

# **AP<sup>®</sup> Computer Science Principles**

---

# Student Handouts

Includes the Exam Reference Sheet

**Effective  
Fall 2020**

## AP COMPUTER SCIENCE PRINCIPLES

# Student Handouts

The following pages contain a student-directed version of the performance task guidelines that you can print out or copy to share with your students.

THIS PAGE IS INTENTIONALLY LEFT BLANK.

# Create Performance Task

---




Programming is a collaborative and creative process that brings ideas to life through the development of software. In the Create performance task, you will design and implement a program that might solve a problem, enable innovation, explore personal interests, or express creativity. Your submission must include the elements listed in the Submission Requirements section below.

You are allowed to collaborate with your partner(s) on the development of the program only. **The written response and the video that you submit for this performance task must be completed individually, without any collaboration with your partner(s) or anyone else.** You can develop the code segments used in the written responses (parts 3b and 3c) with your partner(s) or on your own during the administration of the performance task.

*Please note that once this performance task has been assigned as an assessment for submission to College Board, you are expected to complete the task without assistance from anyone except for your partner(s) and then only when developing the program code. You must follow the Guidelines for Completing the Create Performance Task section below.*

## General Requirements

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

-  **Final program code** (created independently or collaboratively)
-  **A video that displays the running of your program and demonstrates functionality you developed** (created independently)
-  **Written responses to all the prompts in the performance task** (created independently)

Scoring guidelines and instructions for submitting your performance task are available on the [AP Computer Science Principles Exam page](#) on AP Central.

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

# Submission Requirements



## 1. PROGRAM CODE (CREATED INDEPENDENTLY OR COLLABORATIVELY)

Submit one PDF file that contains all of your program code (including comments). Include comments or acknowledgments for any part of the submitted program code that has been written by someone other than you and/or your collaborative partner(s).

### IMPORTANT:

If the programming environment allows you to include comments, this is the preferred way to acknowledge and give credit to another author. However, if the programming environment does not allow you to include comments, you can add them in a document editor when you capture your program code for submission.

In your program, you must include student-developed program code that contains the following:

- Instructions for input from one of the following:
  - ◆ the user (including user actions that trigger events)
  - ◆ a device
  - ◆ an online data stream
  - ◆ a file
- Use of at least one **list** (or other **collection type**) to represent a collection of data that is stored and used to manage program complexity and help fulfill the program's purpose

### IMPORTANT:

The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the program code).

- At least one procedure that contributes to the program's intended purpose, where you have defined:
  - ◆ the procedure's name
  - ◆ the return type (if necessary)
  - ◆ one or more parameters

### IMPORTANT:

Implementation of built-in or existing procedures or language structures, such as event handlers or main methods, are not considered student-developed.

- An algorithm that includes sequencing, selection, and iteration that is in the body of the selected procedure
- Calls to your student-developed procedure
- Instructions for output (tactile, audible, visual, or textual) based on input and program functionality

### DEFINITION:

#### **List**

A **list** is an ordered sequence of elements. The use of lists allows multiple related items to be represented using a single variable. Lists may be referred to by different names, such as **arrays**, depending on the programming language.

### DEFINITION:

#### **Collection Type**

A **collection type** is a type that aggregates elements in a single structure. Some examples include lists, databases, and sets.

### IMPORTANT:

With text-based program code, you can use the print command to save your program code as a PDF file, or you can copy and paste your code to a text document and then convert it into a PDF file.

With block-based program code, you can create screen captures that include only your program code, paste these images into a document, and then convert that document to a PDF. Screen captures should not be blurry, and text should be at least 10 pt font size.



## 2. VIDEO (CREATED INDEPENDENTLY)

Submit one video file that demonstrates the running of your program as described below. Collaboration is **not** allowed during the development of your video.

Your video must demonstrate your program running, including:

- Input to your program
- At least one aspect of the functionality of your program
- Output produced by your program

Your video may NOT contain:

- Any distinguishing information about yourself
- Voice narration (though text captions are encouraged)

Your video must be:

- Either .mp4, .wmv, .avi, or .mov format
- No more than 1 minute in length
- No more than 30MB in file size



### 3. WRITTEN RESPONSES (CREATED INDEPENDENTLY)

Submit your responses to prompts 3a – 3d, which are described below. Your response to all prompts combined must not exceed 750 words (program code is not included in the word count). Collaboration is **not** allowed on the written responses. Instructions for submitting your written responses are available on the [AP Computer Science Principles Exam Page](#) on AP Central.

**3 a.** Provide a written response that does all three of the following:

*Approx. 150 words (for all subparts of 3a combined)*

i. Describes the overall purpose of the program

ii. Describes what functionality of the program is demonstrated in the video

iii. Describes the input and output of the program demonstrated in the video

- 3b.** Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.

*Approx. 200 words (for all subparts of 3b combined, exclusive of program code)*

- i.** The first program code segment must show how data have been stored in the list.

- ii.** The second program code segment must show the data in the same list being used, such as creating new data from the existing data or accessing multiple elements in the list, as part of fulfilling the program's purpose.

Then, provide a written response that does all three of the following:

- iii.** Identifies the name of the list being used in this response

- iv.** Describes what the data contained in the list represent in your program

- v.** Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list

**DEFINITION:**

**List**

A **list** is an ordered sequence of elements. The use of lists allows multiple related items to be represented using a single variable. Lists may be referred to by different names, such as **arrays**, depending on the programming language.

**DEFINITION:**

**Collection Type**

A **collection type** is a type that aggregates elements in a single structure. Some examples include lists, databases, hash tables, dictionaries, and sets.

**IMPORTANT:**

The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the program code).



- 3 c.** Capture and paste two program code segments you developed during the administration of this task that contain a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

*Approx. 200 words (for all subparts of 3c combined, exclusive of program code)*

- i.** The first program code segment must be a student-developed procedure that:
- Defines the procedure's name and return type (if necessary)
  - Contains and uses one or more parameters that have an effect on the functionality of the procedure
  - Implements an algorithm that includes sequencing, selection, and iteration

- ii.** The second program code segment must show where your student-developed procedure is being called in your program.

Then, provide a written response that does both of the following:

- iii.** Describes in general what the identified procedure does and how it contributes to the overall functionality of the program

- iv.** Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

**IMPORTANT:**

Built-in or existing procedures and language structures, such as event handlers and main methods, are not considered student-developed.

**3d.** Provide a written response that does all three of the following:

*Approx. 200 words (for all subparts of 3d combined)*

- i.** Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First call:

Second call:

- ii.** Describes what condition(s) is being tested by each call to the procedure

Condition(s) tested by the first call:

Condition(s) tested by the second call:

- iii.** Identifies the result of each call

Result of the first call:

Result of the second call:

THIS PAGE IS INTENTIONALLY LEFT BLANK.

# Guidelines for Completing the Create Performance Task

---

## AP Computer Science Principles Policy on Plagiarism

The use of media (e.g., video, images, sound), data, information, evidence, or program code created by someone else in the creation of a program and/or a program code segment(s), without appropriate acknowledgment (i.e., through citation, through attribution, and/or by reference), is considered **plagiarism**. A student who commits plagiarism will receive a score of 0 on the performance task.

To the best of their ability, teachers will ensure that students understand how to ethically use and acknowledge 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.

During the final submission process in the AP Digital Portfolio, students will be asked to attest that they have followed the performance task guidelines and have not plagiarized their submission.

## Preparing for the Performance Task

---

*Prior to beginning the performance task, you **should**:*

- Obtain content knowledge and skills that will help you succeed on the performance task. This can include, but needs not be limited to, the iterative development process, strategies for collaboration, the development of both data and procedural abstractions, and describing an algorithm's purpose and explaining how it functions. A development process includes exploration, investigation, reflection, design, implementation, and testing your program.
- Review the performance task directions and guidelines.
- Brainstorm problems that programming can address, or brainstorm special interests that programming can help develop.
- As needed, seek assistance from your teacher or AP Coordinator on defining your focus and choice of topics (e.g., by asking questions).
- Be prepared to collaborate with peers as necessary.
- Practice and discuss the entire performance task or individual prompts of the task.
- Review the role your teacher can and cannot play in providing assistance during the actual performance task, and take advantage of the opportunity to get assistance and feedback from your teacher during practice.
- Review the scoring guidelines to understand how your work will be assessed. The scoring guidelines align to the prompts in the performance task, so you must respond to all the prompts in your attempt to obtain the highest score possible. The scoring of practice performance tasks, such as those assigned via [AP Classroom](#), may differ from scoring of the performance task for the exam.
- Ensure you know the proper way to cite media or program code, including APIs or other pieces of open-source code, used in the creation of your program. Any media or program code that has not been written by you must be cited, and credit must be given to the author.
- Understand the level of detail expected in writing your responses by reviewing examples of performance task submissions at high, medium, and low levels according to the scoring guidelines. Examples of responses can be found on the [AP Computer Science Principles Exam page](#). If you submit a program that has been used as an example or was discussed in class, you must submit original responses to avoid plagiarism. You cannot submit any work from AP Central samples or practice performance tasks for your final submission.
- 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.

- Read through the [AP Digital Portfolio file submission requirements](#) and process, paying attention to the instructions concerning the file type, size, and length to be uploaded. This is important to ensure that work is sent for AP scoring. This process includes:
  - ♦ uploading your files to the correct part of the task
  - ♦ submitting your work as final
  - ♦ completing the College Board attestations to the originality of your work
- Practice creating a video of your program running, while adhering to file type, size, and length requirements.
- Practice creating a PDF file of your program code to submit for the performance task:
  - ♦ With text-based program code, you can use the print command to save your program code as a PDF file, or you can copy and paste your code to a text document and then convert it into a PDF file.
  - ♦ With block-based program code, you can create screen captures that include only your program code, paste these images into a document, and then convert that document to a PDF file. Screen captures should not be blurry, and text should be at least 10 pt font size.
- Understand that you may not revise your work once you have submitted it as final to the AP Digital Portfolio.

## Completing the Performance Task

---

### *You must:*

- Submit your performance task prior to the submission deadline, which can be found on the [AP Computer Science Principles Exam page](#) on AP Central.
- Follow a calendar or schedule that provides time for all performance task components to be completed and uploaded in advance of the deadline.
- Read the performance task directions and rubric.
- Apply the computer science knowledge you have obtained throughout the course, and when completing the performance task, to your responses to the prompts in the performance task.
- Use acceptable acknowledgment practices when using media (i.e., images, videos, sound), data sources, or program code created by others in your program code to avoid plagiarism. Any media or data sources that have not been created by you or your partner(s) must be acknowledged, and credit must be given to the author. Any program code which has not been written by you, including the use of APIs and open-source code should be acknowledged, and credit should be given to the author.
- Add comments to your program code to clarify the functionality of program code segments or to acknowledge and credit authors of media, data sources, or program code:
  - ♦ If the programming environment allows you to include comments, this is the preferred way to acknowledge and give credit to another author.
  - ♦ If the programming environment does not allow you to include comments, you can add them in a document editor when you capture your program code for submission.

---

### *Once you have started your official administration of the performance task, you may not:*

- Seek assistance in writing, revising, amending, or correcting your work, including debugging the program, writing or designing functionality in the program, testing the program, or making revisions to the program, from anyone other than your collaborative partner(s).
- Submit practice performance tasks or any work that has been revised, amended, or corrected by another individual, other than your collaborative partner(s) or cited program code, as a submission for AP Exam scoring.
- Seek answers or provide feedback on answers to prompts.
- Collaborate during the creation of your video or your written responses.
- Revise your work once you have completed and submitted it as final to the AP Digital Portfolio.

---

*Once you have started your official administration of the performance task, you **may**:*

- Collaborate with your partner(s) only during the ideation and development, including error testing, of your program code, if you choose to do so. NOTE: You are **not** allowed to collaborate on your video or individual written responses.
- Follow a timeline and schedule for completing the performance task.
- Seek assistance from your teacher or AP Coordinator on the formation of groups and resolution of collaboration issues when one collaborative partner is clearly and directly impeding the completion of the performance task.
- Seek clarification from your teacher or AP Coordinator on the prompts and submission requirements for the performance task when you do not understand the directions.
- Work on the performance task outside of designated class time.
- 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 to help with saving files or making movie files.
- Keep a programming journal of the design choices that were made during the development of the program code or code segment and the effect of these decisions on the program's function. You can use this journal as a point of reference when responding to writing prompts.



THIS PAGE IS INTENTIONALLY LEFT BLANK.

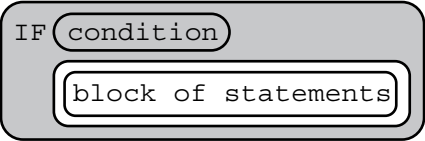
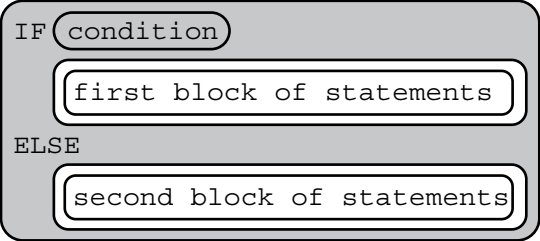
**AP COMPUTER SCIENCE PRINCIPLES**

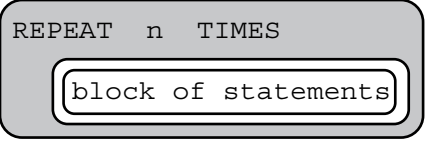

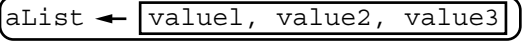
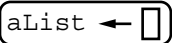
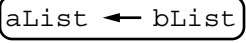
# **AP CSP Exam Reference Sheet**


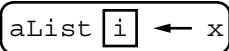
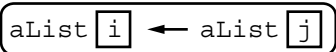
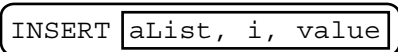


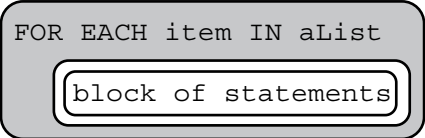
THIS PAGE IS INTENTIONALLY LEFT BLANK.

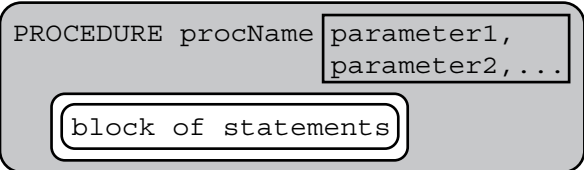
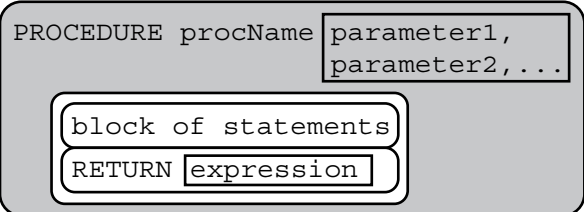


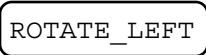
# Exam Reference Sheet

Instruction	Explanation
<b>Assignment, Display, and Input</b>	
Text: <code>a ← expression</code>  Block: <div style="border: 1px solid black; border-radius: 10px; padding: 5px; display: inline-block;"><code>a ← expression</code></div>	Evaluates <code>expression</code> and then assigns a copy of the result to the variable <code>a</code> .
Text: <code>DISPLAY(expression)</code>  Block: <div style="border: 1px solid black; border-radius: 10px; padding: 5px; display: inline-block;"><code>DISPLAY expression</code></div>	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 the input value.
<b>Arithmetic Operators and Numeric Procedures</b>	
Text and Block: <code>a + b</code> <code>a - b</code> <code>a * b</code> <code>a / b</code>	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>17 / 5</code> evaluates to <code>3.4</code> .  The order of operations used in mathematics applies when evaluating expressions.
Text and Block: <code>a MOD b</code>	Evaluates to the remainder when <code>a</code> is divided by <code>b</code> . Assume that <code>a</code> is an integer greater than or equal to <code>0</code> and <code>b</code> is an integer greater than <code>0</code> .  For example, <code>17 MOD 5</code> evaluates to <code>2</code> .  The <code>MOD</code> operator has the same precedence as the <code>*</code> and <code>/</code> operators.
Text: <code>RANDOM(a, b)</code>  Block: <code>RANDOM</code> <span style="border: 1px solid black; border-radius: 10px; padding: 2px 10px;"><code>a, b</code></span>	Generates and returns a random integer from <code>a</code> to <code>b</code> , including <code>a</code> and <code>b</code> . Each result is equally likely to occur.  For example, <code>RANDOM(1, 3)</code> could return <code>1</code> , <code>2</code> , or <code>3</code> .
<b>Relational and Boolean Operators</b>	
Text and Block: <code>a = b</code> <code>a ≠ b</code> <code>a &gt; b</code> <code>a &lt; b</code> <code>a ≥ b</code> <code>a ≤ b</code>	The relational operators <code>=</code> , <code>≠</code> , <code>&gt;</code> , <code>&lt;</code> , <code>≥</code> , and <code>≤</code> are used to test the relationship between two variables, expressions, or values. A comparison using relational operators evaluates to a Boolean value.  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
<b>Relational and Boolean Operators (continued)</b>	
Text: NOT condition  Block: NOT (condition)	Evaluates to true if condition is false; otherwise evaluates to false.
Text: condition1 AND condition2  Block: (condition1) AND (condition2)	Evaluates to true if both condition1 and condition2 are true; otherwise evaluates to false.
Text: condition1 OR condition2  Block: (condition1) OR (condition2)	Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise evaluates to false.
<b>Selection</b>	
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.
Text: IF(condition) { <first block of statements> } ELSE { <second block of statements> }  Block: 	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.

Instruction	Explanation
<b>Iteration</b>	
<p>Text:  REPEAT n TIMES  {  &lt;block of statements&gt;  }  Block:</p> 	<p>The code in <b>block of statements</b> is executed n times.</p>
<p>Text:  REPEAT UNTIL(condition)  {  &lt;block of statements&gt;  }  Block:</p> 	<p>The code in <b>block of statements</b> is repeated until the Boolean expression <b>condition</b> evaluates to true.</p>
<b>List Operations</b>	
<p>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.</p>	
<p>Text:  aList ← [value1, value2, value3, ...]  Block:</p> 	<p>Creates a new list that contains the values <b>value1</b>, <b>value2</b>, <b>value3</b>, and <b>...</b> at indices 1, 2, 3, and <b>...</b> respectively and assigns it to <b>aList</b>.</p>
<p>Text:  aList ← []  Block:</p> 	<p>Creates an empty list and assigns it to <b>aList</b>.</p>
<p>Text:  aList ← bList  Block:</p> 	<p>Assigns a copy of the list <b>bList</b> to the list <b>aList</b>.</p> <p>For example, if <b>bList</b> contains [20, 40, 60], then <b>aList</b> will also contain [20, 40, 60] after the assignment.</p>
<p>Text:  aList[i]  Block:  aList [i]</p>	<p>Accesses the element of <b>aList</b> at index <b>i</b>. The first element of <b>aList</b> is at index 1 and is accessed using the notation <b>aList[1]</b>.</p>

Instruction	Explanation
<b>List Operations (continued)</b>	
Text: <code>x ← aList[i]</code> Block: 	Assigns the value of <code>aList[i]</code> to the variable <code>x</code> .
Text: <code>aList[i] ← x</code> Block: 	Assigns the value of <code>x</code> to <code>aList[i]</code> .
Text: <code>aList[i] ← aList[j]</code> Block: 	Assigns the value of <code>aList[j]</code> to <code>aList[i]</code> .
Text: <code>INSERT(aList, i, value)</code> Block: 	Any values in <code>aList</code> at indices greater than or equal to <code>i</code> are shifted one position to the right. The length of the list is increased by 1, and <code>value</code> is placed at index <code>i</code> in <code>aList</code> .
Text: <code>APPEND(aList, value)</code> Block: 	The length of <code>aList</code> is increased by 1, and <code>value</code> is placed at the end of <code>aList</code> .
Text: <code>REMOVE(aList, i)</code> Block: 	Removes the item at index <code>i</code> in <code>aList</code> and shifts to the left any values at indices greater than <code>i</code> . The length of <code>aList</code> is decreased by 1.
Text: <code>LENGTH(aList)</code> Block: <code>LENGTH aList</code>	Evaluates to the number of elements in <code>aList</code> .
Text: <code>FOR EACH item IN aList</code> <code>{</code> <code>&lt;block of statements&gt;</code> <code>}</code> Block: 	The variable <code>item</code> is assigned the value of each element of <code>aList</code> sequentially, in order, from the first element to the last element. The code in <code>block of statements</code> is executed once for each assignment of <code>item</code> .

Instruction	Explanation
<b>Procedures and Procedure Calls</b>	
<p>Text:</p> <pre>PROCEDURE procName(parameter1,                     parameter2, ...)</pre> <pre>{   &lt;block of statements&gt; }</pre> <p>Block:</p> 	<p>Defines <code>procName</code> as a procedure that takes zero or more arguments. The procedure contains <code>block of statements</code>.</p> <p>The procedure <code>procName</code> can be called using the following notation, where <code>arg1</code> is assigned to <code>parameter1</code>, <code>arg2</code> is assigned to <code>parameter2</code>, etc:</p> <pre>procName(arg1, arg2, ...)</pre>
<p>Text:</p> <pre>PROCEDURE procName(parameter1,                     parameter2, ...)</pre> <pre>{   &lt;block of statements&gt;   RETURN(expression) }</pre> <p>Block:</p> 	<p>Defines <code>procName</code> as a procedure that takes zero or more arguments. The procedure contains <code>block of statements</code> and returns the value of <code>expression</code>. The <code>RETURN</code> statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling statement.</p> <p>The value returned by the procedure <code>procName</code> can be assigned to the variable <code>result</code> using the following notation:</p> <pre>result ← procName(arg1, arg2, ...)</pre>
<p>Text:</p> <pre>RETURN(expression)</pre> <p>Block:</p> 	<p>Returns the flow of control to the point where the procedure was called and returns the value of <code>expression</code>.</p>
<b>Robot</b>	
<p>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.</p>	
<p>Text:</p> <pre>MOVE_FORWARD ( )</pre> <p>Block:</p> 	<p>The robot moves one square forward in the direction it is facing.</p>
<p>Text:</p> <pre>ROTATE_LEFT ( )</pre> <p>Block:</p> 	<p>The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).</p>



Instruction	Explanation
<p>Text: ROTATE_RIGHT ( )</p> <p>Block:</p> <div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content; margin: 5px auto;">ROTATE_RIGHT</div>	<p style="text-align: center;"><b>Robot</b></p> <p>The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).</p>
<p>Text: CAN_MOVE (direction)</p> <p>Block:</p> <div style="border: 1px solid black; padding: 2px; display: inline-block;">CAN_MOVE <span style="border: 1px solid black; padding: 2px 10px;">direction</span></div>	<p>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 <code>direction</code> can be <code>left</code>, <code>right</code>, <code>forward</code>, or <code>backward</code>.</p>