

Anatomy of Pseudo-Code for “Algorithmic” Composition

Vincent Giles, 2014.

All pseudo-code fonts should be Courier.

```
// Comment colour  
( ) parenthesis colour  
{ } curly parenthesis colour
```

Notes on the maths for people who need it:

=, equals: states that the figures on either side are equivalent.

$3 = 3$;

is the same as

$a = 3$, but, because a is a variable, this is a dynamic value that can change with each iteration of the process. However, you could define a as 3 earlier in the algorithm to yield the same result as above ($3=3$).

!=, does not equal: states that the figures are NOT equivalent, of limited use in this context

>, greater than: the figure on the left is greater than the figure on the right

\geq , greater than or equal to: the figure on the left is greater than or equal to the figure on the right.

<, less than: the figure on the left is less than the figure on the right

\leq , less than or equal to: same as above.

All mathematical operators and numbers are the same as anywhere else in mathematics. $3 < \pi < 4 > \pi$, for instance. I'm not sure how useful most of these will be in application, but they are available.

Syntax:

operator.function.action { statement of function }; // the colon indicates the end of the expression and a return to the next line of pseudo-code

operator: sound, silence, gesture, music, etc

function: what happens to the variables, to be defined by score

action: do, doUntil, doWhile, etc

Universal Variables

variable (a, b, c, d, etc) = definition; shortened as '**var**', ie. **var** a = pants; variables do not need to be defined, as they can be "rewritten" in the processes, and are called "dynamic variables"

static variables can be defined by the use of .static suffix, for example **var** a = noterow(a, a#, f#, c).static, these variable cannot be re-written.

Defining Calls/Functions

Function definitions expressed as mathematics; ie. gesture.function { definition }

If using multiple functions simultaneously, they can be defined by variable (separate variables from universal variables), ie. music.function(a) { }; music.function(b) { }; and so on.

Defining functions:

A function needs to be expressed as a process that can be "solved", even if only hypothetically. For instance, the end result may be $a=a^2$, indicating that the variable (a) has been re-written by process to now be the equivalent of itself times itself. If this were expressed in numbers, say, $a = 3$, $a = 3^2$ (or $a = 9$). When used in the context of improvised or non-improvised sonic material, this effectively defines the sonic result of processes. The preceding function lines may involve multiple operations, or simultaneous operations, or multiple functions. For instance:

```
music.function { a+b+c=d }; // { this defines the function to be performed AND
simultaneously, the process by which to do it, though this can be further
elaborated upon using the doWhile action. }
music.function.doUntil { a = d }
```

It would also be possible to further define the order of process using subroutines.

```
music.function { (a+b)+c=d }; // this states that you must add b to a first, then c
to the sum of b and a, and that this is equal to d.
```

Comments

Regular text can be added for any reasons, such as time markers, or additional instructions on sonic material, by prepending it inline with // See above for examples of this.

Thread 1

For any number of improvising performers

Vincent Giles, 2014

```
music.function { x = (x-x2)+(y+y3) };  
music.function.DoUntil;  
music.function { x = x+1 };
```

A stylized, handwritten signature in black ink, consisting of several overlapping loops and a long horizontal stroke at the bottom.

2014

Thread 2

For any number of improvising musicians

Vincent Giles, 2014

```
// Begin here
var x = sound;
var y = silence;

{
  sound.function(1);

  {
    sound.function.do(a = a+x-y);
    sound.function.doWhile(y/x2+a)=(y3)3;
    sound.function.doUntil(y=x2);
  }

  sound.function(0);
}
```

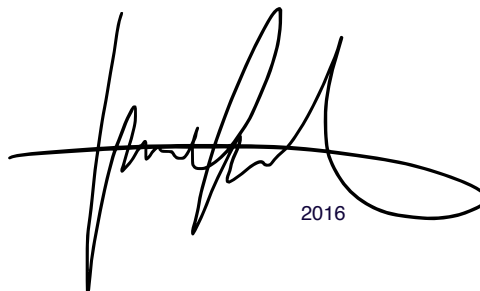


2014

For (a small quantity (20-30) of) massed instruments

Vincent Giles, 2016

```
(  
// Define variables – zeroed by default  
instQuant = 0; // quantity of performers  
instFreqn = 0; // frequency  
instSim(s) | s = 0 |; // similarity value (a value of 1 means that the similarity is  
high // bordering on exact)  
instAmpn = 0; // amplitude  
instRhyn = 0; // rhythm OR gesture  
instFXn = 0; // timbral technique  
  
// The algorithm  
  
set instQuant(n.performers);  
process = {  
  process.doLoop {  
    instQuant(instFreqn, instAmpn, instRhyn, instFXn); // each performer  
independently sets their parameters in real time on each "loop"  
    process.doLoopWhile(instFreqn(n) != instSim(1));  
    process.doLoopUntil (instFreqn(n) == instSim(1), then process.stop);  
  }  
}  
process.start;  
)
```



2016