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 ← expression` Block: `a ← expression`	Evaluates `expression` and assigns the result to the variable `a`.

Text:
`DISPLAY (expression)`
Block:
`DISPLAY expression`
	Displays the value of `expression`, followed by a space.

Text:
`INPUT ()`
Block:
`INPUT`
	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 `+`, `-`, `*`, and `/` are used to perform arithmetic on `a` and `b`.
	For example, `3 / 2` evaluates to `1.5`.

Text and Block: `a MOD b`	Evaluates to the remainder when `a` is divided by `b`. Assume that `a` and `b` are positive integers. For example, `17 MOD 5` evaluates to `2`.

Text:
`RANDOM (a, b)`
Block:
`RANDOM a, b`
	Evaluates to a random integer from `a` to `b`, including `a` and `b`.
	For example, `RANDOM (1, 3)` could evaluate to `1`, `2`, or `3`.

Relational and Boolean Operators	

Text and Block:
`a = b`
`a ≠ b`
`a > b`
`a < b`
`a ≥ b`
`a ≤ b`
	The relational operators `=`, `≠`, `>`, `<`, `≥`, and `≤` are used to test the relationship between two variables, expressions, or values.
	For example, `a = b` evaluates to `true` if `a` and `b` are equal; otherwise, it evaluates to `false`.

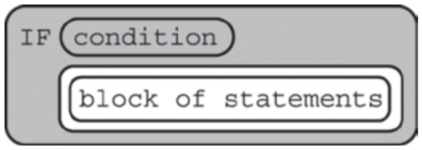
Instruction	Explanation
Relational and Boolean Operators (continued)	

Text:
`NOT condition`
 	
Block:
`NOT` `condition`
	Evaluates to `true` if condition is `false`; otherwise evaluates to `false`.

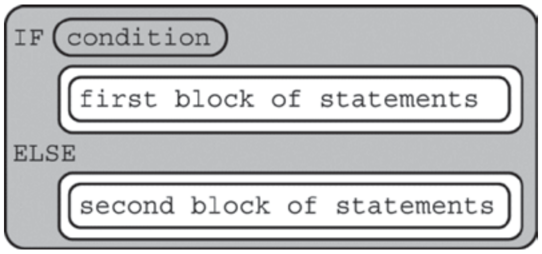
Text:
`condition1 AND condition2`
 	
Block:
`condition1` `AND` `condition2`
	Evaluates to `true` if both `condition1` and `condition2` are `true`; otherwise, evaluates to `false`.

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`.

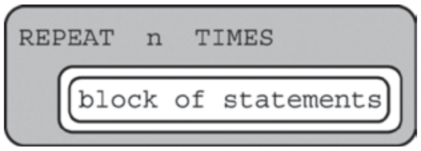
Selection	

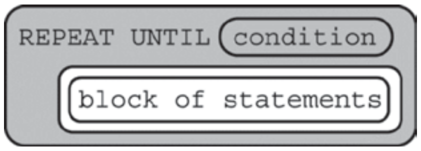
Text:
`IF (condition)`
`{`
`   <block of statements>`
`}`
 	
Block:

	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)	

Text:
`IF (condition)`
`{`
`   <first block of statements>`
`}`
`ELSE`
`{`
`   <second block of statements>`
`}`
Block:
 IF
condition
first block of statements
ELSE
second block of statements
	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.

Iteration	

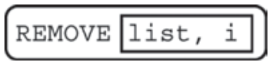
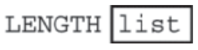
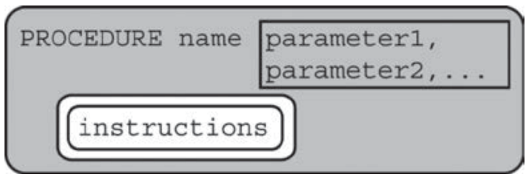
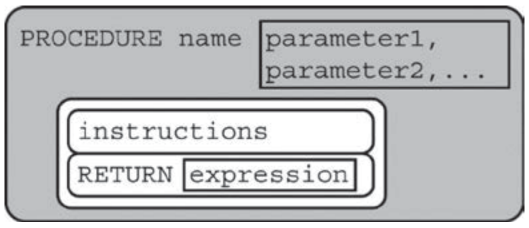
Text:
`REPEAT n TIMES`
`{`
`   <block of statements>`
`}`
Block:

	The code in `block of statements` is executed `n` times.

Text:
`REPEAT UNTIL (condition)`
`{`
`   <block of statements>`
`}`
Block:

	The code in `block of statements` is repeated until the Boolean expression `condition` evaluates to `true`.

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: `list[i]` Block: `list` $\boxed{\texttt{i}}$	Refers to the element of `list` at index `i`. The first element of `list` is at index `1`.
Text: `list[i] ← list[j]` Block: $\boxed{\texttt{list} \boxed{\texttt{i}} \leftarrow \texttt{list} \boxed{\texttt{j}}}$	Assigns the value of `list[j]` to `list[i]`.
Text: `list ← [value1, value2, value3]` Block: $\boxed{\texttt{list} \leftarrow \boxed{\texttt{value1, value2, value3}}}$	Assigns `value1`, `value2`, and `value3` to `list[1]`, `list[2]`, and `list[3]`, respectively.

Text:
`FOR EACH item IN list`
`{`
`   <block of statements>`
`}`
 	
Block:
$\boxed{\texttt{FOR EACH item IN list} \quad \boxed{\boxed{\texttt{block of statements}}}}$
	The variable `item` is assigned the value of each element of `list` sequentially, in order from the first element to the last element. The code in `block of statements` is executed once for each assignment of `item`.

Text: `INSERT (list, i, value)` Block: $\boxed{\texttt{INSERT} \boxed{\texttt{list, i, value}}}$	Any values in `list` at indices greater than or equal to `i` are shifted to the right. The length of list is increased by 1, and `value` is placed at index `i` in `list`.
Text: `APPEND (list, value)` Block: $\boxed{\texttt{APPEND} \boxed{\texttt{list, value}}}$	The length of `list` is increased by 1, and `value` is placed at the end of `list`.

Instruction	Explanation
List Operations (continued)	
Text: `REMOVE (list, i)` Block: 	Removes the item at index `i` in `list` and shifts to the left any values at indices greater than `i`. The length of `list` is decreased by 1.
Text: `LENGTH (list)` Block: 	Evaluates to the number of elements in `list`.
Procedures	
Text: `PROCEDURE name (parameter1, parameter2, ...)` `{` `<instructions>` `}` Block: 	A procedure, `name`, takes zero or more parameters. The procedure contains programming instructions.
Text: `PROCEDURE name (parameter1, parameter2, ...)` `{` `<instructions>` `RETURN (expression)` `}` Block:  PROCEDURE name parameter1, parameter2,... instructions RETURN expression	A procedure, `name`, takes zero or more parameters. The procedure contains programming instructions and returns the value of `expression`. 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:
`MOVE_FORWARD`
	The robot moves one square forward in the direction it is facing.

Text:
`ROTATE_LEFT ()`
Block:
`ROTATE_LEFT`
	The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).

Text:
`ROTATE_RIGHT ()`
Block:
`ROTATE_RIGHT`
	The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).

Text:
`CAN_MOVE (direction)`
Block:
`CAN_MOVE` `direction`
	Evaluates to `true` if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to `false`. The value of direction can be `left`, `right`, `forward`, or `backward`.