

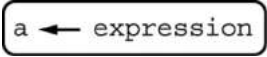
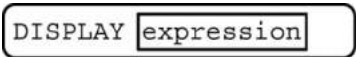
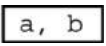

## Appendix 2: AP Computer Science Principles Exam Reference Sheet




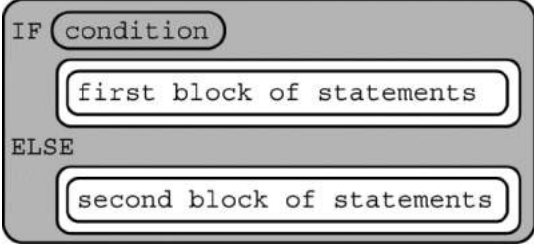
As AP<sup>®</sup> Computer Science Principles does not designate any particular programming language, this reference sheet provides instructions and explanations to help students understand the format and meaning of the questions they will see on the exam. The reference sheet includes two programming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

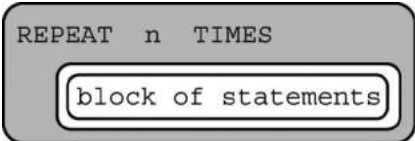
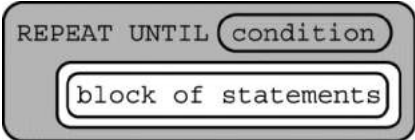

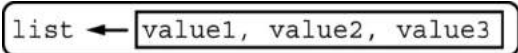
Programming instructions use four data types: numbers, Booleans, strings, and lists.

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

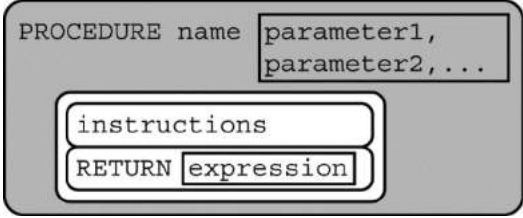



- ▶ Assignment, Display, and Input
- ▶ Arithmetic Operators and Numeric Procedures
- ▶ Relational and Boolean Operators
- ▶ Selection
- ▶ Iteration
- ▶ List Operations
- ▶ Procedures
- ▶ Robot

Instruction	Explanation
<b>Assignment, Display, and Input</b>	
Text: <code>a ← expression</code>  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.
<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>3 / 2</code> evaluates to <code>1.5</code> .
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> 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> .
<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.  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> .
Text: <code>NOT condition</code>  Block: <code>NOT</code> 	Evaluates to <code>true</code> if <code>condition</code> is <code>false</code> ; otherwise evaluates to <code>false</code> .

Instruction	Explanation
<b>Relational and Boolean Operators (continued)</b>	
<p>Text: condition1 AND condition2</p> <p>Block:  </p>	<p>Evaluates to true if both condition1 and condition2 are true; otherwise evaluates to false.</p>
<p>Text: condition1 OR condition2</p> <p>Block:  </p>	<p>Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise evaluates to false.</p>
<b>Selection</b>	
<p>Text: IF (condition) {     &lt;block of statements&gt; }</p> <p>Block:  </p>	<p>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.</p>
<p>Text: IF (condition) {     &lt;first block of statements&gt; } ELSE {     &lt;second block of statements&gt; }</p> <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>

Instruction	Explanation
<b>Iteration</b>	
<p>Text:  REPEAT n TIMES  {      &lt;block of statements&gt;  }</p> <p>Block:  </p>	<p>The code in block of statements is executed n times.</p>
<p>Text:  REPEAT UNTIL (condition)  {      &lt;block of statements&gt;  }</p> <p>Block:  </p>	<p>The code in block of statements is repeated until the Boolean expression condition 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:  list[i]</p> <p>Block:  list [i]</p>	<p>Refers to the element of list at index i. The first element of list is at index 1.</p>
<p>Text:  list[i] ← list[j]</p> <p>Block:  </p>	<p>Assigns the value of list[j] to list[i].</p>
<p>Text:  list ← [value1, value2, value3]</p> <p>Block:  </p>	<p>Assigns value1, value2, and value3 to list[1], list[2], and list[3], respectively.</p>

Instruction	Explanation
<b>List Operations (continued)</b>	
<p>Text:  <pre>FOR EACH item IN list {   &lt;block of statements&gt; }</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>FOR EACH item IN list   block of statements</pre> </div>	<p>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 <code>block of statements</code> is executed once for each assignment of <code>item</code>.</p>
<p>Text:  <pre>INSERT (list, i, value)</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>INSERT list, i, value</pre> </div>	<p>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>.</p>
<p>Text:  <pre>APPEND (list, value)</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>APPEND list, value</pre> </div>	<p>The length of <code>list</code> is increased by 1, and <code>value</code> is placed at the end of <code>list</code>.</p>
<p>Text:  <pre>REMOVE (list, i)</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>REMOVE list, i</pre> </div>	<p>Removes the item at index <code>i</code> in <code>list</code> and shifts to the left any values at indices greater than <code>i</code>. The length of <code>list</code> is decreased by 1.</p>
<p>Text:  <pre>LENGTH (list)</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>LENGTH list</pre> </div>	<p>Evaluates to the number of elements in <code>list</code>.</p>
<b>Procedures</b>	
<p>Text:  <pre>PROCEDURE name (parameter1,                   parameter2, ...)</pre> <pre>{   &lt;instructions&gt; }</pre> </p> <p>Block:</p> <div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <pre>PROCEDURE name parameter1,                 parameter2, ...   instructions</pre> </div>	<p>A procedure, <code>name</code>, takes zero or more parameters. The procedure contains programming instructions.</p>

Instruction	Explanation
<b>Procedures (continued)</b>	
<p>Text:</p> <pre>PROCEDURE name (parameter1,                 parameter2, ...)</pre> <pre>{     &lt;instructions&gt;     RETURN (expression) }</pre> <p>Block:</p> 	<p>A procedure, <code>name</code>, takes zero or more parameters. The procedure contains programming instructions 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 program.</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>
<p>Text:</p> <pre>ROTATE_RIGHT ()</pre> <p>Block:</p> 	<p>The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).</p>
<p>Text:</p> <pre>CAN_MOVE (direction)</pre> <p>Block:</p> <pre>CAN_MOVE <input type="text" value="direction"/></pre>	<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>