in class, you must find new sources and submit original responses to avoid. You cannot submit any work from practice performance tasks.
- ▶ pay attention to the instructions concerning the size of the files to be uploaded; the computational artifact for the Explore performance task and the video for the Create performance task individually cannot exceed 30MB.
- ▶ ensure you know the proper way to evaluate and appropriately cite a source, including program code; any program code which has not been written by you must be cited and credit given to the author;
- ▶ understand the level of detail expected in writing your responses by examining the scoring guidelines and high level samples found on AP Central;
- ▶ understand that you may not revise your work once you have submitted your work as final to the AP Digital Portfolio; and
- ▶ be aware that the scoring process that occurs in the AP Reading may be different from the scoring process that occurs in your classroom; the AP score that you receive may be different than your classroom grade.

Performance Task: Explore – Impact of Computing Innovations

Overview

Computing innovations impact our lives in ways that require considerable study and reflection for us to fully understand them. In this performance task, you will explore a computing innovation of your choice. A computing innovation is an innovation that includes a computer or program code as an integral part of its functionality. Your close examination of this computing innovation will deepen your understanding of computer science principles.

Please note that once this performance task has been assigned as an assessment (rather than as practice), you are expected to complete the task with minimal assistance from anyone. For more clarification see the Guidelines for Completing the Through-Course Performance Tasks section.

You will be provided with a minimum of 8 hours of class time to develop, complete, and submit the following:

- ▶ **A computational artifact**
- ▶ **Written responses**

Scoring guidelines and instructions for submitting your performance tasks are available on the AP Computer Science Principles Course Home Page.

Note: Students in nontraditional classroom environments should consult a school-based AP Coordinator for submission instructions.

When completing the Explore – Impacts of Computing Innovations performance task, you will be expected to conduct investigations on a computing innovation. A computing innovation is an innovation that includes a computer or program code as an integral part of its functionality.

You must ensure you have identified relevant, credible, and easily accessible sources to support your creation of a computational artifact as well as to support your responses to the prompts. You can search for print or nonprint sources as part of your investigation. You can refer to a journal, Web page, or an expert that is being quoted as part of your written response. Avoid plagiarism by acknowledging, attributing, and/or citing sources throughout your responses.

General Requirements

This performance task requires you to select and investigate a computational innovation to:

- ▶ analyze a computing innovations impact on society, economy, or culture and explain how this impact could be beneficial and/or harmful;

- ▶ explain how a computing innovation consumes, produces, or transforms data; and
- ▶ describe how data storage, data privacy, or data security concerns are raised based on the capabilities of the computing innovation.

You are also required to:

- ▶ investigate your computing innovation using a variety of sources (e.g., print, online, expert interviews);
- ▶ provide in-text citations of at least three different sources that helped you create your computational artifact and/or formulate your written responses;
 - › At least two of the sources must be available online or in print; your third source may be either online, in print, or a personal interview with an expert on the computing innovation.
 - › At least two of the sources must have been created after the end of the previous academic year.
- ▶ produce a computational artifact that illustrates, represents, or explains the computing innovation's intended purpose, its function, or its effect; and
- ▶ provide written responses to all the prompts in the performance task about your computational artifact and computing innovation.

Submission Requirements

1. Computational Artifact

Your computational artifact must provide an illustration, representation, or explanation of the computing innovation's intended purpose, its function, or its effect. The computational artifact must not simply repeat the information supplied in the written responses and should be primarily nontextual.

Submit a video, audio, or PDF file. Use computing tools and techniques to create one original computational artifact (a visualization, a graphic, a video, a program, or an audio recording). **Acceptable multimedia file types include .mp3, .mp4, .wmv, .avi, .mov, .wav, .aif, or .pdf format. PDF files must not exceed three pages. Video or audio files must not exceed 1 minute in length and must not exceed 30MB in size.**

2. Written Responses

Submit one PDF file in which you respond directly to each of the prompts below. **Clearly label your responses 2a–2e in order.** Your responses must provide evidence of the extensive knowledge you have developed about your chosen computing innovation and its impact(s). Write your responses so they would be understandable to someone who is not familiar with the computing innovation. Include citations, as applicable, within your written responses. **Your response to prompts 2a–2d combined must not exceed 700 words.** The references required in 2e are not included in the final word count.

Computational Artifact

2a. Provide information on your computing innovation and computational artifact.

- ♦ Name the computing innovation that is represented by your computational artifact.
- ♦ Describe the computing innovation's intended purpose and function.
- ♦ Describe how your computational artifact illustrates, represents, or explains the computing innovation's intended purpose, its function, or its effect.

(Must not exceed 100 words)

2b. Describe your development process, explicitly identifying the computing tools and techniques you used to create your artifact. Your description must be detailed enough so that a person unfamiliar with those tools and techniques will understand your process. *(Must not exceed 100 words)*

Computing Innovation

2c. Explain at least one beneficial effect and at least one harmful effect the computing innovation has had, or has the potential to have, on society, economy, or culture. *(Must not exceed 250 words)*

2d. Using specific details, describe:

- ♦ the data your innovation uses;
- ♦ how the innovation consumes (as input), produces (as output), and/or transforms data; and
- ♦ at least one data storage concern, data privacy concern, or data security concern directly related to the computing innovation.

(Must not exceed 250 words)

References

2e. Provide a list of at least three online or print sources used to create your computational artifact and/or support your responses through in-text citation to the prompts provided in this performance task.

- ♦ At least two of the sources must have been created after the end of the previous academic year.
- ♦ For each online source, include the complete and permanent URL. Identify the author, title, source, the date you retrieved the source, and, if possible, the date the reference was written or posted.
- ♦ For each print source, include the author, title of excerpt/article and magazine or book, page number(s), publisher, and date of publication.
- ♦ If you include an interview source, include the name of the person you interviewed, the date on which the interview occurred, and the person's position in the field.
- ♦ Include in-text citations for the sources you used.
- ♦ Each source must be relevant, credible, and easily accessed.

Preparing for the Explore Performance Task

Prior to your teacher administering this task, you should:

- ▶ understand that a computing innovation (i.e., an innovation that includes a computer or program code as an integral part of its functionality) has a meaningful personal or community emphasis is an appropriate choice, as long as it fulfills the requirements to complete all the prompts in the performance task;
- ▶ practice searching and evaluating sources relevant to computing innovations; all sources cited must be relevant, credible, and easily accessible;
- ▶ practice clearly explaining the impact the intended use of a computing innovation has on society, economy, and culture, clearly justifying both beneficial and harmful effects;
- ▶ practice demonstrating your knowledge of computer science and understanding of how data is input, output, and transformed in your analysis of the data used by the computing innovation.
- ▶ practice making connections between the data used by a computing innovation and a security, privacy, or storage concern.
- ▶ obtain the meaning and purpose of creating a computational artifact; your creation must provide an illustration, representation, or explanation of the computing innovation's intended purpose, its function, or its effect;
- ▶ have exposure to the use of a variety of computational tools that can be used to create effective computational artifacts;
- ▶ understand which computational artifacts would be considered effective and ineffective.

Effective artifacts include:

- › visual, graphical, and/or audio content to help a reader understand the purpose, function, or effect of a computing innovation; and
- › the use of communications media, such as animations, comic strips, infographics, and/or public service announcements, to illustrate the purpose, function, or effect of a computing innovation.

Ineffective artifacts include:

- › artifacts that repeat information previously supplied in the written responses;
 - › multislides presentations with paragraphs of text or bullets;
 - › artifacts that have not been created by the student; and
 - › artifacts that focus on advertising the computing innovation's functionality instead of the purpose of the innovation.
- ▶ practice writing responses based on relevant and credible sources and include in-text citations; and
 - ▶ practice appropriate acknowledgment of sources used in the creation of your computational artifact.

Guidelines for Completing the Explore Performance Task

You must:

- ▶ be aware of the performance task directions, timeline, and scoring criteria;
- ▶ support your written analysis of your computing innovation when responding to all the prompts by using details related to the knowledge and understanding of computer science you have obtained throughout the course and your investigation;
- ▶ provide evidence to support your claims using in-text citations;
- ▶ use relevant and credible sources to gather information about your computing innovation;
- ▶ provide acknowledgments for the use of any media or program code used in the creation of your computational artifact that is not your own; and
- ▶ allow your own interests to drive your choice of computing innovation and computational artifact.

You may:

- ▶ follow a timeline and schedule for completing the performance task;
- ▶ seek clarification from your teacher or AP Coordinator pertaining to the task, timeline, components, and scoring criteria;
- ▶ seek clarification from your teacher or AP Coordinator regarding submission requirements;
- ▶ as needed, seek assistance from your teacher or AP Coordinator in defining your focus and choice of topics; and
- ▶ seek assistance from your teacher or AP Coordinator to resolve technical problems that impede work, such as a failing workstation or difficulty with access to networks, or help with saving files or making movie files.

You may not:

- ▶ collaborate on the Explore performance task;
- ▶ submit work that has been revised, amended, or corrected by another individual;
- ▶ submit work from a practice performance task as your official submission to the College Board to be scored by the AP Program; or
- ▶ seek assistance or feedback on answers to prompts.

Performance Task: Create – Applications from Ideas

Overview

Programming is a collaborative and creative process that brings ideas to life through the development of software. Programs can help solve problems, enable innovations, or express personal interests. In this performance task, you will be developing a program of your choice. Your development process should include iteratively designing, implementing, and testing your program. You are strongly encouraged to work with another student in your class.

Please note that once this performance task has been assigned as an assessment (rather than as practice), you are expected to complete the task with minimal assistance from anyone other than your collaborative peer(s). For more clarification see the Guidelines for Completing the Through-Course Performance Tasks section.

You will be provided with a minimum of 12 hours of class time to complete and submit the following:

- ▶ **A video of your program running**
- ▶ **Individual written responses about your program and development process**
- ▶ **Program code**

Scoring guidelines and instructions for submitting your performance tasks are available on the AP Computer Science Principles Course Home Page.

Note: Students in nontraditional classroom environments should consult a school-based AP Coordinator for instructions.

General Requirements

This performance task requires you to develop a program on a topic that interests you or one that solves a problem. During the completion of this performance task, you will iteratively design, implement, and test your program. You will provide written responses to prompts about your program and specific program code that are significant to the functionality of your program. It is strongly recommended that a portion of the program involve some form of collaboration with another student in your class, for example, in the planning, designing, or testing (debugging) part of the development process. Your program development must also involve a significant amount of independent work writing your program code, in particular, algorithm(s) and abstraction(s) that you select to use as part of your written response to describe how the program code segments help your program run.

You are required to:

- ▶ independently develop an algorithm that integrates two or more algorithms and that is fundamental for your program to achieve its intended purpose;
- ▶ develop an abstraction that manages the complexity of your program;
- ▶ create a video that displays the running of your program and demonstrates its functionality;
- ▶ write responses to all the prompts in the performance task; and
- ▶ submit your entire program code.

Program Requirements

Your program must demonstrate a variety of capabilities and implement several different language features that, when combined, produce a result that cannot be easily accomplished without computing tools and techniques. Your program should draw upon mathematical and logical concepts, such as use of numbers, variables, mathematical expressions with arithmetic operators, logical and Boolean operators and expressions, decision statements, iteration, and/or collections.

Your program must demonstrate:

- ▶ use of several effectively integrated mathematical and logical concepts, from the language you are using;
- ▶ implementation of an algorithm that integrates two or more algorithms and integrates mathematical and/or logical concepts; and
- ▶ development and use of abstractions to manage the complexity of your program (e.g., procedures, abstractions provided by the programming language, APIs).

Submission Requirements

1. Video

Submit one video in .mp4, .wmv, .avi, or .mov format that demonstrates the running of at least one significant feature of your program. **Your video must not exceed 1 minute in length and must not exceed 30MB in size.**

2. Written Responses

Submit one PDF file in which you respond directly to each prompt. **Clearly label your responses 2a–2d in order. Your response to all prompts combined must not exceed 750 words, exclusive of the Program Code.**

Program Purpose and Development

2a. Provide a written response or audio narration in your video that:

- ♦ identifies the programming language;
- ♦ identifies the purpose of your program; and
- ♦ explains what the video illustrates.

(Must not exceed 150 words)

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. *(Must not exceed 200 words)*

2c. Capture and paste a program code segment that implements an algorithm (marked with an **oval** in **section 3** below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. *(Must not exceed 200 words)*

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a **rectangle** in **section 3** below). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. *(Must not exceed 200 words)*

3. Program Code

Capture and paste your entire program code in this section.

- › Mark with an **oval** the segment of program code that implements the algorithm you created for your program that integrates other algorithms and integrates mathematical and/or logical concepts.
- › Mark with a **rectangle** the segment of program code that represents an abstraction you developed.
- › Include comments or acknowledgments for program code that has been written by someone else.

Preparing for the Create Performance Task

Prior to your teacher administering this task, you should:

- ▶ Brainstorm problems that programming can address, or brainstorm special interests that programming can help develop;
- ▶ Ensure you understand the iterative nature of developing a computer program;
- ▶ Be prepared to collaborate with peers as necessary and in different ways;
- ▶ Ensure you are able to analyze program code and code segments and explain the function as it relates to the overall program;
- ▶ Know how to keep a programming journal of the design choices that you will make during the development of your program code and the effect of these decisions on the program's function. You can use this journal as a point of reference when demonstrating your understanding of how:
 - › an algorithm was built as part of the integration of two or more algorithms;
 - › a program functions differently with the inclusion of algorithms and abstractions;
 - › the inclusion of an abstraction has made their program code more compact, readable and/or reusable and how the program would operate differently without the inclusion of the abstraction;
- ▶ obtain programming support as necessary while practicing the skills needed to complete the performance task.

Guidelines for Completing the Create Performance Task

You must:

- ▶ be aware of the performance task directions, timeline, and scoring criteria;
- ▶ apply computer science knowledge you've obtained throughout the course when developing your program and in your explanation of its function;
- ▶ provide acknowledgment for program code used in your program that is not your own; and
- ▶ allow your own interests to drive your choice of program.

You may:

- ▶ follow a timeline and schedule for completing the performance task;
- ▶ seek clarification from your teacher or AP Coordinator pertaining to the task;
- ▶ seek clarification from your teacher or AP Coordinator regarding submission requirements;

- ▶ as needed, seek assistance from your teacher or AP Coordinator in defining your focus or choice of topics;
- ▶ seek assistance from your teacher or AP Coordinator to resolve technical problems that impede work, such as a failing workstation or difficulty with access to networks, or help with saving files or making movie files;
- ▶ obtain assistance from your teacher or AP Coordinator with the formation of peer-to-peer collaboration when completing the Create performance task;
- ▶ seek assistance from your teacher or AP Coordinator in resolving collaboration issues where one partner is clearly and directly impeding the completion of the Create performance task; and
- ▶ seek guidance from your teacher or AP Coordinator to use and acknowledge APIs or other pieces of open-source code. Program code not written by you can be used in programs as long as you are extending the project in some new way. You should provide acknowledgment and credit from program code you did not write.

You may not:

- ▶ collaborate on the video or any of the written responses;
- ▶ submit work that has been revised, amended, or correct by another individual, with the exception of cited program code;
- ▶ submit work from a practice performance task as your official submission to the College Board to be scored by the AP Program; or
- ▶ seek assistance or feedback on answers to prompts.




AP Computer Science Principles Exam Reference Sheet

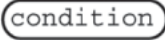
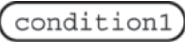
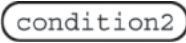


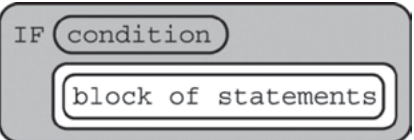
As AP Computer Science Principles does not designate any particular programming language, this reference sheet provides instructions and explanations to help students understand the format and meaning of the questions they will see on the exam. The reference sheet includes two programming formats: text based and block based.

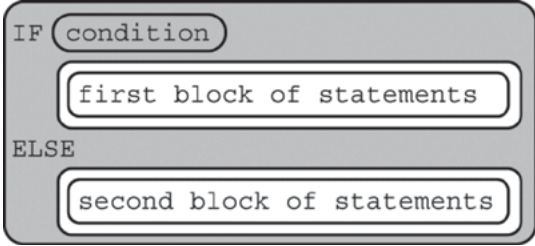
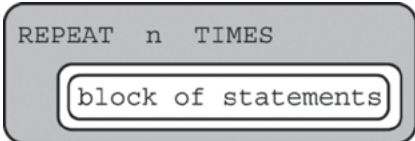
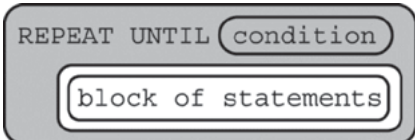
Programming instructions use four data types: numbers, Booleans, strings, and lists.

Instructions from any of the following categories may appear on the exam:



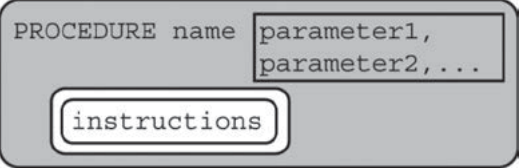
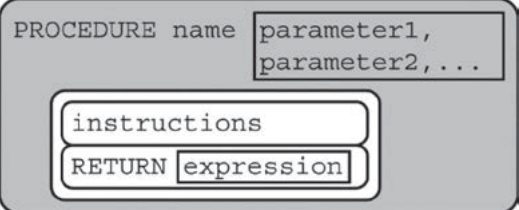
- ▶ Assignment, Display, and Input
- ▶ Arithmetic Operators and Numeric Procedures
- ▶ Relational and Boolean Operators
- ▶ Selection
- ▶ Iteration
- ▶ List Operations
- ▶ Procedures
- ▶ Robot





Instruction	Explanation
Assignment, Display, and Input	
Text: $a \leftarrow \text{expression}$ Block: 	Evaluates <code>expression</code> and assigns the result to the variable <code>a</code> .
Text: <code>DISPLAY (expression)</code> Block: 	Displays the value of <code>expression</code> , followed by a space.
Text: <code>INPUT ()</code> Block: <code>INPUT</code>	Accepts a value from the user and returns it.
Arithmetic Operators and Numeric Procedures	
Text and Block: $a + b$ $a - b$ $a * b$ a / b	The arithmetic operators <code>+</code> , <code>-</code> , <code>*</code> , and <code>/</code> are used to perform arithmetic on <code>a</code> and <code>b</code> . For example, <code>3 / 2</code> evaluates to <code>1.5</code> .
Text and Block: $a \text{ MOD } b$	Evaluates to the remainder when <code>a</code> is divided by <code>b</code> . Assume that <code>a</code> and <code>b</code> are positive integers. For example, <code>17 MOD 5</code> evaluates to <code>2</code> .
Text: <code>RANDOM (a, b)</code> Block: <code>RANDOM</code> 	Evaluates to a random integer from <code>a</code> to <code>b</code> , including <code>a</code> and <code>b</code> . For example, <code>RANDOM (1, 3)</code> could evaluate to <code>1</code> , <code>2</code> , or <code>3</code> .
Relational and Boolean Operators	
Text and Block: $a = b$ $a \neq b$ $a > b$ $a < b$ $a \geq b$ $a \leq b$	The relational operators <code>=</code> , <code>≠</code> , <code>></code> , <code><</code> , <code>≥</code> , and <code>≤</code> are used to test the relationship between two variables, expressions, or values. For example, <code>a = b</code> evaluates to <code>true</code> if <code>a</code> and <code>b</code> are equal; otherwise, it evaluates to <code>false</code> .

Instruction	Explanation
Relational and Boolean Operators (continued)	
Text: NOT condition Block: NOT 	Evaluates to true if condition is false; otherwise evaluates to false.
Text: condition1 AND condition2 Block:  AND 	Evaluates to true if both condition1 and condition2 are true; otherwise, evaluates to false.
Text: condition1 OR condition2 Block:  OR 	Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise, evaluates to false.
Selection	
Text: IF (condition) { <block of statements> } Block: 	The code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.

Instruction	Explanation
Selection (continued)	
<p>Text:</p> <pre>IF (condition) { <first block of statements> } ELSE { <second block of statements> }</pre> <p>Block:</p> 	<p>The code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise, the code in second block of statements is executed.</p>
Iteration	
<p>Text:</p> <pre>REPEAT n TIMES { <block of statements> }</pre> <p>Block:</p> 	<p>The code in block of statements is executed n times.</p>
<p>Text:</p> <pre>REPEAT UNTIL (condition) { <block of statements> }</pre> <p>Block:</p> 	<p>The code in block of statements is repeated until the Boolean expression condition evaluates to true.</p>

Instruction	Explanation
List Operations	
For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates.	
Text: <code>list[i]</code> Block: <code>list</code> <code>i</code>	Refers to the element of <code>list</code> at index <code>i</code> . The first element of <code>list</code> is at index 1.
Text: <code>list[i] ← list[j]</code> Block: <code>list</code> <code>i</code> ← <code>list</code> <code>j</code>	Assigns the value of <code>list[j]</code> to <code>list[i]</code> .
Text: <code>list ← [value1, value2, value3]</code> Block: <code>list</code> ← <code>value1, value2, value3</code>	Assigns <code>value1</code> , <code>value2</code> , and <code>value3</code> to <code>list[1]</code> , <code>list[2]</code> , and <code>list[3]</code> , respectively.
Text: FOR EACH item IN list { <block of statements> } Block: FOR EACH item IN list block of statements	The variable <code>item</code> is assigned the value of each element of <code>list</code> sequentially, in order from the first element to the last element. The code in block of statements is executed once for each assignment of <code>item</code> .
Text: <code>INSERT (list, i, value)</code> Block: <code>INSERT</code> <code>list, i, value</code>	Any values in <code>list</code> at indices greater than or equal to <code>i</code> are shifted to the right. The length of <code>list</code> is increased by 1, and <code>value</code> is placed at index <code>i</code> in <code>list</code> .
Text: <code>APPEND (list, value)</code> Block: <code>APPEND</code> <code>list, value</code>	The length of <code>list</code> is increased by 1, and <code>value</code> is placed at the end of <code>list</code> .

Instruction	Explanation
List Operations (continued)	
Text: REMOVE (list, i) Block: 	Removes the item at index <i>i</i> in <i>list</i> and shifts to the left any values at indices greater than <i>i</i> . The length of <i>list</i> is decreased by 1.
Text: LENGTH (list) Block: 	Evaluates to the number of elements in <i>list</i> .
Procedures	
Text: PROCEDURE name (parameter1, parameter2, ...) { <instructions> } Block: 	A procedure, <i>name</i> , takes zero or more parameters. The procedure contains programming instructions.
Text: PROCEDURE name (parameter1, parameter2, ...) { <instructions> RETURN (expression) } Block: 	A procedure, <i>name</i> , takes zero or more parameters. The procedure contains programming instructions and returns the value of <i>expression</i> . The RETURN statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling program.

Instruction	Explanation
Robot	
If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.	
Text: MOVE_FORWARD () Block: 	The robot moves one square forward in the direction it is facing.
Text: ROTATE_LEFT () Block: 	The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).
Text: ROTATE_RIGHT () Block: 	The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).
Text: CAN_MOVE (direction) Block: CAN_MOVE 	Evaluates to <code>true</code> if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to <code>false</code> . The value of direction can be <code>left</code> , <code>right</code> , <code>forward</code> , or <code>backward</code> .