Create Performance Tasks Definitions/Clarifications

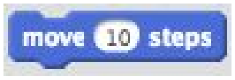
This glossary of terms is meant to help you understand words or phrases in the directions of the Performance Tasks that may be vague or difficult to comprehend.

Struggle/opportunity (Handout 5: Development) – a time in the coding of your program where something was difficult, made your program not work as intended, or created an error when your program was executed.

Resolution (Handout 5: Development) – the specific steps you took to overcome a problem or opportunity. Resolution should contain a step-by-step description so that a person unfamiliar with your program could resolve the same problem.

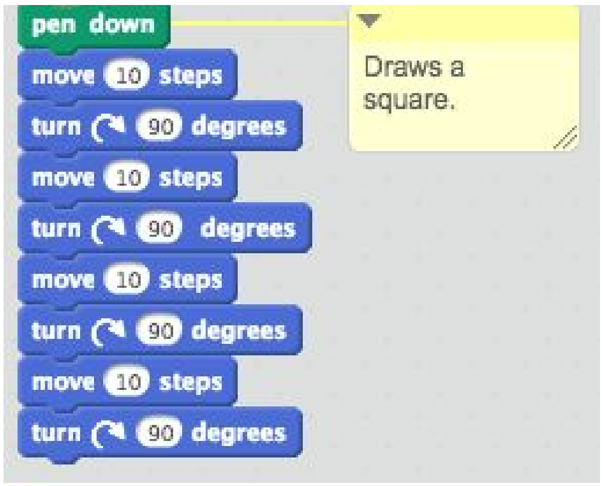
Algorithm (Handout 6: Algorithm) – a series of commands to execute a task. An algorithm must be able to accomplish a task independently.

Not an algorithm (as defined by AP CSP):



```
move 10 steps
```

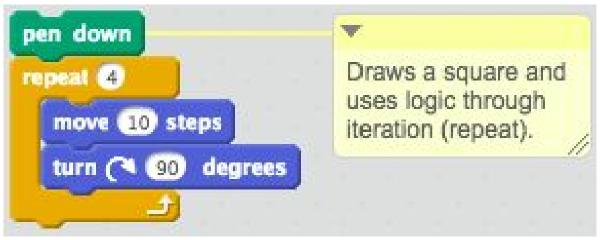
An algorithm, but does **NOT** include math or logic:



```
pen down
move 10 steps
turn ↻ 90 degrees
move 10 steps
turn ↻ 90 degrees
move 10 steps
turn ↻ 90 degrees
move 10 steps
turn ↻ 90 degrees
```

Draws a square.

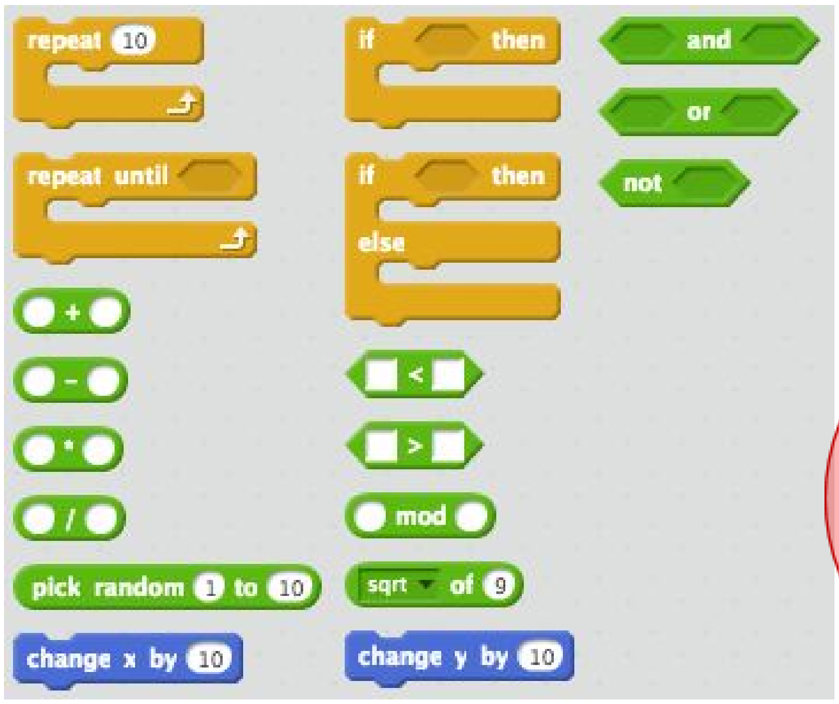
An algorithm that uses **math or logic** (what you want for these responses):

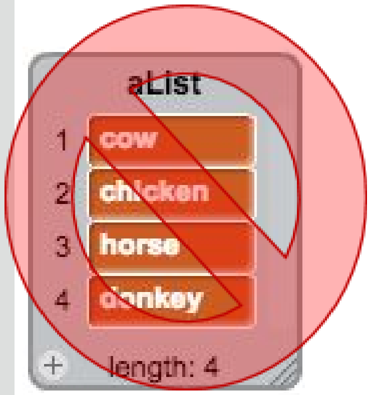


```
pen down
repeat 4
    move 10 steps
    turn ↻ 90 degrees
```

Draws a square and uses logic through iteration (repeat).

Acceptable forms of math or logic in Scratch:





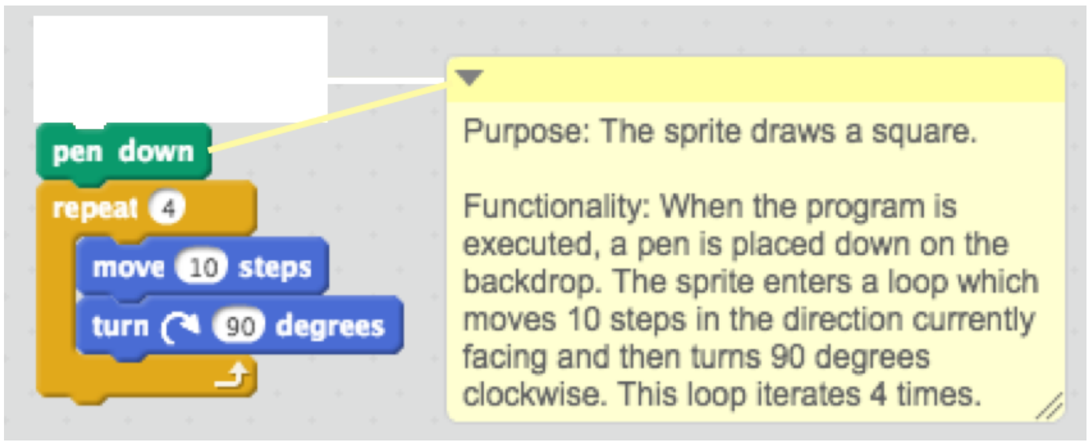
A list ( ) can be used as "math" or "logic" but must contain more than constants.

Acceptable forms of math or logic in Processing:

Operators	Calculation	Conditionals
% (modulo)	abs()	?: (conditional)
* (multiply)	ceil()	break
*= (multiply assign)	constrain()	case
+ (addition)	dist()	continue
++ (increment)	exp()	default
+= (add assign)	floor()	else
- (minus)	lerp()	if
-- (decrement)	log()	switch
-= (subtract assign)	mag()	**Logical Operators**
/ (divide)	map()	! (logical NOT)
/= (divide assign)	max()	&& (logical AND)
Relational Operators	min()	|| (logical OR)
!= (inequality)	norm()	**Iteration**
< (less than)	pow()	for
<= (less than or equal to)	round()	while
> (greater than)	sq()	
>= (greater than or equal to)	sqrt()	
	random()	

Purpose (Handout 6: Algorithm) – a description **what** the algorithm does in its entirety.

Functionality (Handout 6: Algorithm) – a description of **how** each line of code of a program works.

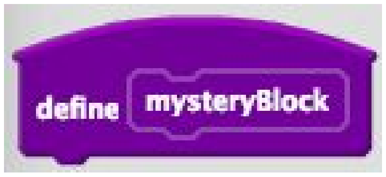
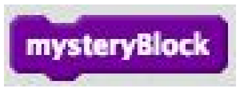
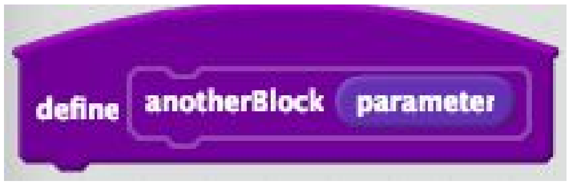
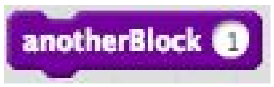
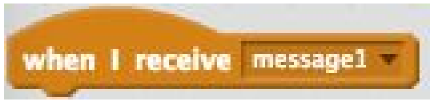
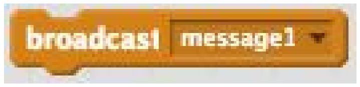
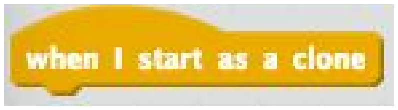
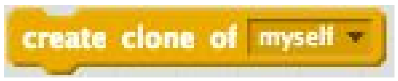


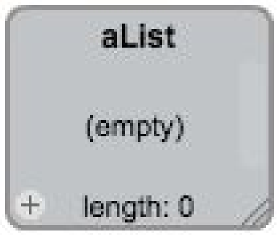
Purpose: The sprite draws a square.

Functionality: When the program is executed, a pen is placed down on the backdrop. The sprite enters a loop which moves 10 steps in the direction currently facing and then turns 90 degrees clockwise. This loop iterates 4 times.

Abstraction (Handout 7: Abstraction) – a technique for hiding the details of an aspect of a computer program. The abstraction can be related to control/procedures or data.

Abstractions in Scratch:

Definition	Call
 define mysteryBlock	 mysteryBlock
 define anotherBlock parameter	 anotherBlock 1
 when I receive message1	 broadcast message1
 when I start as a clone	 create clone of myself

 aList (empty) length: 0	aList

Abstractions in Processing:

Definition	Call

`void aFunction {`
`  statements`
`}`
	`aFunction()`

Procedural abstractions: For more information concerning procedural/control abstraction in Processing, see the Processing documentation on return (https://processing.org/reference/return.html).	
Data abstractions: For more information concerning data abstraction in Processing, see the Processing documentation on Arrays (https://processing.org/reference/Array.html).	